CSC111 Course Project Report

Stockify App

Mahe Chen, Aarya Bhardawaj, Matthew Yu and Daniel Cheng

April 1, 2023

Problem Description and Research Question

- **Background and Overview**

  "You pay a very high price in the stock market for a cheery consensus" - Warren Buffet.

  A stock is the division of ownership of a corporation or company through shares. The stock market is a complex network of exchanges where investors and traders buy and sell shares of publicly traded companies (Stock market 101). It has been a traditional source of funding for companies to expand their businesses and for investors to receive dividends in return for sharing ownership of companies. The stock market is a dynamic and ever-changing place where stock values for different companies rise and fall due to various factors. This makes it a potentially profitable but risky investment option. To make the most informed decisions with their money, stock traders often require guidance.

- **Motivation**

  The stock market offers ample opportunities, yet not everyone takes advantage of them. For many, investing in stocks can be an intimidating prospect, even for seasoned investors. Determining whether a stock is worth investing in is not always straightforward, and predicting its future performance can be challenging. It requires a considerable amount of analysis and investment management to make informed investment decisions.

  To assist beginners in navigating the stock market and making sound investment choices, we have designed a program that is specifically tailored for this purpose. Our program allows users to limit their search by stock sector, value, risk, and other characteristics, and provides clear recommendations of the most fitting stock purchases, along with detailed information about these stocks.

  Our primary objective is to create an application that recommends optimal stocks based on the user's preferences. Whether you are new to investing or looking for a simpler way to manage your investments, our program is an effective solution that can help you achieve your investment goals.

Datasets

- **stock_data.csv, stock_symbols.csv**

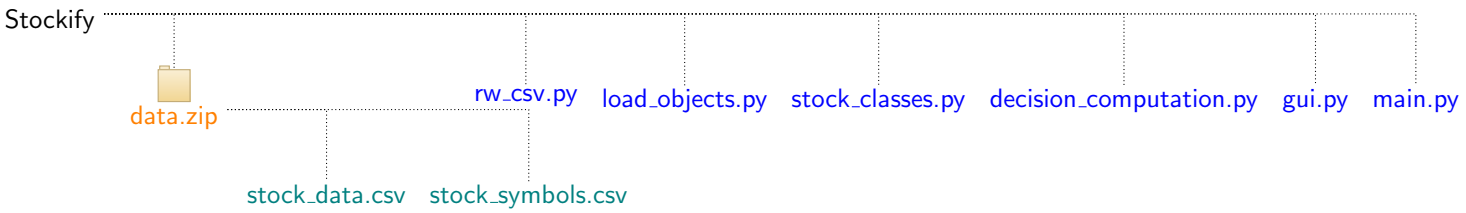
  To create an effective dataset for our stock recommender, we utilized the Python requests library and web scraped data on a total of 3568 stocks from Yahoo Finance. After filtering out stocks with insufficient data, we were left with 3505 stocks. We began by storing a list of all 3505 stock symbols, which were extracted from a CSV file of large cap stocks from the US stock market. Next, we used a function to web scrape relevant stock information for each stock, by iterating over the list of stock symbols and using the requests.get function to change the "stock" portion of the URL. The write_stock_data function recorded all the relevant attributes for each stock and stored this information in a CSV file. This resulted in a dataset containing all 3505 stocks and their respective attributes.

  For each stock, we recorded a total of 24 attributes, with most of them being about the company rather than the stock itself. We included information such as debt to equity ratio, return on equity ratio, and profit margins, as these attributes are crucial in evaluating a stock's future potential. We focused on

computing 7 scores, value score, growth score, quality score, consistency score, risk score, ESG score, and dividend score using the extracted data. To accomplish this, we implemented a load csv function to read all the data from the generated CSV file and store all the instance attributes for each stock. This was the final step in our data accumulation process. By using this method, we can have up-to-date stock information and easily update our dataset in real-time by running our function again.

