Implementing and Testing a Strategy to Pick Stocks

Author: Elliot Cooke
Student Number: 201606831
Supervisors: Rasmus Ibsen-Jensen, Patrick Totzke

The University of Liverpool

Abstract

This project captures the implementation and testing of two pre-defined stock selecting strategies; the first strategy (epsilon strategy), a unique approach posing as the focus of this research project, and the second strategy (quantitative momentum strategy a.k.a QMS), a commonly used strategy for picking stocks. A python program was developed to apply and test these strategies on adjusted closing price data of the top 30 FSTE-100 stocks for the past 15 years from the point of program execution.

The stock data was imported from yahoo finance, cleaned of 'NaN' values and manipulated into a data frame using monthly intervals. The percentage changes of adjusted close prices minus the target value (average percentage change for all stocks over all intervals) from the start to end interval were computed and stored in a separate data frame. For the epsilon formula, a function was created that when applied to this data frame, would use the strategy's formula to return the number of times each stock was picked over the 15 years duration. For the QMS, a separate function was made comprising of this strategy's formula, and it would return the 'momentum', a number ranging from 0 to 1, of each stock when applied on the first data frame. The top 10 stocks based on highest number of investments and highest momentum for both strategies respectively were applied to a 'backtest' function, visualising the percentage return on these stocks after investing £1m pounds into each strategy equally across the top 10 stocks. This enabled statistical comparison of each strategy to highlight which strategy picked the best returning stocks and gave the greatest investment returns.

It was found that the QMS gave the greatest returns on average, with most stocks giving positive returns, whilst the epsilon strategy produced more mixed results where a few stocks returned money, but the majority caused a loss of money (no positive returns). This was expected as whilst the QMS is a known strategy to work sufficiently in stock selection, the epsilon strategy was a new and unique strategy in development that may have needed a longer time duration to compute on the data. It was noted that a smaller epsilon/ target value combination was required to produce the most reliable results whilst being able to invest in any stocks at all. As a key recommendation, this epsilon strategy requires further testing to reveal its true potential in stock selection.

Student Declaration

SIGNATURE ELLIOT WILLIAM COOKE DATE August 20, 2014

Acknowledgements

As part of this project's development and completion, gratitude is given to the two project supervisors, Rasmus and Patrick. Rasmus has provided the epsilon strategy as the main strategy to implement and has assisted in the implementation of this strategy to a suitable stock index. Both Rasmus and Patrick have supervised the assessment of the project specification and final presentation, provided informative feedback to help regulate project development.

Table of Contents

Introduction

The power of artificial intelligence (AI) is greater than ever before and continues to serve as a key tool in an extensive variety of work fields. The finance sector is no exception, with data-driven decisions being witnessed as a smart, effective, and efficient option when it comes to stock investment and optimizing returns. Automating manual analysis of stocks with AI can diminish the risk of human error, constant uncertainty and even negative emotions that ultimately affect investment decisions. There is no doubt that AI serves as a valuable and exciting tool for this sector.

The majority of AI exploitation within this area has tended towards stock price prediction, using machine learning models to predict the past, current and future selling price of stocks to make investment decisions. However, the focus and natural progression of AI in stock investment has shifted towards predefined strategies that follow algorithms, rather than models, applied on stock data to highlight the best stocks to pick for investment. Examples of such strategies include the *moving average (MA) strategy*, which smooths out stock price data by creating a constantly updated average price. This average is taken over a specific time period (such as a week or a month) and provides insight as to how that stock behaves within these intervals over a certain time duration (e.g 10 years), and therefore, an indication as to how investable that stock is.

Within this project, two strategies have been implemented, tested and compared with each other to discover the power of the AI-related stock investment industry and the effectiveness of a new, unique strategy (the epsilon strategy) against a commonly used and well-reviewed approach (the Quantitative Momentum strategy). Whilst it was desired to witness the researched effectiveness of the Quantitative Momentum strategy (QMS), the focus lay within the epsilon strategy as this was untested prior to this project, providing strong interest in how this strategy would react on real-life stock data.

The contribution that this project makes to the knowledge of others is, firstly, spreading the importance of AI in stock selection. The project aims to show how using AI to analyse and pick stocks is a strategical, structured, efficient, and renewable approach. Secondly, the project highlights how effective programming is in not only implementing strategies, but also testing and visualising the results in a time-efficient manner. Finally, this project specifically captures the new and exciting epsilon strategy in implementation and action on real-life stock data, providing a report on its highlights and pitfalls.

Aims and Objectives

The aims and objectives of this project are listed below as practical and supplementary aims:

Practical aims:

- Implement both the epsilon and QMS pre-defined strategies using Python code
- Test both strategies via 'back testing' to make evaluations based on return percentages
- Evaluate both sets of results to conclude the effectiveness of the epsilon strategy against the commonly used and well-reviewed QMS.

