Vrije Universiteit Amsterdam
Computational Thinking
Project Assignment: <*I Trade*>

Group number: <Mykhailo Varha>
Members with student numbers:
<Mykhailo Varha> <lgm 557>

Date: <15/12/2024>

Context Task

AI has both negative and positive effects on the financial stock market. Here are the positive impacts. First, AI systems are more efficient in detecting fraudulent activities than most humans. So, security and integrity of the market are in a much better state than before. Also, AI as a technological prospect has increased the demand for tech-oriented stocks and thus increasing the market variety.

On the other hand, with the rise of AI technology appeared the Trading bots, that make split-second decisions in their trading patterns, and trade with monstrous efficiency. Because of these trading bots, the stock market becomes more liquid. After extensive research the liquid stock market is not good in the long term, for a few reasons: insecurity as stocks can be sold in bulk by big companies, susceptibility to flash crashes and high upkeep cost. These weaknesses are not inherently caused by AI technology, as they are mostly caused by human nature itself. The only actual weakness of AI in the stock market is that not all people have equal access to it, and therefore some may have an unfair edge over others.

However, a liquid stock market is very stable and highly competitive, with low entries and low exits so many people can join and leave anytime they wish. This also means that more entrepreneurs may get enough money from selling stocks to enhance their business. Also, AI trading bots are more efficient than most humans thus making the stock market operations much easier and more uniform.

Overall, I think the AI has a positive impact on the stock platform, even though use of it has some downsides. The liquidity of the market is just too big of a benefit no matter how one sees it. It allows for much faster progression for all the population on condition they are willing to risk their money.

Design process

So for the design of the project, I decided to come up with this approach. First, I decided to create 3 classes. I also used git hub. Just regular practice. My approach is a type of Divide and Conquer, as I separate work to be done modularly.

stock – class that describes a Stock unit with all its parameters
user_profile – class that describes users input into the algorithm with all the filters
stock_recommender – class that handles the database of stocks

However, I needed to decide how I was going to handle the database, am I going to use pandas or dask. I have used pandas as a programmer more, so I decided to go with pandas. For reference dask is a module NVIDIA Developer program gives free courses on. Capitalize on that however you wish. I only started to learn dask so I did not use it.

One issue I faced, not major but still an issue – I don't even need class stock. It is useless as I don't do transformations on stocks at all. Pandas does everything for me in terms of sorting and transformation as its dataframe class is very useful. I wonder if dask also has that much utility.

Next, I decided that I need to find an approach to solve the problem. I have 3 parameters to limit my search: Top in the industry, year of establishment and ESG score.

Top in the industry parameter is a bit unclear. So I decided to take liberty and define it as the (100 - N)th percentile in the industry in the stock market where N is such number that (100 – N)th percentile in the industry in the stock market will contain as much stocks as user asks for.

ESG score I defined as the some of Environmental, Social and Governance Scores.
Ratio 1 : 1 : 1;

The most limiting factor is year of establishment. If this filter parameter is active no company younger than that year is out. So I decided to make my algorithm first to limit companies on the year of establishment if the parameter is active. However, the next priority parameter was an issue. After a long thought, I came to a decision to make the top in the industry parameter come next, as I would rather help my user achieve more financial success as the my program is targeted for beginners and if my user fails there is a trend for beginners to quit the stock market and that is my worst case scenario. At last the last search parameter is ESG rating. Here I also allowed my user to select which one of the ESG score he want to look at individually.

