STOCK ANALYTICAL TRADING SCANNER

THE SATSY

Aleksei Makushin

Content

Background Research/Statement of the problem 2
Identification of end users 2
Interview with end user 3
Summary of the end user feedback 4
Existing/Similar systems 4
..... 5
Analysis of my system features based on previous findings 6
Acceptable Limitations of My System 7
Proposed Solution Details 8
Set of objectives 8
High Level Structure diagram 14
File structure (7 python files that operate my program) 15
Database Structure: 17
The Flowchart Diagram 24
The SATSY Scanner (Extended) Flowchart Diagram 29
Pseudocode: 29
Pseudocode of Algorithm 2: 31
The Gapper Scanner 32
Pseudocode on SQL tables: 32
Pseudocode for : Algorithm 4 33
The US Stock Table 34
Pseudocode on SQL Tables: 34
The ML Algorithm 36
Pseudocode on SQL tables: 36
Graphical User Interface. Flowchart diagram 39
Flowchart diagram. Gapper Scanner GUI 40
Flowchart diagram. SATSY Scanner GUI 41
UML DIAGRAM 44
Video of testing and screenshots of some of the testing: 99
Full system video overview: 103
Evaluating the Success of the Solution 103
User Feedback 112
Analysis of feedback 113
Conclusion 113

ANALYSIS SECTION

Background Research/Statement of the problem

Stock scanners are an essential tool for investors and traders alike, as they offer an efficient way to sift through the vast amount of stocks and identify potential trades that meet specific criteria. One of the significant advantages of using a stock scanner is the ability to identify opportunities before institutional traders and banks, which increases the likelihood of achieving a higher return on investment. Moreover, stock scanners provide a systematic approach to trading by using pre-determined criteria to identify potential trades. This helps traders and investors avoid making emotional decisions based on market fluctuations. Furthermore, using a stock scanner can be a cost-effective way to identify potential trades, particularly compared to paying for expensive investment advice or spending time analyzing stocks manually.

Stock scanners are generally divided into two main categories fundamental and technical stock scanners. Fundamental stock scanners examine a company's financial situation, current market and economic conditions to sort securities based on intrinsic value. While technical stock scanners sort stocks depending on different technical tools that consider the price and volume of the stock. Having done some research on the availability of both technical and fundamental stock scanners, I have found that the market has an unfilled niche – technical stock scanners that work on an intraday level. Many of the scanners that I have analyzed were primarily focused on providing a fundamental analysis of the stocks. Additionally, the few scanners that did provide technical analysis were focused on the daily level or weekly level, rather than intraday level. Moreover, technical scanners were not customizable, and quite a lot of them were poorly designed and inconvenient in use. Furthermore, many stock scanners provided an outdated information on some essential values and made mistakes in calculations when calculating technical indicators. Lastly, many of them were unaffordable for beginner day trader, due to high price (up to 200$ per month).

To address these issues, I have decided to develop a customizable, affordable day trading scanner that is convenient to use, and which provides the end user with updated values on stocks. Such a scanner would provide day traders worldwide with the necessary tools to execute their trading strategies and achieve success in the stock market.

Identification of end users

The end user would be day traders who are looking for a customizable, affordable, and user-friendly day trading scanner that focuses on technical analysis at an intraday level.

Interview with end user

To gain a deeper understanding of the necessary requirements for a day-trader stock scanner, I interviewed Alexander Makuvin, an experienced trader with three years of experience. During our conversation, he shared his frustrations with existing day-trading scanners and reiterated the problems I've encountered myself.

1. What specific areas should a stock scanner program address that basic stock scanners do not?

I've come across a lot of programs that use outdated data like the stock's float or price, which really frustrates me because my trading strategy relies heavily on these values being accurate. Moreover, I think it's important for the program to have a user-friendly GUI that is easy to navigate, with only the necessary fields and no unnecessary information cluttering the screen. A simple GUI is what I would like to see.

2. Can you recommend any stock scanners that I should use as a reference for my project? Additionally, I'm interested to know what aspects of these scanners you appreciate and what aspects you don't.

As a day trader, I have primarily relied on two scanners. One of them is FINVIZ.com, which I find to be a valuable tool for my trading style. Its simple GUI enables me to quickly spot opportunities in the market, and it also includes helpful indicators like RSI, volume, and gap. However, I do find the large number of indicators on the tableau to be overwhelming, which can make it difficult to find the ones that I need.

Another scanner that I use is Day Trader Dash, which also has a straightforward interface that I appreciate. Additionally, it includes a colour sorting feature that makes it more convenient to use.

However, one major downside of both scanners is that they are web-based, which can be inconvenient since I have to log in and set them up every morning before trading, which can be time-consuming. Also, another drawback of some of these scanners is that they provide outdated information on various stocks, which is a significant disadvantage. Furthermore, they can be quite costly, with prices that can be several hundred dollars per month, making them unaffordable for some of my colleagues.

3. From a technical analysis perspective, which tools do you believe should be included in my program?

In my opinion, some of the programs we've discussed aim for the wrong target by trying to incorporate every possible indicator into their platform. This is unnecessary, as most traders I know focus their trading on only six indicators, which cover about 99% of what day traders use. Personally, I cannot trade without seeing the stock's Gap, Float, Volume, Price, and maybe relative volume. These are the most crucial indicators, but if you want to add something extra to the program, I suggest including the RSI and the Number of Consecutive Candles indicator, as they may also be beneficial to day traders.

Incorporating other technical tools is not necessary, and it may slow down the program, which I believe is entirely unacceptable. The purpose of these programs is to identify opportunities quickly, not slowly.

4. Would it be beneficial to create two separate programs, one for a general stock scanner and another for a gap scanner, or do you think it's better to combine them into one program?

I believe that dividing the general stock scanner and gap scanner into two separate entities would be a great idea. These are two different strategies and having separate programs for each would benefit end-users.

5. How can I further enhance my program to make it more competitive, assuming that the scanner is already functioning properly?

   I suggest adding a stock analysis algorithm to enhance the program's competitiveness. This feature could provide users with a theoretical expected increase in the stock price. Additionally, incorporating a news feed into the program could be beneficial as the stock's potential can be directly influenced by market news.

Summary of the end user feedback

After conducting an interview with Aleksander Makuvin and reflecting on my own experience, I have identified several key features that end-users would like to see in my stock scanner:

1. A user-friendly and straightforward GUI that is convenient and easy to navigate.
2. Technical indicators such as Gap, Volume, RSI, Last Candle Volume, Number of Consecutive Candles, Aggregate Volume and Relative Volume should be available.
3. Fundamental indicators such as Price, Float, Sector, and Country should also be available.
4. Users should be able to view news related to selected stocks.
5. To cater to different strategies, I plan to create two programs in one: a Gapper Scanner and an Extended Scanner.
6. To enhance the program's competitiveness, I am considering incorporating a Machine Learning Algorithm to provide users with a theoretical possible increase in the price of the stock.

Existing/Similar systems

During my research, I examined several popular stock screeners that are commonly used by day-traders, including "FINVIZ.COM" and "DAY TRADER DASH". After using them for a few weeks, I identified some of their advantages and disadvantages.

Existing System 1 (Finviz.com)

Overview:

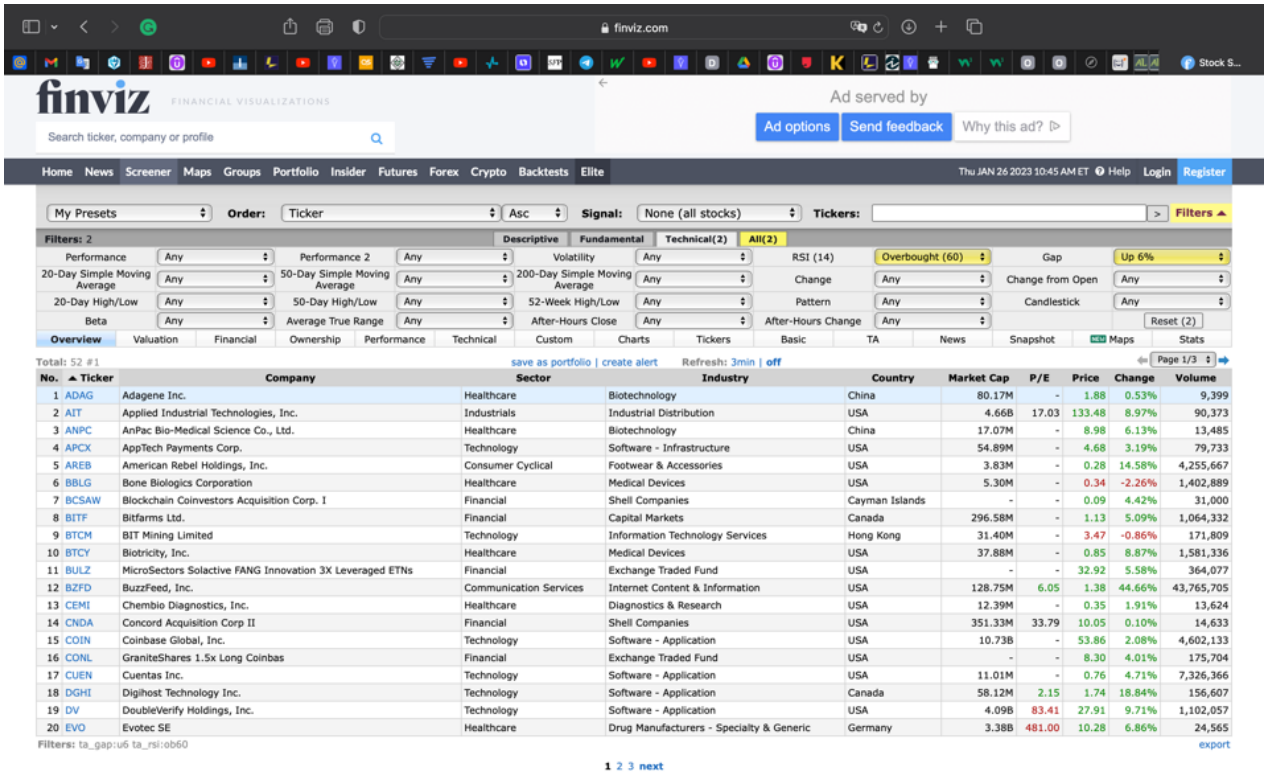
FINVIZ.com is a popular stock screener used by many day traders. It is a web-based platform that allows users to scan stocks based on a wide range of criteria. Additionally, FINVIZ.com includes various charts, news feeds, and other information that traders may find helpful. FINVIZ.com has user-friendly interface which presents results in the form of ordered table.

Advantages:

1. User-friendly interface with easy-to-use filters.
2. Wide range of screening options including technical, fundamental and descriptive criteria.
3. Provides real-time data with minimal delay, making it useful for day trading.
4. Offers various visualization tools, such as heat maps and charts, for better understanding of data.
5. Provides email alerts for specific screening criteria.

Disadvantages:

1. Overwhelming amount of data and options on the homepage, which can be confusing.
2. Limited number of free screening options, with more advanced options requiring a subscription to FINVIZ Elite/
3. Data can be delayed for non-Elite members, making it less useful for those who do not have a subscription.
4. The free version of the screener displays ads, which can be distracting for users.



Existing System 2 (Day Trader Dash)

Overview:

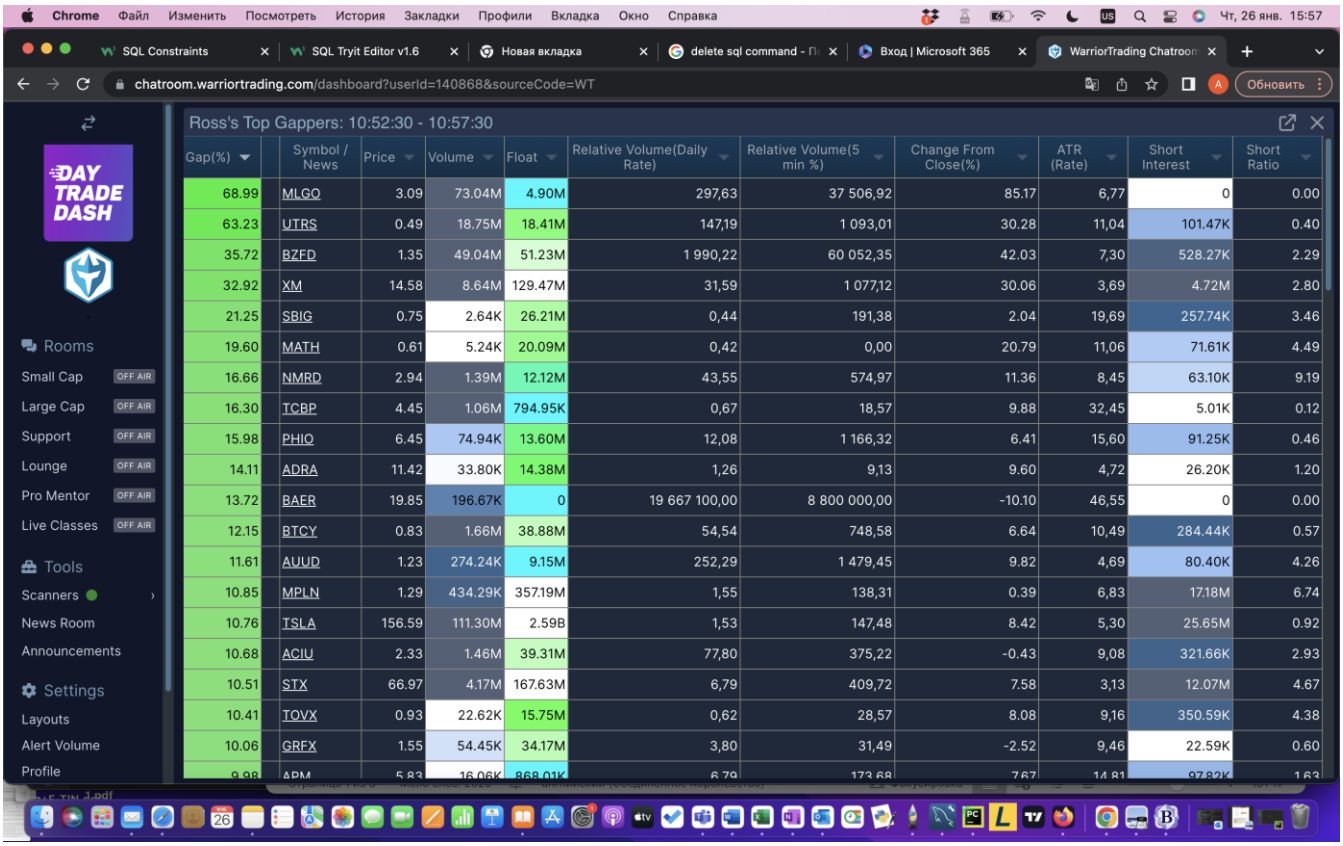
DayTradeDash is a platform created for day traders, featuring various types of scanners, such as gap scanner, momentum scanner, news scanner, and reversal trading scanner. Additionally, it allows users to interact with other traders on the platform through the chat on that platform.

Advantages:

1. The interface is designed to be user-friendly and includes a colour sorting feature.
2. The program offers real-time stock data that updates automatically every five minutes.
3. Users can communicate with each other through the chat feature.
4. The program includes various scanners for the user to choose from and allows to use multiple scanners simultaneously.

Disadvantages:

1. The platform does not permit end-users to personalize their stock scanners to suit their needs. Modifications to the scanners are not possible.
2. Certain scanners provide outdated information on stocks.
3. The system experiences delays as the stock data is refreshed every 5 minutes.



Gap(%)	Symbol / News	Price	Volume	Float	Relative Volume(Daily Rate)	Relative Volume(5 min %)	Change From Close(%)	ATR (Rate)	Short Interest	Short Ratio
68.99	MLGO	3.09	73.04M	4.90M	297,63	37 506,92	85.17	6,77	0	0.00
63.23	UTRS	0.49	18.75M	18.41M	147,19	1 093,01	30.28	11,04	101.47K	0.40
35.72	BZFD	1.35	49.04M	51.23M	1 990,22	60 052,35	42.03	7,30	528.27K	2.29
32.92	XM	14.58	8.64M	129.47M	31,59	1 077,12	30.06	3,69	4.72M	2.80
21.25	SBIG	0.75	2.64K	26.21M	0,44	191,38	2.04	19,69	257.74K	3.46
19.60	MATH	0.61	5.24K	20.09M	0,42	0,00	20.79	11,06	71.61K	4.49
16.66	NMRD	2.94	1.39M	12.12M	43,55	574,97	11.36	8,45	63.10K	9.19
16.30	TCBP	4.45	1.06M	794.95K	0,67	18,57	9.88	32,45	5.01K	0.12
15.98	PHIO	6.45	74.94K	13.60M	12,08	1 166,32	6.41	15,60	91.25K	0.46
14.11	ADRA	11.42	33.80K	14.38M	1,26	9,13	9.60	4,72	26.20K	1.20
13.72	BAER	19.85	196.67K	0	19 667 100,00	8 800 000,00	-10.10	46,55	0	0.00
12.15	BTCY	0.83	1.66M	38.88M	54,54	748,58	6.64	10,49	284.44K	0.57
11.61	AUUD	1.23	274.24K	9.15M	252,29	1 479,45	9.82	4,69	80.40K	4.26
10.85	MPLN	1.29	434.29K	357.19M	1,55	138,31	0.39	6,83	17.18M	6.74
10.76	TSLA	156.59	111.30M	2.59B	1,53	147,48	8.42	5,30	25.65M	0.92
10.68	ACIU	2.33	1.46M	39.31M	77,80	375,22	-0.43	9,08	321.66K	2.93
10.51	STX	66.97	4.17M	167.63M	6,79	409,72	7.58	3,13	12.07M	4.67
10.41	TOVX	0.93	22.62K	15.75M	0,62	28,57	8.08	9,16	350.59K	4.38
10.06	GRFX	1.55	54.45K	34.17M	3,80	31,49	-2.52	9,46	22.59K	0.60
9.98	APM	5.83	16.06K	868.01K	6.79	173.68	7.67	14.81	97.82K	1.63

Analysis of my system features based on previous findings

Having analysed the existing systems, I have come up with a list of features that my system should have to be a good system.

Feature	Why incorporating this feature?

Simple and easy to use GUI.	Makes system user-friendly and easy to navigate, without causing any disruptions to the end users, allowing them to remain focused on the intended purpose of the system.
Program should provide real-time data with minimal delay, making it useful for day trading.	Enables the system to produce results quickly, thus fulfilling the primary purpose of the program.
Program should have two scanners gappers scanner and extended stock scanner with functionality of reversal scanner and momentum scanner.	Enables the utilization of various scanners, with the data being saved in the text field of that scanner. So that user could easily access the results of the stock analysis.
News should be shown below the stock scanner.	Enables to get an understanding of the news catalyst, thus enabling informed investment decisions to be made.
The program should have only the indicators that are necessary for three types of scanners, nothing extra.	The program should include only the essential indicators for the three types of scanners, without any unnecessary extras.
The program should provide up-to-date information on stocks.	Provides up to date information as certain criteria like float is very important for day traders.
The user could easily jump from one scanner to another, and the information is saved, when he changes scanner.	Provides temporary storage.

Acceptable Limitations of My System

1. My system may not be able to perform instantaneous data analysis like Finviz.com, as I will not be able to distribute the processing across multiple servers and request data concurrently.
2. The prediction of stock price increase by my machine learning algorithm may not be accurate. As it requires a deep data analysis and lots of processing power to analyse lots of data to come up with an algorithm that predicts stock prices almost perfectly.
3. My system may not be able to provide real data because it is very expensive to request real time data up to 200$ per month and instead may provide data with a 15-minute delay.

Proposed Solution Details

Deciding on HARDWARE:

To develop a program I will use laptop running on MacOS operating system.

Deciding on Programming Language:

I have chosen Python programing language to create Stock screener because of the following reasons:

1. Python's simplicity and ease of use mean that it typically takes less time to develop a project compared to Java or C++. This is particularly true when using pre-built libraries.
2. Python has a large and active community of developers, which means there are plenty of resources and ideas shared online, forums like stackoverflow.com.
3. Python has a vast number of libraries and packages that make it easy to build complex applications quickly. For example, libraries like Pandas and NumPy can be used for data analysis, while libraries like BeautifulSoup can be used for web scraping, which I all plan to use when developing my application.
4. Python is a cross-platform language, which means it can be used on Windows, Mac, and Linux. This makes it possible to develop applications that can be ran on different operating systems.
5. Python has several powerful data analysis and visualization libraries such as Pandas, NumPy, and Matplotlib, which can be used to analyze and visualize stock data effectively.

Selecting software for program developing:

I have chosen PyCharm as an IDE (integrated development environment), because of:

1. User-friendly interface - simple and easy-to-use
2. Error highlighting - highlights errors and warnings, which helps in debugging the code quickly.
3. Intelligent code completion - predicts the next line of code, which saves time.
4. Debugging capabilities - aloows to debug the code quickly and efficiently.

Set of objectives

After my research was completed, I was able to create a set of main operations and objectives that my program would have to meet to be a good quality program. My program will strive to incorporate the main advantageous features of the systems that I have analysed and merge them into one program.

Main objective:

Development of the affordable, user friendly and convenient program Stock Analytical Trading Scanner “THE SATSY" that would incorporate three main scanners like gap scanner, momentum scanner and reversal scanner to analyse stocks traded on NASDAQ, NYSE and AMEX. The SATSY would

also provide a news feed, for the found stocks, and predict a potential increase in price for stocks selected by the gapper scanner.

The system will consist of following main operations:

1. Gappers Scanner
2. Extended Scanner (Reversal scanning and Momentum scanning)
3. Machine Learning Algorithm which provides with an estimate on the potential increase in price of the stock.
4. Convenient Graphical User Interface.
5. Program that requests data from an external API , and gathers data from websites
6. Program that analyses received data.
7. Program that is responsible for the connection with database.

Set of Objectives:

1. The program needs to establish a connection between Python and a MySQL database.
    a. Create a class called "AppConnection" that will handle the database connection.
    b. The class method should be able to print a message to the console confirming that the connection has been established.
    c. The class method should also be able to create a database.

2. The program needs to be able to interact with the database.
    a. Create a class called "Database" that will connect to a specific database on the MySQL server.
    b. The class should have a "commit" method that will save any changes made to the database.
    c. The class should also have an attribute that allows changes to be made to the database.

3. The program needs to be able to gather data on stocks.
    a. The program should be able to collect data from the marketwatch.com website. (from the “KEY DATA ” table)
        i. The data should be stored in a list with table names serving as keys in a dictionary.
        ii. The data should be reformatted, with numbers like "T" for trillion, "B" for billion, "M" for million, and "K" for thousand replaced by the appropriate numerical values.
        iii. The data in these cells should be reformatted into acceptable types such as integers, floats and strings.
        iv. Exception handling methods should be implemented at both the formatting and request levels. (Exception handling)

        a. If a request fails, the data should be returned as a dictionary with 0 values for all parameters.
        b. If formatting fails, the particular parameter should be returned as 0.
    v. The method should return a dictionary in the form of {'Open': value, 'Day Range': value, '52 Week Range': value, 'Market Cap': value, 'Shares Outstanding': value, 'Public Float': value, 'Beta': value, 'Rev. per Employee': value, 'P/E Ratio': value, 'EPS': value, 'Yield': value, 'Dividend': value, 'Ex-Dividend Date': value, 'Short Interest': value, 'Average Volume': value }.
b. The program should be able to request aggregate data on stocks for the Gapper scanner by sending requests to external API.
    i. The method should be able to interpret values that come in the form of JSON.
    ii. The method should be able to format data into the correct data types, such as integers and floats.
    iii. The method should have an exception handling method so that if it fails to receive data or format data, the program won't crush.
    iv. The method should return values in the form of a dictionary {"gap": value, "price": value, "volumeDay": value, "volumeMin": value}.
c. d. The program should have the ability to request data on stocks at the 'daily level' for the SATSY scanner through API requests.
    i. The method should be capable of requesting data on the stock within the specified day range, and specified timespan.
    ii. The method should have an exception-handling mechanism.
    iii. The method should return data in the form of a simplified array, without any meta-data.
d. The program should be capable of requesting the MACD indicator through API requests.
    i. The method should be capable of returning the value of the MACD.
    ii. The value of the MACD should be formatted into the correct data type.
    iii. The value of the MACD should be returned in the form of a dictionary. {'timestamp': value, 'value': value, 'signal': value, 'histogram': value}
e. The program must be able to request the RSI value, through API requests.
    a. The method should be capable of requesting the RSI value.
    b. The method must include an exception handling mechanism. If the method fails, the RSI value returned should be -1. Otherwise, the correct value should be returned.
    c. The method should return value as a float number
f. The program should be able to request news.
    a. The method should be able to request news by sending API requests.
    b. The method should have an exception handling mechanism, so that if exception is returned, the values are returned as a list ['NA'].
    c. The method should be able to return news in the form of a list.

4. The program should be able to process data, and calculate the stock's price, gap, aggregate volume, volume, price volatility, extension of vwap, relative volume, RSI, and the number of consecutive candles.
    a. The program should have a method that converts each item in the array into an object of the Candle class.

i. The Candle class should have a following attributes: volume, open, close, high, low, number_of_transactions, vwap.
ii. The Candle class should have a method that determines the colour of the candle, and stores the value as an attribute.
iii. The program should have a method that appends Candle objects to a list of Candle objects.

b. The program should have a Candle_AGG class that is responsible for returning values for the stock, such as price, gap, aggregate volume, volume, price volatility, extension of vwap, relative volume, RSI, and the number of consecutive candles.
    i. The Candle_AGG class should be created.
    ii. The class should have a method that returns the closing price of the last candle.
    iii. The class should have a method that returns the number of consecutive candles as a tuple.
    iv. The method inside the class should be able to calculate the relative volume over a period of 11 candles.
    v. The method inside the class should be able to return the gap of the stock.
    vi. The method inside the class should be able to return the value of the total volume experienced by the stock.
    vii. The method inside the class should be able to return the extension of vwap
    viii. The method inside the class should be able to return the value of price volatility.
    ix. The method inside the class should be able to return the value RSI.
    x. The method inside the class should be able to return the value of macd.
    xi. The public method should be able to return all the values price, gap, aggregate volume, volume, price volatility, extension of vwap, relative volume, RSI and number of consecutive candles in the form of dictionary.

5. The program should be capable of generating a database table that stores parameters for each stock obtained from both a CSV file sourced from Nasdaq.com and through web scraping from marketwatch.com.
    a. The class US_Stock_Table should be implemented to handle the creation of the US_Stock_Table, created upon file path parameter.
    b. The class should have an attribute that utilizes the Database class to establish a connection with the database.
    c. The class method should be able to generate the table inside the database.
    d. The class method should be able to upload data from the CSV file into the database table.
    e. The class method should be able to leverage the Stock class's functionality, specifically the method that scrapes data from marketwatch.com, to obtain values for each stock in the table.
    f. The class method should be able to compare public float and shares outstanding values, and the return the value of one of the parameters.
    g. The class method should be created to streamline the process of table creation, value updates, and insertion by placing them all under a single method.

6. The program should be able to find gappers on the market.

a. The class should be created Gappers_Table, which would be responsible for creation of the gappers, selection and insertion into the database.
b. The method inside the class should be able to create table in the database.
c. The method inside the class should be able to select stocks with float and price in the specified range, and select stocks with correct sector and country specified in the constructor.
    i. The method should be able to access the US_Stocks table and querry results with specified parameters.
d. The method should be able to take the results of the selected stocks and find among selected stocks gappers that are within the specified gap range.
e. The results of the method should than be uploaded to the Gappers table
f. The method should be created that would be responsible, for creating, selecting, and uploading data into the table by calling one public method
g. The method should be created that would allow to access the results in the database, and request news on these results.
    i. The method should be created that would query the results in the database, and order by the gap
    ii. The method should be created to find news on every gapper in the gappers table
    iii. The method than should return tuple, with gappers results and news on these gappers.

7. The program should be able to screen stocks on the market (reversal scanning and momentum scanning)
    a. The class Scanner should be created and the constructor of this class should take parameters like gap, price, float, sector, country, RSI, relative volume, extension of vwap, price volatility, number of consecutive candles, aggregate volume and volume of the last candle.
    b. The method inside the class should be able to create table in the database.
    c. The method inside the class should be able to select stocks with float and price in the specified range, and select stocks with correct sector and country specified in the constructor.
        i. The method should be able to access the US_Stocks table and query results with specified parameters.
    d. The method should be able to take the results of the selected stocks and find among selected stocks, stocks that are within the parameters specified.
        i. The method should be able to request data on stocks on the daily level, the RSI of the stock and MACD.
        ii. The method should be able to compare the specified parameters in the constructor with the parameters of every stock, and save the stock that follows all the parameters.
    e. Saved stocks which follow specified parameters should than be inserted into the table.
    f. The method should be created which would be responsible for creating selecting and uploading of stocks, basically linking all the processes above into one public method.
    g. The method should be created that would allow to access the results in the database, and request news on these results.

        i. The method should be created that would query the results in the database, and order by the stock RSI and gap.
        ii. The method should be created to find news on every selected stock in the scanner table.
        iii. The method than should return tuple, with scanner results and news on this selected stock.

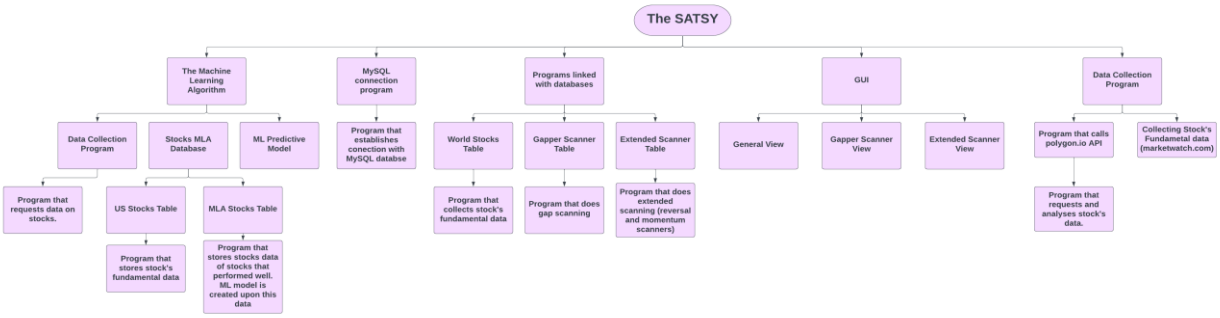
8. The program should have a convenient, beautiful GUI.
    a. The GUI should be resizable.
    b. The GUI should have options to choose from. Gappers Scanner or SATSY Scanner.
    c. The user should be able to specify all the parameters required to create Gappers Table.
        i. Exception handling should check if data is entered correctly, if not popup will be displayed on the screen, asking to renter the data.
    d. The user should be able to specify all the parameters required create Scanner.
        i. Exception handling should check if data is entered correctly, if not popup will be displayed on the screen, asking to renter the data.
    e. The user should be able to see the results of the Gapper Scanner, and SATSY Scanner in two text boxes, one with news for the selected stocks and another with selected stocks and their parameters.
    f. The user should be able to reset the fields.
    g. The user should be able to start the program when he pushes button enter.
    h. The program should be able to change appearance mode.