Supplementary aims:

- Develop knowledge and skills within the AI-related finance sector
- Further improve Python skills (experimenting with new functions such as unit-tests)
- Develop project planning and execution skills
- Develop presentation skills
- Showcase how a program can be utilised to execute a difficult set of human tasks in an efficient and renewable manner

All aims have been achieved apart from experimenting with unit-tests. Upon making these aims, it was planned that Visual Studio Code would be used as the programming environment to implement and execute code. This environment had powerful unit-test capabilities that made certain programming tasks much more efficient. However, due to further analysis of the suitability of this task to a particular environment, Jupyter Notebook was used instead as an environment that provides better sharing and readable elements when it comes to external people viewing implemented code and obtained results. However, unit-testing in Jupyter Notebook realised issues, and this aim was not be achieved.

Background

Although background reading was not a significant part of this project, several sources were reviewed to gain insight on 4 areas:

1. Hyperparameter selection for the pre-defined strategy
2. Data loading, analysis and cleaning techniques
3. Other strategies used
4. Evaluation methods

Conducting research on these four areas allowed for a rigorous and strategic plan for implementing the pre-defined strategy effectively.

1. Hyperparameter selection for the pre-defined strategy

There were 6 hyperparameters that needed to be defined for the implementation of the strategy:

1. **Stock Index** (the set of stocks to be used, whether that was S&P 500, FSTE 100 etc. or a mixture of stocks
2. **Time duration** (the period containing the change of stock value in percentage that we are interested in)
3. **Target value** (the target for how much the average change on a good stock would be during a time duration in percentage of its value)
4. **Starting moment** (the starting time, which would be sometime in the past)
5. **Epsilon** (used in the probability equation for the strategy. Must be greater than 0)
6. **Bound on percentage change of values** (must be large enough so that it is very unlikely to be exceeded for any one stock

Whilst several sources of literature were reviewed to find appropriate stock indexes to use, it was found that stock variability played the most importance. Therefore, the literature review didn't give much insight on which stock index to select. However, a source using FSTE 100 stocks was found *(Hussain, S., 2022)*, and after viewing the top 30 components for this index in Yahoo Finance *(Finance, Y., 2022),* it was found that the stocks within showed reasonable variability in percentage change. This indicated the FSTE 100 index as a suitable selection of stocks to use.

When researching appropriate and commonly used time durations for literature sources conducting similar research, a variety of time durations were found to have been used. From one source, *(Raposa, 2021),* a 15-day time period produced the optimal absolute risk and adjusted returns for the GME (GameStop) stock over a 21-year period. Looking at another source *(Algovibes, 2020)* using the top 30 components for the DJI index, a monthly duration was instead utilised. This source more closely related with the type of work presented in this project, and after witnessing several other online sources also using a month duration, it was concluded that a month would be an appropriate time duration to use.

The target value was researched to be optimum at around +/- 5% average change on a stock within the time duration of a month. This was found from averaging 30 FSTE 100 stock changes during a month (Hargreaves Lansdown., 2021).

A search for an appropriate starting moment was also conducted, but this came down to the overall period used for the strategy as it was desired to set the end moment as whatever the current time of program execution was (always providing the most recent stock data). It was found that trading strategies with holding periods (period of holding before investing again) of more than a month would benefit greater with longer time periods of around 15 years *(Singh, V., 2021)*. A decided time duration of a month made 15 years the ideal overall period for trading, so it was concluded that the starting moment would be 15 years from the current time of program execution.

An epsilon value was not researched in literature for this pre-defined strategy, but other research highlighted that the epsilon value should be lower than the target value. Therefore, an epsilon value below 0.1 would suffice.

The bound on percentage change of values was indicated by the project supervisor to be a value high enough so that a change in value of a stock would not exceed it (therefore, 100% was decided to be a good bound).

2. Data loading, cleaning and analysis techniques

Literature review was, secondly, conducted to explore data loading, analysis and cleaning techniques prior to implementation of the strategy.

Data loading (importing of the stocks) was researched to be executed via using the pandas data reader function and importing the stock index from Yahoo Finance *(Algovibes, 2020)*.

Data analysis was seen to entail plots of certain stocks to visualise patterns in stock value changes over time. This could be useful to gain more insight on some of the chosen stocks.

Based on research, data cleaning was found to include removing NaN values.

3. Other strategies used

Other strategies used for picking stocks were researched, and it was seen that the quantitative momentum strategy and moving average strategy were 2 of the most common strategies used in other literature. An aim in this project was to implement and test the pre-defined strategy effectively, so to maintain key focus on that, the quantitative momentum strategy was researched more deeply to prepare for implementation and comparison with the pre-defined strategy.

4. Evaluation methods

Suitable evaluation methods for discovering strategy performance were researched and it was found that back testing was by far the most common method for evaluating algo-trading strategies (Adithyan, N., 2021).

Ethical Use of Data

The data required for this project contained adjusted close price data, for the past 15 years from the point of program execution, of the top 30 components of the FSTE 100 stock index. This data was obtained from Yahoo Finance via the ‘pandas datareader’ module, which requested the data from the website and loaded it into a pandas data frame.

This data was freely available for use and no other data was used for this project. This project complies with the British Computer Society (BCS) Code of Conduct.