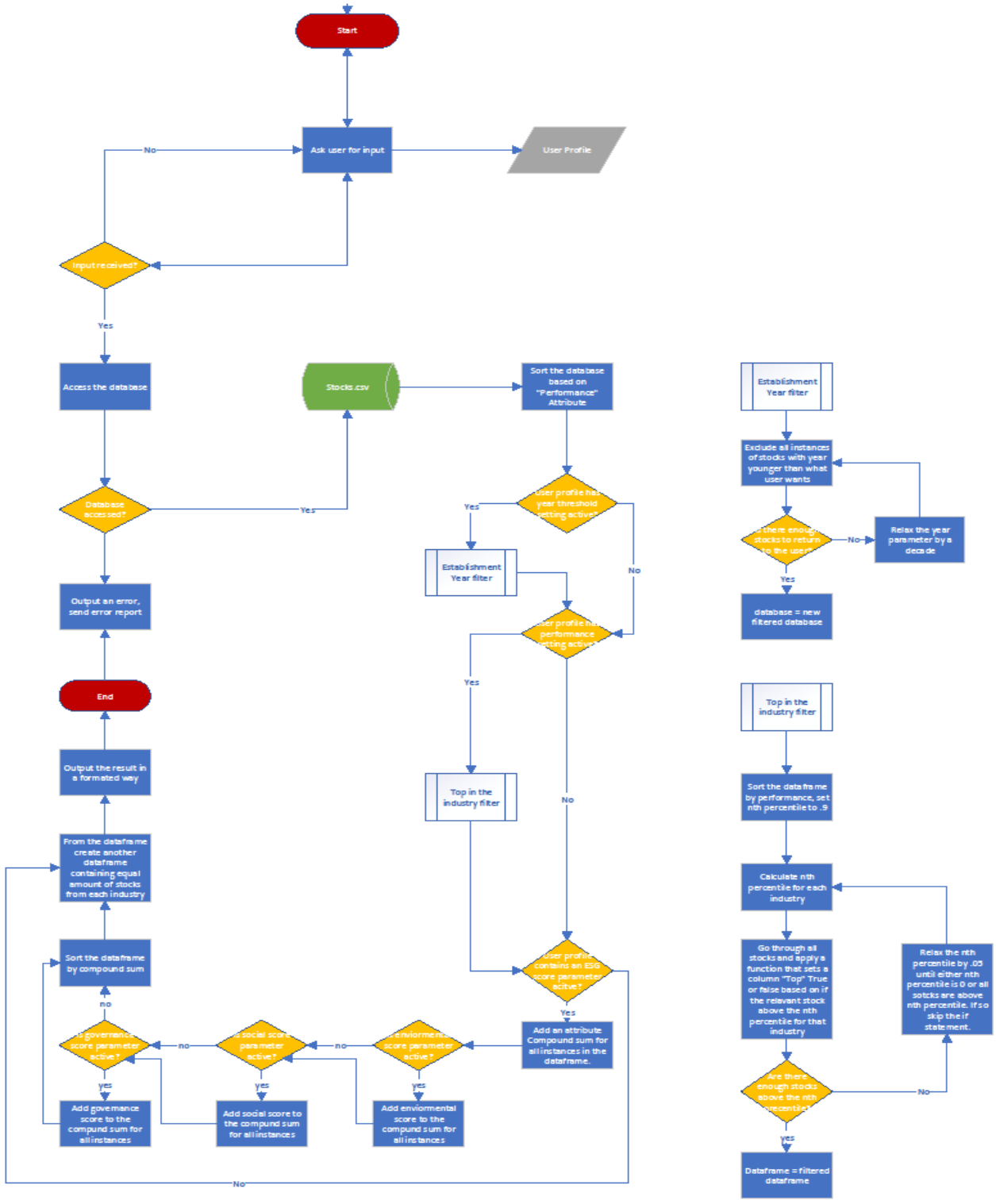
I also want to add more variety for my user in their stock choices so I decided to make an additional task for myself that will ensure that all industries are mentioned equally as long as the size of the sample of stocks allows that.

So the actual process will look like this:

Ask user for input -> Create User profile -> Access the Database of Stocks -> Create the Dataframe of Stocks -> Exclude the Stocks younger than Users year of establishment parameter -> Select top percentile of the stocks based on the amount of stocks user asks for -> Sort the dataframe based on ESG score performance -> Split the dataframes into smaller dataframes containing one industry and from them include top N stocks equally from each industry -> Output the result.

For the output I used module tabulate as it pairs nicely with pandas dataframes.

Flowchart



Pseudocode

```
boot up program

ask user if they want a year setting
    if yes, ask them to enter a year and store the value in year
```

```
ask user if they want to focus on the most important stocks
    if yes, store true in performance variable

ask user if they want to sort them based on ESG scores
    if yes
        ask user to enter if they want to focus on Environment Social and
Governance score individually
            if yes set the respective variable true

ask the user how much stock do they want
store the answer in number

from the variables create user profile

Start the dataframe

filter the dataframe by performance

send the user profile to the dataframe

start processing dataframe based on user profile

if year setting is true
    exclude all entries younger than that year, however if there are not
enough entries to match number of
    stocks n - user wants - reverse changes and repeat the step relaxing
the year parameter by 10 years until there are
    enough stocks

if most important stocks setting is true
    create function is_top_percentile(row of the dataframe){
        if rows performance > nth percentile
            row[is top percentile] = True
            where n is such a number that ensures the inclusion of number
of the stocks user wants displayed
    }

    go through all the entries in the dataframe and apply is_top_percentile
to each row
    starting with percentile is 90

    and if there are not enough True values
    go through all of them again decrementing the percentile by 5, repeat
until either percentile is 0
    or there are n or more stocks in the dataframe with True values

    remove column[is top percentile]

if one of ESG setting is true
    create column Compound sum in all elements in the dataframe

    set the column equally to that columns economical + social + governance
score if and only if
    the respective economical social and governance settings are active in
the user profile

    sort the dataframe based on the compound sum

    remove compound sum column

create result dataframe
```

```
add to result dataframe equal amount of elements from each industry taking
them from original dataframe

send the result to the user in form of a table

Stop operations.
```