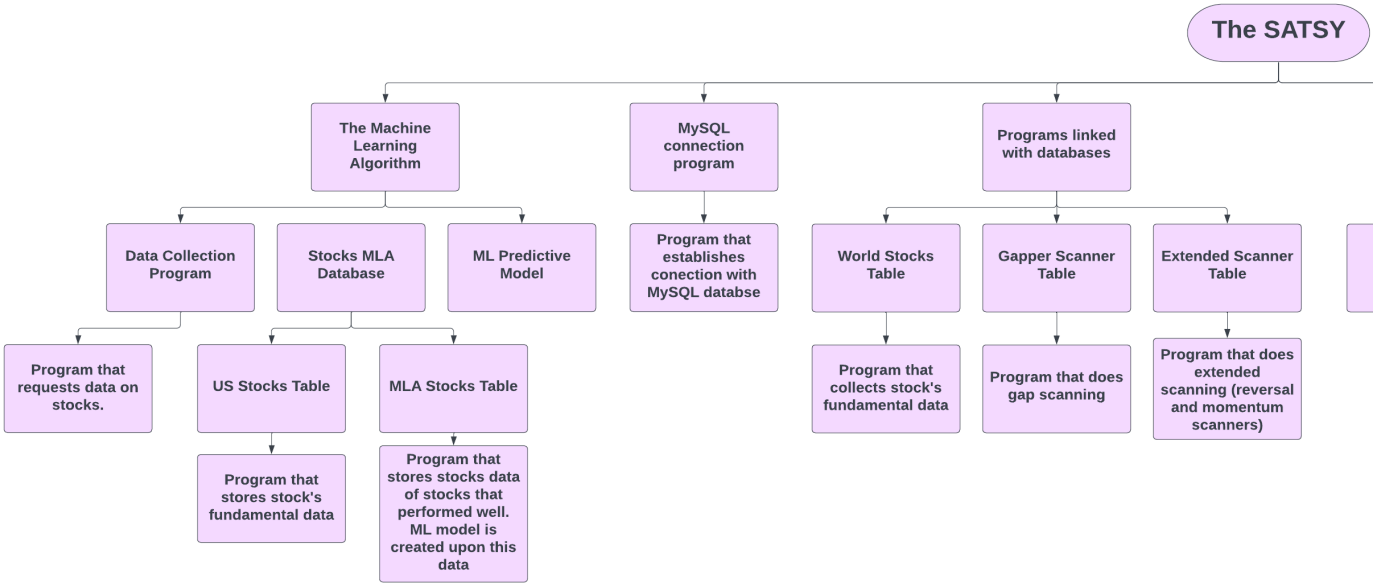
9. The program should be able to predict potential increase in the stock's price
    a. The program should be able to create database with stocks traded on the market, and able to find data on stocks that have gained the most over the specified period of days.
        i. The class should be created ML_Table
        ii. The method in the class should be able to create a table in the database
        iii. The method in the class should be able to get the array of working days in the specified timespan.
        iv. The method in the class should be able to upload data from the csv file into the database
        v. The method in the class should be able to find stock gainers for every working day that lies within the timespan.
        vi. The method in the class should be able to find any extra data on the gainer stock, and upload gainers into the database
        vii. The method in the class should be able to upload data into the table in the database.
    b. The program should be able to create a ML model based on the stock gainers stored in the database.
        i. The function should be able to query data from the table in the database.
        ii. The program should be able to train the model.
        iii. The program should be able to return the model.
    c. The program should have a function predict that would use the model to predict increase in the prices of stocks.

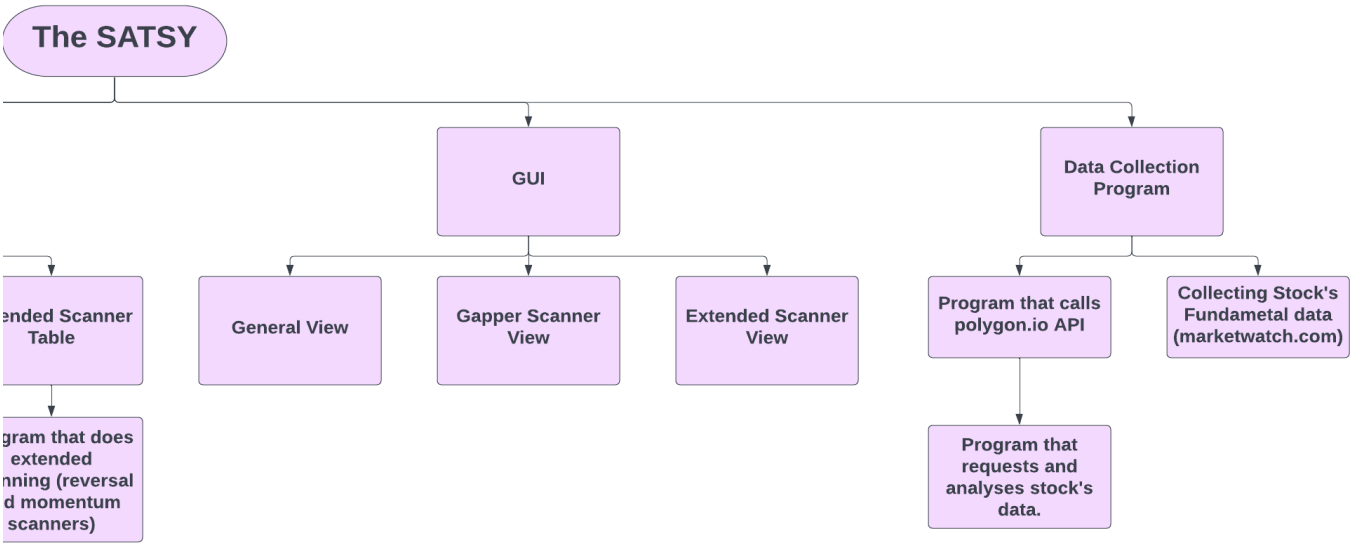
i. The value of the predicted price would be returned from the function

DESIGN SECTION

High Level Structure diagram







File structure (7 python files that operate my program)

GUI.py
DATABASE.py
DATAREQUESTS.py
CANDLESCLASSES.py
SQL.py
MACD.py
ML.py

GUI.py

This file is the main program that is responsible for the graphical user interface, it allows users to interact with the program by allowing them to select different scanners GAPPERS_SCANNER and SATSY_SCANNER, changing the appearance mode and view results displayed as text in the news_text_box and results_text_box. This python file is the main file.

SQL.py

The purpose of this Python file is to facilitate communication between the Python program and a MySQL database. Its primary function is to establish a connection between database and tables in MySQL with the program. It also contains various methods that make the process of saving data to the table more efficient and convenient. In essence, this file serves as a mediator, enabling the program to interact with the database seamlessly.

DATABASE.py

This file contains three primary classes that perform distinct functions:

1. The "US_STOCKS_TABLE" class is responsible for uploading data from a CSV file sourced from Nasdaq.com into the database. It then searches for additional information from the marketwatch.com website and uploads it into the database as well.
2. The "GAPPERS_TABLE" class is responsible for identifying gappers and uploading them into the database. The class sorts the gappers and returns the results in the form of a list. Also, it is responsible for finding news headlines for every stock in the "GAPPERS_TABLE".
3. The "SCANNER" class conducts an extensive search of stocks based on specified parameters. (Responsible for momentum scanner and reversal scanner). Once stocks that fit the criteria are identified, they are uploaded into the database, sorted, and the results are returned in the form of a list. Also, it is responsible for finding news headlines for every stock in "SCANNER" table.

DATAREQUEST.py

The primary role of this file is to manage API calls and data requests. It is achieved through leveraging a request module that enables it to access data from various sources, including the marketwatch.com website and the polygon.ai API server. "Stock" class is responsible for handling all requests to these sources and obtaining the necessary data.

CANDLESCLASSES.py

This file comprises two classes, each serving a distinct purpose. The first class is "Candle," which is responsible for organizing the data of a single candle under the "Candle" class.The second class, "Candle_Agg," performs a more advanced function. It interprets the movements within the market by analyzing specific parameters of every candle for a particular stock during the day. The results of this analysis are returned in the form of a dictionary, which includes information such as price, gap, aggregate volume, volume, price volatility, extension of volume-weighted average price (VWAP), relative strength index (RSI), relative volume, and the number of consecutive candles. These classes serve a crucial role in enabling the program to interpret and analyze the movements within the market.

MACD.py

The primary function of this file is to analyze the Moving Average Convergence Divergence (MACD) indicator over a specified period that spans several candles. Its purpose is to calculate and return the value of the histogram associated with the MACD indicator.

ML.py

The main purpose of this file is to facilitate the creation of a machine learning (ML) model. This is achieved through the "ML_TABLES" class, which is responsible for uploading data from a CSV file sourced from Nasdaq.com into the "USStocks" table within the "StocksMLA" database. The class also sources additional information from the marketwatch.com website and uploads it into the same table. Once the necessary data has been gathered, a second table named "MLAStocks" is created within the same database. It is populted with the stocks that that meet specific criteria, such as price, gap, and change in price during a specified time span. Once identified, these stocks are uploaded to the table.

The ML model is then created based on the data within the "MLAStocks" table. The model's predict function is used to estimate the potential increase in stock prices. Overall, this file serves as a crucial component of the program, enabling it to create an ML model for predicting potential stock price increases.

Database Structure:

DATABASE StocksScreener, Tables:

US_STOCKS_TABLE	Stores parameters for all stocks trading on AMEX, NSDAQ, NYSE. Stores following data: symbolID, name, country, sector, industry, av_price, day_range, 52_week_range, market_cap, shares_outstanding, public_float, beta, short_interest, average_volume
GAPPERS_TABLE	Stores stocks with gaps. Stores following data: symbolID, stock_gap, stock_price, stock_volume, stock_float, stock_dailyRelativeVolume, stock_shortInterest
SCANNER	Stores stocks with specified parameters. Stores following data: symbolID, stock_gap, stock_price, stock_float, stock_agg_volume, stock_volume5min, stock_rv5min, stock_rsi, stock_priceVolatility, stock_extensionVWAP, stock_num_of_conseq_candles, stock_shortInterest

Table US_STOCKS_TABLE:

Field name	Data type	Description of the field	Validation
symbolID	String	This field holds ticker of the stock	Presence check
name	String	This field holds full name of the company	Presence check
country	String	This field holds country, where this company operates.	Presence check
sector	String	This field holds name of the sector where this company operates	Presence check
industry	String	This field holds name of the industry where this company operates	Presence check
av_price	Float	This field holds average price of the stock	Format check (correct to two decimal places)
day_range	String	This field holds range of the stock's price on intraday level.	Presence check
52_week_range	String	This field holds range of the stock's price on 52 week range level.	Presence check
market_cap	Integer	This field holds the market capitalization of the stock.	Range check: Between 0 and 999,999,999,999
shares_outstanding	Integer	This field holds stock's number of shares issued by the company.	Presence check

public_float	Integer	This field holds stock's number of shares traded	Presence check
beta	String	This field holds stock's beta value.	Presence check
short_interest	Float	This field holds stock's short interest value	Format check (correct to two decimal places)
average_volume	Integer	This field holds average total volume	Range check: Between 0 and 999,999,999,999

Table GAPPERS_TABLE:

Field name	Data type	Description of the field	Validation
symbolID	String	This field holds ticker of the stock	Presence check
stock_gap	Float	This field holds stock's gap	Format check (correct to two decimal places)
stock_price	Float	This field holds stock's price	Format check (correct to two decimal places)
stock_volume	Integer	This field holds stock's volume	Range check: Between 0 and 999,999,999,999
stock_float	Integer	This field holds stock's float or number of shares traded	Range check: Between 0 and 999,999,999,999
stock_dailyRelativeVolume	Float	This field holds stock's daily relative volume	Format check (correct to four decimal places)
stock_shortInterest	Float	This field holds stock's short interest	Format check (correct to two decimal places)

Table SCANNER:

Field name	Data type	Description of the field	Validation
symbolID	String	This field holds ticker of the stock	Presence check
stock_gap	Float	This field holds stock's gap	Format check (correct to two decimal places)
stock_price	Float	This field holds stock's price	Format check (correct to two decimal places)
stock_float	Integer	This field holds stock's float or number of shares traded	Range check: Between 0 and 999,999,999,999
stock_agg_volume	Integer	This field holds stock's total volume	Range check: Between 0 and 999,999,999,999
stock_volume5min	Integer	This field holds stock's volume for last 5 min candle	Range check: Between 0 and 999,999
stock_rv5min	Integer	This field holds stock's relative volume over 11 previous candles form 5 min candle	Format check (correct to four decimal places)
stock_rsi	Float	This field holds stock's Relative Strength Index value	Format check (correct to two decimal places)
stock_priceVolatility	Float	This field holds stock's price volatility on the inter day level	Format check (correct to two decimal places)
stock_extensionVWAP	Float	This field holds stock's extension of VWAP value	Format check (correct to two decimal places)
stock_num_of_conseq_candles	String	This field holds stock's candle colour and number of times it is repeated	Presence check

stock_shortInterest	Float	This field holds stock's short interest	Format check (correct to two decimal places)

DATABASE StocksMLA, Tables:

USStocks	Stores parameters for all stocks trading on AMEX, NSDAQ, NYSE. Stores following data: symbolID, stock_name, stock_price, stock_market_cap, country, stock_volume, stock_sectror, stock_industry, date_time, stock_gap, stock_number_of_shares_traded, stock_day_volume, stock_volatility, stock_day_change, stock_total_change
MLAStocks	Stores stocks to be used for ML model. Stores following data: symbolID, stock_price, date, stock_gap, stock_float, stock_drv, stock_vofi, stock_short_interest, beta, stock_market_cap, stock_sector, stock_day_change

USStocks Table:

Field name	Data type	Description of the field	Validation
symbolID	String	This field holds ticker of the stock	Presence check
stock_name	String	This field holds full name of the company	Presence check
stock_price	String	This field holds price of the stock	Format check (correct to two decimal places)
stock_market_cap	String	This field holds the market capitalization of the stock.	Range check: Between 0 and 999,999,999,999

country	String	This field holds country, where this company operates.	Presence check
stock_volume	String	This field holds total volume	Range check: Between 0 and 999,999,999,999
stock_sectror	String	This field holds name of the sector where this company operates	Presence check
stock_industry	String	This field holds name of the industry where this company operates	Presence check
date_time	String	This field holds the time, when the stocks data is collected	Presence check
stock_gap	Float	This field holds stock's gap	Format check (correct to two decimal places)
stock_number_of_shares_traded	Integer	This field holds stock's float	Range check: Between 0 and 999,999,999,999
stock_day_volume	Integer	This field holds stock's total volume intraday	Range check: Between 0 and 999,999,999,999
stock_volatility	Integer	This field holds stock's price volatility	Format check (correct to two decimal places)
stock_day_change	Float	This field holds stock's change in price during the day	Format check (correct to two decimal places)

stock_total_change	Float	This field holds stock's total change in price	Format check (correct to two decimal places)

MLAStocks Table:

Field name	Data type	Description of the field	Validation
symbolID	String	This field holds ticker of the stock	Presence check
stock_price	Float	This field holds price of the stock	Format check (correct to two decimal places)
date	String	This field holds the time, when the stocks data is collected	Presence check
stock_gap	Float	This field holds stock's gap	Format check (correct to two decimal places)
stock_float	Integer	This field holds stock's float or number of shares traded	Range check: Between 0 and 999,999,999,999
stock_drv	Float	This field holds stock's relative volume on daily level	Format check (correct to four decimal places)
stock_vofi	Float	Index that represents strength of the stock	Format check (correct to four decimal places)
stock_short_interest	Float	This field holds stock's short interest	Format check (correct to two decimal places)
beta	Float	The field holds beta value of the stock	Format check (correct to two decimal places)

stock_market_cap	Integer	The field holds the market capitalization of the stock	Range check: Between 0 and 999,999,999,999
stock_sector	String	The field holds the sector where this company operate.	Presence check
stock_day_change	Float	The field holds the value of the stock's change in price as percentage.	Format check (correct to two decimal places)

The Flowchart Diagram

The data flow diagram provides a comprehensive overview of all the processes occurring within the system. This includes the system's interaction with databases, as well as the user's interaction with the graphical user interface. It also provides an overview of the main programs responsible for running the scanners (Gapper scanner and Extended Satsy Scanner), as well as an outline of the algorithms used in the program.

Figure 1 (System Flowchart Diagram)

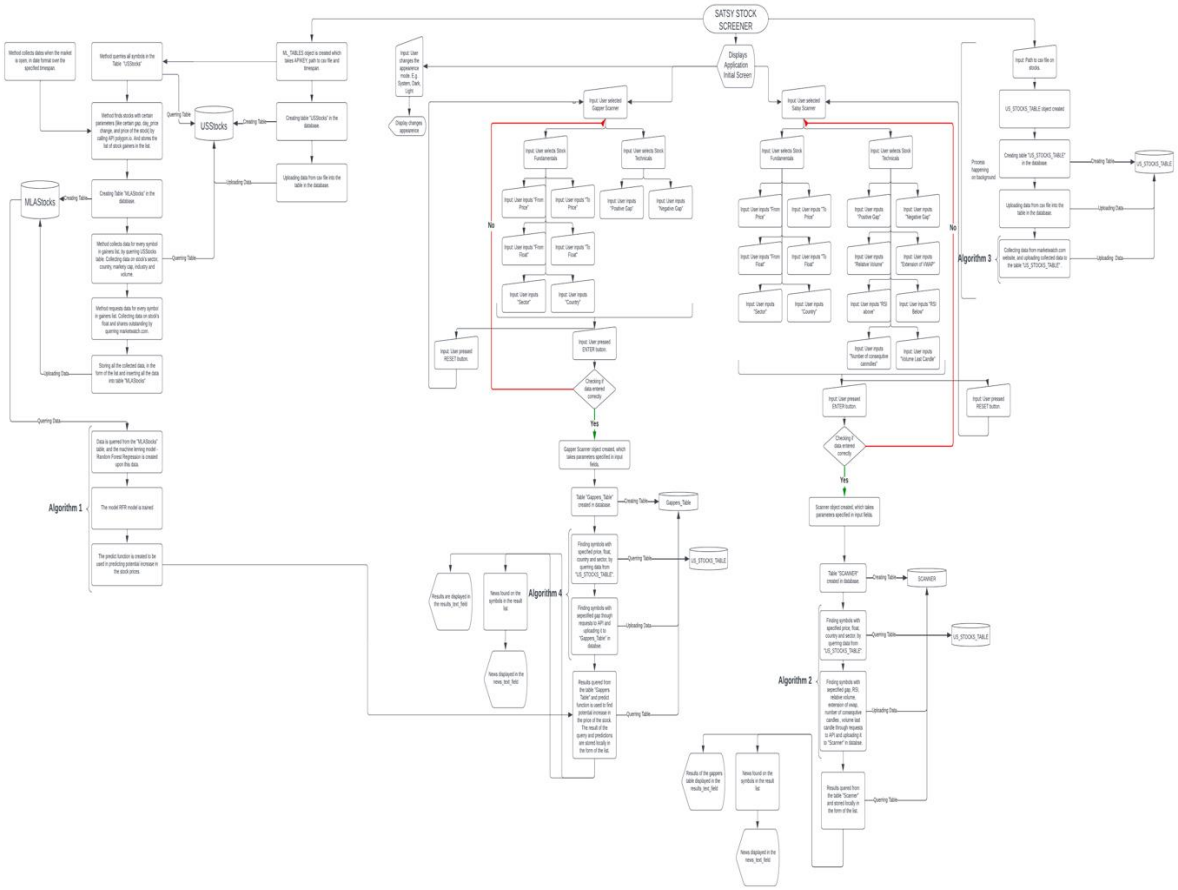

SATSY STOCK SCREENER
Displays Application Initial Screen
Input: User changes the appearence mode. E.g. System, Dark, Light
Display changes appearence
Input: User selected Gapper Scanner
Input: User selects Stock Fundamentals
Input: User selects Stock Technicals
Input: User inputs "From Price"
Input: User inputs "To Price"
Input: User inputs "Positive Gap"
Input: User inputs "Negative Gap"
Input: User inputs "From Float"
Input: User inputs "To Float"
Input: User inputs "Sector"
Input: User inputs "Country"
Input: User pressed RESET button.
Input: User pressed ENTER button.
Checking if data entered correctly
No
Yes
Gapper Scanner object created, which takes parameters specified in input fields.
Table "Gappers_Table" created in database.
Creating Table
Gappers_Table
Finding symbols with specified price, float, country and sector, by querying data from "US_STOCKS_TABLE".
Querying Table
US_STOCKS_TABLE
Finding symbols with specified gap through requests to API and uploading it to "Gappers_Table" in database.
Uploading Data
Results queried from the table "Gappers_Table" and predict function is used to find potential increase in the price of the stock. The result of the query and predictions are stored locally in the form of the list.
Querying Table
Results are displayed in the results_text_field
News found on the symbols in the result list.
News displayed in the news_text_field
Algorithm 4
Input: User selected Satsy Scanner
Input: User selects Stock Fundamentals
Input: User selects Stock Technicals
Input: User inputs "From Price"
Input: User inputs "To Price"
Input: User inputs "Positive Gap"
Input: User inputs "Negative Gap"
Input: User inputs "From Float"
Input: User inputs "To Float"
Input: User inputs "Relative Volume"
Input: User inputs "Extension of VWAP"
Input: User inputs "Sector"
Input: User inputs "Country"
Input: User inputs "RSI above"
Input: User inputs "RSI Below"
Input: User inputs "Number of consequtive candles"
Input: User inputs "Volume Last Candle"
Input: User pressed ENTER button.
Input: User pressed RESET button.
Checking if data entered correctly
No
Yes
Scanner object created, which takes parameters specified in input fields.
Table "SCANNER" created in database.
Creating Table
SCANNER
Finding symbols with specified price, float, country and sector, by querying data from "US_STOCKS_TABLE".
Querying Table
US_STOCKS_TABLE
Finding symbols with specified gap, RSI, relative volume, extension of vwap, number of consequtive candles, volume last candle through requests to API and uploading it to "Scanner" in database.
Uploading Data
Results queried from the table "Scanner" and stored locally in the form of the list.
Querying Table
Results of the gappers table displayed in the results_text_field
News found on the symbols in the result list.
News displayed in the news_text_field
Algorithm 2
Input: Path to csv file on stocks.
US_STOCKS_TABLE object created.
Process happening on background
Creating table "US_STOCKS_TABLE" in the database.
Creating Table
US_STOCKS_TABLE
Uploading data from csv file into the table in the database.
Uploading Data
Collecting data from marketwatch.com website, and uploading collected data to the table "US_STOCKS_TABLE".
Uploading Data
Algorithm 3
Method collects data when the market is open, in date format over the specified timespan.
Method queries all symbols in the Table "USStocks"
Querying Table
USStocks
Creating Table
NS_TABLE2 object is created which takes APIKEY path to csv file and timespan.
Creating table "USStocks" in the database.
Uploading Data
Uploading data from csv file into the table in the database.
Method finds stocks with certain parameters (like certain gap, day, price change, and price of the stock) by calling API polygon.io. And stores the list of stock gainers in the list.
Creating Table "MLAStocks" in the database.
MLAStocks
Creating Table
Querying Table
Method collects data for every symbol in gainers list, by querying USStocks table. Collecting data on stock's sector, country, marketcap, industry and volume.
Method requests data for every symbol in gainers list. Collecting data on stock's float and shares outstanding by querying marketwatch.com.
Storing all the collected data, in the form of the list and inserting all the data into table "MLAStocks"
Uploading Data
Querying Data
Data is queried from the "MLAStocks" table, and the machine lerning model - Random Forest Regression is created upon this data.
The model RFR model is trained.
The predict function is created to be used in predicting potential increase in the stock prices.
Algorithm 1

Figure 2 (Left side of the flowchart diagram)

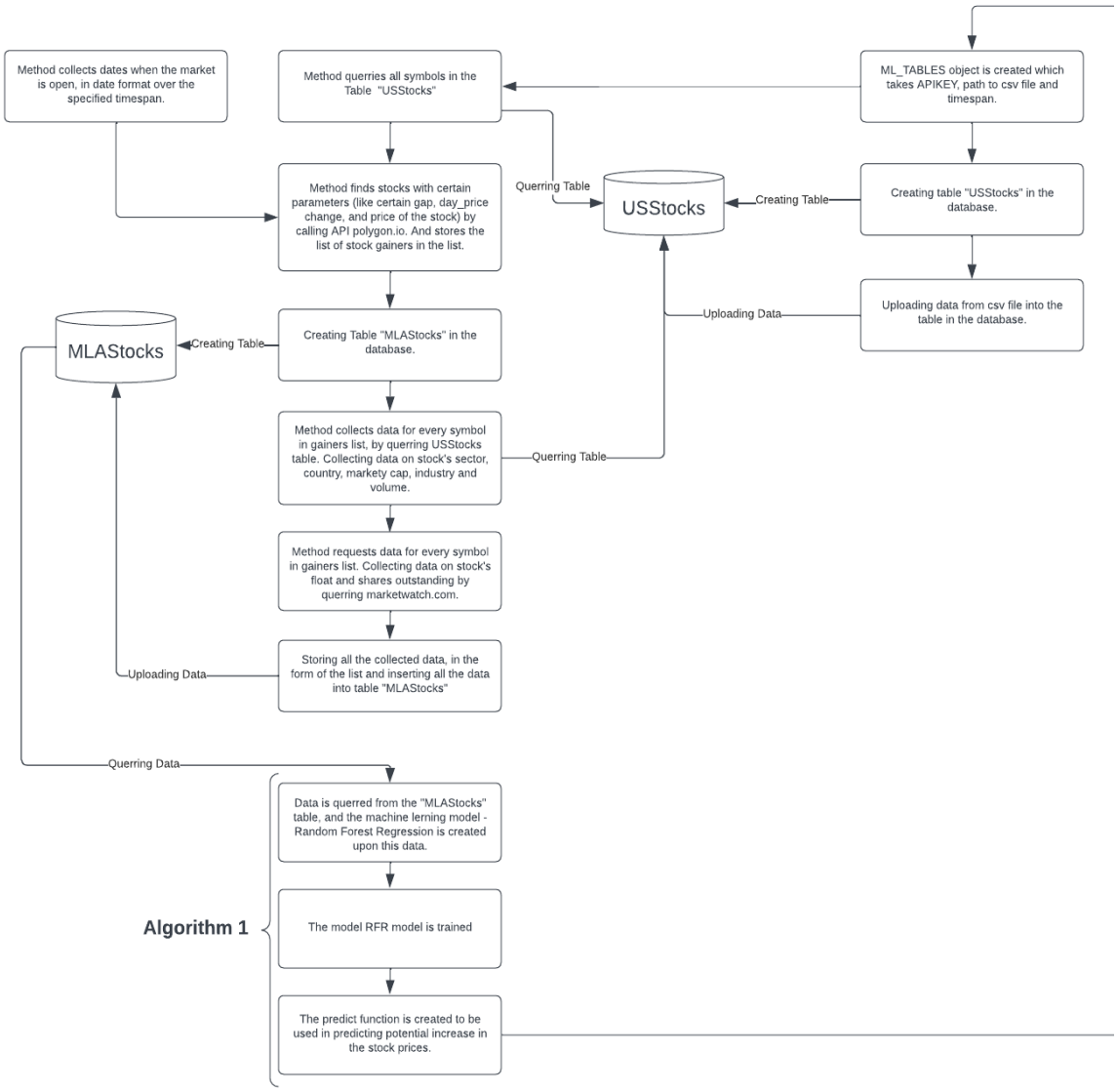


Figure 3 (Middle side of the flowchart diagram)