Design

This section is separated into 3 parts:

1. Software design
2. Epsilon strategy design
3. QMS design
4. Backtest design

1. Software design

Software design is documented below in the form of a UML diagram (Figure 1):

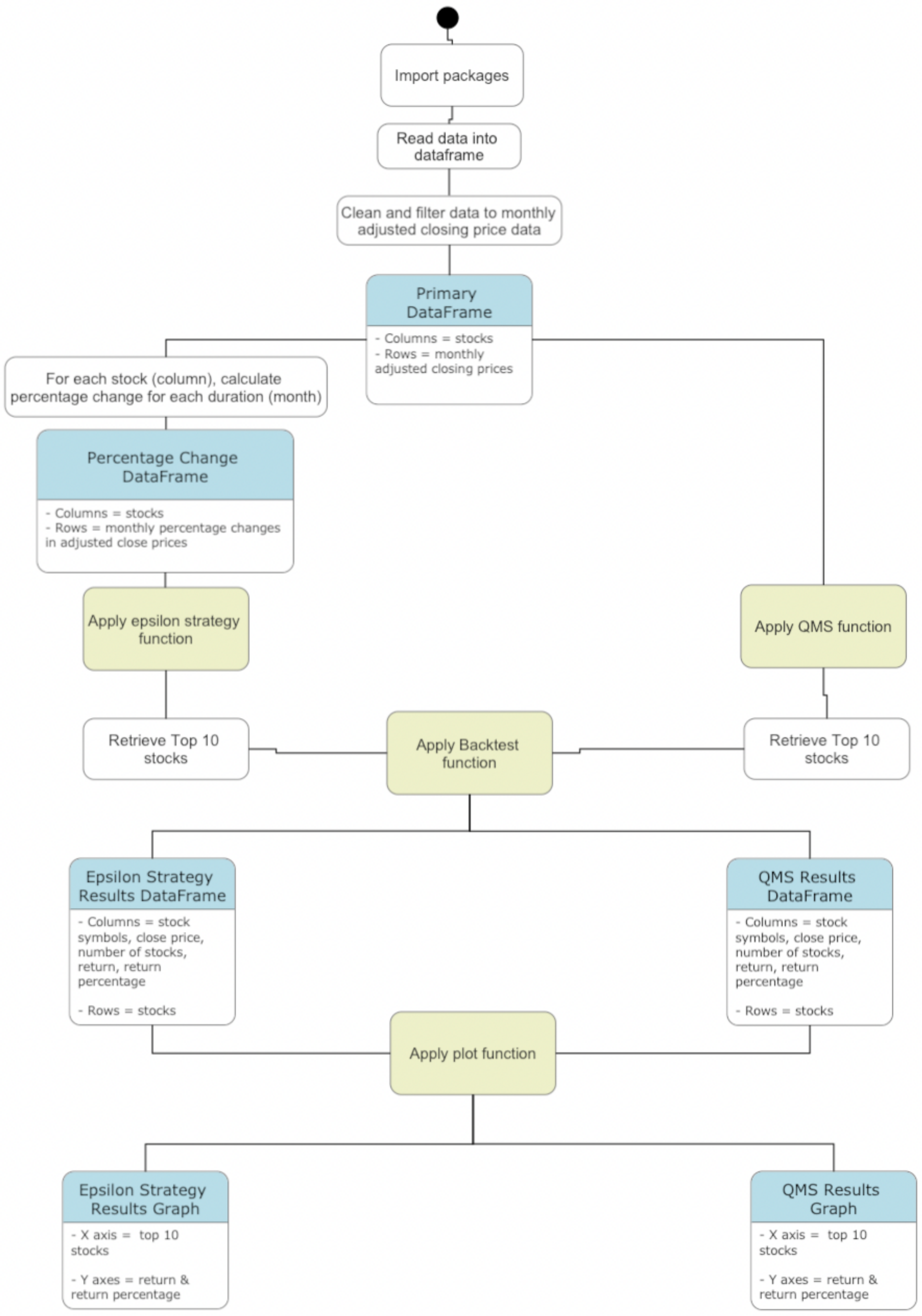

Import packages
Read data into dataframe
Clean and filter data to monthly adjusted closing price data
Primary DataFrame
- Columns = stocks
- Rows = monthly adjusted closing prices
For each stock (column), calculate percentage change for each duration (month)
Percentage Change DataFrame
- Columns = stocks
- Rows = monthly percentage changes in adjusted close prices
Apply epsilon strategy function
Apply QMS function
Retrieve Top 10 stocks
Apply Backtest function
Retrieve Top 10 stocks
Epsilon Strategy Results DataFrame
- Columns = stock symbols, close price, number of stocks, return, return percentage
- Rows = stocks
QMS Results DataFrame
- Columns = stock symbols, close price, number of stocks, return, return percentage
- Rows = stocks
Apply plot function
Epsilon Strategy Results Graph
- X axis = top 10 stocks
- Y axes = return & return percentage
QMS Results Graph
- X axis = top 10 stocks
- Y axes = return & return percentage

Figure 1

Figure 1 details the software structure from start to finish. White boxes represent actions, blue and white boxes represent outputs (data frames and graphs) with their structure documented and green boxes represent functions.

Firstly, the necessary packages are imported which enables a series of computations described in the **implementation section**. The data required is then read into a data frame, which is then cleaned and filtered to the desired format (Primary DataFrame). From this point, the software will follow 2 paths:

1. The primary data frame will be converted to a percentage change data frame, which will be applied to the epsilon strategy function. This will return the top 10 stocks given by this strategy.

