
Python Project

Stock Performance Analysis and Stock Recommendation

Group Member: Zhenning Li, Nevina Bankim Dalal

Lecturer: POPINEAU Fabrice

Table of Contents

Python Project.....	1
Preamble:	2
Introduction:.....	3
Goal of Each Stage:.....	4
Step by step explanation:.....	6
Stage V0:.....	6
Stage V1:.....	11
Stage V2:.....	14
Bugs fixed & difficulties:	17
Future Scope	20

Preamble:

The goal of this project was to use python to understand the US Market in an interactive way using Graphic User Interface in python. The customers can input the name of the stocks that they wish to invest in, the date of their purchase, the cost and quantity that they are willing to pay for the above-mentioned stock. In return, they get a beautiful interface which gives relevant information about their stocks in a graphical format.

Having never worked with python professionally before, we decided to use Spyder to write and implement different parts of our code.

As our work progressed, we encountered several difficulties which we have addressed in different sections of the project. To handle these difficulties, we divided the project into 3 phases.

1. Stage V0
2. Stage V1
3. Stage V2

Each phase consists of some modules which are python files required to run the phase. For the sake of simplicity please follow the order mentioned in the ReadMe.txt file which explicitly states the way to run each file.

Required Libraries:

For the smooth functioning of this project in all three phases, the users need to install the following libraries :

1. fix_yahoo_finance==0.1.37
2. numpy==1.19.2
3. matplotlib==3.3.3
4. pandas_datareader==0.9.0
5. yfinance==0.1.55
6. plotly==4.12.0

All the above-mentioned libraries can be installed via pip. The method to install it is:
pip install <name of the library>

Introduction:

Portfolio management is the selection, prioritization and control of an organization's programs and projects, in line with its strategic objectives and capacity to deliver.

The goal is to balance the implementation of change initiatives and the maintenance of business-as-usual, while optimizing return on investment. The goal of this project is to visualize the investment's made the user using different interface graphs that helps the user to understand his position.

Explanation about all the files:

There are all together 6 “.py” files, as the following:

- No1: stage1&2_investment_performance_analysis_and_visualization.py
- No2: stage3_stock_interface_recommendation.py
- No3: recommendation.py
- No4: input_process.py
- No5: stock_interface.py
- No6: class_input_investor_tickers.py

Among them:

Stage V0 mainly includes: No.1, No4, No6

Stage V1 mainly includes: No1, No5

Stage V2 mainly includes: No2, No3

We have put separate files into different files of different stages.

Something to be mentioned: Our **Stage V1** is an optimization of **Stage V0**, which is an interface of the investor input process defined in **Stage V0**. Since the investor input process and the corresponding visualization is so complicated, we split the performance analysis part into 2 stages: **Stage V0** and **V1**. In the zip file there are 3 files, and in the file of **Stage V0**, only the unoptimized raw code we wrote was shown. The true performance analysis code is in **Stage V1 file**, and the recommendation system code is in **Stage V2 file**.

Goal of each stage:

Stage V0:

Target:

1. investors can get a summary table (DataFrame) of all their input portfolio stock basic information.
2. investors can get a summary table (DataFrame and HTML interactive chart) in which:
 1. the respective performance of their own stock will be compared with the S&P 500 to show investor's ability to distinguish good stocks.
 2. The difference between stocks' highest price since acquisition date and current price will be visualized to show investors' ability to sell at the proper time.
 3. compare YTD (year to date, the return rate of buying the stock at the beginning of this year and selling it now) returns of investor's stocks and S&P 500.
 4. compare the total return of the chosen stocks with the return if S&P 500 index was bought at the same time with the same amount of capital.
 5. cumulative returns and total cumulative investment over time will be shown.

Stage V1:

An interactive interface has been created with Tkinter, so that investors would find it easier to input the information and see the graphs.

We created a label in which all the necessary messages would appear, such as all the error input messages, input success messages, and "your information has been saved" things like that. Similarly, investors cannot input any illegal numbers and after inputting the numbers and tickers, they need to click "**Finish Input**" button to see if there are any mistakes in their inputs, and then if no error happens, they need to click the "**Save**" button to assign their inputs to the corresponding variables and saved into a DataFrame. Once they clicked "**Save**", they will see a pop-up message box asking them if they want to add more tickers. If the answer is yes, then they will be able to input all the information once again, beginning with blanks in each input box (automatically erase the previous inputs). When the investor clicks "**No**", the input information will be processed and then 5 buttons of 5 different graph options would appear. The investor can see the description of each graph above, and after clicking one button, the corresponding **graph()** method would be triggered, and the browser would open and show the interactive stock or index performances etc.

Stage V2:

In this stage, we wanted to offer our customer an opportunity to see if the stock he chooses is worth investing or not with the basic machine learning techniques. We searched the internet and collected the rating information from wall street main investment banks including

Goldman Sachs, Morgan Stanley etc. of the typical 30 US listed companies, and we labelled some with the number one indicating that this stock is well worth buying and labelled some zero indicating that this one is not good to buy now. Only those stocks which are rated “Highly recommended” by those investment banks would get the label 1.