Sample row of the csv file with the first few headers:							
symbol	sector	name	address	previous	dividend	high	low
AAPL	Technology	Apple	Cupertino	160.77	0.0057	178.49	124.17

Computational Plan



- **Data Wrangling: stock_classes.py**

  This file contains the StockData, SectorAverage, and the StockScore classes. The StockData class contains all the attributes extracted from the stock_data.csv file. These attributes are used to compute the 7 scores our stock recommender will use to recommend a stock based on user preferences. This file contains 7 'get score' functions, these are the get_value_score, get_growth_score, getvquality_score, get_consistency_score, get_risk_score, get_esg_score, and get_dividend_score functions. The formulas for each function were created by referencing information from Investopedia and The Motley Fool. The scores are calculated using the following formulas:

  Value score:

  The value score is computed by multiplying the price to earning ratio by the price to book ratio. This will provide an evaluation of the stock based on expectations for future growth and the market value of a company's shares.

  Growth score:

  The growth score is computed by dividing the earnings before interest, taxes, depreciation, and amortisation by companies sales (ebitda) by the company's total total sales and then multiplying that value by the companies earnings growth. Dividing ebitda by sales will provide insight on how financially stable a company is and then multiplying by a company's earnings growth will provide insight on how likely the company stock is projected to grow in the near future.

  Quality score:

  The quality score is computed by dividing a company stock's profit margin by its return on equity ratio and then dividing by its debt to equity ratio. This gives a quality score on the stock based on how risky and profitable the company's investments are for that stock.

  Consistency score:

  The consistency score is computed by subtracting the stocks fifty two week high by the fifty two week low, and then dividing by the two hundred day average. This value will give insight on how consistent the stock has been performing over the last fiscal year. A higher score will mean that the stock price is not jumping up or down.

  Risk score:

  The risk score is computed by dividing the total cash by the current liabilities. This measures how risky it would be to invest in this stock based on the company's financial status. A higher risk score indicates that the company has significantly higher amounts of current liabilities, short term debt, compared to their total cash at hand. This means the stock has a higher chance of going into bankruptcy due to insufficient cash.

ESG score:

The esg score is computed by taking the average of the company's social score, environmental score, and governance score. By taking the average of a company stocks social, environmental, and governance score, this esg score is able to represent how well a company is able to address risks with respect to these attributes. A higher esg score means the company is able to effectively address the risk.

Dividend score:

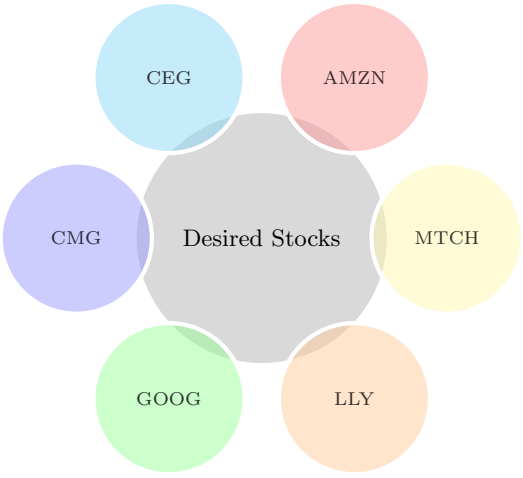
The dividend score is directly an attribute of the stock StockData class, thus there is no need to compute this score. The dividend score is simply the dividend yield of a company stock. A high dividend score indicates that the company is returning a lot of its profits to investors.

The StockScore class is then used to record these respective value scores for each stock. The SectorAverage class is used to record the sector average of all the data which may have missing values. These are the attributes with an 'Optional' type annotation in the StockData class. If these attributes have missing values and are 'None', then we will use the get_avg_of_sector function to compute the average value of that respective attribute for all stocks in the same sector as the one with the missing data. This value is then used in the computation in the 'get score' functions.

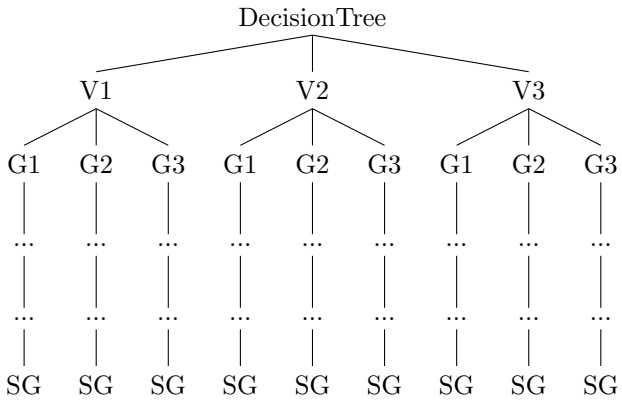
- **Data Computation: decision_computation.py**