2. The primary data frame will be applied to the QMS function, returning the top 10 stocks given by this strategy.

Both sets of top 10 stocks will be passed individually to the Backtest function, which will return an epsilon strategy results data frame and a QMS results dataf rame for the top 10 stocks returned by the epsilon and quantitative momentum strategies respectively.

Finally, the return and return percentage columns of these data frames will be plotted via the plot function to visualise and compare performance of both strategies.

2. Epsilon strategy design

The epsilon strategy was built as a function with 3 parameters:

1. Epsilon value (e)
2. Target value (target)
3. Dataset (dataset)

The algorithm implemented to allow this strategy to function is documented in a UML diagram below (Figure 2):

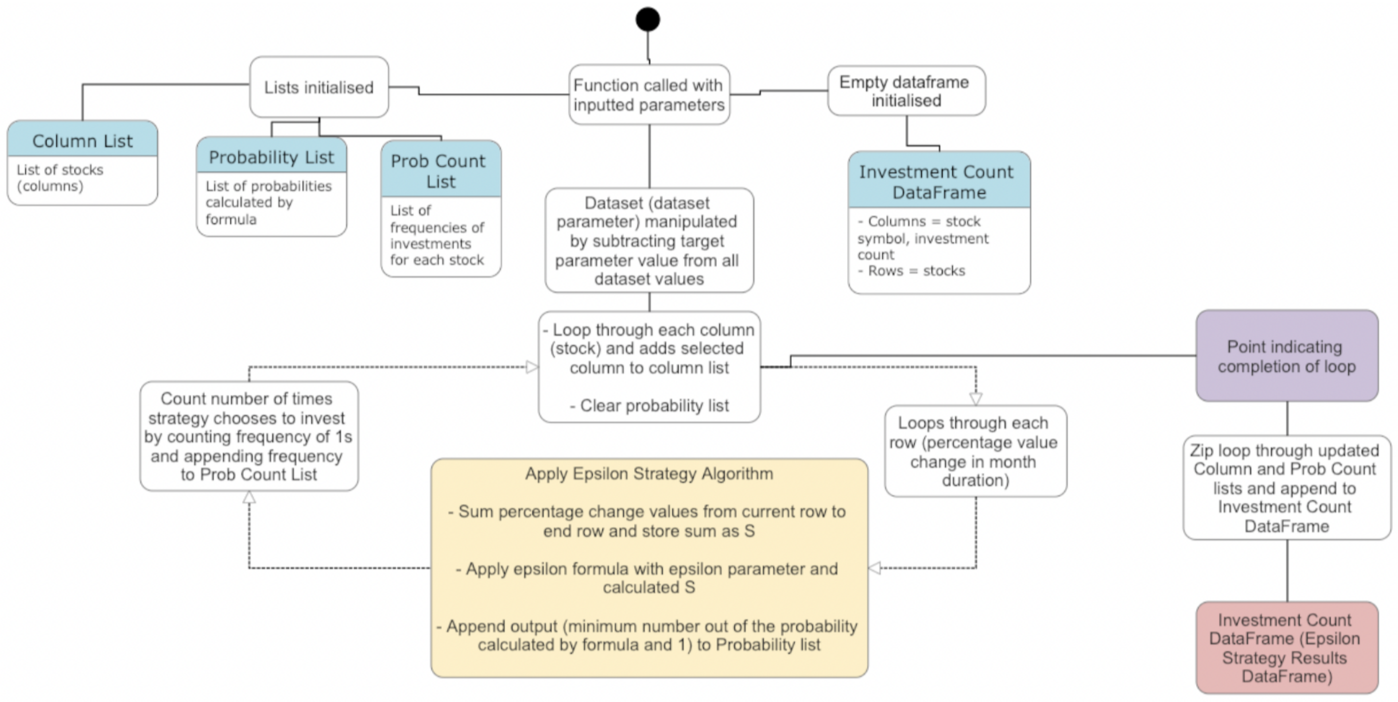

Lists initialised
Function called with inputted parameters
Empty dataframe initialised
Column List
List of stocks (columns)
Probability List
List of probabilities calculated by formula
Prob Count List
List of frequencies of investments for each stock
Dataset (dataset parameter) manipulated by subtracting target parameter value from all dataset values
Investment Count DataFrame
- Columns = stock symbol, investment count
- Rows = stocks
- Loop through each column (stock) and adds selected column to column list
- Clear probability list
Point indicating completion of loop
Count number of times strategy chooses to invest by counting frequency of 1s and appending frequency to Prob Count List
Loops through each row (percentage value change in month duration)
Zip loop through updated Column and Prob Count lists and append to Investment Count DataFrame
Apply Epsilon Strategy Algorithm
- Sum percentage change values from current row to end row and store sum as S
- Apply epsilon formula with epsilon parameter and calculated S
- Append output (minimum number out of the probability calculated by formula and 1) to Probability list
Investment Count DataFrame (Epsilon Strategy Results DataFrame)

Figure 2

Figure 2 shows the epsilon strategy function design. Blue and white boxes represent certain outputs (lists and data frames) with their structure detailed, the yellow box represents the strategy algorithm, the dotted lines represent a loop, the purple box represents the end point of the loop and the pink box represents the output of the entire function (a data frame).

