Vrije Universiteit Amsterdam
Computational Thinking
Project Assignment: iTrade

Group number: 54.
Members with student numbers:
Abdoellah, Kyan 2849022
Bakker, Skick 2862824
Meijerink, Jasper 2862115
Moustafa, Abir 2863783
Tran, Demy 2839607

Date: 15-12-2024

Context Task

Over the past few years, we've become more dependent on technology. With the advancement of this technology, we need more computing power. Enter Moore's Law, which, as Kirilenko and Lo (2013) explain, observes that every 2 years the number of transistors on a microchip doubles with minimal increase in cost. This means an exponential growth of computing power every 2 years. While this sounds amazing, we should keep track of Murphy's Law as well, which states in its technology-specific corollary: '*Whatever can go wrong, will go wrong faster and bigger when computers are involved*'.

The financial industry is a lot more dependent on human behavior. A computer algorithm might be able to see historical patterns, but will struggle with real time changes, like bitcoin rising due to the election of Donald Trump, or investments based on an influencer's opinion. Besides that, people oftentimes make decisions based on emotions such as fear or greed, while an algorithm is blind to such things.
An algorithm could suggest investing in a stock that could be deemed risky, which will lead to people not investing, which causes a loss on your end. On the other hand, an algorithm could advise selling your stock, but others might hope that it'll rise more and invest more. This would lead to missed gains due to the greed of others.

Algorithmic day trading might seem like a very powerful tool at first, but it has many flaws that are very dangerous for your pockets. I think it's very nice that technology gives the opportunity to help with finance and the stock market, but I don't like the thought of people becoming dependent on it, since it's much more likely to lead to a bad outcome, if we're to believe Murphy's Law.

The problem

When you're a new trader, it can be quite difficult to start. There are so many different industries, each with their own brands. They are all fluctuating in ways that are impossible for a beginner to understand. And not only that, what if you want to make sure the company that you are investing in is good for the environment? What is the average opinion of this company? How much is the company's value affected by governance factors?

There are way too many complications to keep track of when you start trading. However, we can help these novice traders by having them use *iTrade*. This algorithm looks at the performance of the individual stocks within the large stock market and gives recommendations to the user based on the preference settings that can be given by the user. The system makes use of a database which includes the information of about 500 different stocks. Information can always be added to this database to ensure more secure choices.

Simply want to know which top 10 stocks are currently the most profitable? No problem! Here is a list of the 10 most profitable stocks. It'll also tell you the industry of the stocks, the foundation year and an average score based on how socially responsible the stock is. With the help of *iTrade*, even beginning traders can have a shot at making a profit within the financial industry.