Because the market is changing every day and the information of those top companies is also changing, so in the recommendation stage every time the customer runs the file, it will extract the latest data from the market and make a new calculation, which include:

1. Making a DataFrame of the data just collected and putting the correct label on them.
2. Training the random forest model with the best parameters we tested before with GridSearch method.
3. Generating the recommendation information based on the outcome of the model on the test dataset, which is the stock that the investor wants to require for advice.

After all these are done, when you run the **Stage V2** file, you will see a window in which it requires you to input the stock you want to get advice. After inputting the ticker and wait for 3 seconds, you will see in the label if AI thinks it is a good idea to buy or not, along with the accuracy score generated by the model, which can be a supportive information for investors to take into consideration.

Step by step explanation:

Stage V0:

See the file “**class_input_investor_tickers.py**” first:

This file is written for creating a DataFrame for investors to input the related information.

This file contains 199 lines.

Line 17-21: **dataframe_initialization()** is used for initializing the skeleton of the DataFrame we want to get as an outcome.

```
# initialization of DataFrame
def dataframe_initialization():
    global data_base
    parameters = ['Acquisition Date', 'Ticker', 'Quantity', 'Unit Cost', 'Cost Basis', 'Start of Year' ]
    data_base = pd.DataFrame(columns = parameters)
    return data_base
#
```

Here is the how it runs:

```
In [16]: dataframe_initialization()
Out[16]:
Empty DataFrame
Columns: [Acquisition Date, Ticker, Quantity, Unit Cost, Cost Basis, Start of Year]
Index: []
```

Line: 24-49: **input_process()** is used for making investors input some information including the name of the company, the quantity he bought, the unit cost etc. with the **input()** function.

```
# Initialization of the things that investors are ordered to input
def input_process():

    global ticker
    ticker = 0
    global ticker_acquisition_date
    ticker_acquisition_date = 0
    global ticker_quantity
    ticker_quantity = 0
    global ticker_unit_cost
    ticker_unit_cost = 0
    global ticker_cost_basis
    ticker_cost_basis = 0
    global ticker_start_of_year
    ticker_start_of_year = 0

    # basic error information input validation process
    ticker = input('please input your ticker')
    ticker = ticker.upper()
    ticker_acquisition_date = input('please input your acquisition date in the form: "Year-Month-Day"')
    ticker_quantity = input('please input your quantity')
    ticker_unit_cost = input('please input your unit cost')

    ticker_cost_basis = float(ticker_quantity)* float(ticker_unit_cost)

    # we will use the price at the end of last year to make some graphs
    ticker_start_of_year = datetime.datetime(2019,12,31)
```

Here is the how it runs:

```
In [15]: input_process()
please input your tickerBABA
please input your acquisition date in the form: "Year-Month-Day"2020-11-25
please input your quantity20
please input your unit cost150
```

Line 53-108: There are 5 functions committed to judge if the investor's input information is legal or not, if it is legal, then return True, if not, return False. This is used for later data retrieving job since if the input is wrong, only error messages will come out when putting it into the API database.

The 5 functions are:

```
judge_whether_ticker_is_legal(ticker)
judge_whether_ticker_acquisition_is_legal_in_format(ticker_acquisition_date)
judge_whether_ticker_acquisition_is_legal_in_weekdays(ticker_acquisition_date)
judge_whether_ticker_quantity_is_legal(ticker_quantity)
judge_whether_ticker_unit_cost_is_legal(ticker_unit_cost)
```

Line 110-116: **change_acquisition_date_format(ticker_acquisition_date)** is used for changing the acquisition date format for convenience of merging DataFrames in the “stage1_input_process.py” file.

```
# change the acquisition date format for the convenience of merging dataframes in another file
def change_acquisition_date_format(ticker_acquisition_date):
    ticker_acquisition_date = ticker_acquisition_date.replace('-', ' ')
    ticker_acquisition_date = ticker_acquisition_date.split()
    k = []
    for i in ticker_acquisition_date:
        k.append(int(i))
    return datetime.datetime(k[0], k[1], k[2])
```

Line 122-187: **main()** is used for returning the “data_base” (in DataFrame type) which includes all the correct and legal information the investors just input, and if the investor wants to input more than one stock, he can choose to continue and input more, and all the data will be put into the “data_base”.

```
In [17]: main()

please input your tickerBABA
please input your acquisition date in the form: "Year-Month-Day"2020-11-26
please input your quantity20
please input your unit cost201
BABA
2020-11-26

Do you want to add more tickers? y/ny
please input your tickerNIO
please input your acquisition date in the form: "Year-Month-Day"2020-11-24
please input your quantity20
please input your unit cost40
NIO
2020-11-24

Do you want to add more tickers? y/nn
Out[17]:
  Acquisition Date Ticker Quantity Unit Cost  Cost Basis  Start of Year
1      2020-11-26   BABA      20      201    4020.0    2019-12-31
2      2020-11-24   NIO      20      40     800.0    2019-12-31
```

Secondly, see the file “stage1_input_process.py”:

This file is written for creating a DataFrame for investors to input the related information.