Our program will rely on decision trees to provide the stock recommendations. Each root(excluding the root of the main tree) will hold some question as an attribute. The subtrees for each internal root correspond to the possible answers the decision may branch into. The path taken depends on the user's answer to that question, with the leaves at the end of the tree representing the resulting stock(s) from taking that path down the tree. This file contains the DecisionTree class and the _StockValue class. The decision tree uses the scaled function to convert all the score attributes into skill attributes. These skill attributes are values from 1 - 3 for each stock. This is used to determine which way to go down the tree based on the user preference. The main recursive function to generate the decision tree is the insert_stock_sequence function. This function adds all the stocks into the decision tree. The _StockValue is a recursive class which holds a stock in a StockGraph. The StockGraph class is a recursive class which contains the last node of the tree. Accessing the neighbours returns a list of all recommended stocks.

Our Decision Factors:			
Stock Value	Stock Growth	Stock Quality	Consistency
Social Responsibility	Dividend	Financial Risk	Sectors



StockGraph



DecisionTree

Two diagrams that illustrate how DecisionTree Class and StockGraph Class works in our backend.

- **Data Visualization: gui.py**

  The user interface is implemented in the `gui.py` file using the `Pygame` module. `Pygame`'s `Surface` objects and the `display` module's `update` method are essential for displaying the interface on the user's screen. In the *_set_dimension_variables* function, the interface adjusts the size of buttons and positions of text according to the user's screen size.

  To initialise the buttons for each panel screen, the program uses several initialisation functions. Buttons are objects that collect user input and trigger the program's response to the input. There are various button classes, each with unique behaviours. For instance, a Checkbox can be selected to record the user's choices, while a NaviButton allows the user to navigate between pages when clicked. Specialty buttons, such as the NextStockButton or SelectAllButton, have been implemented to tie our user interface to decision tree computations or allow users to select all options with a single click.

  Each button uses `pygame.Rect` objects to represent the button's actual display on the user interface. Additionally, the `ButtonGroup` class has been created to organise the checkboxes for the select-one page, ensuring only one button is selected for each question.

  The contents are displayed on the screen within a continuously running while loop. When a change is made, the program redisplay the contents on the screen. However, during panel transitions, an *_fade_out* function has been implemented to display a smoother fade-in and fade-out transition. The *get_pos* function from the `pygame.mouse` module helps us determine the user's mouse position, while the `pygame.event` module allows us to determine when the user left clicks. By combining these two, we can determine when the user clicks on a button and run the button's action method and other program behaviours accordingly.

  The program uses modules such as `pygame.font` and `pygame.image` to enhance communication and visual appeal. Additionally, functions such as *draw_text*, *_center_coordinates*, and *_split_str_to_paragraph* have been implemented to display nicely formatted and centred text, as well as buttons.

  When the program initialises, the user will first be greeted by a loading screen. The *_loading_screen* function has been implemented to display loading screens to the user interface anytime the program may take time for computations. From the main menu, the user may start their search, navigate to the instructions or credits panel, or update the stocks in the csv file.

  When the user conducts their search, they select their preferences through the checkboxes, and the results containing the recommended stock information are displayed on the results screen. If there are multiple recommended stocks, the user can view each one by clicking the NextStockButton. The ResetButton allows the user to clear their search and begin a new one. Finally, the Quit button allows the user to quit the program anytime.

User Instructions

- **Requirements**

  1. Install all required Python libraries in the requirements.txt file.

  2. Download all files in our repository. Inside data.zip, there should be 2 csv files: stock_data.csv and stock_symbols.csv. Please save this folder on the same level as the other .py files, which are rw_csv.py, load_objects.py, stock_classes.py, decision_computation.py, gui.py and main.py

- **Initializing the program**

  1. Open Pycharm and run the main.py file in the Python console.

  2. Your screen should be filled with a Pygame screen. This screen may say "Loading...". Anytime you encounter this screen (A), please be patient - the program takes time to run.

  Note: Your screen must be at minimum 1000 x 600 for our program to properly fill the screen. If it is smaller, a 1000 x 600 screen will display(the window will be too large for your screen).