Python code

```
import pandas as pd
from pandas.core.interchange import dataframe
from tabulate import tabulate
## User profile is a custom class used for containing the information about
the user and their prefered choices
# during the search or user input processes
class user_profile:
    #user id
    id: str;

    #if we are searching for highest performance
    performance: bool;

    #if we are also searchign for specific industry
    industry: str;

    #for year of establishment parameter
    establishment_year: int;

    #for ESG Parameters search when active
    environment: bool;
    social: bool;
    governance: bool;

    def __init__(self):
        self.id = "000"
        self.performance = False;
        self.industry = ""
        self.establishment_year = 9999;
        self.social = False;
        self.environment = False;
        self.governance = False ;

    def __init__(self, id="000", performance=False,
establishment_year=9999,
                 social=False, environment=False, governance=False,
industry = ""):
        self.id = id[0:2]

        self.performance = performance
        self.industry = industry;

        self.establishment_year = int(establishment_year);
        self.social = social
        self.environment = environment
        self.governance = governance

    def __setattr__(self, __name, __value):
        super().__setattr__(__name, __value)
class stock_recommender:
```

```
    path: str;
    df: dataframe;
    tdf: dataframe;
    perentile: dataframe;

    def __init__(self):
        self.path = "";
        self.df = None;
        self.tdf = None;

    def __init__(self, relative_path):
        self.path = relative_path;

        self.df = pd.read_csv(self.path);
        self.df = self.df.sort_values(by = "Performance", ascending=False);
        self.tdf = None;

    # this function will sort based on Enviornmental, Social and Governance
parameters separately
    def sort_by_ESG(self, df: dataframe, user) -> dataframe:
        # to do this I am going to create a new column with pandas df
method, and assign it such values
        # equal to previously stated parameters, if their counterpart in
the user_profile is active - True
        df["CompoundScore"] = (df["Environment"] if user.environment else 0
+
                               df["Social"] if user.social else 0 +
                               df["Governance"] if user.governance else 0);

        # Here I just sort based on these parameters and remove the column
afterwards
        df = df.sort_values(by="CompoundScore", ascending=False);

        df = df.drop(columns=["CompoundScore"]);

        return df;

    #this function ensures equal inclusion to the maximum for each industry
for the output
    def get_unique(self, df: dataframe, n) -> dataframe:
        # There is no need to go trhough this function if the length of the
dataframe is less or equal to n
        if n >= len(df):
            return df;

        # unique_ industries is a numpy_array that contains all unique
industries
        unique_industries = df["Industry"].unique();

        # this is going to be a list of dataframes containing solely a
specific industry
        partition: list[dataframe] = []

        # this list 'lengths' is of integers that track the length of each
df in partition
        lengths = [];
        #this list will track how much elements we take from each df in
partition
        takes = [];
        #i is just an index
        i = 0;
```

```
        # here I make it so partition has all instances fully filled for
each industry
        # what it is in result and array of dataframes where each dataframe
has only one type of industry
        for element in unique_industries:
            partition.append(
                df[df["Industry"].str.contains(element, case=False)]
            )

        # in this loop I initialize all elements in lengths and take
        for dfa in partition:
            lengths.append(len(dfa));
            takes.append(0);

        #while i is less than n
        while i < n:

            # I go through each element in length
            # and if it is not 0 I transfer 1 from lengths to takes
            for j in range(len(lengths)):
                if (i < n):

                    if (lengths[j] != 0):
                        lengths[j] -= 1;
                        takes[j] += 1;
                        i += 1;


        i = 0;
        res = pd.DataFrame();

        # now I go thorugh each df in partition and add respective amount
of elements determined in takes[i] to
        # res dataframe
        for dfa in partition:
            res = pd.concat([res, dfa.head(takes[i])], ignore_index=True);
            i += 1;

        return res

    # function that I am going to pass onto pandas df apply method wich
basically goes through each row
    # in the dataframe and applies a function to that row, in this case it
is going to check if that row
    # is within the looked for percentile and returns True if it is within
the precentile or false if not
    def is_top_percentile(self, row):
        industry = row["Industry"]
        performance = row["Performance"]

        return performance >= self.percentile[industry]

    # this is a recursive function that dynamically adjust its needed
percentile for each industry
    def get_top_percentile(self, df: dataframe, n, quan):
        grouped = df.groupby("Industry"); # I group them by industry