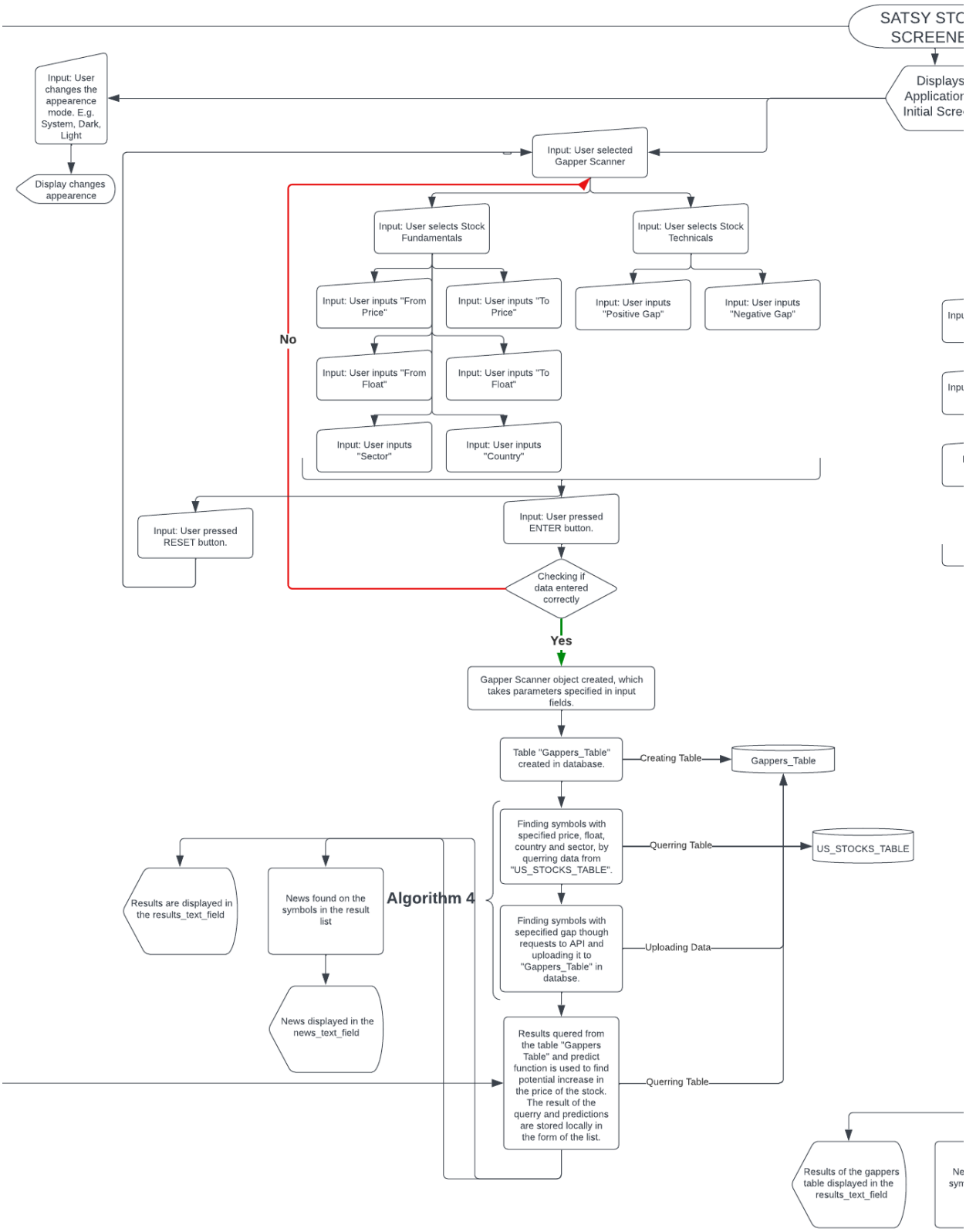
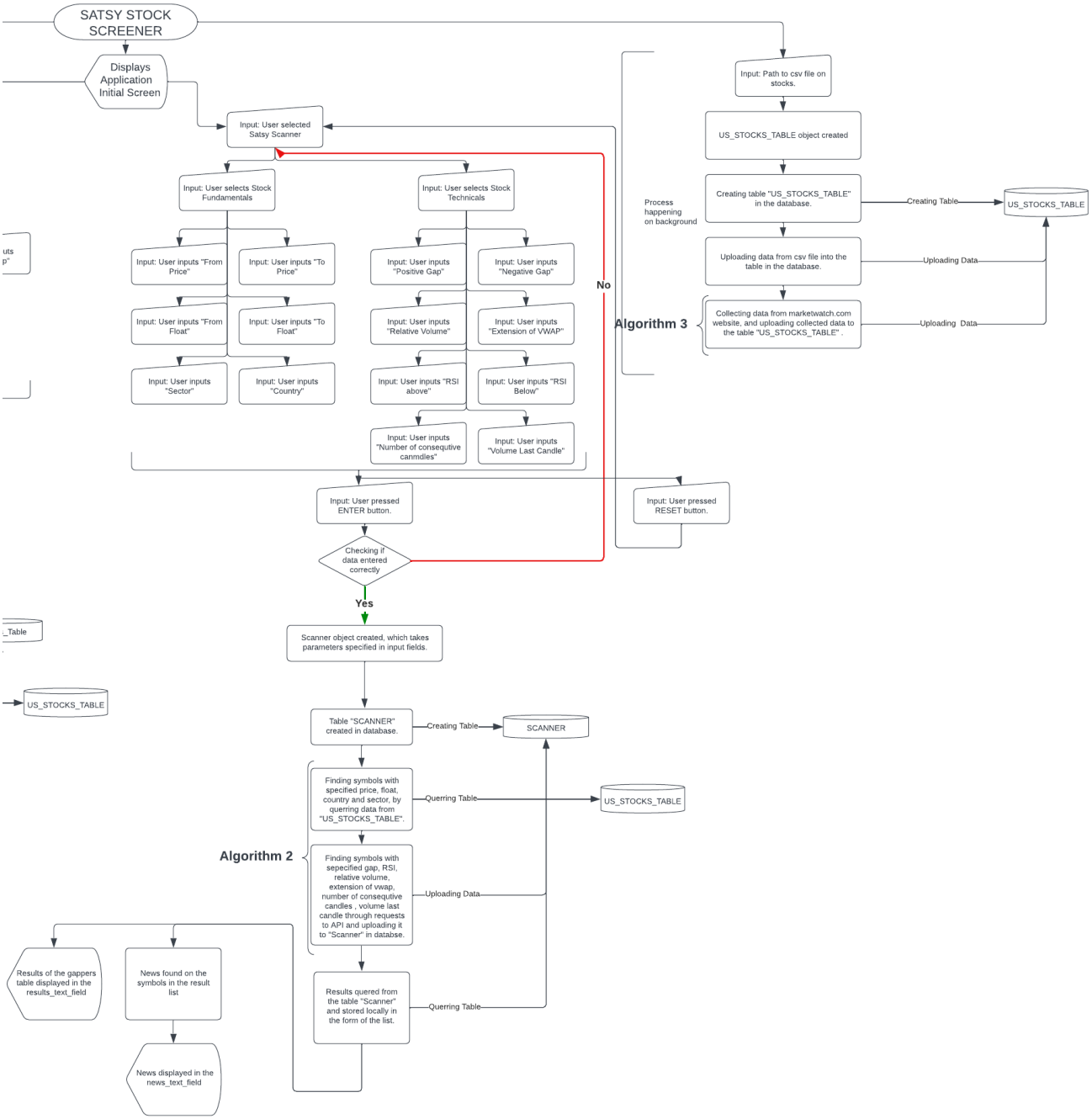
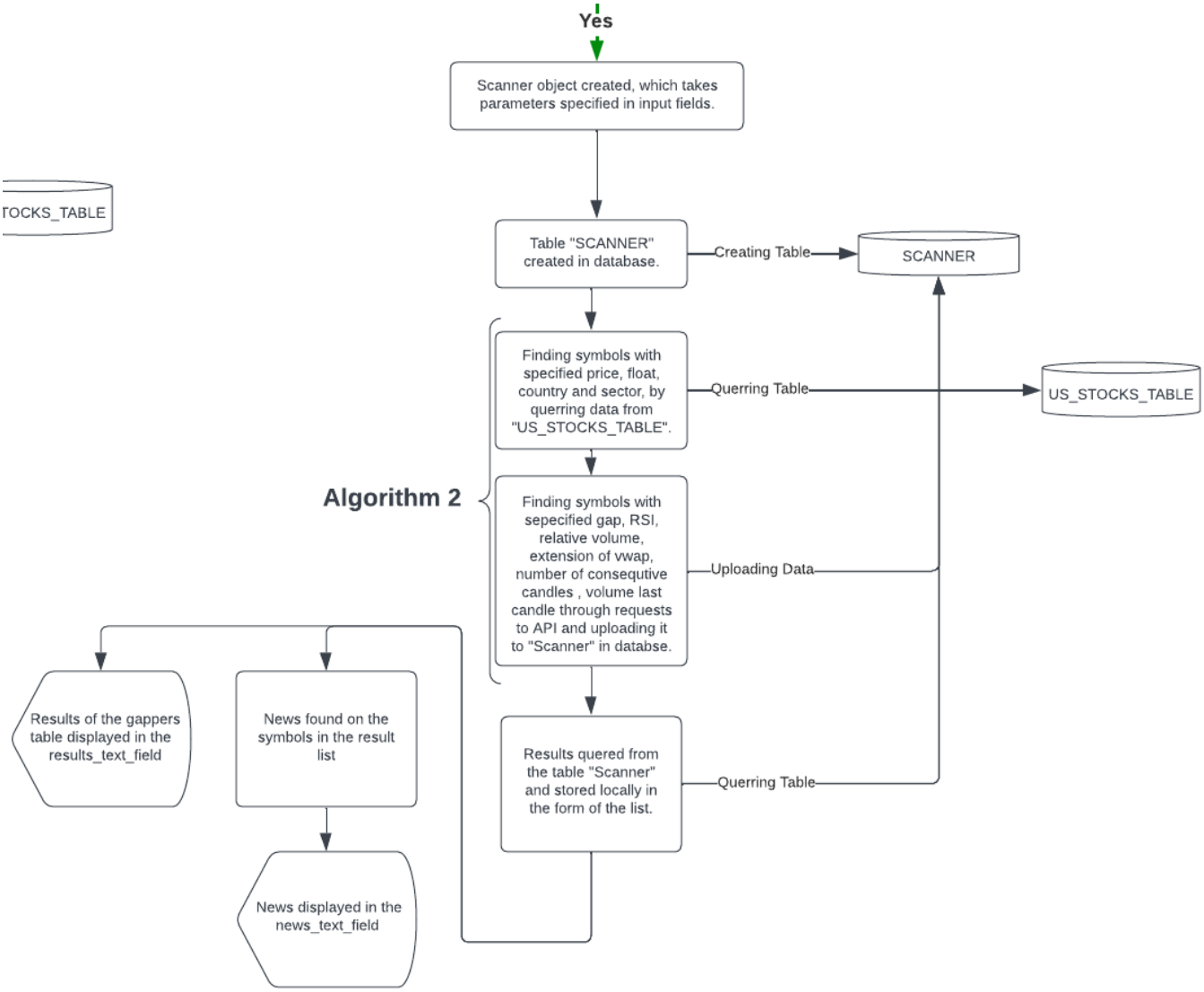


Figure 4 (right side of the flowchart diagram)



The SATSY Scanner (Extended) Flowchart Diagram



Pseudocode:

Pseudocode for creating a "Scanner" table:

```
DEFINE function __creating_table
   #Creates Scanner table in database StocksScreener

   EXECUTE SQL query "DROP TABLE SCANNER" using self.__db.mycursor
   EXECUTE SQL query "CREATE TABLE SCANNER(
      symbolID VARCHAR(255) PRIMARY KEY,
      stock_gap FLOAT(3),
      stock_price FLOAT(2),
```

```
    stock_float BIGINT(255),
    stock_agg_volume BIGINT(255),
    stock_volume5min BIGINT(255),
    stock_rv5min BIGINT(255),
    stock_rsi FLOAT(2),
    stock_priceVolatility FLOAT(2),
    stock_extensionVWAP FLOAT(2),
    stock_num_of_conseq_candles VARCHAR(5),
    stock_shortInterest FLOAT(4))" using self.__db.mycursor
  COMMIT changes using self.__db
  PRINT "CREATION OF TABLE SCANNERS FINISHED"
```

Pseudocode for querying a "US_STOCKS_TABLE" table:

```
DEFINE function __finding_symbols_with_p_f_s_c
  #Finds stocks with specified parameters float, avPrice, country, sector from US_STOCKS_TABLE and returning dict

  SET publicFloat to self.__public_float
  SET avPrice to self.__avPrice
  SET sector to self.__sector
  SET country to self.__country

  SET sectorQuerry to an empty string
  IF sector.upper() is not equal to "ALL" THEN
    SET sectorQuerry to "AND sector='{sector.title()}'"

  SET countryQuerry to an empty string
  IF country.upper() is not equal to "ALL" THEN
    SET countryQuerry to "AND country='{country.title()}'"

  EXECUTE SQL query "SELECT symbolID, av_price, public_float, short_interest
              FROM StocksScreener.US_STOCKS_TABLE
              WHERE public_float BETWEEN {publicFloat['minValue']}
              AND {publicFloat['maxValue']} {sectorQuerry} {countryQuerry}
              AND av_price BETWEEN {avPrice['minValue']} AND {avPrice['maxValue']}" using self.__db.mycursor
  SET results to the fetched data from the query

  RETURN a dictionary with a key 'r' and value as the fetched results
```

Pseudocode for uploading data into a "SCANNER" table:

```
DEFINE function __uploading_selected_stocks
  SET selected_stocks to the result of calling function __finding_symbols_with_sp

  SET sqlFormula to """INSERT INTO SCANNER(symbolID, stock_gap, stock_price,
      stock_float, stock_agg_volume, stock_volume5min, stock_rv5min,
      stock_rsi, stock_priceVolatility, stock_extensionVWAP,
      stock_num_of_conseq_candles, stock_shortInterest)
      VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""

  EXECUTE SQL query sqlFormula with selected_stocks as input using self.__db.mycursor

  COMMIT changes using self.__db

  PRINT "UPLOADING AND INSERTION INTO SCANNERS FINISHED"
```

```
  RETURN selected_stocks
```

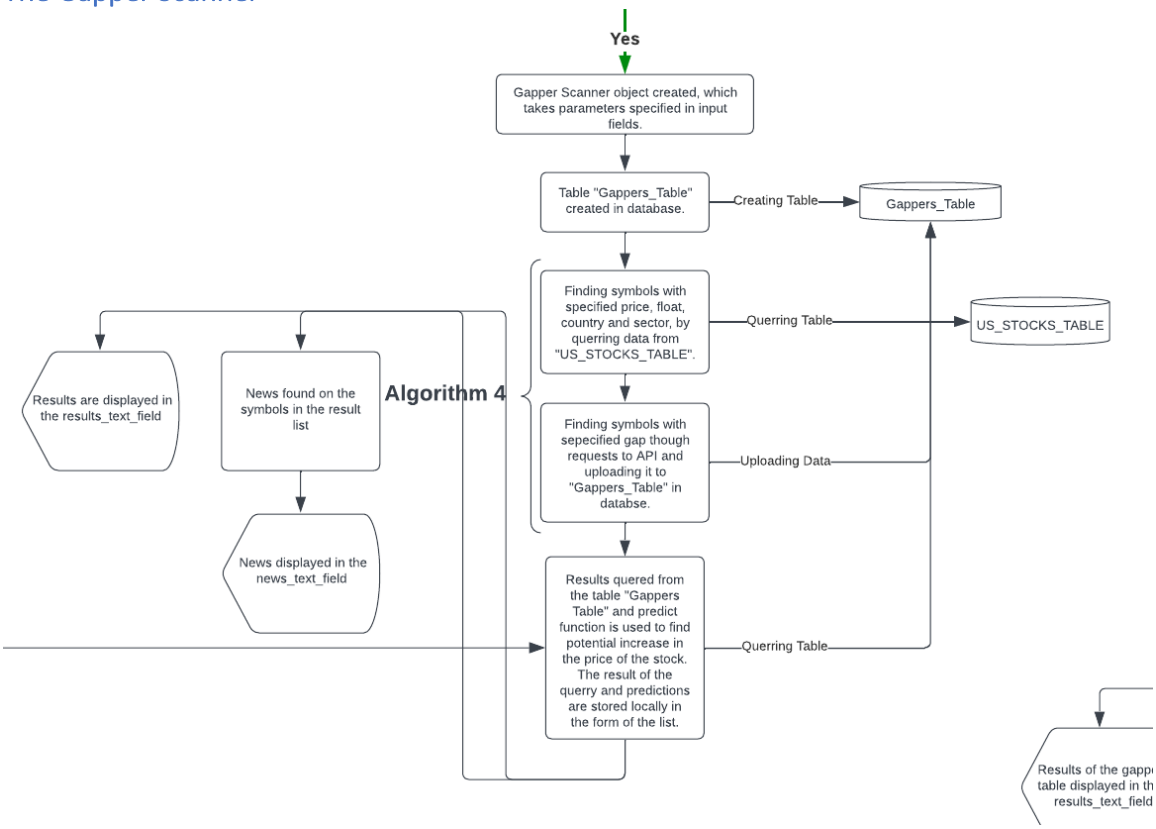
Pseudocode for querying "SCANNER" table:

```
DEFINE function SATSY_SCANNER_RESULTS
# method responsible for finding news and returning results in the form of list

  EXECUTE SQL query "SELECT symbolID, stock_gap, stock_price,
      stock_float, stock_agg_volume, stock_volume5min, stock_rv5min,
      stock_rsi, stock_priceVolatility, stock_extensionVWAP,
      stock_num_of_conseq_candles, stock_shortInterest
      FROM StocksScreener.SCANNER
      ORDER BY stock_rsi ASC, stock_gap ASC;" using self.__db.mycursor
  SET results to the fetched data from the query

  RETURN results
```

Pseudocode of Algorithm 2:

```
FUNCTION finding_stocks_with_specified_req(selected_symbols_array)

    IF ((indic_values['price'] >= self.__avPrice['minValue']) AND
(indic_values['price'] <= self.__avPrice['maxValue'])) THEN
        IF ((pos_min <= gap AND gap <= pos_max) OR (gap <= neg_min AND gap >=
neg_max)) AND (rsi >= self.__rsi['above'] OR rsi <= self.__rsi['below']) AND (rv
>= self.__rv) THEN
            IF (indic_values['n_conseq_candles'][1] >= self.__n_conseq_candles)
AND (indic_values['agg_volume'] >= self.__agg_volume) AND indic_values['volume']
>= self.__volume THEN
                tuple_array = (symbol, indic_values['gap'],
indic_values['price'], public_float, indic_values['agg_volume'],
indic_values['volume'], indic_values['rv'], indic_values['rsi'],
indic_values['price_volatility'], indic_values['e_vwap'],
f"{indic_values['n_conseq_candles'][0]}, {indic_values['n_conseq_candles'][1]}",
short_interest)
                array_of_selected_data.append(tuple_array)
            END IF
        END IF
    END IF
END FUNCTION
```

The Gapper Scanner



Pseudocode on SQL tables:

Pseudocode for creating "GAPPERS_TABLE" table:

```
DEFINE function __creating_table
#Creates GAPPERS_TABLE table in database StocksScreener
   EXECUTE SQL query "DROP TABLE GAPPERS_TABLE" using self.__db.mycursor
   EXECUTE SQL query """CREATE TABLE GAPPERS_TABLE(
   symbolID VARCHAR(255) PRIMARY KEY,
   stock_gap FLOAT(3),
   stock_price FLOAT(2),
   stock_volume BIGINT(255),
   stock_float BIGINT(255),
   stock_dailyRelativeVolume FLOAT(4),
   stock_shortInterest FLOAT(4))""" using self.__db.mycursor
   COMMIT changes using self.__db
```

Pseudocode for querying the "US_STOCKS_TABLE" table:

```
DEFINE FUNCTION __finding_symbols_with_p_f_s_c():

   SET sectorQuerry TO ""
   IF sector.upper() != "ALL":
      SET sectorQuerry TO "AND sector='" + sector.title() + "'"
```

```
  SET countryQuerry TO ""
  IF country.upper() != "ALL":
    SET countryQuerry TO "AND country='" + country.title() + "'"

  EXECUTE SQL QUERY "SELECT symbolID, av_price, public_float, short_interest FROM StocksScreener.US_STOCKS_TABLE
              WHERE public_float BETWEEN " + publicFloat['minValue'] + " AND " + publicFloat['maxValue'] + "
              " + sectorQuerry + "
              " + countryQuerry + "
              AND av_price BETWEEN " + avPrice['minValue'] + " AND " + avPrice['maxValue'] + ";"
  FETCH ALL RESULTS INTO results
```

Pseudocode for uploading of the data into the "GAPPERS_TABLE" table:

```
FUNCTION __uploading_selected_stocks()
  SET gappers_array TO __finding_symbols_with_gap()
  SET sqlFormula TO "INSERT INTO GAPPERS_TABLE(symbolID, stock_gap, stock_price, stock_volume, stock_float,
        stock_dailyRelativeVolume, stock_shortInterest)
        VALUES (%s, %s, %s, %s, %s, %s, %s)"
  CALL __db.mycursor.executemany(sqlFormula, gappers_array)
  CALL __db.commit()
  PRINT "UPLOADING AND INSERTION INTO GAPPERS_TABLE FINISHED"
  RETURN gappers_array
END FUNCTION
```

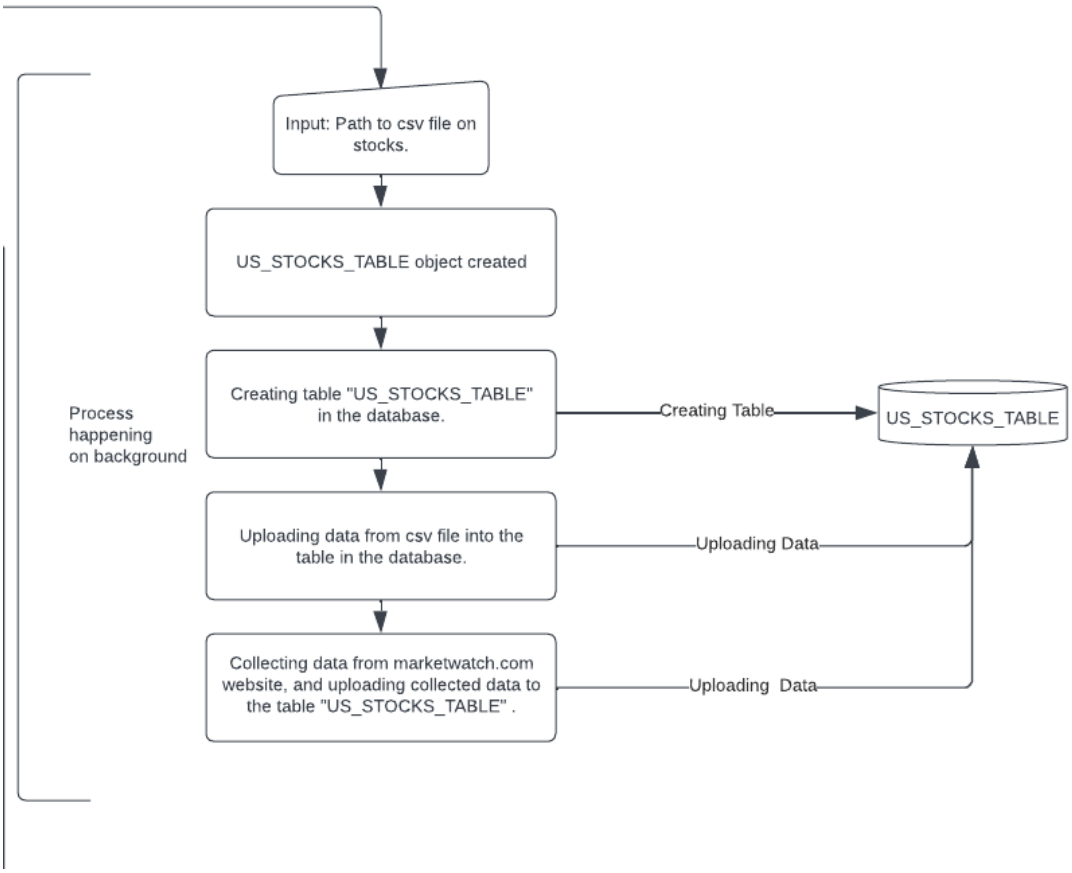
Pseudocode for querying of "GAPPERS_TABLE" table:

```
FUNCTION GAPPER_SCANNER_RESULTS()
  CALL __db.mycursor.execute("SELECT symbolID, stock_gap, stock_price,
        stock_volume, stock_float, stock_shortInterest
        FROM StocksScreener.GAPPERS_TABLE ORDER BY stock_gap ASC;")
  SET results TO __db.mycursor.fetchall()
END FUNCTION
```

Pseudocode for : Algorithm 4

```
FUNCTION finding_gappers(symbolsData)
        IF (pos_min <= gap AND gap <= pos_max) OR (gap <= neg_min AND gap >=
neg_max) THEN
        tuple = (symbolsData[0], gpv_data['gap'], gpv_data['price'],
gpv_data['volumeDay'], symbolsData[2], 0, symbolsData[3])
        gappersArray.append(tuple)
    END IF
END FUNCTION
```

The US Stock Table



Pseudocode on SQL Tables:

Pseudocode for creating "US_STOCKS_TABLE" table:

```
FUNCTION __creating_table()
   CALL __db.mycursor.execute("DROP TABLE US_STOCKS_TABLE")
   CALL __db.mycursor.execute("""CREATE TABLE US_STOCKS_TABLE(
   symbolID VARCHAR(255),
   name VARCHAR(255),
   country VARCHAR(255),
   sector VARCHAR(255),
   industry VARCHAR(255),
   av_price FLOAT(3),
   day_range TINYTEXT,
   52_week_range TINYTEXT,
   market_cap BIGINT(255),
   shares_outstanding BIGINT(255),
   public_float BIGINT(255),
   beta VARCHAR(255),
   short_interest FLOAT(3),
   average_volume BIGINT(255))""")
```

```
    CALL __db.commit()
END FUNCTION
```

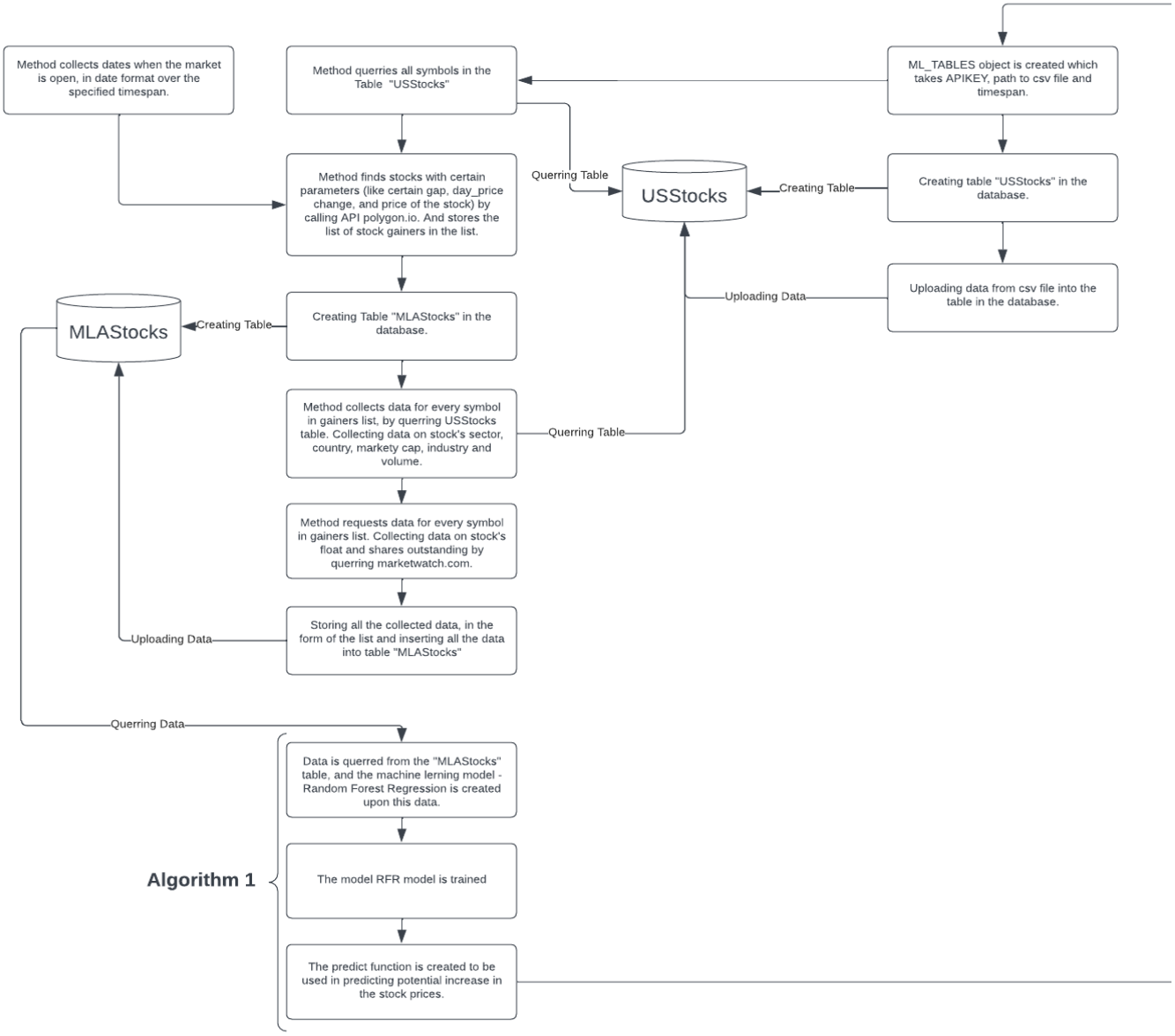
Pseudocode for uploading data into the "US_STOCKS_TABLE" table:

```
FUNCTION __uploading_data_from_csv(number_of_rows: INTEGER)
    SET data TO READ_CSV(FILE_PATH=self.__file_path, nrows=number_of_rows)
    SET df TO CREATE_DATA_FRAME(data)
    CALL FILL_NA(df, "Nan")
    Copy code
    FOR row IN ITER_TUPLES(df)
        SET formatedRow TO (row.Symbol, row.Name, row.Country, row.Sector, row.Industry, 0, "Nan",
"Nan", 0, 0, 0, 0, 0, 0)
        SET sqlFormula TO "INSERT INTO US_STOCKS_TABLE(symbolID, name, country, sector, industry,
av_price, day_range,
        52_week_range, market_cap, shares_outstanding, public_float, beta, short_interest,
average_volume)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"
        CALL __db.mycursor.execute(sqlFormula, formatedRow)
        CALL __db.commit()
    PRINT "UPLOADING FROM CSV TO US_STOCKS_TABLE FINISHED"
```

Pseudocode for uploading data to "US_STOCKS_TABLE" table:

```
FUNCTION update_data(av_price, day_range, fiftytwo_week_range, market_cap,
shares_outstanding,
            public_float, short_interest, average_volume, beta, symbol, index):
    SET sqlFormula TO "UPDATE StocksScreener.US_STOCKS_TABLE
        SET av_price = {av_price},
        day_range = {day_range},
        52_week_range = {fiftytwo_week_range},
        market_cap = {market_cap},
        shares_outstanding = {shares_outstanding},
        public_float = {public_float},
        short_interest = {short_interest},
        average_volume = {average_volume},
        beta = {beta}
        WHERE symbolID = '{symbol}';"
    CALL self.__db.mycursor.execute(sqlFormula)
    CALL self.__db.commit()
    PRINT f"{index}. {symbol}'s info added to US_STOCKS_TABLE"
END FUNCTION
```

The ML Algorithm



Pseudocode on SQL tables:

Pseudocode for creating a "USStocks" table:

```
FUNCTION creating_table_USStocks()
   CALL __db.mycursor.execute('DROP TABLE USStocks')
   CALL __db.mycursor.execute("""CREATE TABLE USStocks(
                    symbolID VARCHAR(255),
                    stock_name VARCHAR(255),
                    stock_price VARCHAR(255),
```

```
                    stock_market_cap VARCHAR(255),
                    country VARCHAR(255),
                    stock_volume VARCHAR(255),
                    stock_sector VARCHAR(255),
                    stock_industry VARCHAR(255),
                    date_time VARCHAR(255),
                    stock_gap FLOAT(5),
                    stock_number_of_shares_traded BIGINT(255),
                    stock_day_volume BIGINT(255),
                    stock_volatility BIGINT(255),
                    stock_day_change FLOAT(5),
                    stock_total_change FLOAT(5))""")
  CALL __db.commit()
  PRINT 'Table USStocks created'
END FUNCTION
```

Pseudocode for uploading data into "USStocks" table:

```
FUNCTION uploading_data_to_USStocks(numberOfRows=8000):
  SET file_path TO r"" + f"{self.__NASDQ_STOCKS_CSV}"
  SET data TO pd.read_csv(file_path)
  SET df TO pd.DataFrame(data)
  CALL df.fillna("Nan", inplace=True)
  SET csv_data TO []
  FOR EACH row IN df.itertuples():
    SET formated_row TO (row[1], row[2], row[3], row[6], row[7], row[9], row[10], row[11], 0,0,0,0,0,0,0)
    CALL csv_data.append(formated_row)
  SET sqlFormula TO """INSERT INTO USStocks(symbolID, stock_name,
                stock_price, stock_market_cap, country,
                stock_volume, stock_sector, stock_industry,
                date_time, stock_gap,
                stock_number_of_shares_traded,
                stock_day_volume, stock_volatility,
                stock_day_change, stock_total_change)
                VALUES (%s, %s, %s, %s, %s, %s, %s,
                %s, %s, %s, %s, %s, %s, %s, %s)"""
  TRY:
    CALL self.__db.mycursor.executemany(operation=sqlFormula, seq_params=csv_data)
    CALL self.__db.commit()
    PRINT "SQL injection from csv finished"
  EXCEPT:
    RAISE Exception
```

Pseudocode for querying data from "USStocks" table:

```
FUNCTION __symbols_array():
  SET symbols_array TO []
  SET sqlOrderFormula TO "SELECT symbolID FROM USStocks"
  CALL self.__db.mycursor.execute(sqlOrderFormula)
  SET myResults TO self.__db.mycursor.fetchall()
  FOR row IN myResults:
```

```
        symbols_array.append(row[0])
    RETURN symbols_array
```

Pseudocode for creating “MLAStocks” table (and dropping the previous one)

```
FUNCTION creating_table_MLAStocks():
    CALL __db.mycursor.execute('DROP TABLE MLAStocks')
    SET sqlFormula TO """CREATE TABLE MLAStocks(
        symbolID VARCHAR(255),
        stock_price FLOAT(5),
        date VARCHAR(255),
        stock_gap FLOAT(5),
        stock_float BIGINT(255),
        stock_drv FLOAT(5),
        stock_vofi FLOAT(5),
        stock_short_interest FLOAT(5),
        beta FLOAT(5),
        stock_market_cap BIGINT(255),
        stock_sector VARCHAR(255),
        stock_day_change FLOAT(5))"""
    CALL __db.mycursor.execute(sqlFormula)
    PRINT 'Table MLAStocks created'
    CALL __db.commit()
    END FUNCTION
```

Pseudocode for querying data from “USStocks” table:

```
FUNCTION finding_marketCap_country_volume_sector_indusrty(forSymbol: STRING):
    CALL __db.mycursor.execute(f"""SELECT symbolID, stock_market_cap, country,
    stock_volume, stock_sector, stock_industry
    FROM StocksMLA.USStocks
    WHERE symbolID='{forSymbol}'""")
    SET myResult TO __db.mycursor.fetchall()
    RETURN myResult
```