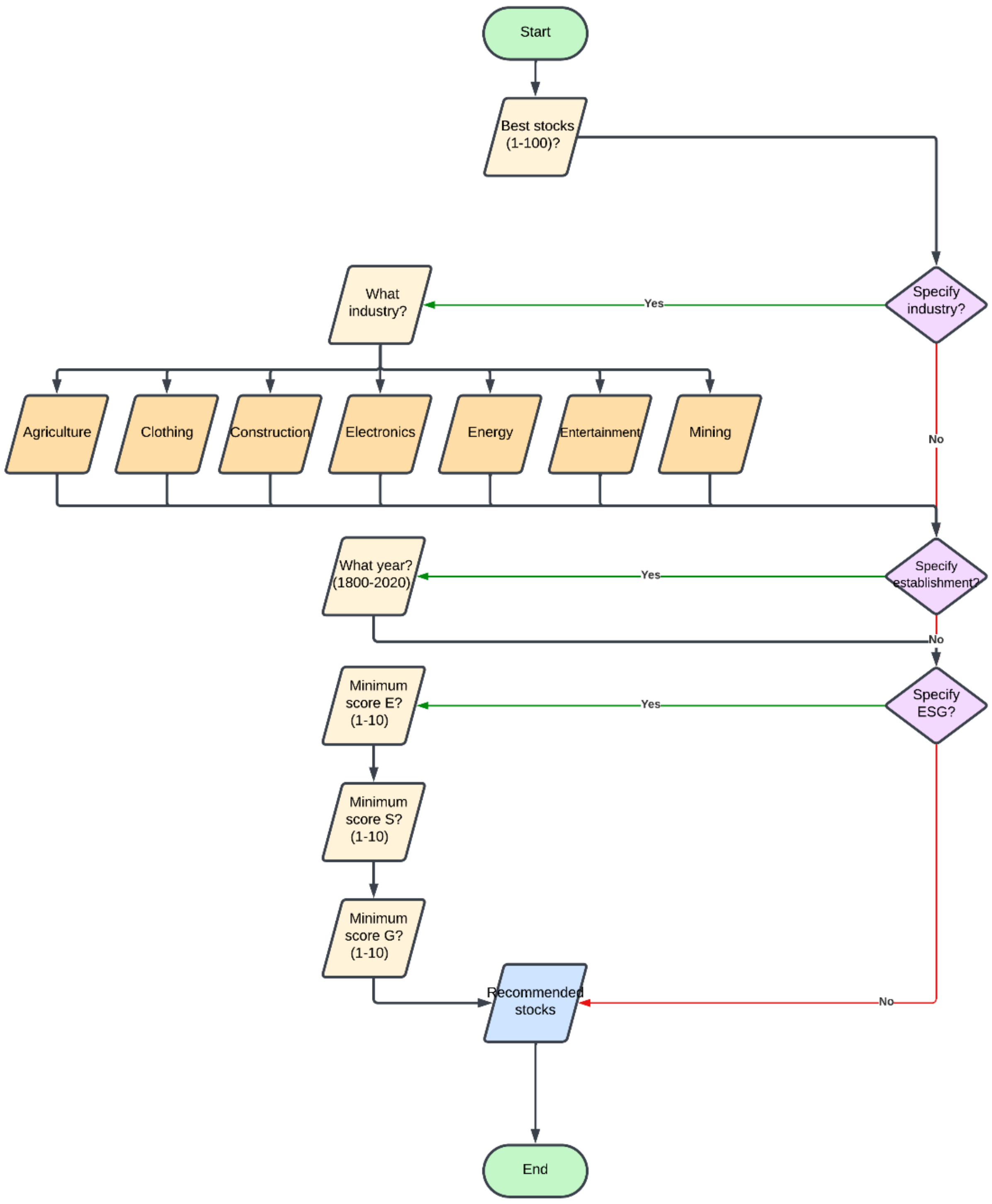
Design process

For the project we decided to divide the problem into smaller problems, that we could then work out more easily. The first step was to load in the database and read the data from it. For easy access we have stored all the data in a dictionary with the labels as keys and their corresponding values and change the values to integers if needed. For a nice presentation of the data, we also needed to be able to print it from a table. The next step was to get all the user inputs. We have decided to split this up into different categories with each their own functions which return the values the user gives. It is split up into the number of stocks, the industry, the foundation year, and the ESG scores of the stocks. Then for each of these input values we have made their corresponding functions that filter the database and return the stocks with the required criteria. These functions start with an empty list and add all the stocks that have the required criteria and add these to the list and return it when it looped through all the stocks in the database. The last step was to bring everything together to make it a working program.

The loading of the database was relatively new for us, so we needed to figure out how we could do this in the best way that we can still do calculations with it and modify it if needed. The function for printing the database was a new aspect that we have never touched before, so it was a lot of trial and error to get the result we wanted. The functions that filter the database for the stocks with the right criteria were also quite tough to figure out how we could make them work the way we wanted. But besides these points the rest went quite smoothly.