        # and from each grouped variant, I get something call quantile,
which is a method in pandas dataframe
        # that returns a float representing a percentile we gave to
```

```
quantile method, for each unique instances
        # in a dataframe, which is in this case each indsutry
        self.percentile = grouped["Performance"].quantile(quan)

        # now I just apply a function from above, and store the boolean
value in a column TopPercentile
        # I use lamda row to avoid a specific copy warning during the
slicing proccess when panda uses apply method
        # took a while to find out I should do this as .apply method is not
really that safe
        df["TopPercentile"] = df.apply(lambda row:
self.is_top_percentile(row), axis = 1);

        #res is just a list right now that stores only a row with
TopPercentile column set to True,
        # I use it to count the amount of True values in the column to see
        # if I have enough stocks to fulfill users request
        res = df[df["TopPercentile"]]

        # here if the result does not have enough numbers I lower the
percentile parameter by a margin of 0.05 and call
        # the function again, with an updated parameter
        if len(res) < n or quan <= 0:
            return self.get_top_percentile(df, n, quan-0.05);

        # however If the dataframe has enough stocks it will just return
        res = res.drop(columns = ["TopPercentile"])
        return res;

    # this function is a recursive function that ensures only stocks before
certain year will be included
    # if there are not enough stocks, the year parameter will be adjusted
to fit in at least N number of stocks
    def sort_year(self, df: dataframe, n: int, year: int):
        res = df[df["FoundationYear"] <= year];

        if len(res) < n:
            # incrementing the year by a decade as an increasing margin
            return self.sort_year(df, n, year + 10);

        # otherwise I am just going to preint a message that will print out
a year and return res
        print(f"Including stocks from year up to {year} to fit user
parameters");
        return res;


    def get_stocks(self, user: user_profile, n: int) -> dataframe:

        # according to my algorythm we are first going to exclude some
stocks based on their establishment year
        # if the user did not select a specific year the user_profile is
set to select year 9999 which is far above the
        # current year

        self.tdf = self.sort_year(self.df, n, user.establishment_year);

        # now I should do transformations if I am looking at a specific top
percentile of the industry
        if user.performance:
            self.tdf = self.get_top_percentile(self.tdf, n, 0.90)
```

```
        #here I filter the top percentile or the non transformed tdf based
on the ESG criteria
        if (user.governance or user.social or user.environment):
            # this line of code does it
            self.tdf = self.sort_by_ESG(self.tdf, user);

        else:
            self.tdf = self.tdf.sort_values(by = "Performance", ascending =
False);

        # now however I want to include all unique industries at least once
as long as it is in confines
        # of number of stocks the user entered

        res = self.get_unique(self.tdf, n)

        # and now I just sort the table to show all indsutries close to
each other for easy comparison
        res = res.sort_values(by = ["Industry", "Performance"],
ascending=[False, False]);

        return res;

def user_input()->user_profile:

    user = user_profile();

    a: str = input("Would you like to prioritize top in the indsutry
stocks? -> Yes or No\n");

    if "yes" in a.lower():
        user.performance = True;

    a = input("Would you like to search for establishments created before a
certain year? Yes or No\n");

    if "yes" in a.lower():
        a = input("Type a year -> ");
        try:
            user.establishment_year = int(a);
        except:
            user.establishment_year = 9999;

    a = input("Would you like to search for establishments with high
enviornment score? Yes or No\n");

    if "yes" in a.lower():
        user.environment = True;

    a = input("Would you like to search for establishments with high social
score? Yes or No\n");

    if "yes" in a.lower():
        user.social = True;

    a = input("Would you like to search for establishment with high
governance score? Yes or No\n");

    if "yes" in a.lower():
        user.governance = True;
```

```
    return user;

def main():
    path = "../rescs/stocks.csv"; ## local path tot the database of stocks
change it based on your enbiornment

    user = user_input();

    n = 15; ## safe barier for user input

    try:
        n = int(input("Please enter the amount of stocks you want to us to
recommend -> "));
    except:
        n = 15;

    recommender = stock_recommender(path);

    df = recommender.get_stocks(user, n)

    table = tabulate(df, headers = "keys", tablefmt = "grid",
showindex=False);

    print(table)

if __name__ == "__main__":
    main();

main();
```

Checklist for submission:

- ✓ Your project report as a pdf.
- ✓ Your Python code as a .py file.
- ✓ Optional: any additional files (such as .csv files) you might have created which are required for your program to run.
- ✓ Each of the above included in a .zip file with the name CT_PROJECT_*GROUPNUMBER*.zip