Pseudocode for uploading data to “MLAStocks”

```
FUNCTION INSERTING_INTO_MLASTOCKS
    SET gainers_data TO FINDING_INFO_ON_STOCKS()
    SET formula TO """INSERT INTO MLAStocks(symbolID, stock_price, date, stock_gap,
        stock_float, stock_drv, stock_vofi, stock_short_interest,
        beta, stock_market_cap, stock_sector, stock_day_change)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""
    EXECUTE SQL COMMAND "EXECUTEMANY" ON self.__db.mycursor WITH PARAMETERS formula, gainers_data
    COMMIT CHANGES TO self.__db
    PRINT "INSERTION INTO MLASTOCKS FINISHED"
END FUNCTION
```

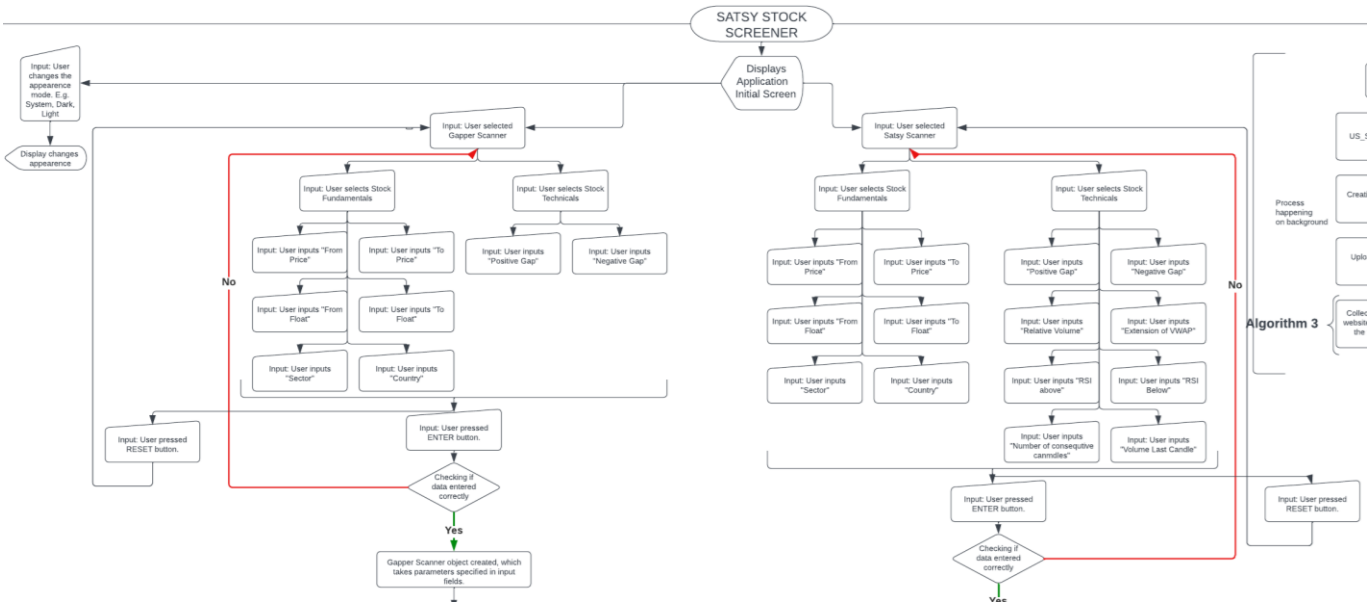
Pseudocode for querying data from “MLAStocks” table:

```
DEFINE function fetching_gainers_data()
   SET db = Database object with database_name = 'StocksMLA'
   EXECUTE the SQL query "SELECT stock_price, stock_gap, stock_float,
                  stock_vofi, stock_short_interest, beta,
                  stock_day_change FROM MLAStocks" using db's mycursor
   SET myResults = db's mycursor's fetchall() method
   CREATE a new DataFrame named df using myResults with column names ['stock_price',
'stock_gap', 'stock_float', 'stock_vofi', 'stock_short_interest', 'beta', 'stock_day_change']
   RETURN df
END function
```

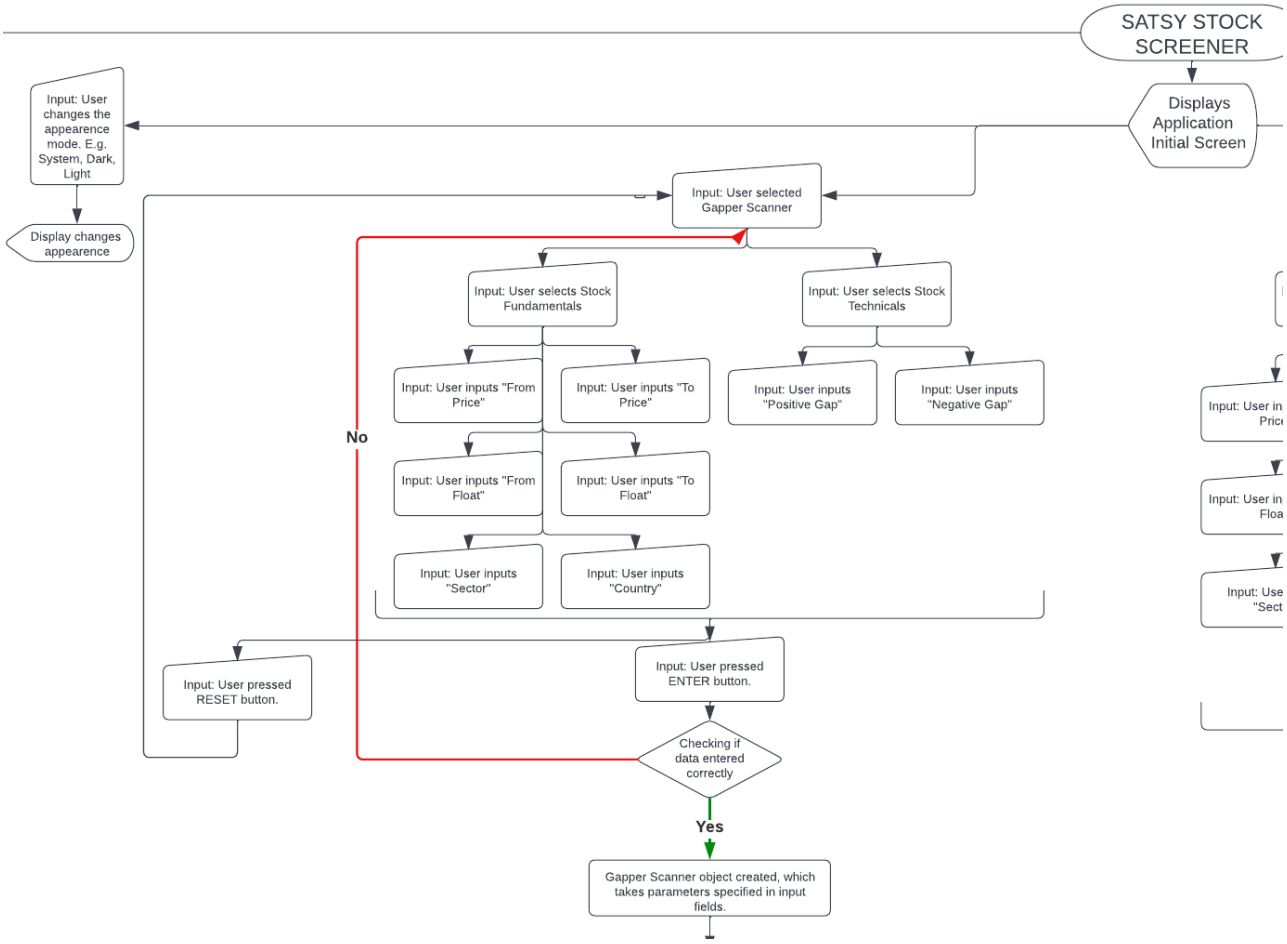
Algorithm 1 :

Algorithm 1 is responsible for creating Random Forest Regression model from dataset of stock gainers. The Random Forest Regression algorithm works by repeatedly selecting random subsets of the training data and building decision trees on them. These decision trees are combined to make predictions for new data points by applying each decision tree to the data and obtaining a prediction, and then combining these predictions to produce a final prediction. The algorithm uses bootstrapping to ensure that each decision tree has a different view of the data, reducing the risk of overfitting to the training data. To average the results for regression tasks, the predictions from all decision trees are combined.

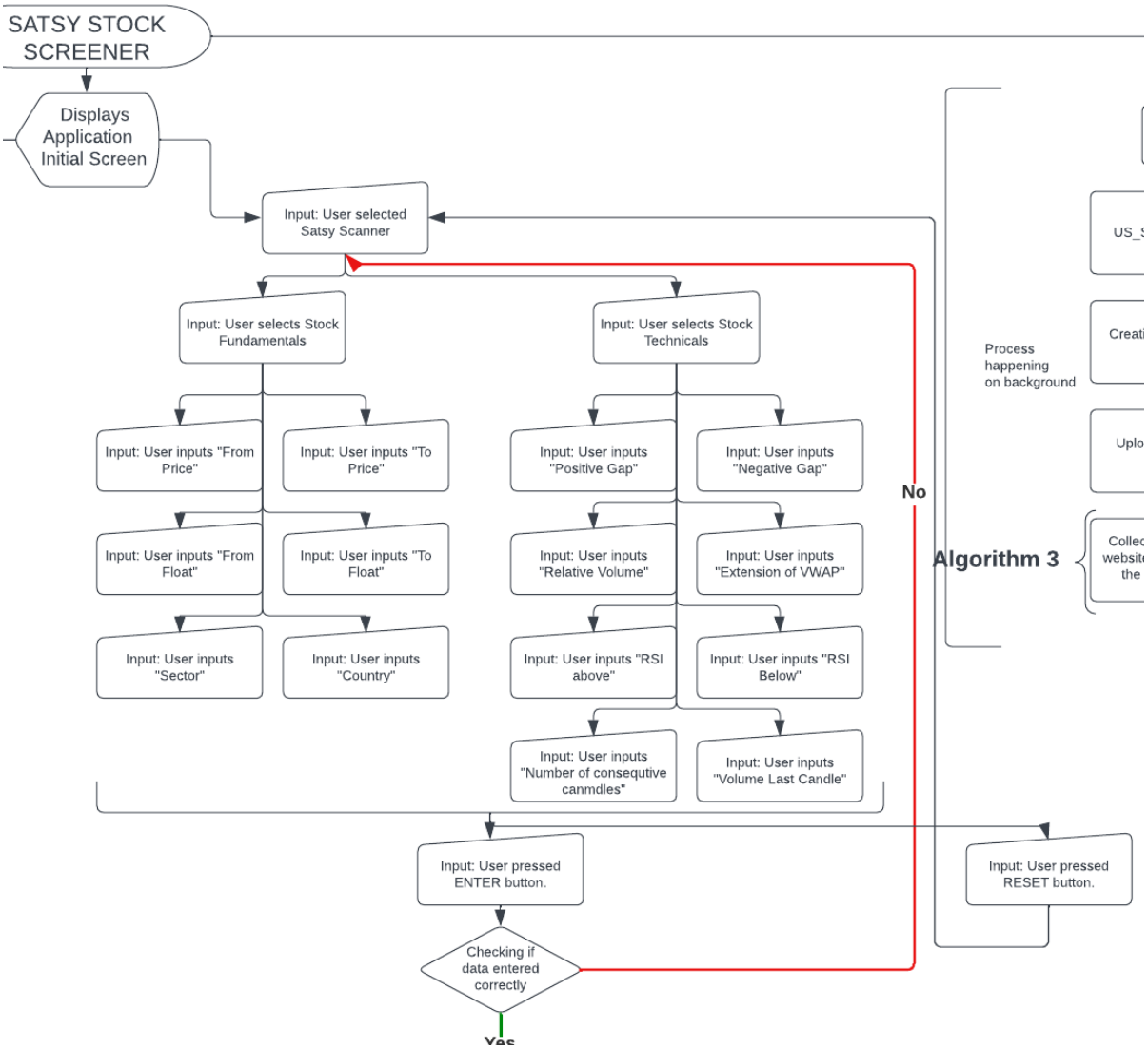
Graphical User Interface. Flowchart diagram



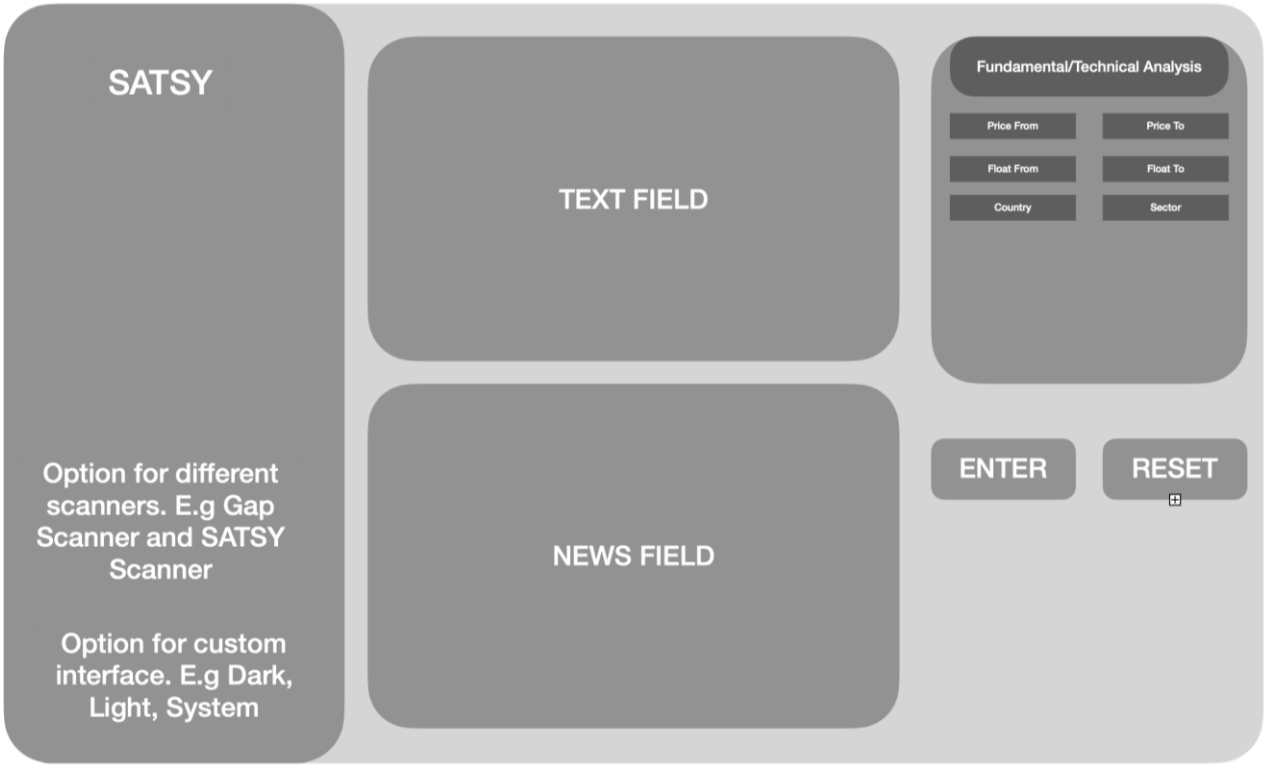
Flowchart diagram. Gapper Scanner GUI



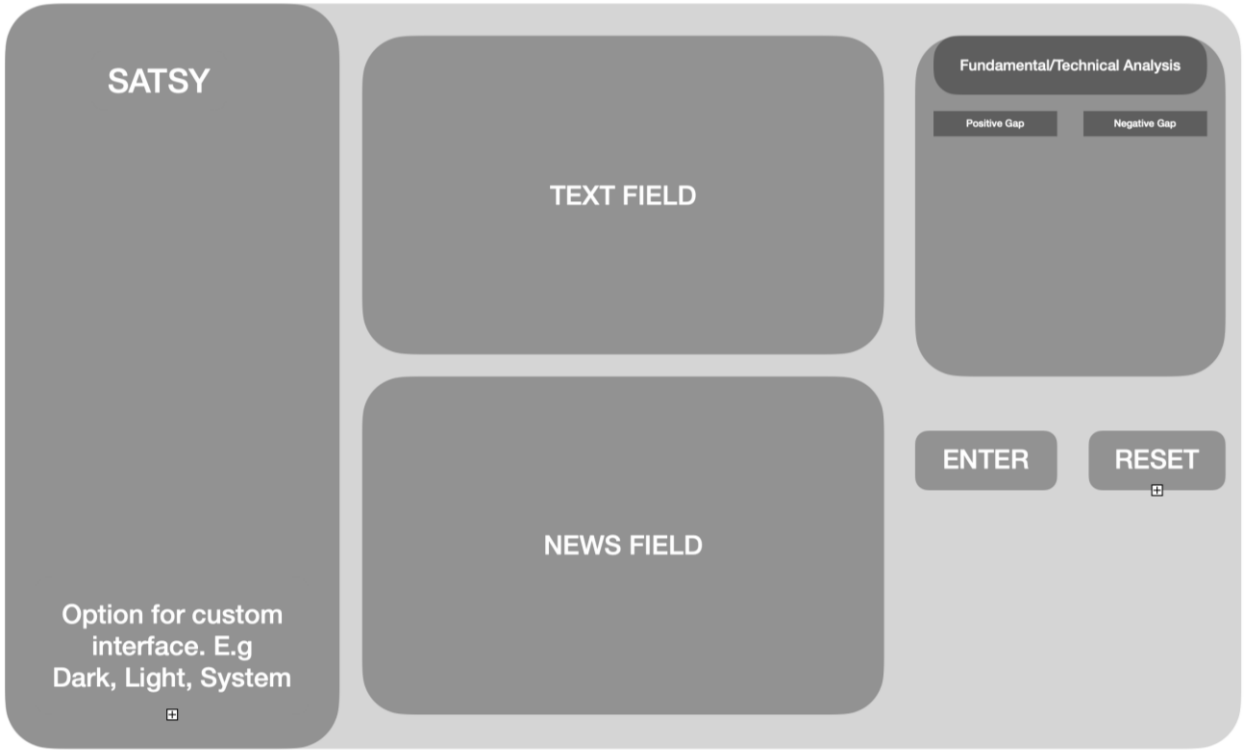
Flowchart diagram. SATSY Scanner GUI



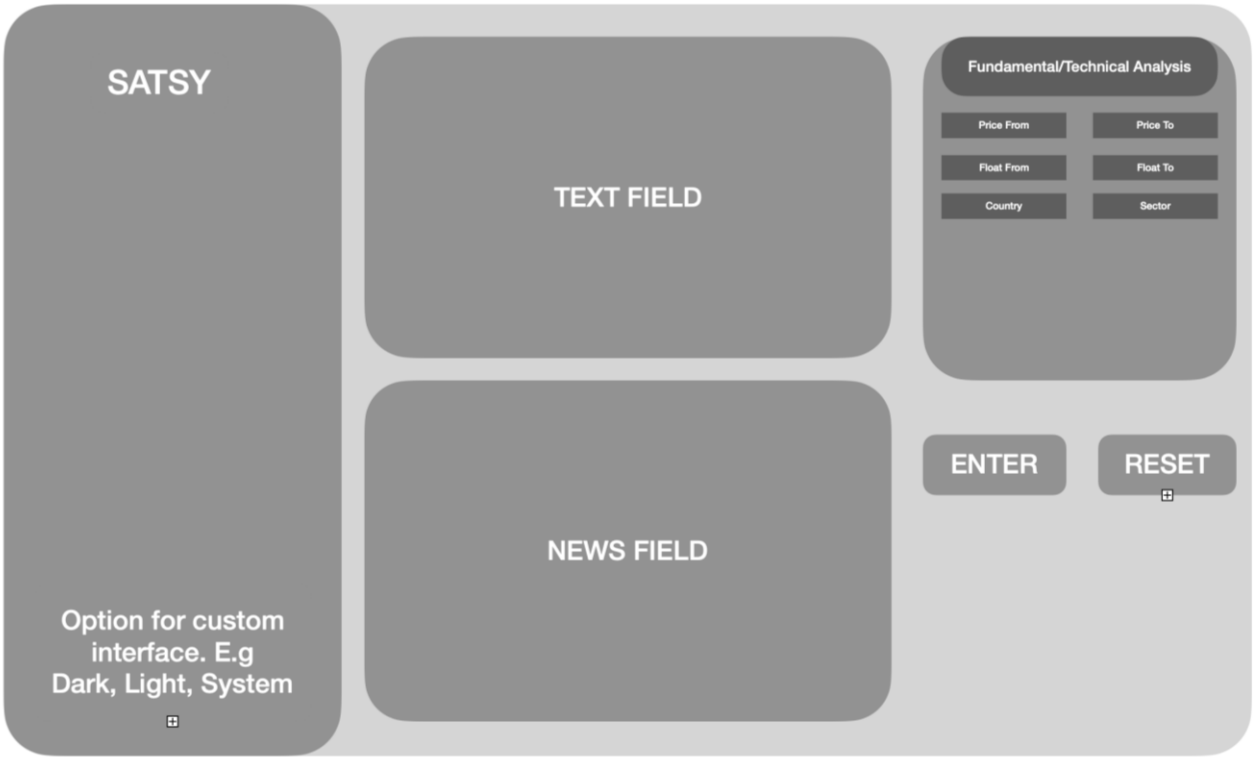
GUI Screen#1 GAP Screener, Fundamental input window



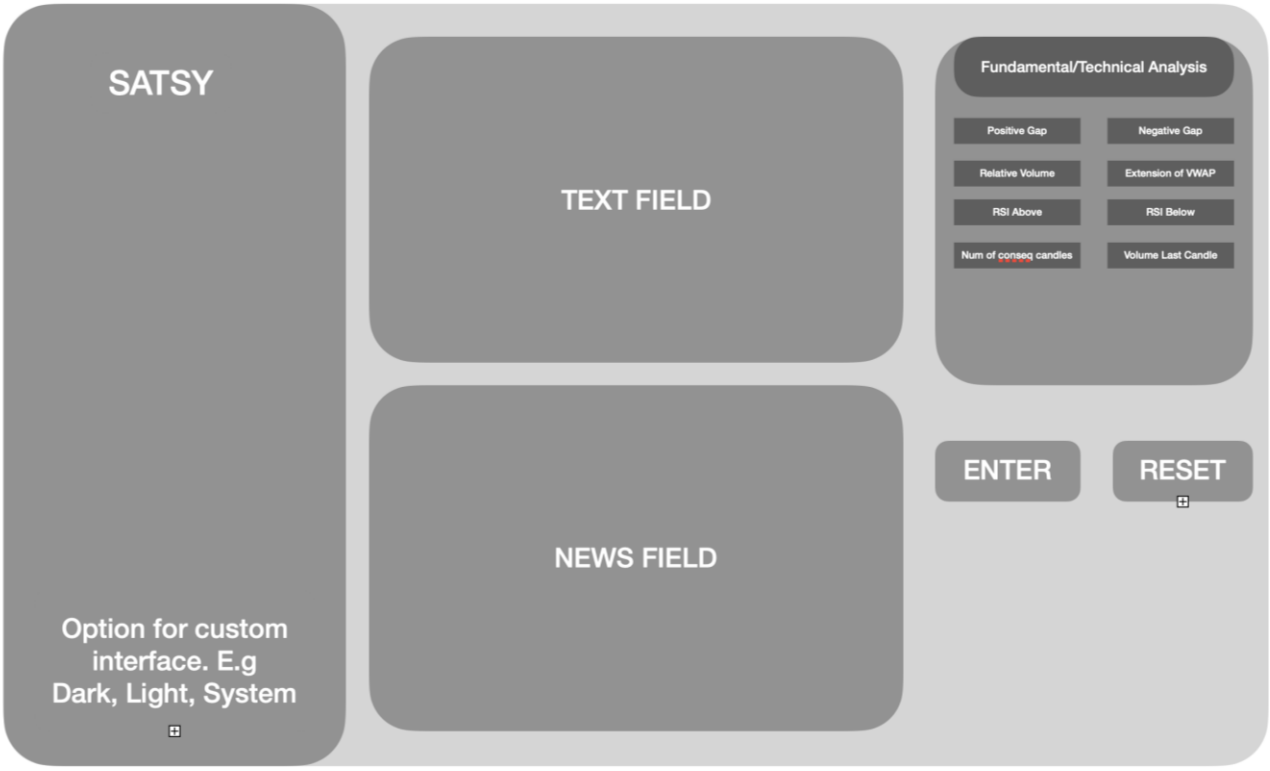
GUI Screen#2 GAP Screener, Fundamental input window



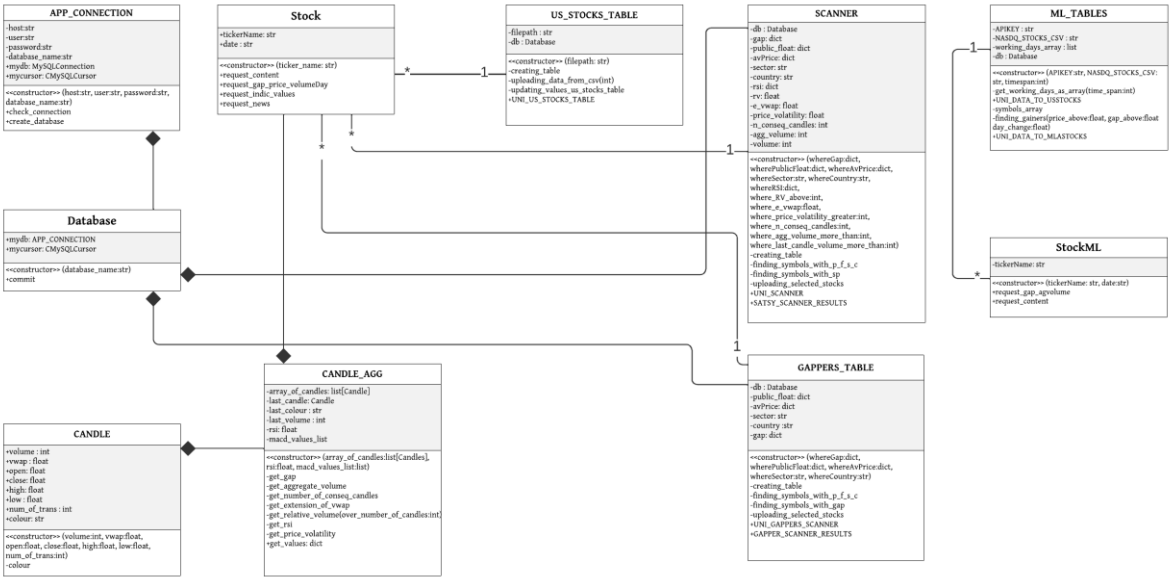
GUI Screen#3 SATSY Screener, Fundamental input window

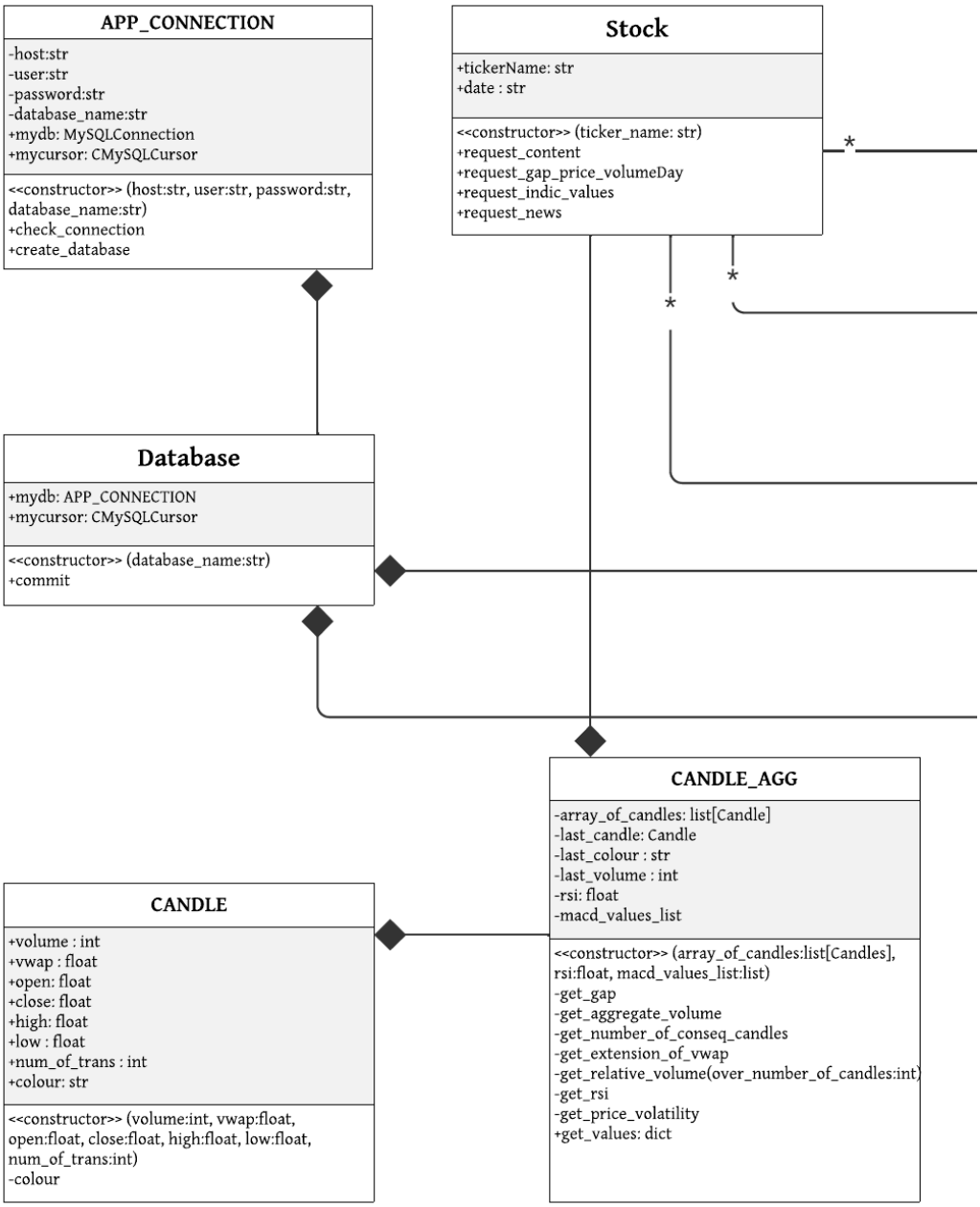


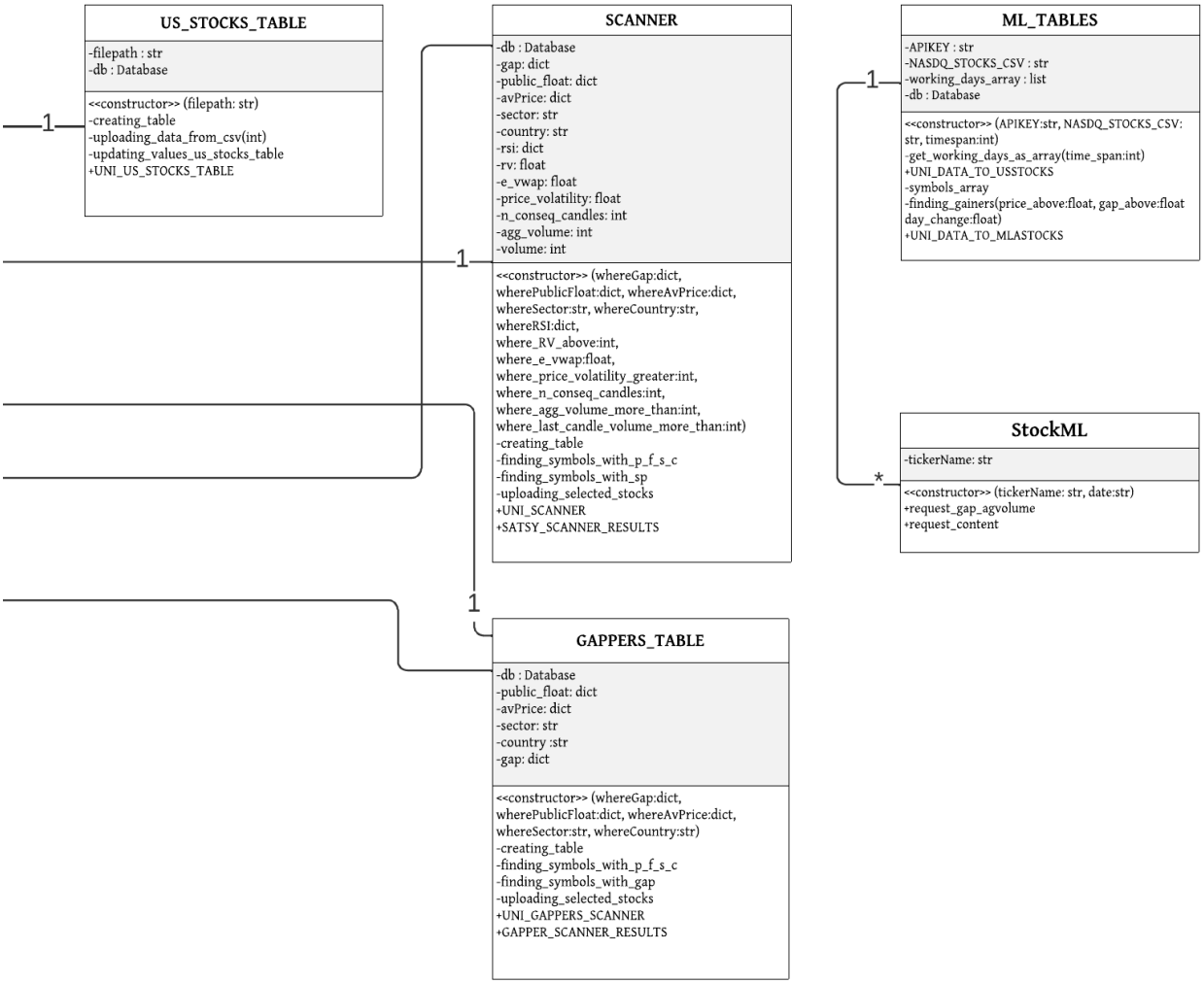
GUI Screen#4 SATSY Screener, Technical input window