The function is called with the 3 inputted parameters and an empty data frame, and 3 empty lists, are initialised. The dataset used as the dataset parameter of the function will then be manipulated as described above with the target parameter. A loop will then begin which applies the implemented strategy algorithm to the entire dataset, starting from the first stock (column) and first duration (row) within that column, progressing to the last row. This algorithm sums the percentage change values in the Percentage Change Data Frame (Figure 1) from the current row to the end row and stores the value as an S variable. The epsilon formula is then applied with the epsilon parameter and the S variable. This formula is shown below:

$$min\left(e^{3 \cdot (1-e)^{\left(\frac{S}{2}\right)}}, 1\right)$$

The output of this formula (the minimum number out of the calculated probability given by the formula and 1), will then determine if the strategy chooses to invest in the stock at the given time duration (row). If 1, this indicates an investment and this will be counted, and the probability count list will be appended with a 1. This process continues in the loop to the next duration (row) all the way to the end of the data frame.

Once complete, the stock symbols and their respective investment count will be added to the initialised 'Investment Count DataFrame' to report the number of times the strategy chose to invest in each stock.

3. QMS design

The second method, the Quantitative Momentum Strategy, was designed to test its effectiveness in investment and compare its performance against that of the epsilon strategy. The QMS works by picking stocks that increased in price the most and identifying stocks with the greatest uptrend. The dependent variable, the momentum, was measured and stocks with the greatest momentum indicated as the best to invest in.

The QMS function was built with only one parameter: dataset (the dataset to be used for strategy application). This dataset was a manipulated version of the 'Primary DataFrame' shown in Figure 1, where the dates column had been removed.

The process of this function is shown below (Figure 3):