  You should eventually be brought to the main screen (B). You have a lot of options here! We'll briefly explore each one.

Click ‘Instructions’ to be brought to the instructions panel (C). You’ll find a brief description of the program here, as well as definitions for the stock qualities. Click ‘Back’ to return to the main screen.

Click ‘Credits’ to view the credits panel (D). Click ‘Back’ to return to the main screen.

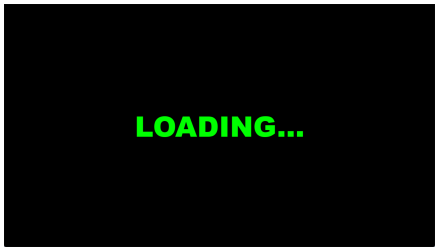
Click ‘Update Stocks’ to update the stock data file to real-time, updated data gathered from Yahoo Finance. You should be brought to a confirmation page (E).

Click ‘Confirm’ to continue with the process, where you will be greeted with a loading screen and automatically returned to the menu once the process is finished. (Note1: If the program crashes, it may indicate a connection issue. To resolve the issue, try rerunning the ’update stocks’ function until it works. Alternatively, you can download the data folder again.)

Please also note that updating the stocks may take a significant amount of time, typically 10 or more minutes. Exiting the program prematurely may result in a corrupt CSV file. Additionally, once the update process finishes, the program will close automatically instead of returning you to the main menu.

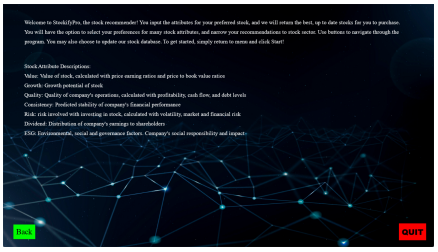
Click ‘Search’ to begin your search! Walkthrough below.

Click the ‘Quit’ button in the bottom right corner anytime you wish to exit the program. (F)



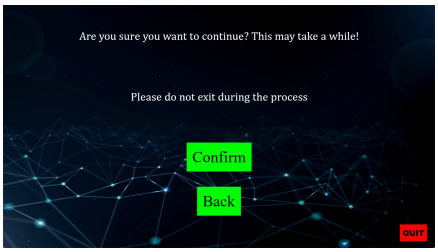
A

B



C

D



E

F

- **Completing a Search**

  You should first be brought to a screen with a table of checkboxes (G). As indicated by the title, select one checkbox for each row. A checkbox will turn yellow when selected. Once each row has a selection, the 'Next' button should become enabled and you can continue on to the next step.

  Next, you'll be brought to another screen with checkboxes and labels (H). As indicated by the title, select the checkbox corresponding to each sector if you wish to receive recommendations for that sector. You can use the 'Select All' and 'Deselect All' buttons at the bottom of the screen to quickly select or deselect all checkboxes. Now just click 'Results' to get your results!

  Recommended stock information will be displayed (I)(J). If there are no recommended stocks, then no stocks will be displayed. If there is more than one recommended stock, you can use the arrows in the bottom left corner to navigate between your recommended stocks. Click 'Restart' to clear your search and return to the menu.

G

H

I

J

Revised Project Plan

We chose to not display a bubble diagram of each stock connected to its respective sector. Although we initially thought it would be a fun visualisation, we decided it would be more useful to display the actual stock information. To incorporate more freedom, we wanted to allow the user to navigate between recommended stocks as they choose instead of displaying all of them at once. Nonetheless, we still incorporated the graphs in the backend as described in our procedure for computation.

We also decided to webscrape updated information on all large cap stocks from Yahoo Finance to create our dataset. This helps improve the practicality of our program, keeping it relevant with the changing stock market. Originally, we planned to use a pre-created data set from the internet. However, we found this not to be feasible as most of the datasets did not contain all the large cap stocks in the USA and were not up to date. For the application of our stock recommender, it would not make sense to be assessing a stock based on its stock value and company information from the past. Therefore, we decided it would be best to webscrape our stock information from Yahoo Finance and create our own csv file to store the data. With this data, we tested the implementation of unique trees with different split conditions. We ended up keeping the tree that evenly split stocks amongst attribute groups while still being realistic and truthful with its categorization.