UML DIAGRAM


APP_CONNECTION
-host:str
-user:str
-password:str
-database_name:str
+mydb: MySQLConnection
+mycursor: CMySQLCursor
<<constructor>> (host:str, user:str, password:str, database_name:str)
+check_connection
+create_database
Stock
+tickerName: str
+date : str
<<constructor>> (ticker_name: str)
+request_content
+request_gap_price_volumeDay
+request_indic_values
+request_news
US_STOCKS_TABLE
-filepath : str
-db : Database
<<constructor>> (filepath: str)
-creating_table
-uploading_data_from_csv(int)
-updating_values_us_stocks_table
+UNI_US_STOCKS_TABLE
SCANNER
-db : Database
-gap: dict
-public_float: dict
-avPrice: dict
-sector: str
-country: str
-rsi: dict
-rv: float
-e_vwap: float
-price_volatility: float
-n_conseq_candles: int
-agg_volume: int
-volume: int
<<constructor>> (whereGap:dict, wherePublicFloat:dict, whereAvPrice:dict, whereSector:str, whereCountry:str, whereRSI:dict, where_RV_above:int, where_e_vwap:float, where_price_volatility_greater:int, where_n_conseq_candles:int, where_agg_volume_more_than:int, where_last_candle_volume_more_than:int)
-creating_table
-finding_symbols_with_p_f_s_c
-finding_symbols_with_sp
-uploading_selected_stocks
+UNI_SCANNER
+SATSY_SCANNER_RESULTS
ML_TABLES
-APIKEY : str
-NASDQ_STOCKS_CSV : str
-working_days_array : list
-db : Database
<<constructor>> (APIKEY:str, NASDQ_STOCKS_CSV: str, timespan:int)
-get_working_days_as_array(time_span:int)
+UNI_DATA_TO_USSTOCKS
-symbols_array
-finding_gainers(price_above:float, gap_above:float day_change:float)
+UNI_DATA_TO_MLASTOCKS
Database
+mydb: APP_CONNECTION
+mycursor: CMySQLCursor
<<constructor>> (database_name:str)
+commit
StockML
-tickerName: str
<<constructor>> (tickerName: str, date:str)
+request_gap_agvolume
+request_content
CANDLE_AGG
-array_of_candles: list[Candle]
-last_candle: Candle
-last_colour : str
-last_volume : int
-rsi: float
-macd_values_list
<<constructor>> (array_of_candles:list[Candles], rsi:float, macd_values_list:list)
-get_gap
-get_aggregate_volume
-get_number_of_conseq_candles
-get_extension_of_vwap
-get_relative_volume(over_number_of_candles:int)
-get_rsi
-get_price_volatility
+get_values: dict
GAPPERS_TABLE
-db : Database
-public_float: dict
-avPrice: dict
-sector: str
-country :str
-gap: dict
<<constructor>> (whereGap:dict, wherePublicFloat:dict, whereAvPrice:dict, whereSector:str, whereCountry:str)
-creating_table
-finding_symbols_with_p_f_s_c
-finding_symbols_with_gap
-uploading_selected_stocks
+UNI_GAPPERS_SCANNER
+GAPPER_SCANNER_RESULTS
CANDLE
+volume : int
+vwap : float
+open: float
+close: float
+high: float
+low : float
+num_of_trans : int
+colour: str
<<constructor>> (volume:int, vwap:float, open:float, close:float, high:float, low:float, num_of_trans:int)
-colour
*
1
*
*
1
1
1
*
1
*


APP_CONNECTION
-host:str
-user:str
-password:str
-database_name:str
+mydb: MySQLConnection
+mycursor: CMySQLCursor
<<constructor>> (host:str, user:str, password:str, database_name:str)
+check_connection
+create_database
Stock
+tickerName: str
+date : str
<<constructor>> (ticker_name: str)
+request_content
+request_gap_price_volumeDay
+request_indic_values
+request_news
*
*
*
Database
+mydb: APP_CONNECTION
+mycursor: CMySQLCursor
<<constructor>> (database_name:str)
+commit
CANDLE_AGG
-array_of_candles: list[Candle]
-last_candle: Candle
-last_colour : str
-last_volume : int
-rsi: float
-macd_values_list
<<constructor>> (array_of_candles:list[Candles], rsi:float, macd_values_list:list)
-get_gap
-get_aggregate_volume
-get_number_of_conseq_candles
-get_extension_of_vwap
-get_relative_volume(over_number_of_candles:int)
-get_rsi
-get_price_volatility
+get_values: dict
CANDLE
+volume : int
+vwap : float
+open: float
+close: float
+high: float
+low : float
+num_of_trans : int
+colour: str
<<constructor>> (volume:int, vwap:float, open:float, close:float, high:float, low:float, num_of_trans:int)
-colour


US_STOCKS_TABLE
-filepath : str
-db : Database
<<constructor>> (filepath: str)
-creating_table
-uploading_data_from_csv(int)
-updating_values_us_stocks_table
+UNI_US_STOCKS_TABLE
SCANNER
-db : Database
-gap: dict
-public_float: dict
-avPrice: dict
-sector: str
-country: str
-rsi: dict
-rv: float
-e_vwap: float
-price_volatility: float
-n_conseq_candles: int
-agg_volume: int
-volume: int
<<constructor>> (whereGap:dict, wherePublicFloat:dict, whereAvPrice:dict, whereSector:str, whereCountry:str, whereRSI:dict, where_RV_above:int, where_e_vwap:float, where_price_volatility_greater:int, where_n_conseq_candles:int, where_agg_volume_more_than:int, where_last_candle_volume_more_than:int)
-creating_table
-finding_symbols_with_p_f_s_c
-finding_symbols_with_sp
-uploading_selected_stocks
+UNI_SCANNER
+SATSY_SCANNER_RESULTS
ML_TABLES
-APIKEY : str
-NASDQ_STOCKS_CSV : str
-working_days_array : list
-db : Database
<<constructor>> (APIKEY:str, NASDQ_STOCKS_CSV: str, timespan:int)
-get_working_days_as_array(time_span:int)
+UNI_DATA_TO_USSTOCKS
-symbols_array
-finding_gainers(price_above:float, gap_above:float day_change:float)
+UNI_DATA_TO_MLASTOCKS
StockML
-tickerName: str
<<constructor>> (tickerName: str, date:str)
+request_gap_agvolume
+request_content
GAPPERS_TABLE
-db : Database
-public_float: dict
-avPrice: dict
-sector: str
-country :str
-gap: dict
<<constructor>> (whereGap:dict, wherePublicFloat:dict, whereAvPrice:dict, whereSector:str, whereCountry:str)
-creating_table
-finding_symbols_with_p_f_s_c
-finding_symbols_with_gap
-uploading_selected_stocks
+UNI_GAPPERS_SCANNER
+GAPPER_SCANNER_RESULTS
1
1
1
1
*

TECHNICAL SOLUTION

```python
import concurrent.futures
import requests
import datetime
import numpy as np
from SQL import Database
import pandas as pd
from bs4 import BeautifulSoup
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from KEY_FILE import APIKEY



class ML_TABLES:
    def __init__(self, APIKEY, NASDQ_STOCKS_CSV, timespan=200):
        self.__APIKEY = APIKEY
        self.__NASDQ_STOCKS_CSV = NASDQ_STOCKS_CSV
        self.__working_days_array = self.__get_working_days_as_array(time_span=timespan)
        self.__db = Database(database_name='StocksMLA')
        self.__index = 0

    def __get_working_days_as_array(self, time_span):
        #method returns list of dates, when the market is open

        working_days_array = []
        today = datetime.date.today()
        for i in range(time_span):
            delta = datetime.timedelta(days=i)
            day = today - delta
            if day.isoweekday() == 6 or day.isoweekday() == 7:
                continue
            else:
                working_days_array.append(day)
        return working_days_array

    def UNI_DATA_TO_USSTOCKS(self):
        def creating_table_USStocks():
            #Creates Table in StocksMLA
            self.__db.mycursor.execute('DROP TABLE USStocks')
            self.__db.mycursor.execute("""CREATE TABLE USStocks(
                                symbolID VARCHAR(255),
                                stock_name VARCHAR(255),
                                stock_price VARCHAR(255),
                                stock_market_cap VARCHAR(255),
                                country VARCHAR(255),
                                stock_volume VARCHAR(255),
                                stock_sector VARCHAR(255),
                                stock_industry VARCHAR(255),
                                date_time VARCHAR(255),
                                stock_gap FLOAT(5),
                                stock_number_of_shares_traded BIGINT(255),
```

Obj 9 a. Complex OOP model

Simple Data model

```
                                    stock_day_volume BIGINT(255),
                                    stock_volatility BIGINT(255),
                                    stock_day_change FLOAT(5),
                                    stock_total_change FLOAT(5))""")
            self.__db.commit()
            print('Table USStocks created')
            #DATETIME

        def uploading_data_to_USStocks(numberOfRows=8000):
            #Uploades data from csv to table USStocks in StocksMLA database

            #Data formating
            data = pd.read_csv(r"" + f"{self.__NASDQ_STOCKS_CSV}") # nrows=22 skiprows=range(1,4)
            df = pd.DataFrame(data)
            df.fillna("Nan", inplace=True)

            #Retrieving data from csv
            csv_data = []
            for row in df.itertuples():
                #Symbol,Name,Last Sale,Net Change,% Change,Market Cap,Country,IPO Year,Volume,Sector,Industry
                formated_row = (row[1], row[2], row[3],
                                row[6], row[7], row[9],
                                row[10], row[11], 0,0,0,0,0,0,0)
                csv_data.append(formated_row)

            #Inserting Data
            try:
                sqlFormula = """INSERT INTO USStocks(symbolID, stock_name, stock_price,
                                                     stock_market_cap, country, stock_volume,
                                                     stock_sector, stock_industry, date_time,
                                                     stock_gap, stock_number_of_shares_traded,
                                                     stock_day_volume, stock_volatility,
                                                     stock_day_change, stock_total_change)
                                                     VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""
                self.__db.mycursor.executemany(operation=sqlFormula,
                                                   seq_params=csv_data)
                self.__db.commit()
                print('SQL injection from csv finished')
            except:
                raise Exception

        creating_table_USStocks()
        uploading_data_to_USStocks()
```

CSV file reading

```python

    def __symbols_array(self):
        symbols_array = []
        sqlOrderFormula = "SELECT symbolID FROM USStocks"
        self.__db.mycursor.execute(sqlOrderFormula)
        myResults = self.__db.mycursor.fetchall()
        for row in myResults:
            symbols_array.append(row[0])
        return symbols_array

    def __finding_gainers(self, price_above=1.0, gap_above=5.0, day_change=8.0):
        #Finding stocks which have risen the most over the specified period, with the specified gap, price and day_change

        results = []
        print(f'Analyse of these days in process {self.__working_days_array}')
        for day in self.__working_days_array:
            print(f'Finding gainers for {day}')
            #for symbol in symbols_array():
            def gappers(symbol):
                stock = StockML(tickerName=symbol, date=f'{day}')
                data = stock.request_gap_agvolume()
                if float(data['day_change']) > day_change and float(data['price'])>price_above and float(data['gap']) > gap_above:
                    print(data, symbol)
                    results.append((day, symbol, data))
                self.__index += 1
                print(self.__index)

            with concurrent.futures.ThreadPoolExecutor(max_workers=12) as executor:
                executor.map(gappers, self.__symbols_array(), timeout=10)

        number_of_days = len(self.__working_days_array)
        print(f' Gainers in {number_of_days} of these days {self.__working_days_array} with following results = {results}')
        return results

    def UNI_DATA_TO_MLASTOCKS(self, price_above=1.0, gap_above=5.0, day_change=8.0):
        gainers_array = self.__finding_gainers(price_above=price_above, gap_above=gap_above, day_change=day_change)
        def creating_table_MLAStocks():
            #Creates Table in StocksMLA
            self.__db.mycursor.execute('DROP TABLE MLAStocks')
            self.__db.mycursor.execute("""CREATE TABLE MLAStocks(
                                        symbolID VARCHAR(255),
                                        stock_price FLOAT(5),
                                        date VARCHAR(255),
                                        stock_gap FLOAT(5),
```

```
                                        stock_float BIGINT(255),
                                        stock_drv FLOAT(5),
                                        stock_vofi FLOAT(5),
                                        stock_short_interest FLOAT(5),
                                        beta FLOAT(5),
                                        stock_market_cap BIGINT(255),
                                        stock_sector VARCHAR(255),
                                        stock_day_change FLOAT(5))""")
            print('Table MLAStocks created')
            self.__db.commit()

        def finding_info_on_stocks():
            #Findes data on gainer stocks

            GAINERS_DATA = []

            def finding_marketCap_country_volume_sector_indusrty(forSymbol: str):
                self.__db.mycursor.execute(f"""SELECT symbolID, stock_market_cap, country,
                                    stock_volume, stock_sector, stock_industry
                                    FROM StocksMLA.USStocks
                                    WHERE symbolID='{forSymbol}'""")  # ORDER BY public_float DESC
                myResult = self.__db.mycursor.fetchall()
                return myResult

            def appending_to_stocks_data_array(symbol, stock_price, date, gap, number_of_shares_traded,
                stock_drv, vofi, short_interest, str_beta, market_cap, sector, day_change):
                if str_beta == 'N/A':
                    beta = 1
                else:
                    try:
                        beta = float(str_beta)
                    except:
                        beta = 1

                formatedRow = (symbol, "{:.2f}".format(stock_price), date,
                               "{:.2f}".format(gap), number_of_shares_traded,
                               "{:.2f}".format(stock_drv), "{:.5f}".format(vofi),
                               "{:.5f}".format(short_interest), beta, market_cap, sector,
                               "{:.2f}".format(day_change))
                print(formatedRow)
                GAINERS_DATA.append(formatedRow)
```

```python

            for row in gainers_array:
                date = row[0]
                stock_symbol = row[1]
                av_price = row[2]['price']
                gap = row[2]['gap']
                volume = row[2]['day_volume']
                day_change = row[2]['day_change']


                stock = StockML(tickerName=stock_symbol, date=date)
                stock_content = stock.request_content()

                MVICS = finding_marketCap_country_volume_sector_indusrty(forSymbol=stock_symbol)
                sector = MVICS[0][4]

                shares_outstanding = stock_content['Shares Outstanding']
                shares_float = stock_content['Public Float']


                stock_drv = 0
                if shares_outstanding > 0 and shares_float > 0:
                    number_of_traded_shares = shares_float
                    short_interest = (stock_content['Short Interest'] / number_of_traded_shares) * 100
                    vofi = volume / number_of_traded_shares
                    appending_to_stocks_data_array(symbol=stock_symbol, stock_price=av_price, date=date, gap=gap,
                                                   number_of_shares_traded=number_of_traded_shares, stock_drv=stock_drv,
                                                   vofi=vofi, short_interest=short_interest,
                                                   str_beta=stock_content['Beta'],
                                                   market_cap=stock_content['Market Cap'],
                                                   sector=sector, day_change= day_change)
                else:
                    if shares_outstanding > 0:
                        number_of_traded_shares = shares_outstanding
                        short_interest = (stock_content['Short Interest'] / number_of_traded_shares) * 100
                        vofi = volume / number_of_traded_shares
                        appending_to_stocks_data_array(symbol=stock_symbol, stock_price=av_price, date=date, gap=gap,
                                                       number_of_shares_traded=number_of_traded_shares,
                                                       stock_drv=stock_drv,
```

```
                                                        vofi=vofi, short_interest=short_interest,
                                                        str_beta=stock_content['Beta'],
                                                        market_cap=stock_content['Market Cap'],
                                                        sector=sector, day_change=day_change)
                    elif shares_float > 0:
                        number_of_traded_shares = shares_float
                        short_interest = (stock_content['Short Interest'] / number_of_traded_shares) * 100
                        vofi = volume / number_of_traded_shares
                        appending_to_stocks_data_array(symbol=stock_symbol, stock_price=av_price, date=date, gap=gap,
                                                        number_of_shares_traded=number_of_traded_shares,
                                                        stock_drv=stock_drv,
                                                        vofi=vofi, short_interest=short_interest,
                                                        str_beta=stock_content['Beta'],
                                                        market_cap=stock_content['Market Cap'],
                                                        sector=sector, day_change=day_change)
                    else:
                        print(f"Haven't found data for {stock_symbol}")
            return GAINERS_DATA

        def inserting_into_MLAStocks():
            #Inserting data into the MLAStocks table
            gainers_data = finding_info_on_stocks()
            formula = """INSERT INTO MLAStocks(symbolID, stock_price, date, stock_gap,
                                stock_float, stock_drv, stock_vofi, stock_short_interest,
                                beta, stock_market_cap, stock_sector, stock_day_change)
                                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""
            self.__db.mycursor.executemany(formula, gainers_data)
            self.__db.commit()
            print("Insertion into MLAStocks finished")

        creating_table_MLAStocks()
        inserting_into_MLAStocks()



class StockML:
```

```
    def __init__(self, tickerName, date):
        self.__tickerName = str(tickerName).upper()
        self.__date = datetime.date.today()
        #self.date = date
        #self.dt = '2022-11-11'

    def request_gap_agvolume(self):
        #method requests data on stocks through API calls
        try:
            url = f'https://api.polygon.io/v1/open-close/{self.__tickerName}/{self.__date}?adjusted=true&apiKey={APIKEY}'
            request = requests.get(url)
            data = request.json()
            status = data['status']
            if status == 'OK':
                price = "{:.2f}".format((data['close'] + data['open']) /2)
                open_to_premarket_change = "{:.2f}".format(((data['open'] - data['preMarket']) / data['preMarket']) *100)
                close_to_open_change = "{:.2f}".format(((data['close'] - data['open']) / data['open'])*100)
                close_to_premarket_change = "{:.2f}".format(((data['close'] - data['preMarket']) / data['preMarket'])*100)
                volume = "{:.0f}".format(data['volume'])
                market_open_volatility = "{:.2f}".format((abs(data['high'] - data['low']) / data['low'])*100)
                data = {"price": float(price),
                        "gap": float(open_to_premarket_change),
                        "day_change": float(close_to_open_change),
                        "total_change":float(close_to_premarket_change),
                        "day_volume": int(volume),
                        "volatility": float(market_open_volatility)}
                return data
            else:
                data = {"price": 0,
                        "gap": 0,
                        "day_change": 0,
                        "total_change": 0,
                        "day_volume": 0,
                        "volatility": 0}
                return data
        except:
            data = {"price":0,
                    "gap": 0,
                    "day_change": 0,
                    "total_change": 0,
                    "day_volume": 0,
                    "volatility": 0}
            return data

    def request_content(self):
        # Method collects content on stocks from the marketwatch.com
```

API requests

```python
        ticker = str(self.__tickerName).lower()
        stockDict = {'Open': 0, 'Day Range': 0, '52 Week Range': 0, 'Market Cap': 0, 'Shares Outstanding': 0, 'Public Float': 0, 'Beta': 0, 'Rev. per Employee': 0, 'P/E Ratio': 0, 'EPS': 0, 'Yield': 0, 'Dividend': 0, 'Ex-Dividend Date': 0, 'Short Interest': 0, 'Average Volume': 0}
        try:
            url = f"https://www.marketwatch.com/investing/stock/{ticker}"
            htmlContent = requests.get(url)
            soup = BeautifulSoup(htmlContent.text, 'html.parser')
            result = soup.find_all('li', attrs={'class': 'kv__item'})
            for value in result:
                titel = value.find('small').text
                titelValue = value.find('span', attrs={'class': 'primary'}).text
                dict = {titel: titelValue}
                for key, value in dict.items():
                    if key == "Market Cap" or key == "Open" or key == "Shares Outstanding" or key == 'Public Float' or key == "Rev. per Employee" or key == "Short Interest" or key == "% of Float Shorted" or key == "Average Volume":
                        plainValue = value.strip('$')
                        lastChar = plainValue[-1]
                        if lastChar == "K":
                            try:
                                formatedValue = plainValue.strip(lastChar)
                                floatValue = float(formatedValue) * 1000
                                stockDict[key] = int(floatValue)
                            except:
                                print(f"Error when trying to convert str to int in {self.__tickerName}")
                                stockDict[key] = 0
                        elif lastChar == "M":
                            try:
                                formatedValue = plainValue.strip(lastChar)
                                floatValue = float(formatedValue) * 1000000
                                stockDict[key] = int(floatValue)
                            except:
                                print(f"Error when trying to convert str to int in {self.__tickerName}")
                                stockDict[key] = 0
                        elif lastChar == "B":
                            try:
                                formatedValue = plainValue.strip(lastChar)
                                floatValue = float(formatedValue) * 1000000000
                                stockDict[key] = int(floatValue)
```

Web data parsing

```
                            except:
                                print(f"Error when trying to convert str to int in {self.__tickerName}")
                                stockDict[key] = 0
                        elif lastChar == "T":
                            try:
                                formatedValue = plainValue.strip(lastChar)
                                floatValue = float(formatedValue) * 1000000000000
                                stockDict[key] = int(floatValue)
                            except:
                                print(f"Error when trying to convert str to int in {self.__tickerName}")
                                stockDict[key] = 0
                        else:
                            try:
                                floatValue = float(plainValue)
                                stockDict[key] = floatValue
                            except:
                                stockDict[key] = 0
                    else:
                        stockDict[key] = value
            return stockDict
        except:
            return stockDict


# Ctreates Random Forest Classification model
def RFR_model():
    def fetching_gainers_data():
        db = Database(database_name='StocksMLA')
        db.mycursor.execute(f"""SELECT stock_price, stock_gap,
                                    stock_float, stock_vofi, stock_short_interest,
                                    beta, stock_day_change
                                    FROM MLAStocks""")  # ORDER BY public_float DESC
        myResults = db.mycursor.fetchall()
        df = pd.DataFrame(data=myResults, columns=['stock_price', 'stock_gap',
                                    'stock_float', 'stock_vofi', 'stock_short_interest',
                                    'beta', 'stock_day_change'])
        return df
    data = fetching_gainers_data()
    print(data)

    # Importing the dataset
    X = data.iloc[:, :-1].values
    y = data.iloc[:, -1].values

    X_train, X_test, y_train, y_test = train_test_split(X, y,
```

Obj 9.b

Complex ML model

```
test_size = 0.4, random_state = 0)

    regressor = RandomForestRegressor(n_estimators=50, random_state=0)
    regressor.fit(X_train, y_train)
    return regressor


#creates function that uses the model to predict potential increase in the price of the stock
def predict(list, model=RFR_model()):
    y_pred = model.predict(list)
    np.set_printoptions(precision=2)
    predicted_value = 0
    for value in y_pred:
        predicted_value = round(value, 3)
    return predicted_value
```

Obj 9.c

Obj 8

Complex OOP - inheritance

```python
from DATABASE import SCANNER, GAPPERS_TABLE, US_STOCKS_TABLE
import tkinter
import tkinter.messagebox
import customtkinter
from ML import ML_TABLES, predict
from KEY_FILE import APIKEY, NASDQ_STOCKS_CSV


customtkinter.set_appearance_mode("System")  # Modes: "System" (standard), "Dark", "Light"
customtkinter.set_default_color_theme("blue")  # Themes: "blue" (standard), "green", "dark-blue"


class App(customtkinter.CTk):
    def __init__(self):
        super().__init__()
        font_tuple_title = ("Comic Sans MS", 26, "bold")
        font_tuple_subtitle = ("Comic Sans MS", 14, "bold")
        font_tuple_text = ("Comic Sans MS", 13, "bold")

        # configure window
        self.title("SATSY")
        self.geometry(f"{1366}x{768}")

        # configure grid layout (4x4)
        self.grid_columnconfigure(1, weight=1)
        self.grid_columnconfigure(2, weight=0)
        self.grid_rowconfigure((0, 1, 2), weight=1)

        # create sidebar frame with widgets
        self.sidebar_frame = customtkinter.CTkFrame(self, width=180, corner_radius=0)
        self.sidebar_frame.grid(row=0, column=0, rowspan=4, sticky="nsew")

        self.sidebar_frame.grid_rowconfigure(1, weight=1)
        self.logo_label = customtkinter.CTkLabel(self.sidebar_frame, text="SATSY", font=font_tuple_title)
        self.logo_label.grid(row=0, column=0, padx=20, pady=(20, 10))

        self.options_mode_label = customtkinter.CTkLabel(self.sidebar_frame, text="OPTIONS:", anchor="w",
                                                         font=font_tuple_subtitle)
        self.options_mode_label.grid(row=3, column=0, padx=20, pady=(10, 0))
        self.options_mode_optionemenu = customtkinter.CTkOptionMenu(self.sidebar_frame,
values=["GAPPER SCANNER", "SATSY SCANNER"],
command=self.change_scanner_type,
```

```
43
   font=font_tuple_subtitle)
44         self.options_mode_optionemenu.grid(row=4, column=0, padx=20,
   pady=(10, 10))
45
46         self.appearance_mode_label = customtkinter.CTkLabel(self.
   sidebar_frame, text="APPEARENCE MODE:", anchor="w",
47                                                                font=
   font_tuple_subtitle)
48         self.appearance_mode_label.grid(row=5, column=0, padx=20, pady
   =(10, 0))
49         self.appearance_mode_optionemenu = customtkinter.CTkOptionMenu
   (self.sidebar_frame,
50
    values=["Light", "Dark", "System"],
51
    command=self.change_appearance_mode_event,
52
    font=font_tuple_subtitle)
53         self.appearance_mode_optionemenu.grid(row=6, column=0, padx=20
   , pady=(10, 10))
54
55         self.appearance_mode_optionemenu.set("Dark")
56
57         # ____________________________________________
58
59     def change_scanner_type(self, options: str):
60         #creates Gapper Scanner View
61
62         if options == "GAPPER SCANNER":
63             font_tuple_subtitle = ("Comic Sans MS", 14, "bold")
64             font_tuple_text = ("Comic Sans MS", 13, "bold")
65             # Creating TextBox
66             self.textbox_scanner_results = customtkinter.CTkTextbox(
   self, width=400, activate_scrollbars=True,
67
   font=font_tuple_text)
68             self.textbox_scanner_results.grid(row=0, column=1, padx=(
   20, 0), pady=(38, 0), sticky="nsew")
69
70             self.textbox_news_results = customtkinter.CTkTextbox(self
   , width=400, activate_scrollbars=True,
71                                                                    font=
   font_tuple_text)
72             self.textbox_news_results.grid(row=1, column=1, padx=(20,
   0), pady=(20, 20), sticky="nsew")
73
74             # Creating TabView
75             self.tabview = customtkinter.CTkTabview(self, width=250)
76             self.tabview.grid(row=0, column=2, padx=(20, 20), pady=(20
   , 0), sticky="nsew")
77             self.tabview.add("Fundamentals")
78             self.tabview.add("Technicals")
```

```
            self.tabview.tab("Fundamentals").grid_columnconfigure(0, weight=1)  # configure grid of individual tabs
            self.tabview.tab("Technicals").grid_columnconfigure(0, weight=1)

            self.stock_from_price_range_label = customtkinter.CTkLabel(self.tabview.tab("Fundamentals"),
                                                                       anchor="center",
                                                                       text="From Price",
                                                                       font=font_tuple_subtitle)
            self.stock_from_price_range_label.grid(row=0, column=0, padx=20, pady=(10, 2))
            self.stock_from_price_combo_box = customtkinter.CTkComboBox(self.tabview.tab("Fundamentals"),
                                                                        values=["0$", "1$", "5$", "10$",
                                                                                "20$", "30$", "40$", "50$",
                                                                                "100$", "150$", "200$"])
            self.stock_from_price_combo_box.grid(row=1, column=0, padx=20, pady=(2, 5))
            self.stock_to_price_range_label = customtkinter.CTkLabel(self.tabview.tab("Fundamentals"), anchor="center",
                                                                     text="To Price",
                                                                     font=font_tuple_subtitle)
            self.stock_to_price_range_label.grid(row=0, column=1, padx=20, pady=(10, 2))
            self.stock_to_price_combo_box = customtkinter.CTkComboBox(self.tabview.tab("Fundamentals"),
                                                                      values=["5$", "10$", "20$", "30$", "40$", "50$",
                                                                              "100$", "150$", "200$", "300$", "500$",
                                                                              "1000$", "10000$"])
            self.stock_to_price_combo_box.grid(row=1, column=1, padx=20, pady=(2, 5))

            self.stock_from_float_label = customtkinter.CTkLabel(self.tabview.tab("Fundamentals"), text="From Float",
                                                                 font=font_tuple_subtitle)
            self.stock_from_float_label.grid(row=2, column=0, padx=20, pady=(5, 2))
            self.stock_from_float_combo_box = customtkinter.CTkComboBox(self.tabview.tab("Fundamentals"),