This project was a joint effort divided among team members. Parts of the project were assigned based on each member's individual strengths. The tasks were divided as followed: Skick was responsible for developing the python code for the algorithm, Jasper created the pseudocode, Demy designed the flowchart, and Abir and Kyan worked together on the presentation while working on things that needed to be written inside the document such as the context task, problem definition and design process. Of course, we helped with each other's tasks where possible as well. The report was a collective effort where everyone worked together to make one well flowing report. The final presentation was given by Abir, Demy, and Kyan. Over the course of two weeks, the team dedicated a total of about 25 hours to the project, with each member contributing approximately 4 hours.

Flowchart



Start
Best stocks (1-100)?
Specify industry?
Yes
No
What industry?
Agriculture
Clothing
Construction
Electronics
Energy
Entertainment
Mining
Specify establishment?
Yes
No
What year? (1800-2020)
Specify ESG?
Yes
No
Minimum score E? (1-10)
Minimum score S? (1-10)
Minimum score G? (1-10)
Recommended stocks
End

Pseudocode

Algorithm 1 Stock Recommendation Algorithm

```
function RecommendStocks
    // Initial input
    INPUT number_of_stocks (1-100)
    // Industry selection
    DISPLAY "Specify industry? (Yes/No)"
    INPUT specify_industry
    if specify_industry = "Yes" then
        DISPLAY "Select industry:"
        DISPLAY OPTIONS ["Agriculture", "Clothing", "Construction",
            "Electronics", "Energy", "Entertainment", "Mining"]
        INPUT selected_industry
    end if
    // Establishment year
    DISPLAY "Specify establishment year? (Yes/No)"
    INPUT specify_year
    if specify_year = "Yes" then
        DISPLAY "Enter year (1800-2020):"
        INPUT establishment_year
    end if
    // ESG Scores
    DISPLAY "Specify ESG scores? (Yes/No)"
    INPUT specify_esg
    if specify_esg = "Yes" then
        DISPLAY "Enter minimum Environmental score (1-10):"
        INPUT min_e_score
        DISPLAY "Enter minimum Social score (1-10):"
        INPUT min_s_score
        DISPLAY "Enter minimum Governance score (1-10):"
        INPUT min_g_score
    end if
    // Apply filters
    stocks $\leftarrow \emptyset$
    if specify_industry = "Yes" then
        stocks $\leftarrow$ FILTER stocks WHERE industry = selected_industry
    end if
    if specify_year = "Yes" then
        stocks $\leftarrow$ FILTER stocks WHERE establishment_year $\leq$ input_year
    end if
    if specify_esg = "Yes" then
        stocks $\leftarrow$ FILTER stocks WHERE
            (environmental_score $\geq$ min_e_score AND
            social_score $\geq$ min_s_score AND
            governance_score $\geq$ min_g_score)
    end if
    // Return results
    return TOP number_of_stocks FROM stocks
end function
```

Python code

```python
from os import system, name
import csv
from typing import List, Dict, Any, Optional, Union

def clear_screen():
    #windows
    if name == 'nt':
        _ = system('cls')
    #mac and linux
    else:
        _ = system('clear')

#region dataset_tools
def read_dataset(path: str) -> List[Dict[str, any]]:
    '''
    Reads the dataset from the given path into a list of dictionaries.
    '''

    newlist = []
    with open(path, 'r') as file:
        csvFile = csv.DictReader(file)

        for row in csvFile:
            # Convert values, so we can calculate with them.
            row['Performance'] = float(row['Performance'])
            row['FoundationYear'] = int(row['FoundationYear'])
            row['Environment'] = int(row['Environment'])
            row['Social'] = int(row['Social'])
            row['Governance'] = int(row['Governance'])

            newlist.append(row)

    return newlist


def print_dataset(dataset):
    '''
    Prints out a table with the stocks in the given dataset.
    '''

    print('ID    |    Performance     |     Industry          |     Foundation Year  |  Environment   |  Social   |  Governance   |')
    print('------|-----------------|------------------|--------------------|-----------------|------------|----------------|')

    for stock in dataset:
        for stock_property in stock:
            #The industry property needs extra space for the long words like: 'entertainment' and 'electronics'
            if stock_property == 'Industry':
                spaces: int = len(stock_property) + 8
            else:
                spaces: int = len(stock_property) + 4
            spaces = spaces-len(str(stock[stock_property]))

            print(stock[stock_property], end=f'{spaces * " "}|  ')

        print('')
#endregion