This file contains 452 lines which can be divided into 2 parts.

The first part is: **investor_please_input(data_base)**:

The aim of this function is to return a DataFrame (named

“merged_portfolio_sp_latest_YTD_sp_closing_high” in the file) that contains all the investors’ input information and performance analysis processes and outcomes.

Line 19-21: **Changestamp(dt1)** is used for making the format of datetime into datestamp, or there will be bugs when merging different DataFrames later.

```
# make the format of datetime into timestamp, or there will be bugs when merging different dataframe
def Changestamp(dt1):
    Unixtime = time.mktime(time.strptime(dt1.strftime('%Y-%m-%d %H:%M:%S'), '%Y-%m-%d %H:%M:%S'))
    return Unixtime
```

Line 24-27: add a new column “datestamp” in “portfolio_df” to store the datestamp type date.

Line 34-50: **getYesterday()** and **getToday()** are used for getting the date of yesterday and today in some specific datetime type, which will be used later for extracting data of S&P and customers’ input.

```
# we need to define yesterday, especially to the format that can be recognized by
# 'Date' column: datetime.datetime(year, month, day, hour, minute)
def getYesterday():
    today = datetime.date.today()
    oneday = datetime.timedelta(days=1)
    yesterday = today-oneday

    year = int(yesterday.strftime('%Y'))
    month = int(yesterday.strftime('%m'))
    day = int(yesterday.strftime('%d'))
    return datetime.datetime(year, month, day, 0, 0)

# same reason as getYesterday()
def getToday():
    today = datetime.date.today()
    year = int(today.strftime('%Y'))
    month = int(today.strftime('%m'))
    day = int(today.strftime('%d'))
    return datetime.datetime(year, month, day, 0, 0)
```

Line 55-81: We created some variables including: **start_sp** (Extracting S&P from year 2005), **end_sp** (which is today), **end_of_last_year** (Year 2019 Dec 31), **stocks_start** (Also as early as Year 2005, since it is quite impossible that someone holds a stock for more than 15 years), **stocks_end_yesterday** (yesterday’s date) and **stocks_end_today** (today’s date).

Line 84-118: This part is about dealing with the data information of S&P 500. We created several variables including **sp500** (The DataFrame of S&P index from the date of start_sp to end_sp), **sp_500_adj_close** (just the closing index number of S&P from sp500), **sp_500_adj_close_start** (the S&P index of the ending of last year), and a function **get(tickers, startdate, enddate)** which is used for making all chosen stock together in one DataFrame.

```
# make all chosen stock together in one DataFrame
def get(tickers, startdate, enddate):
    # This function is only used for later map()'s convenience!
    def data(ticker):
        return (pdr.get_data_yahoo(ticker, start = startdate, end = enddate))

    datas = map(data, tickers)
    # to merge the stocks we chose
    return(pd.concat(datas, keys = tickers, names = ['Ticker', 'Date']))
```

Line 120-308: This part deals with the calculation since performance analysis is based on calculation between different indicators. Since it will take a long time to explain what we have done in each step, I will just show you what is the result of this part. The result is a DataFrame which contains all the calculation processes and needed ratios. The final columns of the outcome mainly include:

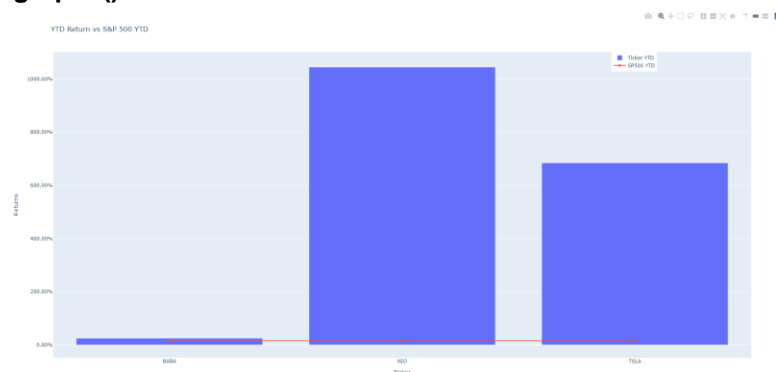
- “**ticker return**” (the rate of return of investor’s ticker on whole time, line 157),
- “**Equiv SP Shares**” (what SP 500 equivalent purchase would have been at the purchase date of stock, line 176),
- “**SP Return**” (the return rate of SP 500 index from the initial date),

“**Absolute Return Compare**” (calculate the difference between SP 500 return and the Ticker return),
 “**SP 500 Value**” (calculate what the ticker price should be if we use the return rate of SP 500),
 “**Stock Gain / (Loss)**” (calculates profit / loss for total stock position taking into consideration of the amount purchased),
 “**Share YTD**” (Year-to-date return for portfolio position),
 “**SP 500 YTD**” (Year-to-date return for SP to run compares),
 “**Cum Ticker Returns**” (cumulative sum of Ticker Share Value (latest FMV based on initial quantity purchased)),
 “**Cum Ticker ROI Mult**” (Cumulative CoC multiple return for stock investments),
 “**Pct off High**” (calculate the percentage difference between the highest price and the current price in the period starting from the start date and now, which is a good indicator to show if the investor is good at selling or not)
 Etc.