```

```
    values=["0", "500K",
                "1M", "10M", "20M",
                "50M", "100M", "200M", "500M"])
            self.stock_from_float_combo_box.grid(row=3, column=0, padx=20, pady=(2, 5))
            self.stock_to_float_label = customtkinter.CTkLabel(self.tabview.tab("Fundamentals"), text="To Float",
                                                               font=font_tuple_subtitle)
            self.stock_to_float_label.grid(row=2, column=1, padx=20, pady=(5, 2))
            self.stock_to_float_combo_box = customtkinter.CTkComboBox(self.tabview.tab("Fundamentals"),
     values=["1M", "5M", "10M",
            "20M", "50M", "100M", "200M", "500M",
            "1T", "100T"])
            self.stock_to_float_combo_box.grid(row=3, column=1, padx=20, pady=(2, 5))

            self.stock_sector_label = customtkinter.CTkLabel(self.tabview.tab("Fundamentals"), text="Sector",
                                                             font=font_tuple_subtitle)
            self.stock_sector_label.grid(row=4, column=0, padx=20, pady=(5, 2))
            self.stock_sector_combo_box = customtkinter.CTkComboBox(self.tabview.tab("Fundamentals"),
    values=['ALL', 'Finance', 'Industrials',
            'Consumer Discretionary',
            'Health Care', 'Real Estate', 'Utilities',
            'Telecommunications', 'Technology',
            'Miscellaneous',
            'Consumer Staples', 'Basic Materials'])
            self.stock_sector_combo_box.grid(row=5, column=0, padx=20, pady=(2, 5))

            self.stock_country_label = customtkinter.CTkLabel(self.tabview.tab("Fundamentals"), text="Country",
                                                              font=font_tuple_subtitle)
            self.stock_country_label.grid(row=4, column=1, padx=20, pady=(5, 2))
```

```
            self.stock_country_combo_box = customtkinter.CTkComboBox(self.tabview.tab("Fundamentals"),
values=['ALL', 'United States', 'United Kingdom',
        'United Arab Emirates', 'Taiwan',
        'Switzerland',
        'Sweden', 'Singapore', 'Peru',
        'Netherlands',
        'Israel', 'India', 'China', 'Brazil',
        'Mexico',
        'Canada', 'Bermuda'])
            self.stock_country_combo_box.grid(row=5, column=1, padx=20, pady=(2, 5))
            #
---------------------------------------------------------------

            self.stock_positive_gap_label = customtkinter.CTkLabel(self.tabview.tab("Technicals"), anchor="center",
text="Positive Gap",
font=font_tuple_subtitle)
            self.stock_positive_gap_label.grid(row=0, column=0, padx=20, pady=(10, 2))
            self.stock_positive_gap_combo_box = customtkinter.CTkComboBox(self.tabview.tab("Technicals"),
     values=["Nan", "Over 50%", "Over 40%",
             "Over 30%",
             'Over 20%', "Over 10%", "Over 5%"])
            self.stock_positive_gap_combo_box.grid(row=1, column=0, padx=20, pady=(2, 5))

            self.stock_negative_gap_label = customtkinter.CTkLabel(self.tabview.tab("Technicals"), anchor="center",
text="Negative Gap",
font=font_tuple_subtitle)
            self.stock_negative_gap_label.grid(row=0, column=1, padx=20, pady=(10, 2))
            self.stock_negative_gap_combo_box = customtkinter.CTkComboBox(self.tabview.tab("Technicals"),

```

```
     values=["Nan", "Over -50%", "Over -40%",
                "Over -30%",
                'Over -20%', "Over -10%", "Over -5%"])
            self.stock_negative_gap_combo_box.grid(row=1, column=1, padx=20, pady=(2, 5))

            # ----------------------------------------------------------------

            self.frame_buttons = customtkinter.CTkFrame(master=self, fg_color="transparent")
            self.frame_buttons.grid(row=1, column=2, padx=(20, 20), pady=(20, 20), sticky="nsew")

            self.enter_button = customtkinter.CTkButton(master=self.frame_buttons, command=self.start_scanning_gappers,
                                                        text='ENTER', border_width=2,
                                                        text_color=("gray10", "#DCE4EE"), font=font_tuple_subtitle)
            self.enter_button.grid(row=0, column=0, padx=(20, 10), pady=(20, 20), sticky="nsew")

            self.reset_button = customtkinter.CTkButton(master=self.frame_buttons, command=self.reset_gapper_scanner,
                                                        text='RESET', border_width=2,
                                                        text_color=("gray10", "#DCE4EE"), font=font_tuple_subtitle)
            self.reset_button.grid(row=0, column=1, padx=(10, 20), pady=(20, 20), sticky="nsew")

            self.reset_gapper_scanner()




        # ------------------------------------------------------------------------------------------------

        elif options == "SATSY SCANNER":
            #creates Satsy Scanner view

            font_tuple_title = ("Comic Sans MS", 26, "bold")
            font_tuple_subtitle = ("Comic Sans MS", 14, "bold")
            font_tuple_text = ("Comic Sans MS", 13, "bold")

            # Creating TextBox
            self.textbox_scanner_results = customtkinter.CTkTextbox(self, width=400, activate_scrollbars=True,
```

```
197
    font=font_tuple_text)
198             self.textbox_scanner_results.grid(row=0, column=1, padx=(
    20, 0), pady=(38, 0), sticky="nsew")
199
200             self.textbox_news_results = customtkinter.CTkTextbox(self
    , width=400, activate_scrollbars=True,
201                                                                  font
    =font_tuple_text)
202             self.textbox_news_results.grid(row=1, column=1, padx=(20
    , 0), pady=(20, 20), sticky="nsew")
203
204             # Creating TabView
205             self.tabview = customtkinter.CTkTabview(self, width=250)
206             self.tabview.grid(row=0, column=2, padx=(20, 20), pady=(
    20, 0), sticky="nsew")
207             self.tabview.add("Fundamentals")
208             self.tabview.add("Technicals")
209             self.tabview.tab("Fundamentals").grid_columnconfigure(0,
    weight=1)  # configure grid of individual tabs
210             self.tabview.tab("Technicals").grid_columnconfigure(0,
    weight=1)
211
212             self.stock_from_price_range_label = customtkinter.
    CTkLabel(self.tabview.tab("Fundamentals"),
213
      anchor="center",
214
      text="From Price",
215
      font=font_tuple_subtitle)
216             self.stock_from_price_range_label.grid(row=0, column=0,
    padx=20, pady=(10, 2))
217             self.stock_from_price_combo_box = customtkinter.
    CTkComboBox(self.tabview.tab("Fundamentals"),
218
       values=["0$", "1$", "5$", "10$",
219
               "20$", "30$", "40$", "50$",
220
               "100$", "150$", "200$"])
221             self.stock_from_price_combo_box.grid(row=1, column=0,
    padx=20, pady=(2, 5))
222             self.stock_to_price_range_label = customtkinter.CTkLabel(
    self.tabview.tab("Fundamentals"), anchor="center",
223
    text="To Price",
224
    font=font_tuple_subtitle)
225             self.stock_to_price_range_label.grid(row=0, column=1,
    padx=20, pady=(10, 2))
226             self.stock_to_price_combo_box = customtkinter.CTkComboBox
    (self.tabview.tab("Fundamentals"),
```

```
227
     values=["5$", "10$", "20$", "30$", "40$", "50$",
228
             "100$", "150$", "200$", "300$", "500$",
229
             "1000$", "10000$"])
230             self.stock_to_price_combo_box.grid(row=1, column=1, padx=
    20, pady=(2, 5))
231
232             self.stock_from_float_label = customtkinter.CTkLabel(self
    .tabview.tab("Fundamentals"), text="From Float",
233                                                                  font
    =font_tuple_subtitle)
234             self.stock_from_float_label.grid(row=2, column=0, padx=20
    , pady=(5, 2))
235             self.stock_from_float_combo_box = customtkinter.
    CTkComboBox(self.tabview.tab("Fundamentals"),
236
       values=["0", "500K",
237
               "1M", "10M", "20M",
238
               "50M", "100M", "200M", "500M"])
239             self.stock_from_float_combo_box.grid(row=3, column=0,
    padx=20, pady=(2, 5))
240             self.stock_to_float_label = customtkinter.CTkLabel(self.
    tabview.tab("Fundamentals"), text="To Float",
241                                                                    font=
    font_tuple_subtitle)
242             self.stock_to_float_label.grid(row=2, column=1, padx=20,
    pady=(5, 2))
243             self.stock_to_float_combo_box = customtkinter.CTkComboBox
    (self.tabview.tab("Fundamentals"),
244
     values=["1M", "5M", "10M",
245
             "20M", "50M", "100M", "200M", "500M",
246
             "1T", "100T"])
247             self.stock_to_float_combo_box.grid(row=3, column=1, padx=
    20, pady=(2, 5))
248
249             self.stock_sector_label = customtkinter.CTkLabel(self.
    tabview.tab("Fundamentals"), text="Sector",
250                                                                  font=
    font_tuple_subtitle)
251             self.stock_sector_label.grid(row=4, column=0, padx=20,
    pady=(5, 2))
252             self.stock_sector_combo_box = customtkinter.CTkComboBox(
    self.tabview.tab("Fundamentals"),
253
    values=['ALL', 'Finance', 'Industrials',
254
```

```
        'Consumer Discretionary',

        'Health Care', 'Real Estate', 'Utilities',

        'Telecommunications', 'Technology',

        'Miscellaneous',

        'Consumer Staples', 'Basic Materials'])
            self.stock_sector_combo_box.grid(row=5, column=0, padx=20, pady=(2, 5))

            self.stock_country_label = customtkinter.CTkLabel(self.tabview.tab("Fundamentals"), text="Country",
                                                              font=font_tuple_subtitle)
            self.stock_country_label.grid(row=4, column=1, padx=20, pady=(5, 2))
            self.stock_country_combo_box = customtkinter.CTkComboBox(self.tabview.tab("Fundamentals"),
                                                                     values=['ALL', 'United States', 'United Kingdom',

                                                                             'United Arab Emirates', 'Taiwan',

                                                                             'Switzerland',

                                                                             'Sweden', 'Singapore', 'Peru',

                                                                             'Netherlands',

                                                                             'Israel', 'India', 'China', 'Brazil',

                                                                             'Mexico',

                                                                             'Canada', 'Bermuda'])
            self.stock_country_combo_box.grid(row=5, column=1, padx=20, pady=(2, 5))

            #------------------------------------------------------------

            self.stock_positive_gap_label = customtkinter.CTkLabel(self.tabview.tab("Technicals"), anchor="center",
                                                                   text="Positive Gap",
                                                                   font=font_tuple_subtitle)
            self.stock_positive_gap_label.grid(row=0, column=0, padx=20, pady=(10, 2))
            self.stock_positive_gap_combo_box = customtkinter.CTkComboBox(self.tabview.tab("Technicals"),

```

```
282      values=["Nan", "Over 50%", "Over 40%",
283
                 "Over 30%",
284
                 'Over 20%', "Over 10%", "Over 5%"])
285             self.stock_positive_gap_combo_box.grid(row=1, column=0,
    padx=20, pady=(2, 5))
286
287             self.stock_negative_gap_label = customtkinter.CTkLabel(
    self.tabview.tab("Technicals"), anchor="center",
288
    text="Negative Gap",
289
    font=font_tuple_subtitle)
290             self.stock_negative_gap_label.grid(row=0, column=1, padx=
    20, pady=(10, 2))
291             self.stock_negative_gap_combo_box = customtkinter.
    CTkComboBox(self.tabview.tab("Technicals"),
292
         values=["Nan", "Over -50%", "Over -40%",
293
                 "Over -30%",
294
                 'Over -20%', "Over -10%", "Over -5%"])
295             self.stock_negative_gap_combo_box.grid(row=1, column=1,
    padx=20, pady=(2, 5))
296
297             self.stock_rv_label = customtkinter.CTkLabel(self.tabview
    .tab("Technicals"), text="Relative Volume",
298                                                          font=
    font_tuple_subtitle)
299             self.stock_rv_label.grid(row=2, column=0, padx=20, pady=(
    5, 2))
300             self.stock_rv_combo_box = customtkinter.CTkComboBox(self.
    tabview.tab("Technicals"),
301
    values=['Nan', 'Over 2', "Over 5", "Over 10",
302
       "Over 15", "Over 20"])
303             self.stock_rv_combo_box.grid(row=3, column=0, padx=20,
    pady=(2, 5))
304             self.stock_e_vwap_label = customtkinter.CTkLabel(self.
    tabview.tab("Technicals"), text="E_VWAP",
305                                                              font=
    font_tuple_subtitle)
306             self.stock_e_vwap_label.grid(row=2, column=1, padx=20,
    pady=(5, 2))
307             self.stock_e_vwap_combo_box = customtkinter.CTkComboBox(
    self.tabview.tab("Technicals"),
308
    values=['Nan', 'Over 2%', 'Over 4%',
309
           'Over 6%', 'Over 8%', 'Over 10%',
```

```
310
          'Over 15%', 'Over 20%'])
311             self.stock_e_vwap_combo_box.grid(row=3, column=1, padx=20
    , pady=(2, 5))
312
313             self.stock_above_rsi_label = customtkinter.CTkLabel(self.
    tabview.tab("Technicals"), anchor="center",
314                                                                 text=
    "RSI above:",
315                                                                 font=
    font_tuple_subtitle)
316             self.stock_above_rsi_label.grid(row=4, column=0, padx=20
    , pady=(10, 2))
317             self.stock_above_rsi_combo_box = customtkinter.
    CTkComboBox(self.tabview.tab("Technicals"),
318
      values=['Nan', "Over 90", "Over 85", "Over 80",
319
              'Over 75', "Over 70", 'Over 65',
320
              'Over 60', 'Over 55', 'Over 49'])
321             self.stock_above_rsi_combo_box.grid(row=5, column=0, padx
    =20, pady=(2, 5))
322             self.stock_below_rsi_label = customtkinter.CTkLabel(self.
    tabview.tab("Technicals"), anchor="center",
323                                                                 text=
    "RSI below:",
324                                                                 font=
    font_tuple_subtitle)
325             self.stock_below_rsi_label.grid(row=4, column=1, padx=20
    , pady=(10, 2))
326             self.stock_below_rsi_combo_box = customtkinter.
    CTkComboBox(self.tabview.tab("Technicals"),
327
      values=['Nan', 'Below 45', 'Below 40',
328
              'Below 35', 'Below 30',
329
              'Below 25', 'Below 20', 'Below 15',
330
              'Below 10'])
331             self.stock_below_rsi_combo_box.grid(row=5, column=1, padx
    =20, pady=(2, 5))
332
333             self.stock_n_conseq_can_label = customtkinter.CTkLabel(
    self.tabview.tab("Technicals"),
334
    text="Num Of Conseq Candles",
335
    font=font_tuple_subtitle)
336             self.stock_n_conseq_can_label.grid(row=6, column=0, padx=
    20, pady=(5, 2))
337             self.stock_n_conseq_can_combo_box = customtkinter.
```

```
CTkComboBox(self.tabview.tab("Technicals"),
        values=['Nan', 'Over 0', 'Over 2', 'Over 3',
                'Over 4',
                'Over 5', 'Over 6'])
            self.stock_n_conseq_can_combo_box.grid(row=7, column=0, padx=20, pady=(2, 5))

            self.stock_volume_can_label = customtkinter.CTkLabel(self.tabview.tab("Technicals"),
                                                                 text="Volume Last Candle",
                                                                 font=font_tuple_subtitle)
            self.stock_volume_can_label.grid(row=6, column=1, padx=20, pady=(5, 2))
            self.stock_volume_combo_box = customtkinter.CTkComboBox(self.tabview.tab("Technicals"),
                                                                    values=['Nan', 'Over 500', 'Over 1000', 'Over 2000',
                                                                            'Over 6000', 'Over 10000', 'Over 15000'])
            self.stock_volume_combo_box.grid(row=7, column=1, padx=20, pady=(2, 5))

            # ----------------------------------------------------------------

            self.frame_buttons = customtkinter.CTkFrame(master=self, fg_color="transparent")
            self.frame_buttons.grid(row=1, column=2, padx=(20, 20), pady=(20, 20), sticky="nsew")

            self.enter_button = customtkinter.CTkButton(master=self.frame_buttons, command=self.start_scanning_extended,
                                                        text='ENTER', border_width=2,
                                                        text_color=("gray10", "#DCE4EE"), font=font_tuple_subtitle)
            self.enter_button.grid(row=0, column=0, padx=(20, 10), pady=(20, 20), sticky="nsew")

            self.reset_button = customtkinter.CTkButton(master=self.frame_buttons, command=self.reset_satsy_scanner,
                                                        text='RESET', border_width=2,
                                                        text_color=("gray10", "#DCE4EE"), font=font_tuple_subtitle)
            self.reset_button.grid(row=0, column=1, padx=(10, 20), pady=(20, 20), sticky="nsew")

```

```
            self.reset_satsy_scanner()
            # ____________________________

    def start_scanning_gappers(self):
        try:
            # gets input entered by user in Gapper Scanner and initiates scanning
            from_p = int(self.stock_from_price_combo_box.get().replace('$', '').replace(' ', ''))
            to_p = int(self.stock_to_price_combo_box.get().replace('$', '').replace(' ', ''))
            from_f = int(self.stock_from_float_combo_box.get().replace('K', '000').replace('M', '000000').replace('B','000000000').replace('T', '000000000000'))
            to_f = int(self.stock_to_float_combo_box.get().replace('K', '000').replace('M', '000000').replace('B','000000000').replace('T', '000000000000'))
            sector = str(self.stock_sector_combo_box.get())
            country = str(self.stock_country_combo_box.get())

            pos_gap = int(self.stock_positive_gap_combo_box.get().replace('Over', '').replace('%', '').replace(' ', '').replace('Nan', '0').replace('-', ''))
            neg_gap = int(self.stock_negative_gap_combo_box.get().replace('Over', '').replace('%', '').replace(' ', '').replace('Nan', '0').replace('-', ''))
            print(from_p, to_p, from_f, to_f, pos_gap, neg_gap, sector, country)
            print(type(from_p), to_p, type(from_f), to_f, type(pos_gap), neg_gap, type(sector), type(country))
            GAPPERS_TABLE_DB = GAPPERS_TABLE(whereGap={'positive': {'minValue': pos_gap, 'maxValue': 99999},
                                                       'negative': {'minValue': -neg_gap, 'maxValue': -99999}},
                                             wherePublicFloat={'minValue': from_f,
                                                               'maxValue': to_f},
                                             whereAvPrice={'minValue': from_p,
                                                           'maxValue': to_p},
                                             whereCountry=country, whereSector=sector)
            GAPPERS_TABLE_DB.UNI_GAPPERS_SCANNER()
            results = GAPPERS_TABLE_DB.GAPPER_SCANNER_RESULTS()
            array_of_stock_data = []
            for stock_data in results[0]:
                row = [stock_data[0], stock_data[1], stock_data[2], stock_data[3], stock_data[4], stock_data[5],
                       predict([[stock_data[2], stock_data[1], stock_data[4], float(stock_data[3] / stock_data[4]),
                                 stock_data[5], 1]])]
```

```
                array_of_stock_data.append(row)
            for item in array_of_stock_data:
                self.textbox_scanner_results.insert("0.0", f'{item}\n')
            for item in results[1]:
                self.textbox_news_results.insert("0.0", f'{item}\n')
        except:
            tkinter.messagebox.showerror('ERROR', message='ERROR: Please reenter data in the right data format...' )

    def start_scanning_extended(self):
        try:
            # gets input entered by user in Satsy Scanner(Extended) and initiates scanning
            from_p = int(self.stock_from_price_combo_box.get().replace('$', '').replace(' ', ''))
            to_p = int(self.stock_to_price_combo_box.get().replace('$', '').replace(' ', ''))
            from_f = int(self.stock_from_float_combo_box.get().replace('K', '000').replace('M', '000000').replace('B',
                                                                                                '000000000').replace(
                'T', '000000000000'))
            to_f = int(self.stock_to_float_combo_box.get().replace('K', '000').replace('M', '000000').replace('B',
                                                                                                '000000000').replace(
                'T', '000000000000'))
            sector = self.stock_sector_combo_box.get()
            country = self.stock_country_combo_box.get()

            pos_gap = int(
                self.stock_positive_gap_combo_box.get().replace('Over', '').replace('%', '').replace(' ', '').replace(
                    'Nan', '0').replace('-', ''))
            neg_gap = int(
                self.stock_negative_gap_combo_box.get().replace('Over', '').replace('%', '').replace(' ', '').replace(
                    'Nan', '0').replace('-', ''))
            above_rsi = int(
                self.stock_above_rsi_combo_box.get().replace('Over', '').replace(' ', '').replace('Nan', '100'))
            below_rsi = int(
                self.stock_below_rsi_combo_box.get().replace('Below', '').replace(' ', '').replace('Nan', '0'))
            rv = int(self.stock_rv_combo_box.get().replace('Over', '').replace(' ', '').replace('Nan', '0'))
            e_vwap = int(
                self.stock_e_vwap_combo_box.get().replace('Over', '').replace('%', '').replace(' ', '').replace('Nan',
                                                                                                '0'))
            n_conseq_can = int(
```

```
                self.stock_n_conseq_can_combo_box.get().replace('Over', '').replace(' ', '').replace('Nan', '0'))
            volume = int(self.stock_volume_combo_box.get().replace('Over', '').replace(' ', '').replace('Nan', '0'))
            scanner = SCANNER(whereGap={'positive': {'minValue': pos_gap, 'maxValue': 99999},
                                        'negative': {'minValue': neg_gap, 'maxValue': -99999}},
                              wherePublicFloat={'minValue': from_f, 'maxValue': to_f},
                              whereAvPrice={'minValue': from_p, 'maxValue': to_p},
                              whereSector=sector, whereCountry=country,
                              whereRSI={'above': above_rsi, 'below': below_rsi},
                              where_RV_above=rv,
                              where_price_volatility_greater=0,
                              where_e_vwap=e_vwap,
                              where_n_conseq_candles_above=n_conseq_can,
                              where_last_candle_volume_more_than=volume,
                              where_agg_volume_more_than=0)
            scanner.UNI_SCANNER()
            results = scanner.SATSY_SCANNER_RESULTS()
            print(results)
            for item in results[0]:
                self.textbox_scanner_results.insert("0.0", f'{item}\n')
            for item in results[1]:
                self.textbox_news_results.insert("0.0", f'{item}\n')
        except:
            tkinter.messagebox.showerror('ERROR', message='ERROR: Please reenter data in the right data format...')

    def change_appearance_mode_event(self, new_appearance_mode: str):
        # changes appearance of the display
        customtkinter.set_appearance_mode(new_appearance_mode)

    def reset_gapper_scanner(self):
        # resets Gapper Scanner
        self.stock_from_price_combo_box.set('0$')
        self.stock_to_price_combo_box.set('10000$')
        self.stock_from_float_combo_box.set('0')
        self.stock_to_float_combo_box.set('100T')
        self.stock_sector_combo_box.set('ALL')
        self.stock_country_combo_box.set('ALL')

        self.stock_positive_gap_combo_box.set('Nan')
        self.stock_negative_gap_combo_box.set('Nan')

    def reset_satsy_scanner(self):
```

```
        #resets Satsy(Extended) Scanner
        self.stock_from_price_combo_box.set('0$')
        self.stock_to_price_combo_box.set('10000$')
        self.stock_from_float_combo_box.set('0')
        self.stock_to_float_combo_box.set('100T')
        self.stock_sector_combo_box.set('ALL')
        self.stock_country_combo_box.set('ALL')

        self.stock_positive_gap_combo_box.set('Nan')
        self.stock_negative_gap_combo_box.set('Nan')
        self.stock_above_rsi_combo_box.set('Nan')
        self.stock_below_rsi_combo_box.set('Nan')
        self.stock_rv_combo_box.set('Nan')
        self.stock_e_vwap_combo_box.set('Nan')
        self.stock_n_conseq_can_combo_box.set('Nan')


if __name__ == "__main__":
    file_path = str(input('Pleas eneter the path to csv file: '))
    #creates ML model
    ML_TABLES = ML_TABLES(APIKEY=APIKEY, NASDQ_STOCKS_CSV=file_path, timespan=9)
    ML_TABLES.UNI_DATA_TO_USSTOCKS()
    ML_TABLES.UNI_DATA_TO_MLASTOCKS()

    #gathers data on stocks
    US_TOCKS_TABLE = US_STOCKS_TABLE(file_path=file_path)
    US_TOCKS_TABLE.UNI_US_STOCKS_TABLE()

    #starts main program - Satsy Scanner
    app = App()
    app.mainloop()


```

```python
import mysql.connector

class APP_CONNECTION:
    def __init__(self, host, user, password, database_name=''):
        self.__host = host
        self.__user = user
        self.__password = password
        self.__database_name = database_name
        self.mydb = mysql.connector.connect(
            host=self.__host,
            user=self.__user,
            passwd=self.__password,
            database=self.__database_name
        )
        self.mycursor = self.mydb.cursor()

    def check_connection(self):
        print(self.mydb)

    def create_database(self, database_name):
        #creates database
        self.mycursor.execute(f'CREATE DATABASE {database_name}')
    """
    Example of use:
    db = AppConnection(host='localhost', user='root', password='AM2304DB')
    db.create_database(database_name='StocksMLA')
    db.create_database(database_name='StocksScanner')
    """

class Database:
    def __init__(self, database_name):
        self.mydb = APP_CONNECTION(host='localhost', user='root', password='AM2304DB', database_name=database_name)
        self.mycursor = self.mydb.mycursor

    def commit(self):
        self.mydb.mydb.commit()
```

Obj 1

Complex OOP

Obj 2

Obj 3j

```python
class MACD:
    def __init__(self, macd_values_list:list):
        self.__macd_values_list = macd_values_list

    def macd_value(self):
        #returns the value of MACD histogram
        return self.__macd_values_list[0]['histogram']

    def convergence_divergence(self):
        return True


```

```python
from SQL import Database
from DATAREQUESTS import Stock
import concurrent.futures
import multiprocessing as mp
import pandas as pd
mp.set_start_method("fork")


class US_STOCKS_TABLE:
    def __init__(self, file_path):
        self.__file_path = file_path
        self.__db = Database(database_name='StocksScreener')

    def __creating_table(self):
        #Creates table in StocksScreener database
        self.__db.mycursor.execute("DROP TABLE US_STOCKS_TABLE")
        self.__db.mycursor.execute("""CREATE TABLE US_STOCKS_TABLE(
                symbolID VARCHAR(255),
                name VARCHAR(255),
                country VARCHAR(255),
                sector VARCHAR(255),
                industry VARCHAR(255),
                av_price FLOAT(3),
                day_range TINYTEXT,
                52_week_range TINYTEXT,
                market_cap BIGINT(255),
                shares_outstanding BIGINT(255),
                public_float BIGINT(255),
                beta VARCHAR(255),
                short_interest FLOAT(3),
                average_volume BIGINT(255))""")
        self.__db.commit()

    def __uploading_data_from_csv(self, number_of_rows:int):
        #Uploades data from csv to table "US_STOCKS_TABLE" in StocksScreener database
        data = pd.read_csv(r"" + f"{self.__file_path}", nrows=number_of_rows)  # nrows=22 skiprows=range(1,4)
        df = pd.DataFrame(data)
        df.fillna("Nan", inplace=True)

        for row in df.itertuples():
            formatedRow = (row.Symbol, row.Name, row.Country, row.Sector, row.Industry, 0, "Nan", "Nan", 0, 0, 0, 0, 0, 0)
            sqlFormula = """INSERT INTO US_STOCKS_TABLE(symbolID, name, country, sector, industry, av_price, day_range,
            52_week_range, market_cap, shares_outstanding, public_float, beta, short_interest, average_volume)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""
            self.__db.mycursor.execute(sqlFormula, formatedRow)
            self.__db.commit()
        print("UPLOADING FROM CSV TO US_STOCKS_TABLE FINISHED")
```

Obj 5

Complex OOP

Simple data model in database

Reading csv file

```python

    def __updating_values_us_stocks_table(self):
        #Updates values in the table "US_STOCKS_TABLE

        def symbols_array():
            #fetches tickers/symbols from US_STOCKS_TABLE

            symbols_array = []
            sqlOrderFormula = "SELECT symbolID FROM StocksScreener.US_STOCKS_TABLE"
            self.__db.mycursor.execute(sqlOrderFormula)
            myResults = self.__db.mycursor.fetchall()
            for value in myResults:
                symbols_array.append(value[0])
            return symbols_array

        def update_data(av_price, day_range, fiftytwo_week_range, market_cap, shares_outstanding,
                    public_float, short_interest, average_volume, beta, symbol, index):
            sqlFormula = f"""UPDATE StocksScreener.US_STOCKS_TABLE
                                    SET av_price = {av_price},
                                    day_range = {day_range},
                                    52_week_range = {fiftytwo_week_range},
                                    market_cap = {market_cap},
                                    shares_outstanding = {shares_outstanding},
                                    public_float = {public_float},
                                    short_interest = {short_interest},
                                    average_volume = {average_volume},
                                    beta = {beta}
                                    WHERE symbolID = '{symbol}';"""
            self.__db.mycursor.execute(sqlFormula)
            self.__db.commit()
            print(f"{index}. {symbol}'s info added to US_STOCKS_TABLE")

        index = 0
        for symbol in symbols_array():
            try:
                stock = Stock(symbol)
                data = stock.request_content()
                av_price = data["Open"]
                day_range = data["Day Range"]
                fiftytwo_week_range = data["52 Week Range"]
                market_cap = data["Market Cap"]
                shares_outstanding = data["Shares Outstanding"]
                public_float = data["Public Float"]
```

```
                short_interest = data["Short Interest"]
                average_volume = data["Average Volume"]
                beta = data['Beta']
                index += 1

                if shares_outstanding > 0 and public_float > 0:
                    number_of_traded_shares = public_float
                    short_interest = (short_interest / number_of_traded_shares) * 100
                    update_data(av_price=av_price, day_range=day_range, fiftytwo_week_range=fiftytwo_week_range,
                                market_cap=market_cap, shares_outstanding=shares_outstanding, public_float=number_of_traded_shares,
                                short_interest=short_interest, average_volume=average_volume, beta=beta, symbol=symbol, index= index)
                else:
                    if shares_outstanding > 0:
                        number_of_traded_shares = shares_outstanding
                        short_interest = (short_interest / number_of_traded_shares) * 100
                        update_data(av_price=av_price, day_range=day_range, fiftytwo_week_range=fiftytwo_week_range,
                                    market_cap=market_cap, shares_outstanding=shares_outstanding,
                                    public_float=number_of_traded_shares,
                                    short_interest=short_interest, average_volume=average_volume, beta=beta,
                                    symbol=symbol, index=index)
                    elif public_float > 0:
                        number_of_traded_shares = public_float
                        short_interest = ( short_interest/ number_of_traded_shares) * 100
                        update_data(av_price=av_price, day_range=day_range, fiftytwo_week_range=fiftytwo_week_range,
                                    market_cap=market_cap, shares_outstanding=shares_outstanding,
                                    public_float=number_of_traded_shares,
                                    short_interest=short_interest, average_volume=average_volume, beta=beta,
                                    symbol=symbol, index=index)
                    else:
                        update_data(av_price=av_price, day_range=day_range, fiftytwo_week_range=fiftytwo_week_range,
                                    market_cap=market_cap, shares_outstanding=0,
                                    public_float=0,
                                    short_interest=short_interest, average_volume=average_volume, beta=beta,
                                    symbol=symbol, index=index)
            except:
```

```
                print(f"Error {symbol} while adding info to US_STOCKS_TABLE")

    def UNI_US_STOCKS_TABLE(self):
        #All methods responsible for creating table, uploading data from csv and updating values
        self.__creating_table()
        self.__uploading_data_from_csv(number_of_rows=6000)
        self.__updating_values_us_stocks_table()


class GAPPERS_TABLE:
    def __init__(self, whereGap={'positive': {'minValue':5.00, 'maxValue':99999}, 'negative':{'minValue':-5.00, 'maxValue':-99999}},
                 wherePublicFloat={'minValue':500000, 'maxValue': 100000000},
                 whereAvPrice={'minValue':1.00, 'maxValue':200.00},
                 whereSector='ALL', whereCountry='ALL'):
        self.__db = Database(database_name='StocksScreener')
        self.__public_float = wherePublicFloat
        self.__avPrice = whereAvPrice
        self.__sector = whereSector
        self.__country = whereCountry
        self.__gap = whereGap

    def __creating_table(self):
        #Creating GAPPERS_TABLE table in database StocksScreener

        self.__db.mycursor.execute("DROP TABLE GAPPERS_TABLE")
        self.__db.mycursor.execute("""CREATE TABLE GAPPERS_TABLE(
            symbolID VARCHAR(255) PRIMARY KEY,
            stock_gap FLOAT(3),
            stock_price FLOAT(2),
            stock_volume BIGINT(255),
            stock_float BIGINT(255),
            stock_dailyRelativeVolume FLOAT(4),
            stock_shortInterest FLOAT(4))""")
        self.__db.commit()

    def __finding_symbols_with_p_f_s_c(self):
        #Finds stocks with specified parameters float, avPrice, country, sector from US_STOCKS_TABLE and returnes dict

        publicFloat = self.__public_float
        avPrice = self.__avPrice
        sector = self.__sector
        country = self.__country

        sectorQuerry = ""
        if sector.upper() != "ALL":
            sectorQuerry = f"AND sector='{sector.title()}'"

        countryQuerry = ""
```