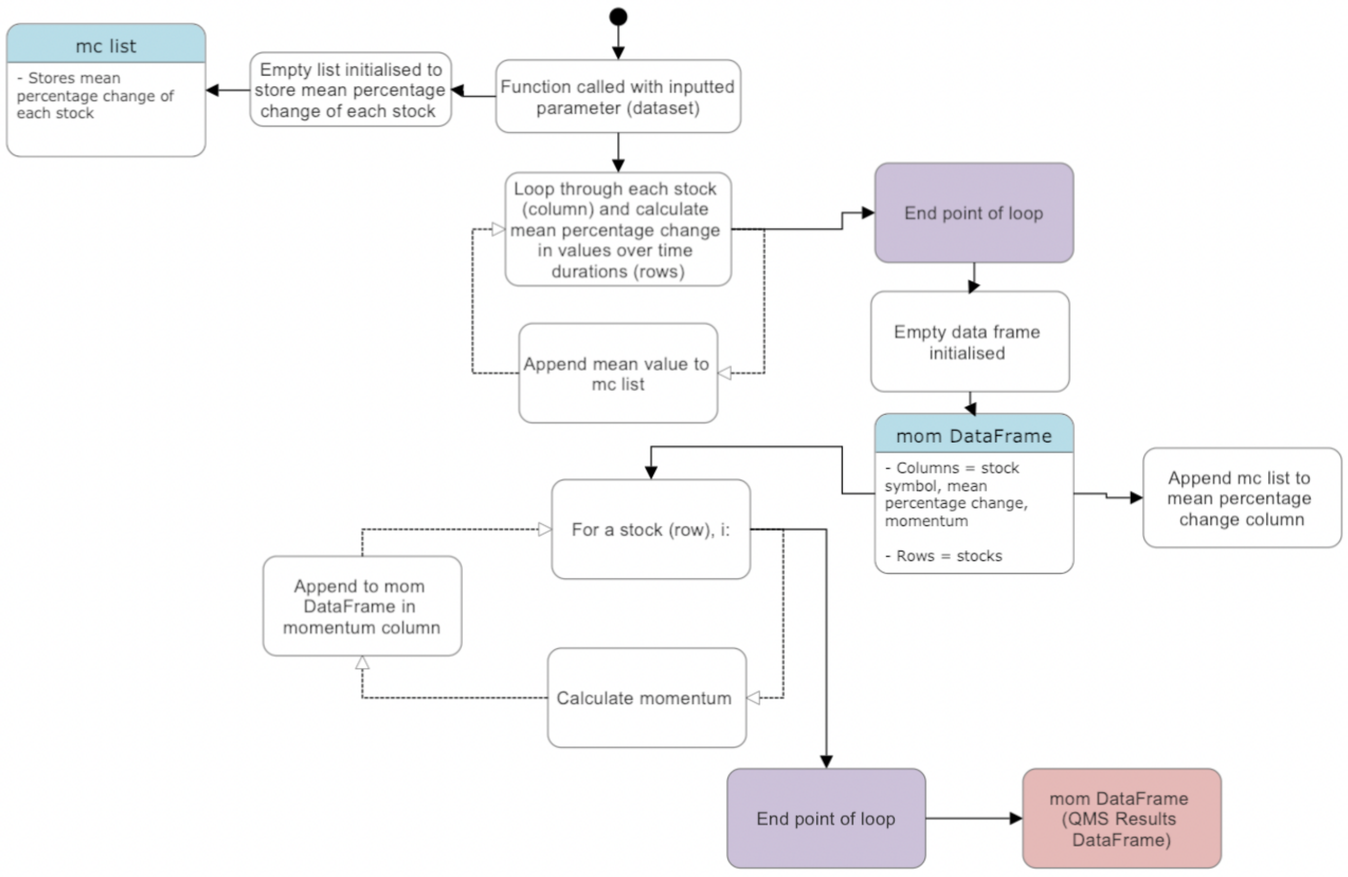

mc list
- Stores mean percentage change of each stock
Empty list initialised to store mean percentage change of each stock
Function called with inputted parameter (dataset)
Loop through each stock (column) and calculate mean percentage change in values over time durations (rows)
End point of loop
Append mean value to mc list
Empty data frame initialised
mom DataFrame
- Columns = stock symbol, mean percentage change, momentum
- Rows = stocks
Append mc list to mean percentage change column
For a stock (row), i:
Append to mom DataFrame in momentum column
Calculate momentum
End point of loop
mom DataFrame (QMS Results DataFrame)

Figure 3

The function is called with the required dataset and will then loop through each column (stock) and calculate the mean percentage change in values over all time durations (rows). This value will be appended to a list (mc list) initialised at the beginning. Once the loop has ended, an empty data frame will be created with a structure documented in the mom DataFrame component of Figure 3. This data frame will be appended with the mc list to fill the mean percentage column (mean percentage changes for each stock), before another loop commences. In this loop, the function will calculate the momentum of each stock using a computation described in the **implementation section.** This momentum value is appended to the respective row (stock) in the 'mom DataFrame'. Once this loop has ceased, the output of the function will be a populated 'mom DataFrame' showing the mean percentage change and momentum of each stock.

4. Backtest design

The back test function was built taking three parameters:

1. The stock data ('stock_data'). The dataset used in this project for this parameter was the 'Primary DataFrame' shown in Figure 1.

2. The top stocks given from applying a strategy ('strat_data')
3. The total amount of money to be invested in the portfolio ('amount_invested')

This function is detailed in the UML diagram below (Figure 4):