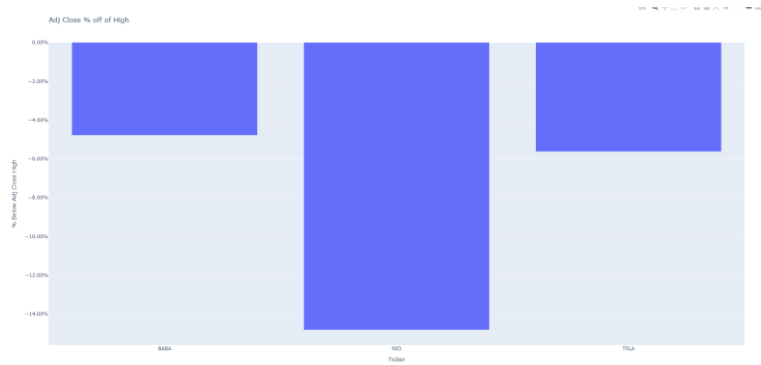
Now after all these data manipulation, the final DataFrame containing all related numbers is done, which is the main job of **Stage V0**.

Line 315-452: The second part of **Stage V0**, which is the visualization of 5 related graphs. When investors input the function name of each graph function, the interactive visualization graph will open in the browser. Here we will use 3 companies as an example:

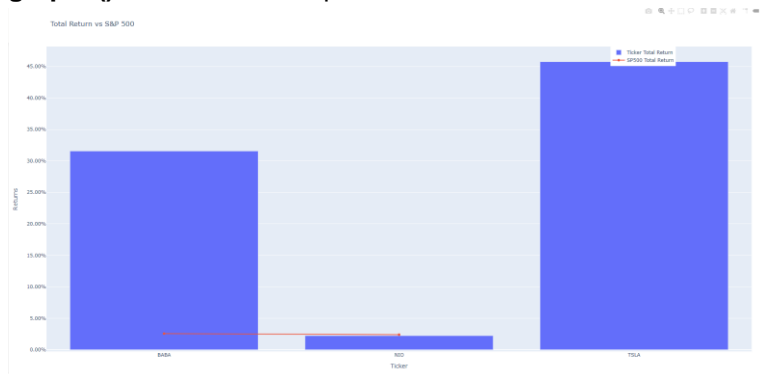
graph1(): Show the returns of investor's stocks and S&P 500.



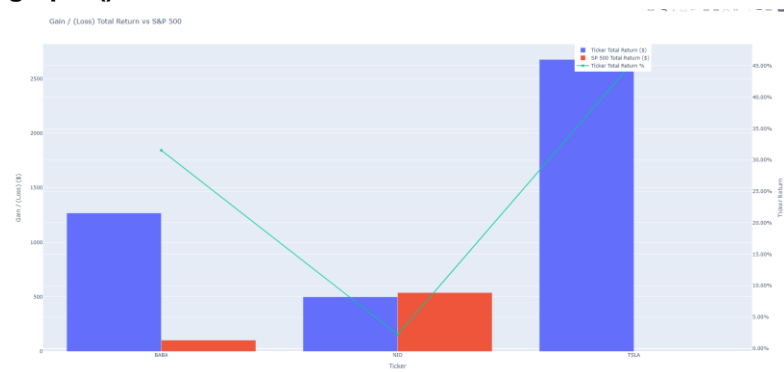
graph2(): Current share price versus closing high (the highest price) since purchased, which can show if this investor is good at selling his stocks at the best time.



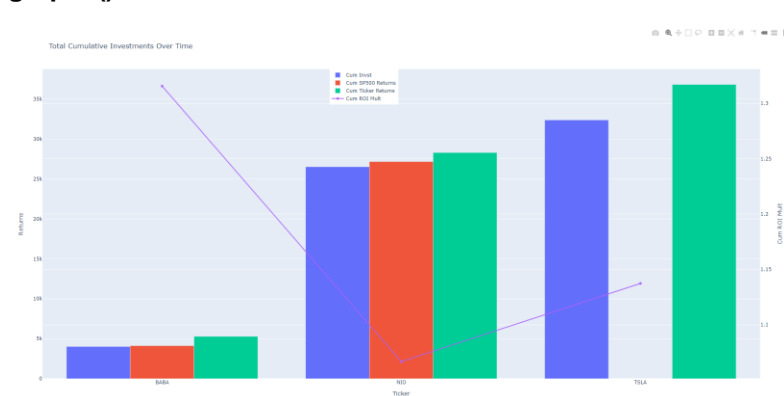
graph3(): Total return comparison charts.



graph4(): Cumulative returns over time.