Obj 6

Complex OOP

```
        if country.upper() != "ALL":
            countryQuerry = f"AND country='{country.title()}'"

        self.__db.mycursor.execute(f"""SELECT symbolID, av_price, public_float, short_interest
                                   FROM StocksScreener.US_STOCKS_TABLE
                                   WHERE public_float BETWEEN {publicFloat['minValue']} AND {publicFloat['maxValue']}
                                   {sectorQuerry}
                                   {countryQuerry}
                                   AND av_price BETWEEN {avPrice['minValue']} AND {avPrice['maxValue']};""")
        results = self.__db.mycursor.fetchall()

        def symbolsArray():
            #Returns symbols of selected stocks
            symbolsArray = []
            for result in results:
                symbolsArray.append(result[0])
            return symbolsArray

        return {'r':results, 's':symbolsArray()}

    def __finding_symbols_with_gap(self):
        #finds gappers within the selected list of stocks addressed with method "__finding_symbols_with_p_f_s_c" and returning list

        gappersArray = []
        selectedStocks = self.__finding_symbols_with_p_f_s_c()['r']
        pos_min = self.__gap['positive']['minValue']
        pos_max = self.__gap['positive']['maxValue']
        neg_min = self.__gap['negative']['minValue']
        neg_max = self.__gap['negative']['maxValue']

        def finding_gappers(symbolsData):
            symbol = symbolsData[0]
            stock = Stock(tickerName=symbol)
            gpv_data = stock.request_gap_price_volumeDay()
            gap = gpv_data['gap']
            if (pos_min <= gap and gap <= pos_max) or (gap <= neg_min and gap >= neg_max):
                """tuple = (ticker, gap, price, volume_day, float, rv_data, short_interest)"""
                tuple = (symbolsData[0], gpv_data['gap'], gpv_data['price'], gpv_data['volumeDay'], symbolsData[2], 0, symbolsData[3])
                gappersArray.append(tuple)

        with concurrent.futures.ThreadPoolExecutor(max_workers=8) as executor:
            executor.map(finding_gappers, selectedStocks, timeout=10)

        return gappersArray

```

```python
    def __uploading_selected_stocks(self):
        #Uploades stocks which follow the specified parameters into the database 'StocksScreener', into table 'GappersTable'
        gappers_array = self.__finding_symbols_with_gap()
        sqlFormula = """INSERT INTO GAPPERS_TABLE(symbolID, stock_gap, stock_price, stock_volume, stock_float,
                    stock_dailyRelativeVolume, stock_shortInterest)
                    VALUES (%s, %s, %s, %s, %s, %s, %s)"""
        self.__db.mycursor.executemany(sqlFormula, gappers_array)
        self.__db.commit()
        print("UPLOADING AND INSERTION INTO GAPPERS_TABLE FINISHED")
        return gappers_array

    def UNI_GAPPERS_SCANNER(self):
        #method responsible for creating and uploading selected stocks into the table in the database
        self.__creating_table()
        self.__uploading_selected_stocks()

    def GAPPER_SCANNER_RESULTS(self):
        #method responsible for finding news and returning results in the form of list
        self.__db.mycursor.execute(f"""SELECT symbolID, stock_gap, stock_price, stock_volume, stock_float, stock_shortInterest
                                    FROM StocksScreener.GAPPERS_TABLE
                                    ORDER BY stock_gap ASC;""")
        results = self.__db.mycursor.fetchall()

        array_of_news_for_selected_stocks = []
        for result in results:
            print(result)
            stock = Stock(tickerName=result[0])
            news_array = stock.request_news()
            for news in news_array:
                array_of_news_for_selected_stocks.append(news)
                print(news)
        return results, array_of_news_for_selected_stocks



class SCANNER:
    def __init__(self, whereGap={'positive': {'minValue':5.00, 'maxValue':99999}, 'negative':{'minValue':-5.00, 'maxValue':-99999}},
                 wherePublicFloat={'minValue':500000, 'maxValue':100000000},
                 whereAvPrice={'minValue':1.00, 'maxValue':200.00},
                 whereSector='ALL', whereCountry='ALL',
                 whereRSI={'above':70, 'below':30},
                 where_RV_above=10,
                 where_e_vwap=0.5,
                 where_price_volatility_greater=15,
                 where_n_conseq_candles_above=5,
                 where_agg_volume_more_than= 10000,
```

Obj 7

Complex OOP

```
                    where_last_candle_volume_more_than = 1000):
        self.__db = Database(database_name='StocksScreener')
        self.__gap = whereGap
        self.__public_float = wherePublicFloat
        self.__avPrice = whereAvPrice
        self.__sector = whereSector
        self.__country = whereCountry
        self.__rsi = whereRSI
        self.__rv = where_RV_above
        self.__e_vwap = where_e_vwap
        self.__price_volatility = where_price_volatility_greater
        self.__n_conseq_candles = where_n_conseq_candles_above
        self.__agg_volume = where_agg_volume_more_than
        self.__volume = where_last_candle_volume_more_than

    def __creating_table(self):
        #Creates Scanner table in database StocksScreener

        self.__db.mycursor.execute("DROP TABLE SCANNER")
        self.__db.mycursor.execute("""CREATE TABLE SCANNER(
            symbolID VARCHAR(255) PRIMARY KEY,
            stock_gap FLOAT(3),
            stock_price FLOAT(2),
            stock_float BIGINT(255),
            stock_agg_volume BIGINT(255),
            stock_volume5min BIGINT(255),
            stock_rv5min BIGINT(255),
            stock_rsi FLOAT(2),
            stock_priceVolatility FLOAT(2),
            stock_extensionVWAP FLOAT(2),
            stock_num_of_conseq_candles VARCHAR(5),
            stock_shortInterest FLOAT(4))""")
        self.__db.commit()
        print("CREATION OF TABLE SCANNERS FINISHED")

    def __finding_symbols_with_p_f_s_c(self):
        #Finds stocks with specified parameters float, avPrice,
    country, sector from US_STOCKS_TABLE and returning dict

        publicFloat = self.__public_float
        avPrice = self.__avPrice
        sector = self.__sector
        country = self.__country

        sectorQuerry = ""
        if sector.upper() != "ALL":
            sectorQuerry = f"AND sector='{sector.title()}'"

        countryQuerry = ""
        if country.upper() != "ALL":
            countryQuerry = f"AND country='{country.title()}'"

        self.__db.mycursor.execute(f"""SELECT symbolID, av_price,
```

Simple data structure in database

```
public_float, short_interest
                            FROM StocksScreener.US_STOCKS_TABLE
                            WHERE public_float BETWEEN {publicFloat['minValue']} AND {publicFloat['maxValue']}
                            {sectorQuerry}
                            {countryQuerry}
                            AND av_price BETWEEN {avPrice['minValue']} AND {avPrice['maxValue']};""")
        results = self.__db.mycursor.fetchall()

        return {'r':results}

    def __finding_symbols_with_sp(self):
        #Finds stocks with specified parameters within the selected list of stocks addressed with method "__finding_symbols_with_p_f_s_c" and returning list
        array_of_selected_data = []
        selected_stocks = self.__finding_symbols_with_p_f_s_c()['r']

        def finding_stocks_with_specified_req(selected_symbols_array):
            pos_min = self.__gap['positive']['minValue']
            pos_max = self.__gap['positive']['maxValue']
            neg_min = self.__gap['negative']['minValue']
            neg_max = self.__gap['negative']['maxValue']

            symbol = selected_symbols_array[0]
            public_float = selected_symbols_array[2]
            short_interest = selected_symbols_array[3]
            stock = Stock(tickerName=symbol)
            indic_values = stock.request_indic_values()
            gap = indic_values['gap']
            rsi = indic_values['rsi']
            rv = indic_values['rv']

            if ((indic_values['price'] >= self.__avPrice['minValue']) and (indic_values['price'] <= self.__avPrice['maxValue'])):
                if ((pos_min <= gap and gap <= pos_max) or (gap <= neg_min and gap >= neg_max)) and (rsi >= self.__rsi['above'] or rsi <= self.__rsi['below']) and (rv >= self.__rv):
                    if (indic_values['n_conseq_candles'][1] >= self.__n_conseq_candles) and (indic_values['agg_volume'] >= self.__agg_volume) and indic_values['volume'] >= self.__volume:
                        tuple_array = (symbol, indic_values['gap'], indic_values['price'], public_float, indic_values['agg_volume'], indic_values['volume'],
                                       indic_values['rv'], indic_values['rsi'], indic_values['price_volatility'],
                                       indic_values['e_vwap'], f"{indic_values['n_conseq_candles'][0]}, {indic_values['n_conseq_candles'][1]}", short_interest)
                        array_of_selected_data.append(tuple_array)

```

```
        with concurrent.futures.ThreadPoolExecutor(max_workers=8) as executor:
            executor.map(finding_stocks_with_specified_req, selected_stocks, timeout=15)


        return array_of_selected_data

    def __uploading_selected_stocks(self):
        #Uploads stocks which follows the specified parameters into the database 'StocksScreener', into table 'SCANNER'
        selected_stocks = self.__finding_symbols_with_sp()
        sqlFormula = """INSERT INTO SCANNER(symbolID, stock_gap, stock_price,
                    stock_float, stock_agg_volume, stock_volume5min, stock_rv5min,
                    stock_rsi, stock_priceVolatility, stock_extensionVWAP,
                    stock_num_of_conseq_candles, stock_shortInterest)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""
        self.__db.mycursor.executemany(sqlFormula, selected_stocks)
        self.__db.commit()
        print("UPLOADING AND INSERTION INTO SCANNERS FINISHED")
        return selected_stocks

    def UNI_SCANNER(self):
        # method responsible for creation of the table in the database and uploading of the results into the table.
        self.__creating_table()
        self.__uploading_selected_stocks()

    def SATSY_SCANNER_RESULTS(self):
        # method responsible for finding news and returning results in the form of list

        self.__db.mycursor.execute(f"""SELECT symbolID, stock_gap, stock_price,
                    stock_float, stock_agg_volume, stock_volume5min, stock_rv5min,
                    stock_rsi, stock_priceVolatility, stock_extensionVWAP,
                    stock_num_of_conseq_candles, stock_shortInterest
                    FROM StocksScreener.SCANNER
                    ORDER BY stock_rsi ASC, stock_gap ASC;""")
        results = self.__db.mycursor.fetchall()


        array_of_news_for_selected_stocks = []
        for result in results:
            print(result)
            stock = Stock(tickerName=result[0])
```

```
            news_array = stock.request_news()
            for news in news_array:
                array_of_news_for_selected_stocks.append(news)
                print(news)
        return results, array_of_news_for_selected_stocks
```

```python

APIKEY = 'm8FDkJFfaeRZWG8GcgalzaB_deSOEufI'
NASDQ_STOCKS_CSV = '/Users/alexeymakushin/Desktop/nasdaq_screener_1668272573095.csv'
```

```python
import requests
import datetime
from bs4 import BeautifulSoup
from CANDLESCLASSES import CANDLE, CANDLE_AGG
from KEY_FILE import APIKEY

class Stock:
    def __init__(self, tickerName):
        self.tickerName = str(tickerName).upper()
        self.date = datetime.date.today()
        #self.date = '2023-02-17'

    def request_content(self):
        #method that collects data from marketwatch.com website
        ticker = str(self.tickerName).lower()
        stockDict = {'Open': 0, 'Day Range': 0, '52 Week Range': 0, 'Market Cap': 0, 'Shares Outstanding': 0, 'Public Float': 0, 'Beta': 0, 'Rev. per Employee': 0, 'P/E Ratio': 0, 'EPS': 0, 'Yield': 0, 'Dividend': 0, 'Ex-Dividend Date': 0, 'Short Interest': 0, 'Average Volume': 0}
        try:
            url = f"https://www.marketwatch.com/investing/stock/{ticker}"
            htmlContent = requests.get(url)
            soup = BeautifulSoup(htmlContent.text, 'html.parser')
            result = soup.find_all('li', attrs={'class': 'kv__item'})
            for value in result:
                titel = value.find('small').text
                titelValue = value.find('span', attrs={'class': 'primary'}).text
                dict = {titel: titelValue}
                for key, value in dict.items():
                    if key == "Market Cap" or key == "Open" or key == "Shares Outstanding" or key == 'Public Float' or key == "Rev. per Employee" or key == "Short Interest" or key == "% of Float Shorted" or key == "Average Volume":
                        plainValue = value.strip('$')
                        lastChar = plainValue[-1]
                        if lastChar == "K":
                            try:
                                formatedValue = plainValue.strip(lastChar)
                                floatValue = float(formatedValue) * 1000
                                stockDict[key] = int(floatValue)
                            except:
                                print(f"Error when trying to convert str to int in {self.tickerName}")
                                stockDict[key] = 0
                        elif lastChar == "M":
                            try:
                                formatedValue = plainValue.strip(lastChar)
```

Obj 3

Web parsing/ scraping

```
                                floatValue = float(formatedValue) * 1000000
                                stockDict[key] = int(floatValue)
                            except:
                                print(f"Error when trying to convert str to int in {self.tickerName}")
                                stockDict[key] = 0
                        elif lastChar == "B":
                            try:
                                formatedValue = plainValue.strip(lastChar)
                                floatValue = float(formatedValue) * 1000000000
                                stockDict[key] = int(floatValue)
                            except:
                                print(f"Error when trying to convert str to int in {self.tickerName}")
                                stockDict[key] = 0
                        elif lastChar == "T":
                            try:
                                formatedValue = plainValue.strip(lastChar)
                                floatValue = float(formatedValue) * 1000000000000
                                stockDict[key] = int(floatValue)
                            except:
                                print(f"Error when trying to convert str to int in {self.tickerName}")
                                stockDict[key] = 0
                        else:
                            try:
                                floatValue = float(plainValue)
                                stockDict[key] = floatValue
                            except:
                                stockDict[key] = 0
                    else:
                        stockDict[key] = value
            return stockDict
        except:
            return stockDict

    def request_gap_price_volumeDay(self):
        #method that requests data on stocks by calling API
        try:
            url = f"https://api.polygon.io/v2/snapshot/locale/us/markets/stocks/tickers?tickers={self.tickerName}&include_otc=false&apiKey={APIKEY}"
            request = requests.get(url)
            data = request.json()
            gap = float(data["tickers"][0]["todaysChangePerc"])
            price = float(data["tickers"][0]["min"]["c"])
            volumeDay = int(data["tickers"][0]["min"]["av"])
            volumeMin = int(data["tickers"][0]["min"]["v"])
```

Reques
t API

```
            data = {"gap": round(gap, 3), "price": round(price,3), "volumeDay": round(volumeDay,0), "volumeMin": round(volumeMin,0)}
            return data
        except:
            data = {"gap": 0, "price": 0, "volumeDay": 0, "volumeMin": 0}
            return data

    def request_indic_values(self):
        #method that requests data on stocks by calling API
        def request_aggregate_data(time_span=5):
            #requests aggregatre data on stocks
            try:
                url2 = f"https://api.polygon.io/v2/aggs/ticker/{self.tickerName}/range/{time_span}/minute/{self.date}/{self.date}?adjusted=true&sort=desc&limit=1200&apiKey={APIKEY}"
                request = requests.get(url2)
                data = request.json()
                candles_data = data['results']
                return candles_data
            except:
                return []

        def requestMACD(timespan='minute'):
            """Requesting MACD

            timespan - 'minute', 'day'
            :return: {'v' : value, 's' : signal, 'h': hist}
            """
            try:
                url = f"https://api.polygon.io/v1/indicators/macd/{self.tickerName}?timestamp={self.date}&timespan={timespan}&adjusted=true&short_window=12&long_window=26&signal_window=9&series_type=close&order=desc&limit=300&apiKey={APIKEY}"
                request = requests.get(url)
                data = request.json()
                result = data['results']['values']
                print(result)
                return result
            except:
                return [{'timestamp': 0, 'value': 0, 'signal': 0, 'histogram': 0}]

        def requestRSI(timespan='minute', window=30):
            """Requesting RSI
            timespan - 'minute', 'day'
            window = int
            """

            try:
                url = f"https://api.polygon.io/v1/indicators/rsi/{self.tickerName}?timestamp={self.date}&timespan={timespan}&adjusted=true&window={window}&series_type=close&order=desc&limit=1&apiKey={
```

```
APIKEY}"
                request = requests.get(url)
                data = request.json()
                result = data['results']['values'][0]['value']
                return result
            except:
                return -1

        def CANDLE_DATA():
            # creates class CandleAgg
            try:
                array_of_candles = []
                candles_data = request_aggregate_data(time_span=5)
                for candle_data in candles_data:
                    array_of_candles.append(
                        CANDLE(volume=candle_data['v'],
                               vwap=candle_data['vw'],
                               open=candle_data['o'],
                               close=candle_data['c'],
                               high=candle_data['h'],
                               low=candle_data['l'],
                               num_of_trans=candle_data['n']))

                CANDLE_DATA = CANDLE_AGG(array_of_candles=array_of_candles,
                                         rsi=requestRSI(timespan='minute', window=30),
                                         macd_values_list=[{'timestamp': 0, 'value': 0, 'signal': 0, 'histogram': 0}])
                VALUES = CANDLE_DATA.get_values()
                return VALUES
            except:
                return {'price': 0, 'gap':0, 'agg_volume':0, 'volume':0, 'price_volatility': 0, 'e_vwap': 0, 'rsi': 0, 'rv': 0,
                        'n_conseq_candles': ('Nan', 0)}

        return CANDLE_DATA()

    def request_news(self):
        # requests stock news by calling an API
        try:
            news_articles_tuple = []
            i = 0
            url = f"https://api.polygon.io/v2/reference/news?ticker={self.tickerName}&published_utc={self.date}&order=desc&apiKey={APIKEY}"
            request = requests.get(url)
            data = request.json()
            numberOfArticles = int(data["count"])
            for i in range(numberOfArticles):
                title = data["results"][i]["title"]
                author = data["results"][i]["author"]
                date = data["results"][i]["published_utc"]
```

```
                description = data["results"][i]["description"]
                newsTuple = f"{i+1}, {date}, {title}, {description}, {author}"
                news_articles_tuple.append(newsTuple)

            return news_articles_tuple
        except:
            return ['NA']

```

```python
from MACD import MACD

class CANDLE:
    def __init__(self, volume, vwap, open, close, high, low, num_of_trans):
        self.volume = volume
        self.vwap = vwap
        self.open = open
        self.close = close
        self.high = high
        self.low = low
        self.num_of_trans = num_of_trans
        self.colour = self.__colour()

    def __colour(self):
        # creates
        price = self.open - self.close
        colour = ''
        if price < 0:
            colour = 'g'
        else:
            colour= 'r'
        return colour


class CANDLE_AGG:
    def __init__(self, array_of_candles:list, rsi:float, macd_values_list=[]):
        self.__array_of_candles = array_of_candles
        self.__last_candle = array_of_candles[0]
        self.__last_colour = array_of_candles[0].colour
        self.__last_volume = array_of_candles[0].volume
        self.__rsi = rsi
        self.__macd_values_list = macd_values_list
        #last candles open - close price is the gap

    def __get_gap(self):
        # calculates gap from the open price of first candle in the day and the close price of the last candle
        first_candle = self.__array_of_candles[-1]
        gap = ((self.__last_candle.close *100) / first_candle.open ) - 100
        return gap

    def __get_aggregate_volume(self):
        # calculates total volume on stock
        agg_volume = 0
        for candle in self.__array_of_candles:
            agg_volume += candle.volume
        return agg_volume

    def __get_number_of_conseq_candles(self):
        # calculates the number of repeated candles of the same colour
```

Obj 4

Complex OOP

Comple x Maths

```python
        num_of_conseq_candles = 0
        for candle in self.__array_of_candles:
            if candle.colour == self.__last_colour:
                num_of_conseq_candles += 1
            else:
                break
        return (self.__last_colour, num_of_conseq_candles)

    def __get_extesion_of_vwap(self):
        # calculates extension in terms of percentage from the last candle close
        extension = round(abs(((self.__last_candle.close * 100) / self.__last_candle.vwap) - 100), 4)
        return extension

    def __get_relative_volume(self, over_number_of_candles=11):
        # calculates relative volume as an index over average of the last 10 candles
        av = 0
        for i in range(0, over_number_of_candles, +1):
            av += self.__array_of_candles[i].volume
        rv = round((self.__last_volume / (av / over_number_of_candles)),4)
        return rv

    def __get_rsi(self):
        # returns RSI
        return self.__rsi

    def __get_macd(self):
        """References class MACD that analyses strategy of MACD
        and returns values of what it has come up with"""
        pass
        """
        macd = MACD(macd_values_list=self.__macd_values_list)

        return macd.macd_value()
        """

    def __get_price_volatility(self):
        #calculates the volatility of the stock from the min price of the stock that day to the maximum price

        min_value = self.__last_candle.close
        max_value = self.__last_candle.close
        for candle in self.__array_of_candles:
            if candle.close > max_value:
                max_value = candle.close
            if candle.close < min_value:
                min_value = candle.close
        result = round(abs((((max_value * 100)/min_value) - 100)), 4)
        return result

```

```python
    def get_values(self):
        # returns all the values in the form of the dict
        price = self.__last_candle.close
        gap = self.__get_gap()
        agg_volume = self.__get_aggregate_volume()
        price_volatility = self.__get_price_volatility()
        extention_of_vwap = self.__get_extesion_of_vwap()
        rsi = self.__rsi
        rv = self.__get_relative_volume(over_number_of_candles=11)
        num_conseq_candles = self.__get_number_of_conseq_candles()
        volume = self.__last_volume
        macd = self.__get_macd()

        return {'price': round(price,2), 'gap':round(gap,2), 'agg_volume': agg_volume, 'volume':volume,'price_volatility': round(price_volatility,4),
                'e_vwap': round(extention_of_vwap,4), 'rsi':round(rsi,2), 'rv':round(rv,2),'n_conseq_candles': num_conseq_candles}


```

TESTING SECTION

In this section, I will test my main sub-objectives and provide evidence through video recordings and screenshots. I will not be testing against Obj 1, 2, 3, and 4 as they are essential for the program to work and successfully achieving my chosen objectives would mean that these objectives have also been achieved. Obj 1 and 2 ensure the database connection and communication, which is necessary for the program to function. Obj 3 handles data scraping and requests, and the successful functioning of any of the scanners indicates that Obj 3 has been achieved. Obj 4 is responsible for calculations for the SATSY scanner, and while most parameters are successfully calculated, the program fails to achieve the calculation of the MACD indicator, thus Obj 4 is only partially met.

Objective		Test Process	Expected Result	Outcome
Obj. 8	Testing the GUI: 1)Could display be resized? 2)Could user choose scanning options? 3)Could user change appearance mode? 4)Could user reset the fields to default values when pushes reset button? 5)Does the popup pops when user enters data in the wrong format? 6)Does the program starts when the user pushes button enter? 7)Could the user see the results of scanning?	In order to test the GUI, my plan is to: 1)Test the program window's ability to resize. 2)Experiment with different scanning options to observe any corresponding changes in the view. 3)Change the appearance mode to see if it affects the appearance. 4)Input data into specific fields, and then press reset to see if it sets values to default. 5)Attempt to input data in incorrect formats to test whether the appropriate pop-up message is displayed. 6)Press the enter button to initiate scanning, and verify that the program begins the scanning process. 7)Wait for the scanning process to complete and check if the results have been uploaded to the result view, as well as whether news corresponding to those results have been uploaded to the news view	1) Window stretches and it does not affect the view of the fields. 2) When scanner type selected the view changes to the view of the selected scanner . 3)Appearance of the view changes. 4)When reset button is pushed the parameters return to default values 5)If data is entered in an incorrect format, a pop-up message appears asking the user to re-enter the data when the enter button is pressed. 6)The program should commence scanning when the user presses the enter button. 7)The results would be displayed in the test fields.	1) Successful 2) Successful 3) Successful 4) Successful 5) Successful 6) Successful 7) Successful – see video on testing of Obj 8

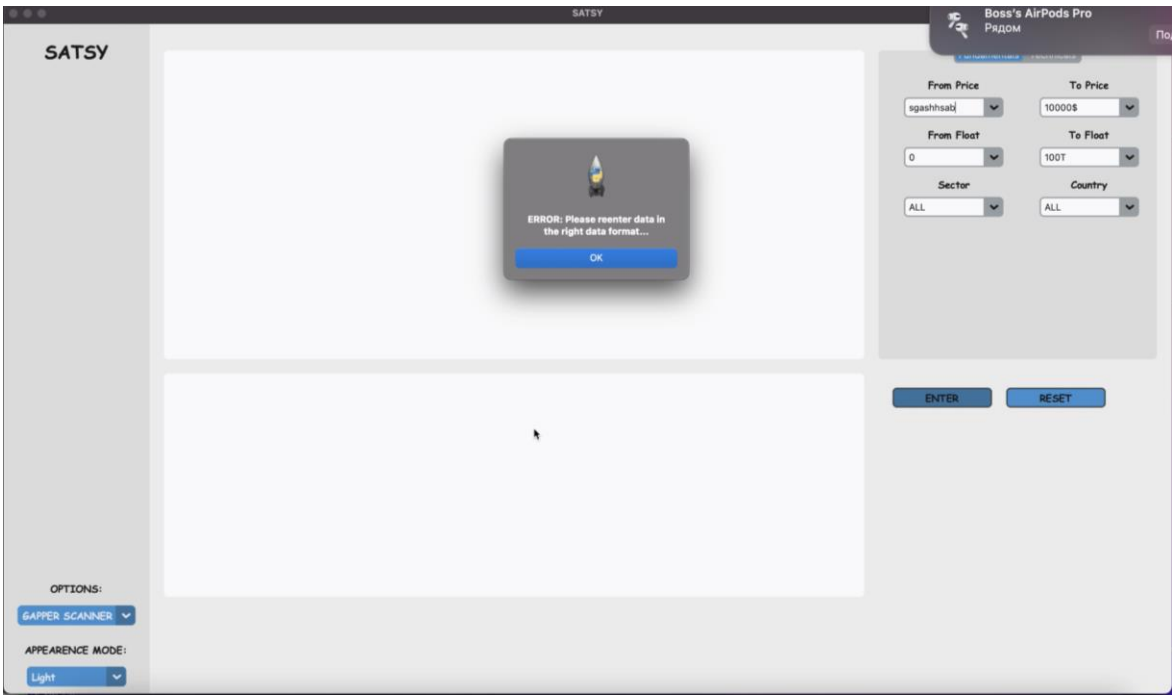
Obj 9.a	Verify if the program can generate a database comprising stocks traded on the market by importing data from a CSV file, and if it can identify the top-performing stocks over a specified timeframe and store them in a separate table within the same database.	To test Obj 9.a I will: 1)Input the file path to the csv file in the directory. 2) I will call the method UNI_DATA_TO_US_STOCKS on ML_TABLES 3) I will call the method UNI_DATA_TO_MLASTOCKS on ML_TABLES 4) Verify that the "US_Stocks" and "MLAStocks" tables have been populated with the appropriate data.	I would expect to see the table US_Stocks in the StocksMLA database populated with data from the csv file. I would expect to see MLAStocks in the StocksMLA database populated with 'gainers' stocks, that follow parameters specified.	Successful – see video on testing of Obj 9.a
Obj 9.b and 9.c	Verify if the program is capable of generating an ML model and using it to predict potential price increases for stocks.	To test Obj 9.b and 9.c I will, start program, and instruct it to use Gapper Scanner. To test whether the program can predict stock prices I will inspect the last parameter in the result text field view.	I would expect to see the predicted increase in price for every stock in results text field as a last parameter.	Successful – see video on testing of Obj 9.b and 9.c

Obj 7	Verify if the program can employ the Satsy Scanner to filter stocks in the market and display the outcomes in the text fields.	In order to test Objective 7, my plan is to select the Satsy Scanner and input certain parameters in both the fundamental and technical windows, and start the scanning process. To verify whether the scanner is able to produce the outcomes I will inspect the two text fields to see if the results of scanning are shown in the text fields and the results are the same as in the database.	I expect to see the scan results in the result text field of the Satsy Scanner view, and I also expect to see the relevant news quotes displayed in the news text field.	Successful – see video on testing of Obj 7
Obj 6	Verify if the program can use the Gapper Scanner to filter stocks in the market and display the results in the text fields.	In order to test Objective 6, my approach is to select the Gapper Scanner and input certain parameters in both the fundamental and technical windows. Next, I will verify whether the scanner works by checking the result text field and news text field for the results of scanning, and will compare it with database records.	I would expect to see, the results of scanning in the result text field of the Gapper Scanner view, and would expect corresponding news to appear in the news text field.	Successful – see video on testing of Obj 6

Obj 5	Verify if the program can create a table called "US_STOCKS TABLE" in the database that stores stock parameters obtained from both a CSV file sourced from Nasdaq.com and through web scraping from marketwatch.com.	To test Objective 5, I will input the file path to the CSV file in the directory and then call the UNI_US_STOCKS_TABLE method on the US_STOCKS_TABLE class. To verify whether the program works, I would inspect table in the database, to see whether it became populated with data.	Initially, I would expect to observe the creation of the "US_STOCKS_TABLE" table in the StockScreener database and the uploading of data from the CSV file into the database. Secondly, I anticipate seeing the gathering of data from marketwatch.com and its subsequent uploading to the database. Eventually, I would expect to see database to be fully populated with data.	Successful – see video on testing of Obj 5