```

```python
#region dataset_modification
def get_best_performing(dataset: List[Dict[str, str]], top_amount: int) -> List[Dict[str, str]]:
    '''
    Returns a list with length of top_amount of the best performing stocks in decending order.
    '''

    top_list: List[Dict[str, str]] = []
    stock_dataset = dataset.copy()

    # Let the user know if there are not enough stocks still in the dataset after the aplied inputs.
    if len(stock_dataset) < top_amount:
        print('There are not enough stocks in the dataset for the specified requirements\n')
        print(f'There are {len(stock_dataset)} stocks, with given input number: {top_amount}\n')
        top_amount = len(stock_dataset)


    for amount in range(top_amount):
        highest_performance: float = float('-inf')
        highest_stock: Dict[str, str] = None

        # Get the highest performing stock.
        for stock in stock_dataset:
            if stock['Performance'] > highest_performance:
                highest_performance = stock['Performance']
                highest_stock = stock

        # Add the highest performing stock to the list and remove it from the databese, so it cannot be chosen again.
        top_list.append(highest_stock)
        stock_dataset.remove(highest_stock)

    return top_list

def get_stocks_from_industry(dataset: List[Dict[str, str]], industry) -> List[Dict[str, str]]:
    '''
    Returns a list of all the stocks in the current dataset from the given industry.
    '''

    stock_list = []

    for stock in dataset:
        if stock['Industry'] == industry:
            stock_list.append(stock)

    return stock_list

def get_below_establishment_year(dataset: List[Dict[str, str]], establishment_year: int) -> List[Dict[str, str]]:
    '''
    Returns a list of all stocks in the current dataset that were founded before the given establishment_year.
    '''

    stock_list = []

    for stock in dataset:
        if stock['FoundationYear'] <= establishment_year:
            stock_list.append(stock)

    return stock_list

    def get_above_ESG_criteria(dataset: List[Dict[str, str]], ESG_criteria: List[int]) -> List[Dict[str, str]]:
        '''
        Returns a list of all the stocks in the current dataset that have scores above the given ESG_criteria.
        '''

        stock_list = []

        for stock in dataset:
            if stock['Environment'] >= ESG_criteria[0] and stock['Social'] >= ESG_criteria[1] and stock['Governance'] >= ESG_criteria[2]:
                stock_list.append(stock)

        return stock_list
    #endregion


    #region user_input
    def ask_user_amount() -> int:
        '''
        Asks the user the amount of stocks he wants in the list and returns it.
        '''

        while True:
            try:
                amount_best_stocks = int(input("How many best stocks do you want (1-100)?\n").strip())

                if amount_best_stocks > 100 or amount_best_stocks < 0:
                    clear_screen()
                    print("Input number must be above 0 and below 100")
                else:
                    return amount_best_stocks

            except ValueError as e:
                clear_screen()
                print('Input must be of type integer.')

    def ask_user_industry() -> str:
        '''
        Asks the user from what industry he wants the stocks and returns the given industry.
        '''

        want_industry = input('Do you want to specify the industry (yes/no)?\n').strip().lower()
        if want_industry == 'yes' or want_industry == 'y':
            clear_screen()
            print('Available industries: agriculture, clothing, construction, electronics, energy, entertainment, mining.')
            industry_input = input('What is the industry you are looking for?\n').strip().lower()

            if industry_input not in ['agriculture', 'clothing', 'construction', 'electronics', 'energy', 'entertainment', 'mining']:
                print('Invalid industry given.')
                return ask_user_industry()

            return industry_input

        return ''

    def ask_user_establishment_year() -> int:
        '''
        Asks the user below which establishment year it wants the stocks to be and returns it.
        '''

        want_establishment_year_input = input('Do you want to specify the establishment year (yes/no)?\n').strip().lower()