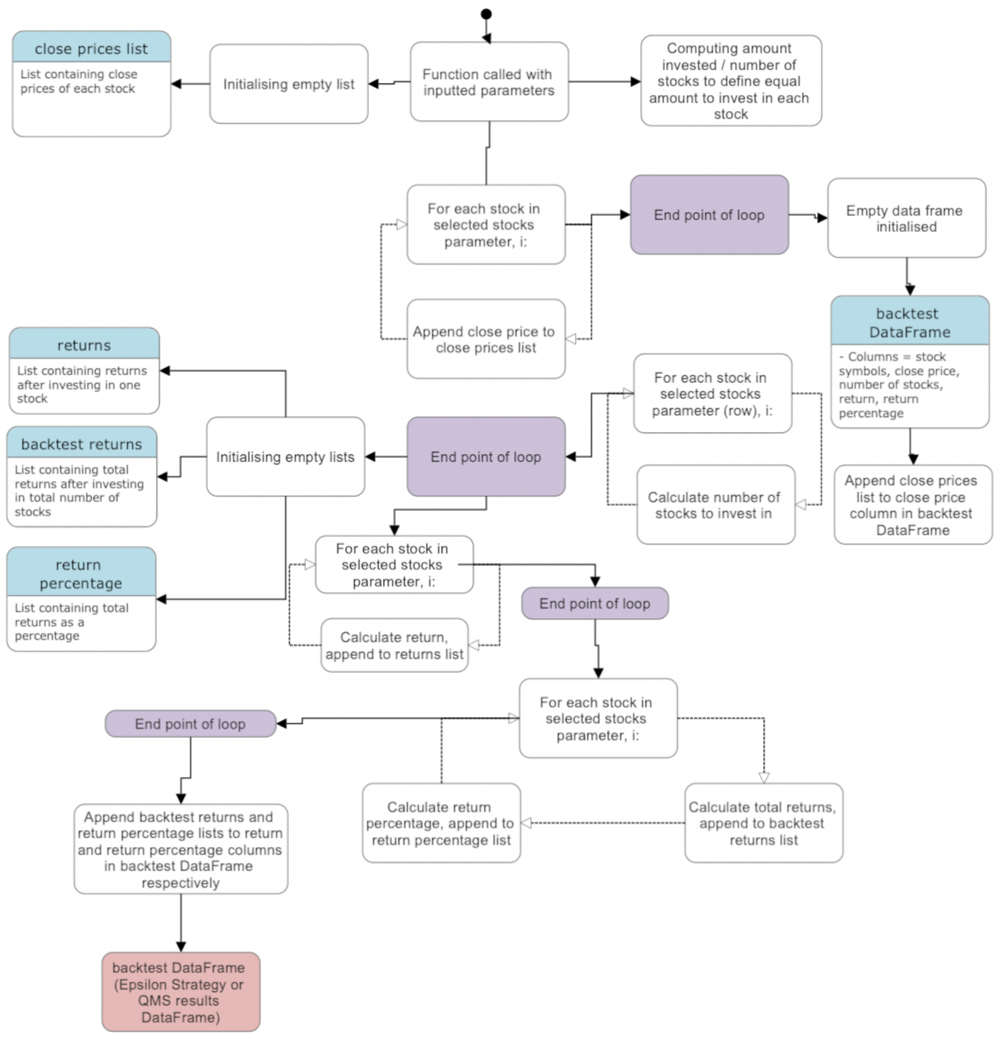

close prices list
List containing close prices of each stock
Initialising empty list
Function called with inputted parameters
Computing amount invested / number of stocks to define equal amount to invest in each stock
For each stock in selected stocks parameter, i:
End point of loop
Empty data frame initialised
Append close price to close prices list
backtest DataFrame
- Columns = stock symbols, close price, number of stocks, return, return percentage
returns
List containing returns after investing in one stock
For each stock in selected stocks parameter (row), i:
backtest returns
List containing total returns after investing in total number of stocks
Initialising empty lists
End point of loop
Calculate number of stocks to invest in
Append close prices list to close price column in backtest DataFrame
For each stock in selected stocks parameter, i:
return percentage
List containing total returns as a percentage
End point of loop
Calculate return, append to returns list
For each stock in selected stocks parameter, i:
End point of loop
Append backtest returns and return percentage lists to return and return percentage columns in backtest DataFrame respectively
Calculate return percentage, append to return percentage list
Calculate total returns, append to backtest returns list
backtest DataFrame (Epsilon Strategy or QMS results DataFrame)

Figure 4

The function is called with the inputted parameters, and then the first empty list (closing prices list) is initialised. An equal amount of money to be invested in each stock is also computed. The first loop then follows, appending the closing price of the top 10 stocks, that were inputted into the function as the 'strat_data' parameter, to the closing prices list. An empty data frame is then initialised with the structure documented in Figure 4. The closing prices for each stock from the closing prices list are appended to this data frame, before a second loop commences. At this point, the number of stocks to be invested in for each stock is calculated. Next, three more empty lists are initialised containing returns, total returns and return percentage after investing in each stock. A third loop will calculate the return giving after investing in one stock for each stock, and this return will be appended to returns list each time. A final loop will then calculate the total returns given after investing in the number of stocks calculated previously, for each stock. This total is appended to 'backtest returns list,' and then return percentage is calculated and appended to the return percentage list. Finally, the ;backtest returns' and 'return percentage' lists are appended to the 'backtest DataFrame' at columns 'return' and 'return percentage' respectively. The output of this function is a data frame containing these statistics, plus the close price and number of stocks invested in for each of the top selected stocks inputted into the function.

Implementation

This section details the implementation of the software design described above. There are six subsections here:

1. Setting up environment
2. Data
3. Epsilon strategy implementation
4. QMS implementation
5. Back testing
6. Visualising results

1. Setting up environment

The environment was firstly set up by importing the required packages for software computations. These packages included:

- 'pandas' (for data handling, and specifically 'pandas_datareader.data' for loading the data into a pandas data frame from Yahoo Finance

- 'datetime' (for handling the period where data would be collected and used)

- 'requests' (for requesting the data from Yahoo Finance)

- 'numpy' (for mathematical calculations included in formulas for both strategies)

- 'percentileofscore' from scipy.stats (used in the QMS)

- 'floor' from math (used in the QMS)

- ‘seaborn’ and ‘matplotlib’ (for plotting results in the ‘Visualising results’ section)

2. Data

Combining the ‘pandas’ function with a get request using the ‘requests’ package and pasting the link to the top 30 components in the FSTE 100 stock index on Yahoo Finance into an argument enabled the necessary stock data to load into a table.

Using the ‘datetime’ package, the start and end dates were defined to create the period that this project was interested in supplying stock data for. This period was defined as 15 years from the point of execution of the program. This was implemented in this way so that both strategies could be tested on the latest stock data, in aim to achieve the most informative results when it came to deciding on which stocks to invest in.

This project was only interested in the ‘adjusted closing prices’ variable amongst others, such as ‘Volume’, ‘Last price’ and ‘Change’. Therefore, using the ‘pandas_datareader.data’ function with parameters ‘start’ (start date, 15 years from ‘end’ date), ‘end’ (end date, which was the point of program execution) and ‘Adjusted Closing Price’ (the variable of interest), the required stock data containing the adjusted closing price data recorded every day for the past 15 years of program execution of 30 FSTE 100 stocks was loaded into a pandas data frame. There were several researched methods of loading stock data, but this way required only a few lines of code, making it the most efficient way of obtaining the data. This code can be seen in the **Appendices** section (Appendix 1).

The data was then checked for null values using ‘df.isnull().values.any()’. Null values were confirmed in the data, and at first, the data was cleaned of all null values by dropping all rows that contained these values. However, this decreased the dataset to a very small size, making it unfit for experimentation. Null value presence was explored deeper by checking the frequency of nulls in each column (stock). It was found that 5 stocks (“HLN.L, “CCH.L”, “AUTO.L”, “MNG.L” and “AAF.L”) contained significant numbers of nulls compared to other stocks. Therefore, these stocks were dropped entirely from the data frame, and the remaining few null values in the data were distinguished by dropping the rows they were present in. As this was a small number, the cleaned data frame was an acceptable size for experimentation.