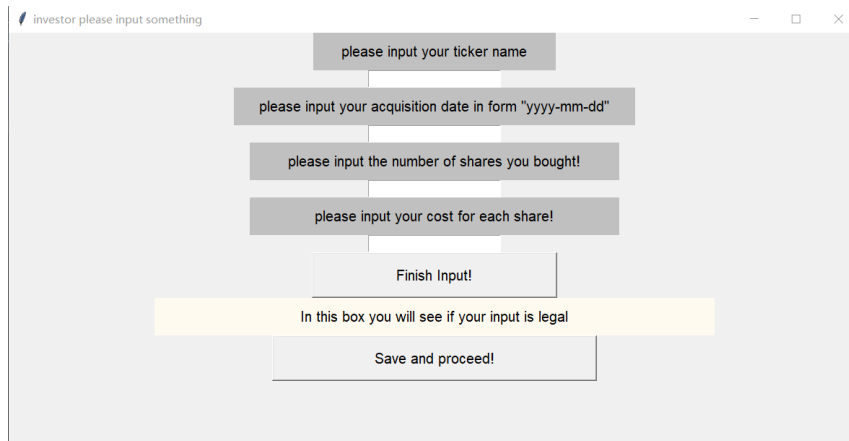
graph5(): Total cumulative investments over time.



Stage V1:

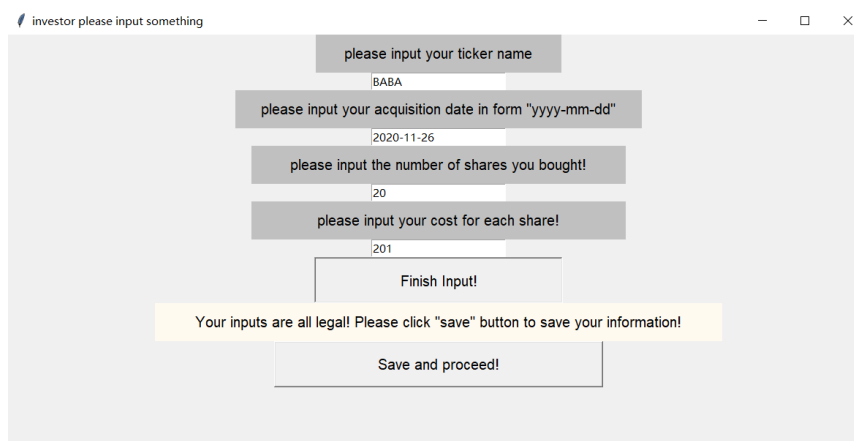
This stage is aiming to create an interface on which investors will be able to input all the needed information (which was previously needed in **Stage V0**).

Firstly, here is the outcome of **Stage V1**:



The screenshot shows a web application window titled "investor please input something". It features a vertical stack of input prompts and buttons. The prompts are: "please input your ticker name", "please input your acquisition date in form 'yyyy-mm-dd'", "please input the number of shares you bought!", and "please input your cost for each share!". Below these is a "Finish Input!" button. Underneath the button is a yellow rectangular area with the text "In this box you will see if your input is legal". At the bottom is a "Save and proceed!" button.

This is the interface investors would see first, and in the white blanks they need to input something legal and related.



This screenshot shows the same interface as the previous one, but with sample data entered into the input fields. The "ticker name" field contains "BABA", the "acquisition date" field contains "2020-11-26", the "number of shares" field contains "20", and the "cost for each share" field contains "201". The "Finish Input!" button is still present. The yellow feedback area now displays the message: "Your inputs are all legal! Please click 'save' button to save your information!". The "Save and proceed!" button remains at the bottom.

When they click the **"Finish Input!"** button, in the label there will be some hints to indicate investors how to continue.

If the investor input something illegal like the ticker name does not exist, the date is on weekend, the number of shares is not integer, or the cost is negative, the respective warnings would come out in the yellow label as below: (some of the scenarios)

investor please input something

please input your ticker name

asdfacca

please input your acquisition date in form "yyyy-mm-dd"

2020-11-24

please input the number of shares you bought!

20

please input your cost for each share!

201

Finish Input!

Your ticker name is not found! Correct your input!

Save and proceed!

investor please input something

please input your ticker name

BABA

please input your acquisition date in form "yyyy-mm-dd"

2020-11-24

please input the number of shares you bought!

20

please input your cost for each share!

201

Finish Input!

Your acquisition date is on weekend! Correct your input!

Save and proceed!

investor please input something

please input your ticker name

BABA

please input your acquisition date in form "yyyy-mm-dd"

2020-11-24

please input the number of shares you bought!

20.1

please input your cost for each share!

201

Finish Input!

Your quantity is illegal! Correct your input!

Save and proceed!

When investors click **"Finish Input!"** with all legal information input, they can decide whether to input more or not:

please input your cost for each share!

201

Finish Input!

Your info

S

Hi

Do you want to input more?

是(Y) 否(N)

If he clicks **Yes**, then all the fill-in blanks will be blank again and he can input once again.

If he clicks **No**, then some other buttons will emerge below as follows:

investor please input something

please input your ticker name

TSLA

please input your acquisition date in form "yyyy-mm-dd"

2020-11-26

please input the number of shares you bought!

20

please input your cost for each share!

400

Finish Input!

Your information has been saved!

Save and proceed!