        if want_establishment_year_input == 'yes' or want_establishment_year_input == 'y':

            print('')
            while True:
                try:
                    establishment_year_input = int(input("What is the establishment year you are looking for (1800-2020)?\n").strip())

                    if establishment_year_input > 2020 or establishment_year_input < 1800:
                        clear_screen()
                        print("Input number must be above 1800 and below 2020")
                    else:
                        return establishment_year_input

                except ValueError as e:
                    clear_screen()
                    print('Input must be of type integer.')
        else:
            return -1

    def ask_user_ESG_criteria() -> List[int]:
        '''
        Asks the user what ESG criteria he wants and returns the given values in a list.
        '''

        want_ESG_criteria = input('Do you want to specify the Environment, Social en Governance scores (yes/no)?\n').strip().lower()

        if want_ESG_criteria == 'yes' or want_ESG_criteria == 'y':

            print('')
            while True:
                try:
                    environment_score_input = int(input("What is the minimal score you are looking for in Environment (0-10)?\n").strip())
                    print('')
                    social_score_input = int(input("What is the minimal score you are looking for in Social (0-10)?\n").strip())
                    print('')
                    governance_score_input = int(input("What is the minimal score you are looking for in Governance (0-10)?\n").strip())

                    for score in [environment_score_input, social_score_input, governance_score_input]:
                        if score > 10 or score < 0:
                            clear_screen()
                            print("Input number must be above 0 and below 10")
                            continue

                    return [environment_score_input, social_score_input, governance_score_input]

                except ValueError as e:
                    clear_screen()
                    print('Input must be of type integer.')
        else:
```

```python
        return []
#endregion


def setup_user_input():
    '''
    Calls all the user input functions and stores their values.
    '''

    clear_screen()
    amount_stocks = ask_user_amount()

    clear_screen()
    wanted_industry = ask_user_industry()

    clear_screen()
    wanted_establishment_year = ask_user_establishment_year()

    clear_screen()
    wanted_ESG_criteria = ask_user_ESG_criteria()

    create_final_stock_list(amount_stocks, wanted_industry, wanted_establishment_year, wanted_ESG_criteria)


def create_final_stock_list(amount_stocks, wanted_industry, wanted_establishment_year, wanted_ESG_criteria):
    '''
    Creates and modifies the dataset to the given input criteria.
    '''

    dataset = read_dataset('stocks.csv')

    if wanted_industry != '':
        dataset = get_stocks_from_industry(dataset, wanted_industry)

    if wanted_establishment_year != -1:
        dataset = get_below_establishment_year(dataset, wanted_establishment_year)

    if wanted_ESG_criteria != []:
        dataset = get_above_ESG_criteria(dataset, wanted_ESG_criteria)

    # Change settings if there are no stocks in the list and rerun this function
    if len(dataset) == 0:
        clear_screen()

        print('The input criteria was too strict, there are no such stocks in the dataset.')
        print('Therefore the we have changed the criteria.\n')


        if wanted_establishment_year != -1:
            print('The establishment year has been set to 1950.')
            wanted_establishment_year = 1950

        if wanted_ESG_criteria != []:
            print('The Environment, Social and Governance scores have all been set to a minimum of 6.')
            wanted_ESG_criteria = [6, 6, 6]

        print('')

        dataset = read_dataset('stocks.csv')

        return create_final_stock_list(amount_stocks, wanted_industry, wanted_establishment_year, wanted_ESG_criteria)


    dataset = get_best_performing(dataset, amount_stocks)

    print_dataset(dataset)


if __name__ == '__main__':
    setup_user_input()
```