Since we deferred from the idea of using a pre constructed data set, we also decided not to use pandas. This is mainly because our data collection process became much more complex and required more steps. We did consider using pandas as a way to visualise the results in a tabular form, but ended up deciding not to in order to keep the user interface more consistent.

Finally, instead of having 10 yes-or-no factors, we decided to use 7 factors where the user selects from either low, medium or high and a sector selection where the user selects all they desire. We felt 10 questions was a bit excessive, and providing 3 options for each factor instead of 2 would create a more tailored recommendation.

Discussion

Our program achieves the project goal; it takes in the user's preferences and outputs the best stocks for them to buy! It even has the ability to update the dataset to the most recent stock information, solving

our project goal for the future too. All these components are delivered through an interactive and visually appealing user interface.

However, we still ran into several challenges during development. Some stocks from the dataset had missing values(in particular ESG scores), so we resorted to using the sector average value to fill in these gaps. However, this could lead to some stock misrepresentation from inaccurate score calculations. Then, the stock symbols are from our current csv file. This means that the stock symbols will not become updated in the case that new stocks get upgraded to large cap stocks or existing large cap stocks become downgraded to mid cap. We also had to learn how to webscrape from the current Yahoo websites to form our datasets. If these websites change their format, we'd have to update our code to properly extract the data again. Finally, we also had some time concerns over how long it took to web scrape the data.

Since our stock recommender is heavily dependent on the data set, it limits the types of stocks we can recommend to the user. Although it is possible to use web scraping to acquire a larger dataset with more stocks, the algorithm may not be as effective with a mix of different types of stocks. More specifically, our stock recommender is mainly built to recommend large cap stocks. It wouldn't necessarily be a problem if we incorporate stocks from other countries into our data set as long as they are large cap stocks. The problem arises if they are not large cap stocks. For example if we were to add penny stocks, our recommender would not be as nearly as accurate as our score values we use to recommend a stock would be all over the place and inconsistent as we designed the algorithm to focus on large cap stocks. Then, despite our efforts to create decision trees that divided the stocks over a large variety of user inputs, there are several instances where the user would get no recommended stocks based on their input criteria.

Unfortunately, one thing we couldn't adjust to the user's screen size were the text sizes, as the font sizes had to be inputted manually while displaying texts. As a next step, we could try creating helper functions that adjust text sizes so we get a consistently sized display no matter the screen size. Another practical extension would be reducing the amount of cases where no stock is recommended. To do this, we could try calculating and adjusting our decision tree for a more even distribution of stocks, or increase the number of stocks in our dataset by web scraping from alternative sites(this approach would further increase the time to update our csv files though). Finally, we could also add the initially planned visual graph as an addition to our recommended stocks. Although it wouldn't provide specific information, the graph would make a nice general overview between related stocks.

References & Acknowledgments

- Mooney, P. (2023, January 16). Stock market data (NASDAQ, NYSE, $S\&P500$). Kaggle.
  `https://www.kaggle.com/datasets/paultimothymooney/stock-market-data`
- Pinkasovitch, A. (2022, August 2). How to pick a stock: Basic best practices for new investors. Investopedia.
  `https://www.investopedia.com/articles/basics/11/how-to-pick-a-stock.asp`
- 5g networking technology background with blue digital line Free Vector. (2021, July 25). Freepik.
  `https://www.freepik.com/free-vector/5g-networking-technology-background-with-blue-digital-line_16406299.htm`
- Elmerraji, J. (2022, June 6). Analyze investments quickly with ratios. Investopedia.
  `https://www.investopedia.com/articles/stocks/06/ratios.asp`
- Stock market 101. What is Stock Market? (n.d.).
  `https://www.td.com/ca/en/investing/direct-investing/articles/what-is-stock-market`.
- Wilkins, G. (2022, November 10). 6 basic financial ratios and what they reveal. Investopedia.
  `https://www.investopedia.com/financial-edge/0910/6-basic-financial-ratios-and-what-they-tell-you.aspx`
- Stock Datasets
  `https://query2.finance.yahoo.com/v10/finance/quoteSummary/+stock+?modules=summaryProfile%2CsummaryDeta2CesgScores%2CincomeStatementHistory%2CdefaultKeyStatistics%2CfinancialData%2CbalanceSheetHistory%2Cprice`

Made with LaTeX