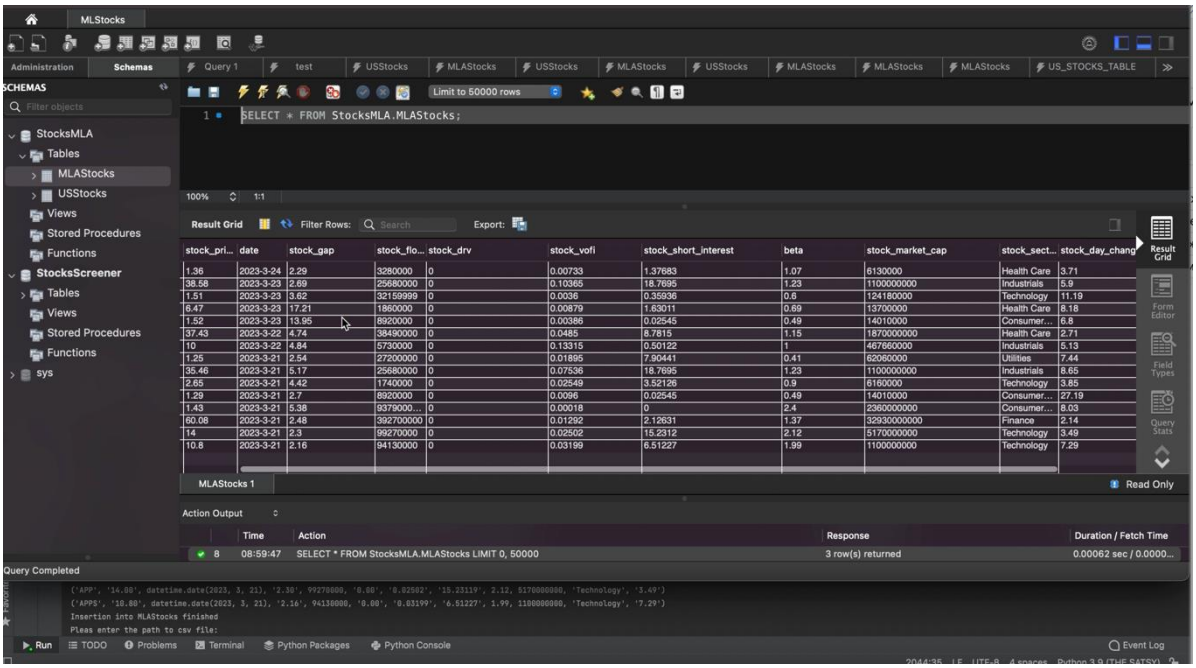
Video of testing and screenshots of some of the testing:

Testing for Obj 8:



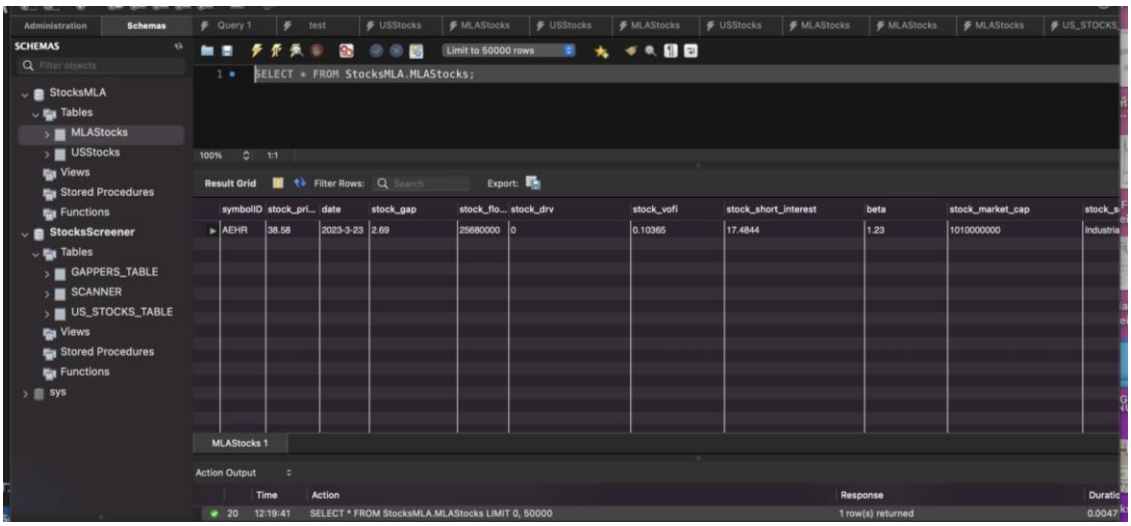
https://www.dropbox.com/s/4tpp5d8mz40jeve/Testing%20Obj%208%20.mov?dl=0

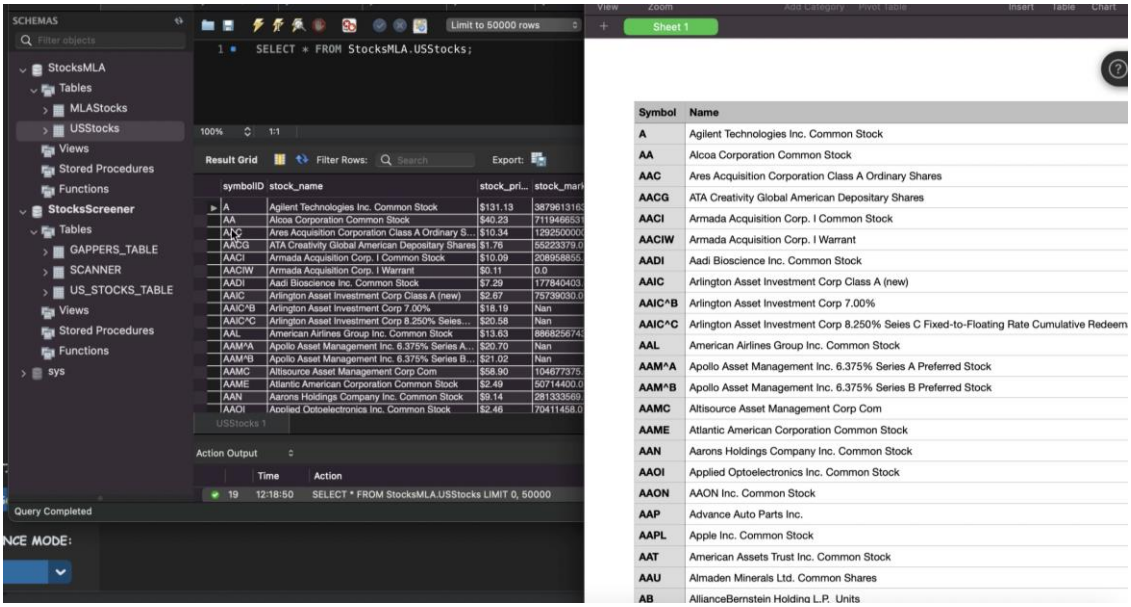
Testing for Obj 9 a:



https://www.dropbox.com/s/g64pjj168j8y5al/Test%20of%20Obj%209a.mov?dl=0

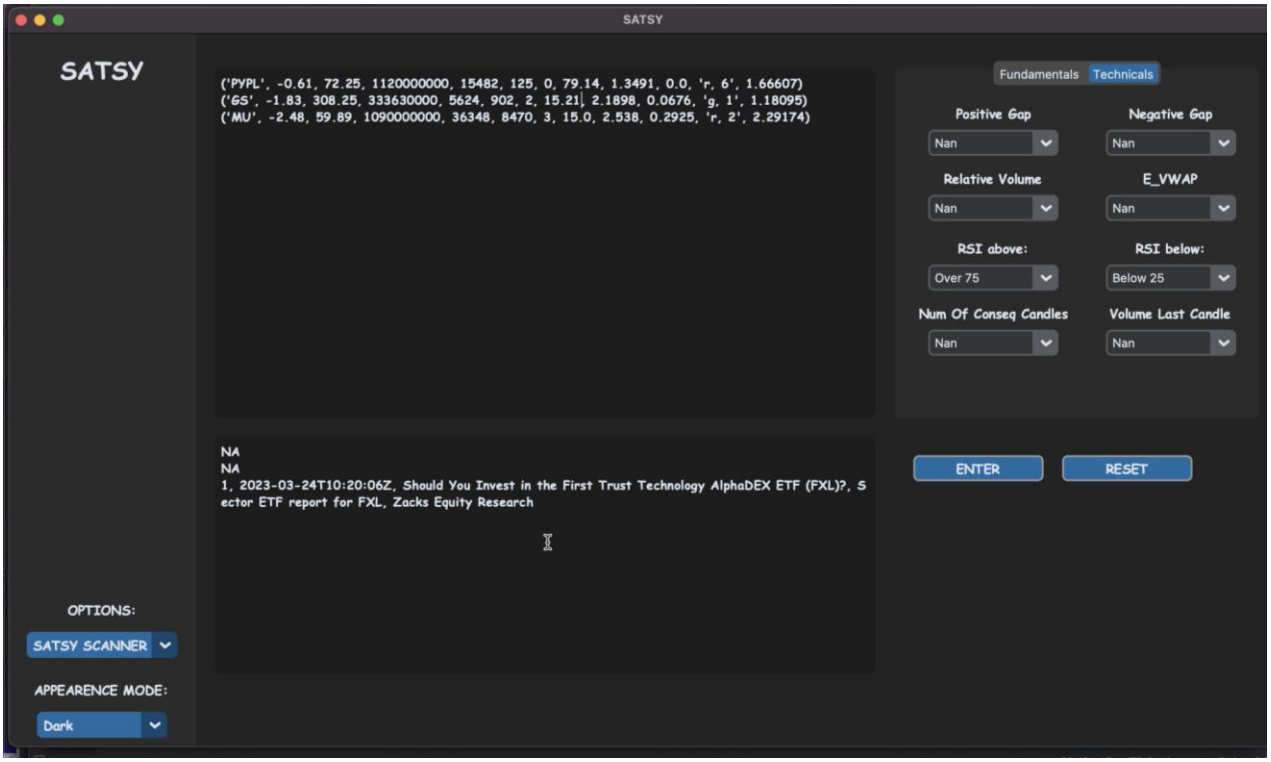
Testing for Obj 9 b,c:





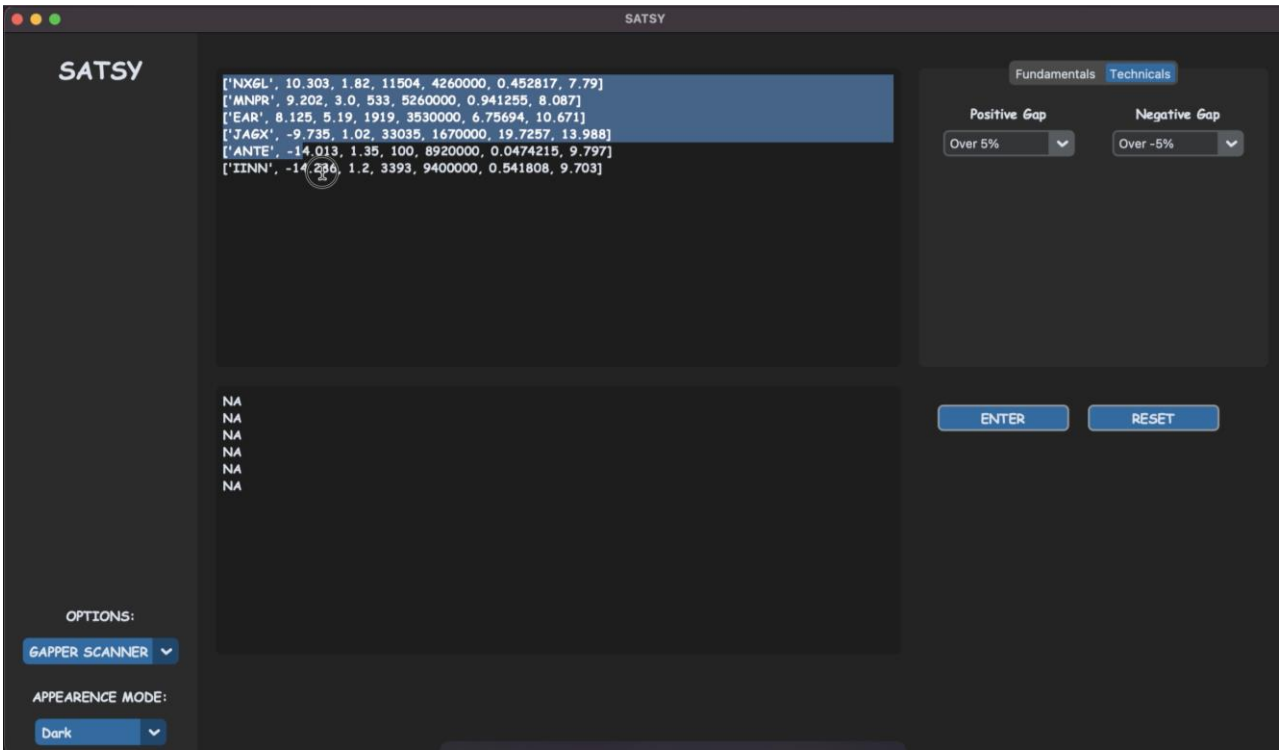
https://www.dropbox.com/s/pdp2dzqjawzdbi8/Test%20of%20Obj%209b%2C%209c.mov?dl=0

Testing for Obj 7:



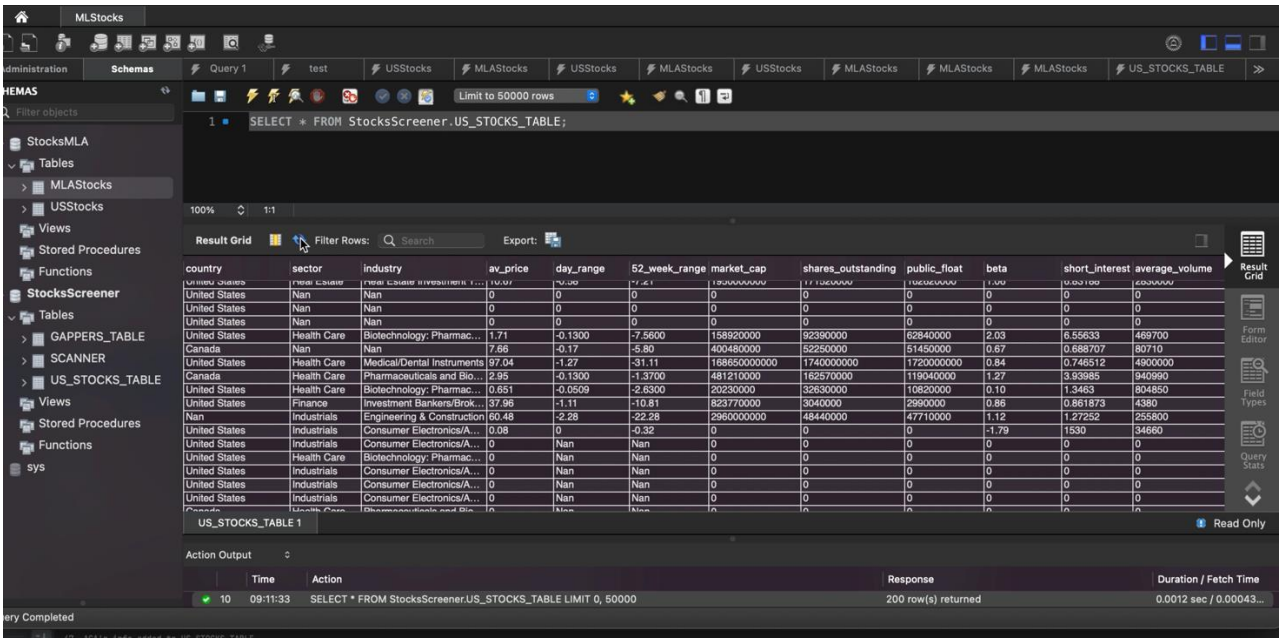
https://www.dropbox.com/s/20vtv69schy7a0h/Test%20of%20Obj%207.mov?dl=0

Testing for Obj 6:



https://www.dropbox.com/s/b62hdsg9j0yadel/Test%20of%20Obj%206.mov?dl=0

Testing for Obj 5:



https://www.dropbox.com/s/b62hdsg9j0yadel/Test%20of%20Obj%206.mov?dl=0

Full system video overview:

https://www.dropbox.com/s/z4s9q1wi0i3n188/CS%20COURSEWORK%20Overall%20view.mov?dl=0

EVALUATION SECTION

Based on my assessment, the project has been well developed and the conducted tests have demonstrated that the main components of the system function as intended. Overall, the program effectively fulfills its primary objective of serving as a scanner for day traders.

Evaluating the Success of the Solution

I would like to give deeper evaluation of whether my objectives were successfully met.

Objective	Reflection
1. The program needs to establish a connection between Python and a MySQL database. a. Create a class called "AppConnection" that will handle the database connection. b. The class method should be able to print a message to the console confirming that the connection has been established. c. The class method should also be able to create a database.	Successfulness: The stated objective has been achieved in its entirety. The system overview video confirms that the program has effectively established a connection between itself and the MySQL database, and has the capability to create a database. Potential Improvements: Based on the performance, it appears that no further enhancements are required as the Python code has fully met the objectives specified.
2. The program needs to be able to interact with the database. a. Create a class called "Database" that will connect to a specific database on the MySQL server. b. The class should have a "commit" method that will save any changes made to the database. c. The class should also have an attribute that allows changes to be made to the database.	Successfulness: The objective has been successfully accomplished, as the program has demonstrated its ability to create tables within the database and execute queries on them. Moreover, the program has proven its proficiency in saving all the necessary modifications made to the database,

	while also enabling other manipulations to be performed on the tables in the database. As seen from the video overview of the system. Potential Improvements: Based on the performance, it appears that no further enhancements are required as the Python code has fully met the objectives specified.
3. The program needs to be able to gather data on stocks. g. The program should be able to collect data from the marketwatch.com website. (from the “KEY DATA ” table) vi. The data should be stored in a list with table names serving as keys in a dictionary. vii. The data should be reformatted, with numbers like "T" for trillion, "B" for billion, "M" for million, and "K" for thousand replaced by the appropriate numerical values. viii. The data in these cells should be reformatted into acceptable types such as integers, floats and strings. ix. Exception handling methods should be implemented at both the formatting and request levels. (Exception handling) a. If a request fails, the data should be returned as a dictionary with 0 values for all parameters. b. If formatting fails, the particular parameter should be returned as 0. x. The method should return a dictionary in the form of {'Open': value, 'Day Range': value, '52 Week Range': value, 'Market Cap': value, 'Shares Outstanding': value, 'Public Float': value, 'Beta': value, 'Rev. per Employee': value, 'P/E Ratio': value, 'EPS': value, 'Yield': value, 'Dividend': value, 'Ex-Dividend Date': value, 'Short Interest': value, 'Average Volume': value }.	Successfulness: The objective has been successfully accomplished, as the program has the capability to extract data from the marketwatch.com website, and request news, aggregate data and data on stocks at a intraday level by sending requests to an external API called polygon.io. Additionally, the program handles errors well due to successfully implemented exception handling mechanism which ensures that the program does not crash even if it is unable to gather data for the stock. Potential Improvements: Despite the successful implementation of the program, there is still room for improvement in terms of API requests, such as limiting the size of the received JSON. This may lead to faster requests, potentially enhancing the overall performance of the program. Additionally, there is a possibility to improve the format of the returned values of the methiods, specifically by implementing Enums for the news instead of using a list as a return value. Such a modification could enhance the structure of the

h. The program should be able to request aggregate data on stocks for the Gapper scanner by sending requests to external API. v. The method should be able to interpret values that come in the form of JSON. vi. The method should be able to format data into the correct data types, such as integers and floats. vii. The method should have an exception handling method so that if it fails to receive data or format data, the program won't crush. viii. The method should return values in the form of a dictionary {"gap": value, "price": value, "volumeDay": value, "volumeMin": value}. i. d. The program should have the ability to request data on stocks at the 'daily level' for the SATSY scanner through API requests. iv. The method should be capable of requesting data on the stock within the specified day range, and specified timespan. v. The method should have an exception-handling mechanism. vi. The method should return data in the form of a simplified array, without any meta-data. j. The program should be capable of requesting the MACD indicator through API requests. iv. The method should be capable of returning the value of the MACD. v. The value of the MACD should be formatted into the correct data type. vi. The value of the MACD should be returned in the form of a dictionary. {'timestamp': value, 'value': value, 'signal': value, 'histogram': value} k. The program must be able to request the RSI value, through API requests. a. The method should be capable of requesting the RSI value. b. The method must include an exception handling mechanism. If the method fails, the RSI value returned should be -1. Otherwise, the correct value should be returned.	program and make it easier for others to comprehend.

c. The method should return value as a float number l. The program should be able to request news. a. The method should be able to request news by sending API requests. b. The method should have an exception handling mechanism, so that if exception is returned, the values are returned as a list ['NA']. c. The method should be able to return news in the form of a list.	
4. The program should be able to process data, and calculate the stock's price, gap, aggregate volume, volume, price volatility, extension of vwap, relative volume, RSI, and the number of consecutive candles. a. The program should have a method that converts each item in the array into an object of the Candle class. i. The Candle class should have a following attributes: volume, open, close, high, low, number_of_transactions, vwap. ii. The Candle class should have a method that determines the colour of the candle, and stores the value as an attribute. iii. The program should have a method that appends Candle objects to a list of Candle objects. b. The program should have a Candle_AGG class that is responsible for returning values for the stock, such as price, gap, aggregate volume, volume, price volatility, extension of vwap, relative volume, RSI, and the number of consecutive candles. i. The Candle_AGG class should be created. ii. The class should have a method that returns the closing price of the last candle. iii. The class should have a method that returns the number of consecutive candles as a tuple.	Successfulness: Generally speaking, the program has been successful in achieving these objectives. It is capable of calculating essential indicators such as RSI, price, gap, aggregate volume, volume, price volatility, extension of vwap, relative volume, RSI, and the number of consecutive candles. These values are conveniently returned in a dictionary format, making them easy to utilize within the program. However, the program falls short of meeting objective 4.b.x, which involves computing the MACD indicator. Potential Improvements: To enhance the program, it could be modified to request the MACD indicator and analyze its values to provide users with insights into the trend of the stock, whether the stock is in the uptrend or downtrend.

iv. The method inside the class should be able to calculate the relative volume over a period of 11 candles. v. The method inside the class should be able to return the gap of the stock. vi. The method inside the class should be able to return the value of the total volume experienced by the stock. vii. The method inside the class should be able to return the extension of vwap viii. The method inside the class should be able to return the value of price volatility. ix. The method inside the class should be able to return the value RSI. x. The method inside the class should be able to return the value of macd. xi. The public method should be able to return all the values price, gap, aggregate volume, volume, price volatility, extension of vwap, relative volume, RSI and number of consecutive candles in the form of dictionary.	
5. The program should be capable of generating a database table that stores parameters for each stock obtained from both a CSV file sourced from Nasdaq.com and through web scraping from marketwatch.com. a. The class US_Stock_Table should be implemented to handle the creation of the US_Stock_Table, created upon file path parameter. b. The class should have an attribute that utilizes the Database class to establish a connection with the database. c. The class method should be able to generate the table inside the database. d. The class method should be able to upload data from the CSV file into the database table. e. The class method should be able to leverage the Stock class's functionality, specifically the method that scrapes data from marketwatch.com, to obtain values for each stock in the table.	Successfulness: The program has been successful in achieving these objectives, as the program is capable of creating the table in the database, and populating it with the essential parameters from csv file and by sending requests to marketwatch.com. Potential Improvements: The collected values from the marketwatch.com very quickly become outdated, particularly the average stock price. Therefore, to keep the program's values up to date, the stock price could be updated every hour by requesting information from an external API.

	f. The class method should be able to compare public float and shares outstanding values, and the return the value of one of the parameters. g. The class method should be created to streamline the process of table creation, value updates, and insertion by placing them all under a single method	
6.	The program should be able to find gappers on the market. a. The class should be created Gappers_Table, which would be responsible for creation of the gappers, selection and insertion into the database. b. The method inside the class should be able to create table in the database. c. The method inside the class should be able to select stocks with float and price in the specified range, and select stocks with correct sector and country specified in the constructor. i. The method should be able to access the US_Stocks table and querry results with specified parameters. d. The method should be able to take the results of the selected stocks and find among selected stocks gappers that are within the specified gap range. e. The results of the method should than be uploaded to the Gappers table f. The method should be created that would be responsible, for creating, selecting, and uploading data into the table by calling one public method g. The method should be created that would allow to access the results in the database, and request news on these results. i. The method should be created that would query the results in the database, and order by the gap ii. The method should be created to find news on every gapper in the gappers table iii. The method than should return tuple, with gappers results and news on these gappers.	Successfulness: The program has achieved the stated objectives, as evident from the testing video and screenshots. Specifically, the Gappers Scanner can effectively store data in the database, identify market gappers that satisfy specific end-user parameters, and provide the results in the form of a list that can be utilized later in the program. Potential Improvements: Considering the program's performance, it appears that the Gappers Scanner has successfully accomplished all the tasks required for its intended purpose. Therefore, there may be no further improvements necessary.

7. The program should be able to screen stocks on the market (reversal scanning and momentum scanning) a. The class Scanner should be created and the constructor of this class should take parameters like gap, price, float, sector, country, RSI, relative volume, extension of vwap, price volatility, number of consecutive candles, aggregate volume and volume of the last candle. b. The method inside the class should be able to create table in the database. c. The method inside the class should be able to select stocks with float and price in the specified range, and select stocks with correct sector and country specified in the constructor. i. The method should be able to access the US_Stocks table and query results with specified parameters. d. The method should be able to take the results of the selected stocks and find among selected stocks, stocks that are within the parameters specified. i. The method should be able to request data on stocks on the daily level, the RSI of the stock and MACD. ii. The method should be able to compare the specified parameters in the constructor with the parameters of every stock, and save the stock that follows all the parameters. e. Saved stocks which follow specified parameters should than be inserted into the table. f. The method should be created which would be responsible for creating selecting and uploading of stocks, basically linking all the processes above into one public method. g. The method should be created that would allow to access the results in the database, and request news on these results. i. The method should be created that would query the results in the database, and order by the stock RSI and gap. ii. The method should be created to find news on every selected stock in the scanner table.	Successfulness: The program has generally successfully achieved its objectives as it allows to screen stocks on the market by various parameters such as gap, price, float, sector, country, RSI, relative volume, extension of vwap, price volatility, number of consecutive candles, aggregate volume, and volume of the last candle. Additionally, the program can create a table and store data in the database, as demonstrated by the testing video and screenshots. However, the program cannot screen stocks based on the MACD indicator, which means it fails to meet objective 7.d. Potential Improvements To improve the Satsy Scanner, the program could be enhanced to include screening by the MACD indicator. Such an improvement could potentially enhance the program's overall functionality and usefulness for end-users.

	iii. The method than should return tuple, with scanner results and news on this selected stock.	
8.	The program should have a convenient, beautiful GUI. a. The GUI should be resizable. b. The GUI should have options to choose from. Gappers Scanner or SATSY Scanner. c. The user should be able to specify all the parameters required to create Gappers Table. i. Exception handling should check if data is entered correctly, if not popup will be displayed on the screen, asking to renter the data. d. The user should be able to specify all the parameters required create Scanner. i. Exception handling should check if data is entered correctly, if not popup will be displayed on the screen, asking to renter the data. e. The user should be able to see the results of the Gapper Scanner, and SATSY Scanner in two text boxes, one with news for the selected stocks and another with selected stocks and their parameters. f. The user should be able to reset the fields. g. The user should be able to start the program when he pushes button enter. h. The program should be able to change appearance mode.	Although the program has mostly met these objectives, such as the creation of a user-friendly GUI, an exception-handling mechanism to prevent crashes, ability to provide the results in a structured way sorted by various parameters, there are still areas for improvement. In particular, the text field presents information in a non-convenient way, the lack of specified field names may make it difficult for users to understand the results of the scanner. Potential Improvements: To address this issue, the program's GUI could be enhanced in a few ways. For instance, implementing colour coding could make it easier for users to visually grasp the data. Additionally, presenting the data in a table format could increase convenience for users.
9.	The program should be able to predict potential increase in the stock's price a. The program should be able to create database with stocks traded on the market, and able to find data on stocks that have gained the most over the specified period of days. ii. The class should be created ML_Table iii. The method in the class should be able to create a table in the database iv. The method in the class should be able to get the array of working days in the specified timespan.	Although the program has mostly met these objectives, such as capability of gathering data on gainer stocks, uploading them into the database, and creating model from the gainer stocks, the program still in most cases fails to predict potential increase in stock prices. Therefore, this objective was only partially met. Potential Improvements:

v. The method in the class should be able to upload data from the csv file into the database vi. The method in the class should be able to find stock gainers for every working day that lies within the timespan. vii. The method in the class should be able to find any extra data on the gainer stock, and upload gainers into the database viii. The method in the class should be able to upload data into the table in the database. d. The program should be able to create a ML model based on the stock gainers stored in the database. i. The function should be able to query data from the table in the database. ii. The program should be able to train the model. iii. The program should be able to return the model. e. The program should have a function predict that would use the model to predict increase in the prices of stocks. i. The value of the predicted price would be returned from the function	To improve the prediction method and the overall ML model, several potential improvements can be considered. For instance, alternative regression models such as SVR or polynomial regression algorithms could be implemented. Additionally, deep learning approaches could be explored to better predict stock prices. Moreover, techniques for enhancing the quality of the collected data, such as eliminating variables with high correlation that may adversely affect the model's performance, could be adopted.

User Feedback

Question	Interviewee response
Are you satisfied with the program's graphical user interface (GUI)? If so, what aspects of the GUI do you find appealing and what aspects do you find unappealing?	In general, I think that the GUI you have created is very user-friendly and easy to navigate. Its design is also impressive. One of the things that I appreciate about the GUI is that it has a simple layout, with only essential input fields and buttons, avoiding anything that might confuse or overwhelm users. Additionally, I also like that the scanner is customizable, and that the input fields are presented as dropdown lists, which makes it easy to specify parameters and also pretty fast. However, the program has a major drawback in terms of how it presents data in text fields. The data is displayed as rows of unstructured information, making it challenging for me to interpret the values and comprehend the meaning of each one.
Have you used the Gapper Scanner? If so, what aspects did you find appealing and what aspects did you find unappealing?	After using the gapper scanner for a while to locate gappers in the market, I can confidently say that I am satisfied with the overall performance that you have achieved, in particular the speed of scanning and the quality of the data collected. I have also liked that your system provides with an estimated increase of the stock price, which I used as a confidence level index, when deciding on decision, which was useful and beneficial to me as a trader. Nevertheless, the ML algorithm's precision is still inadequate, which at times influences the trading outcomes.
Have you used the SATSY Scanner? If so, what aspects did you find appealing and what aspects did you find unappealing?	In general, I am very satisfied with the Satsy scanner because it effectively serves its purpose for both momentum and reversal trading. The results are generated quickly, which is a great advantage, and the data collected and presented to me was extremely useful, in making trading decisions. However, after using it for both trading strategies, I believe there are a few additional values that could be added to make the system even better. These values include the MACD indicator, as well as the EMA 20 and EMA 50, to get a better understanding of the overall trend of the stock. But overall, I relly liked it.

Analysis of feedback

Based on the feedback received, it can be inferred that the system has successfully achieved its primary objective of being a relatively fast stock screener. It successfully performs all the necessary scanning like gapper scanning, reversal scanning and momentum scanning, as mentioned by the interviewee. However, given that he generally liked the system that I have created, he still suggested the areas for improvement, particularly in making the system more convenient and user-friendly. Specifically, the data displayed in the text fields could be formatted differently, with different colors and presented in a table format, to make the results more visually appealing and easier to comprehend. Furthermore, adding a MACD indicator could make the system even more suitable for day traders by providing an understanding of the stock trend. Overall, I agree with the third party's comments, especially regarding the importance of improving data representation and introducing the MACD technical indicator. Implementing these changes could take my program to the next level, and I believe it is a very feasible objective.

Conclusion

In my opinion, my program effectively caters to the requirements of day traders who seek a customizable, affordable, and fast day trading scanner that prioritizes technical analysis at an intraday level. Although my program meets most of its objectives, I acknowledge that there is scope for improvement, especially in providing a more accurate ML algorithm for predicting potential stock price increases, creating a more visually appealing GUI, and enabling users to scan stocks using the MACD indicator. If I were to continue with the project, I would incorporate additional trading indicators, improve data request speed, develop algorithms for more frequent information updates, and prioritize creating an even more personalized and user-friendly interface. Overall, I consider my project to be a success, but there is always room for further improvement.