Graph1: Compare stock's YTD with S&P 500

Show graph1

Graph2: Compare current price with highest price since purchased

Show graph2

Graph3: Total return comparison charts

Show graph3

Graph4: Cumulative returns over time

Show graph4

Graph5: Total cumulative investment over time

Show graph5

quit

The following 5 buttons are corresponding to the 5 graphs we just created in **Stage V0**, and once you click some button, the graph will come out in your browser.

For the code explanation:

See the file "**stock_interface.py**":

There are 209 lines in this file.

Line 21-23: Initialize a window using Tkinter and set some basic settings.

Line 33-199: **input_in_interface()** is used for generating an interactive Tkinter window with all the necessary logics.

Line 202-209: initializing the "**data_base**" variable and use the function **investor_interface_input()** to connect the interface with the input information we collected in **Stage V0**.

See the file "**stage1&2_investment_performance_analysis_and_visualization.py**":

This file is only for running the **Stage V1** interface. After running this file, an information input interface will come out.

Stage V2:

We wrote 2 files about this stage, one is “**recommendation.py**” and the other one is “**stage3_stock_interface_recommendation.py**”.

See the file “**class_input_investor_tickers.py**” first: This file is used for generating a model for the training set and test set and defining some functions to determine the classification of the input ticker.

This file has 235 lines.

Line 12-16: initialization of the empty DataFrame and get the largest 30 US listed companies’ tickers.

Line 19-49: **market_cap_list_function(stock_name, i)** this function is used for returning a DataFrame which will contain some features of one specific stock (the machine learning feature selection process and the corresponding data collection process)

```
In [53]: all_information_needed
Out[53]:
```

	market_cap_list	beta	...	trailingEps	priceToBook
0	7.081906e+11	1.034645	...	9.310	10.484716
1	1.652650e+12	0.823897	...	6.199	13.399742
2	1.606431e+12	1.291049	...	34.202	19.416883
3	1.168364e+12	0.997375	...	51.752	5.509805
4	1.170079e+12	0.997375	...	51.752	5.494558
5	7.072673e+11	1.193175	...	8.778	6.691036
6	7.081906e+11	1.034645	...	9.310	10.484716
7	6.587912e+11	2.151181	...	0.523	41.099940
8	4.700233e+11	0.089971	...	2.230	10.448898
9	4.658139e+11	0.960104	...	4.888	14.424875
10	4.129349e+11	0.435170	...	6.930	5.074051
11	4.067537e+11	0.675065	...	6.360	6.308075
12	3.629797e+11	1.210498	...	7.669	1.505836
13	3.447650e+11	0.364829	...	5.233	7.289886
14	3.392835e+11	1.183727	...	6.663	58.576590
15	3.286147e+11	1.468766	...	6.117	21.430649
16	3.210620e+11	0.732283	...	17.409	4.922822
17	3.124226e+11	1.170078	...	-1.588	3.743990
18	2.911665e+11	1.045669	...	11.561	189.523480
19	2.770461e+11	1.103412	...	2.650	14.993658
20	2.480181e+11	1.591601	...	2.030	1.011823
21	2.501889e+11	0.403149	...	4.420	3.845077
22	2.412747e+11	0.967979	...	10.830	18.162941
23	2.361173e+11	0.932283	...	6.192	22.850487
24	2.328838e+11	1.017322	...	2.230	2.702416
25	2.309439e+11	0.566929	...	1.930	12.411086
26	2.155063e+11	0.845669	...	1.689	23.291483
27	2.149443e+11	0.669767	...	16.356	1.245061
28	2.094405e+11	0.652493	...	1.538	3.208447
29	2.095044e+11	0.672965	...	1.520	1.193230

This is the returned DataFrame that contains all the features of the largest 30 companies listed on US market.

```
In [54]: all_information_needed.columns
Out[54]:
```

```
Index(['market_cap_list', 'beta', 'isEsgPopulated', 'quoteType',  
      'enterpriseToRevenue', 'Fifty_two_WeekChange', 'trailingEps',  
      'priceToBook'],  
      dtype='object')
```

This is the specific typical and influential features we chose.

Line 51-88: some data manipulation job that transfer the original DataFrame into the one we want which is more convenient.

Line 90-101: add another important feature: the price fluctuation.

Line 118-128: put correct labels to each of the companies in training dataset based on the equity researchers’ reports found on the internet from investment banks’ websites.

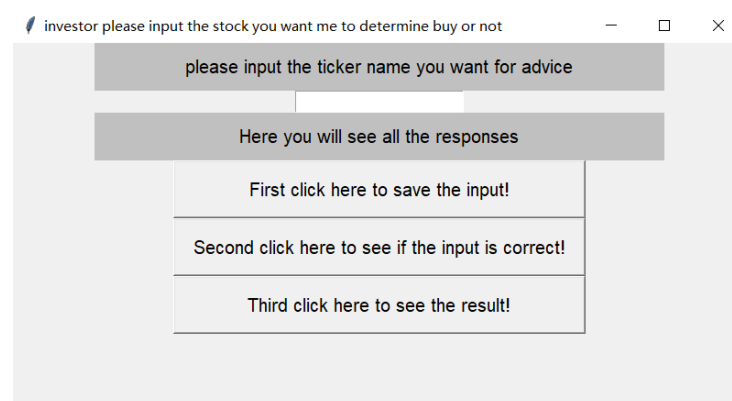
Line 130-163: training the model and find the f1 score and accuracy, which will be later shown on the interface telling the investors how accurate the model is.

Line 165-235: the function **machine_learning_to_recommend_buy_or_not(ticker)** is used for making the investor's input target stock into the same DataFrame as the one we did for the training data. This function needs a parameter "**stock name**" and will return a DataFrame which is the same as the one we used for training dataset, so this will make the process of getting the outcome on the unknown or new dataset more conveniently.

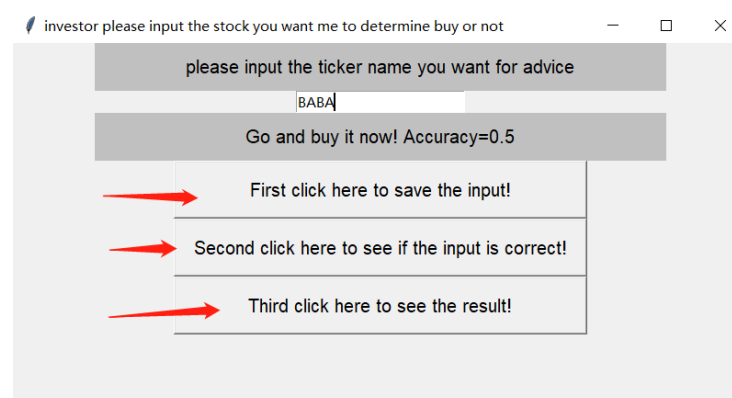
Now see the file "**stage3_stock_interface_recommendation.py**":

This is the last part of **Stage V2**, in which we created an interface to help investors input the stocks they want to consult and through machine learning data process, they will get an immediate reply telling them what's the AI's opinion regarding their input ticker.

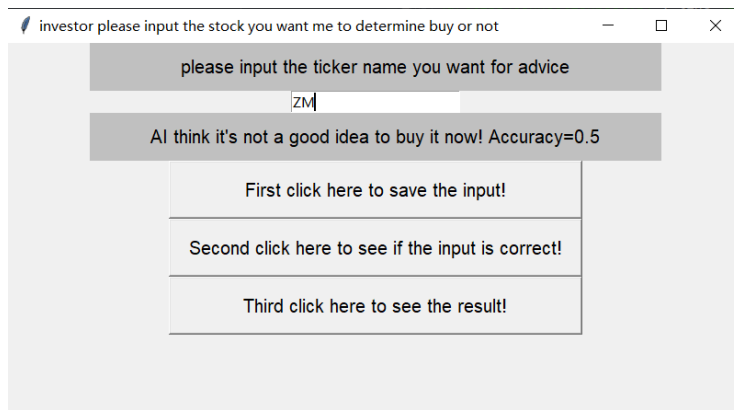
This is what investor would see firstly:



If the investor correctly inputs the ticker name, after clicking in sequence the following 3 buttons (the first button is to save the result, the second button is to judge whether the input is legal or not, the third one is for showing the result in the label), he will find the following notes in the label:

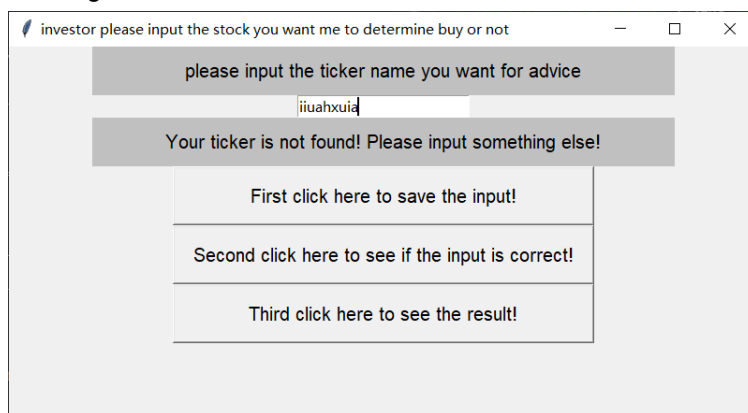


This is the result that the AI recommends him to buy, and the related accuracy ratio is show right afterwards telling the investors how accurate the model is.



Similarly, this is the result that AI thinks it is not good to buy now.

If the investor inputs something illegal and does not exist, similarly as in **Stage V1**, some error messages would come out as follows:



This guarantees that all the data processing is rightly undergoing without errors.

For the code part:

The file "**stage3_stock_interface_recommendation.py**" has 58 lines.

Line 7-9: initialization of the Tkinter window.

Line 11-47: **recommendation_interface()** function is used for creating the window as shown previously, and this includes some other functions like **save_recommendation_input()** and **check_recommendation_name_function()**, which are the parameters in some buttons.

Line 50-57: **show_recommendation_result()**, this function will show the result in the label defined before, and it is also some parameter used in the related button before.

These are all the things we did in coding for all 3 stages.

Bugs fixed & difficulties:

There are mainly 3 kinds of bugs: the ones related to **DataFrame**, **plotly visualization**, and **input data manipulation**.

About DataFrame:

1. The format of time. We aim to make the investor's input information into a DataFrame so that we will be able to use "**Pandas**" package to do further calculations and other manipulations. However, once we created the DataFrame, in "Date" column, the investor's input is in format "String", if we want to merge several DataFrames according to the "Date" value, we should transform all the value into the same type. We firstly used "**dtypes**" function to see what all those types were, and we found that we needed to unify the "String" and "Datestamp", but the "Datestamp" time returned by API is not a number but looks the same as "datetime". We tried many times but still cannot unify them. Finally, I decided to convert all of them into the "Datestamp" number type: firstly, changed the investor's "String" type "Date" into "datetime", then converted both into "Datestamp" type, and then when we tried to merge the DataFrames, it worked.

2. The value assignment in DataFrame. After calculating some ratio based on the numbers in 2 columns of DataFrame, we wanted to assign it to another new column by just using `DF['column_name'][i] = some_calculated_number`, but the **ERROR**: "A value is trying to be set on a copy of a slice from a DataFrame" kept coming out. After reading the documentation, we solved this problem by using a similar function: `DF.at[i, 'column_name'] = some_calculated_number`.

3. When we wanted to merge 2 DataFrames based on the same values of two columns, we found that no matter how many tickers we input, the outcome would always be a DataFrame of 2 rows only containing the first 2 tickers we input. This is because this merge is based on a "multiple to one" mode and we did not mention that in parameters. After changing the line to `merged_portfolio_sp = pd.merge(merged_portfolio, sp_500_adj_close, left_on = 'datestamp', right_on = 'datestamp2', how = 'left', validate = 'm:1')` in which I indicated validate and how, the problem was solved.

About input data manipulation

1. we must make sure that all the inputs that investors typed are in legal form so that we can do the further calculation.

1. For the Ticker part, we must make sure that the ticker he inputs exists, so every time he inputs a ticker, I will have to use the API to see if something can be returned and it is not an empty DataFrame with the following codes: (if no error exists, then it means the input ticker is legal)

```
pdr.get_data_yahoo(ticker, datetime.datetime(
    int(today.strftime('%Y')),
    int(today.strftime('%m')),
    int(today.strftime('%d')),0,0))
```

2. For the acquisition date part, we must make sure that the inputs are in the right form: 'yyyy-mm-dd', or the datetime package cannot recognize, and then change the string into the 'datetime' type. To solve this difficulty, I used the 'replace' module to change the '-' into blank, and used split to convert each part into list, and used 'datetime()' to change the string type into the 'datetime' type.

Besides, we also must make sure that the input date must be some weekday because the market opens only on weekdays. This bug puzzled me a long time because the output was sometimes an empty DataFrame after I randomly typed in the portfolio information (because no price was available on weekends). To solve this, I used the 'calendar.weekday()' method to determine if the date is on weekends.

2. Besides, since we have many different functions to handle the input data, so in order not to make the number of parameters of those functions too big, we decided to globalize every main variable, so that no bugs like "XX was referenced before the function" would come out.

About interface

1. The biggest difficulty is that the interface thing must relate to nearly all functions in **Stage V0**, which are around 16, thus I spent a long time clearing my mind to make the logic of the interface correct.

2. When an investor has input the information of one ticker and he chooses that he wants to input another one, in this case I must delete all the labels & buttons before the recurring so that it looks the same as before.

3. For different errors there will be different error messages in the main label, but because many errors are parallel and the newly defined functions are not well-defined, so I spent a long time connecting each error scenario with the corresponding error message. Honestly, I could insert many separate labels to show the investor if each blank they filled is legal or not, yet it is too ugly and space consuming.

4. A very serious problem is that I did not draft the Tkinter code in a defined function, because I thought it is easier to debug and run, however a problem is that because I defined many global variables for convenience, each time I run the part of the code I need, probably the value of some variables was not updated so that an error would occur. A typical example is that each time as long as I don't run the whole ".py" file, an error which says that "the base level window cannot be constructed on the top-

level window; thus, I need to use **tk.Toplevel(window)** to solve", but actually all I needed to do is run again and the error would not occur.

Future Scope

With the limited time and experience, we did not manage to add the company information that would have helped the investors make a better decision. With this added information, the investors would be able to view the company information like screening the most recent financial data of the companies in this industry, he can receive some stock recommendations within this industry (the logic is: screen every company of this industry, pick those whose market capacity is among top 30%, and pick the ones with the highest ROE).

We also would like to improve the accuracy of the recommendations given to the investors. Currently, the accuracy for some of the stocks is only around 50% even though the stock is worth investing in. This is mainly because the training dataset is so small, and I could not find much useful equity researchers' insights about the stocks. However, we could also improve this through enlarging the dataset and take in more labels (multi classification), and maybe with more data, the model can be better trained in the future.