Finally, the data was filtered by selecting intervals of 30 rows from the start to the end of the data frame. This was done to achieve a data frame where each row represented a monthly, instead of daily, interval. It is noted that not every day’s data was accessible upon loading from Yahoo Finance, and so these monthly intervals were not completely accurate. However, they marked a good indication of the trend in stock value changes in the 15-year period, deeming acceptable.

3. Epsilon Strategy Implementation

With the cleaned data available, the epsilon strategy was implemented and applied via three steps:

1. Converting the cleaned data to a 'percentage change data frame'

2. Implementing the epsilon strategy function following its design documented in Figure 2.

3. Applying the strategy to the percentage change data frame with different combinations of parameters

4. Obtaining results given by the best combination of parameters and filtering the top 10 stocks indicated by highest frequency of investment.

In the first step, the percentage changes in adjusted close prices from the start to end of the data frame of each stock was calculated and recorded in a separate data frame. This data frame is shown in **Appendix 2.** The percentage changes were computed using python's 'pct_change()' function. This data frame was then cleaned of the 'na' values seen at the first row across all columns (as the percentage change of the very first row will not be a number). The target value (average percentage change for all stocks within the period) was calculated using simple mathematical computations, and this percentage (1.7%) was stored as a decimal number in the 'target' variable.

In the second step, the epsilon strategy was implemented as a function following the specified design. By creating three function parameters (e for epsilon, target, and dataset), the function was able to apply the strategy to any given dataset using different epsilon and target value combinations efficiently. There were no complex calculations required for implementing this strategy's formula, but it is noted that a 'zip' function was used to loop through multiple lists at the end of the function to create the outputted data frame, documenting the strategy results. 'Zip' was used for the first time in this project and worked effectively at cutting down code. The output of this function can be viewed at **Appendix 3.**

The third step included applying the strategy to the cleaned dataset using different combinations of the epsilon and target value parameters. Implementation of this series of testing included a series of function calls with different parameters, and their tabulated results were analysed by checking if any investments were made for each stock at all, for a certain trend in investments for certain stocks and any outlying results. The first combination of parameters tested were the calculated target value (average percentage change on a stock) of 0.017 and an epsilon of 0.01. These values were then inflated (multiplied by 20, 50 and 75) to see any changes in investment counts. Other values were tested as well, such as the same target value but a larger epsilon, and even a larger epsilon than target value (although this was only tested out of interest, as it was researched that epsilon had to be lower than the target value for the strategy to avoid unreliable results).

Once the best witnessed parameter combination had been discovered, the results from this combination were collected as a table containing the investment count of each stock. Using the ‘nlargest’ function, the top 10 stocks with highest number of investments were filtered into a new data frame to be used in back testing. This line of code can be seen in **Appendix 4**.

4. QMS implementation

The QMS was implemented as a function, like the epsilon strategy. Before implementing this function, the cleaned dataset of monthly adjusted close price values was manipulated by removing the dates column. Then, the QMS function was created with a single parameter, ‘dataset’. This took in the manipulated dataset and once called, the mean percentage changes of each stock were calculated and appended to a list. After making a ‘momentum’ dataframe, following the design in Figure 3, momentum was calculated for each stock by using the ‘percentileofscore’ function from the ‘scipy.stats’ package. The computation to calculate momentum for a stock used this score function with the mean percentage change of the stock and the mean percentage changes of all other stocks as the two parameters for the function. Calling this function would compute the percentile rank of the score relative to the list of scores, divided by 100 to compute the momentum. In other words, if the mean change of a stock was greater relative to the mean changes of all other stocks, that stock would have a higher momentum.

The output of the QMS function was a dataframe, ‘mom’, containing the mean change and momentum of all stocks.

As in the epsilon strategy implementation, the top 10 stocks with highest momentum were filtered using the ‘nlargest’ function.

5. Back testing

A function was made to back test both sets of top 10 stocks given by the two strategies. This function took in the cleaned monthly adjusted closing price data, a set of top stocks as the data frame given from the ‘nlargest’ computation and an amount of money invested into the entire investment portfolio. The function followed the design documented in Figure 4, with simple python computations used throughout. The ‘floor’ function was used when calculating the number of stocks to be invested in for each stock, which returned the largest number not greater than the argument inside the floor() function. This argument was the equal amount of money invested in a stock divided by the close price of that stock, allowing for a fixed number of stocks to be invested in that would keep within the allocated budget provided for that stock’s investment.

Returns were calculated by using ‘np.diff’ from the ‘numpy’ package to calculate the nth order discrete difference along the given input axis, which was the series of monthly adjusted closing price data points. The total returns were calculated by multiplying a stock’s returns by the number of stocks invested in for that stock, with the return percentage computed via division of the total returns by the amount invested in that stock, multiplied by 100.

The output of the back test function was a data frame containing the close price, number of stocks invested in, return, and return percentage of each of the top stocks provided to the function. This was implemented so that the function could be applied on results from both strategies efficiently, and this worked very well for visually comparing these results next.

6. Visualising results

A final function was created that contained one parameter: the output data frame of the back test results of the epsilon strategy or QMS. This plot function set up a figure and used 'seaborn' with 'matplotlib' to plot the return percentage and return of the given top 10 stocks. Return was given on the right y-axis, whilst return percentage was given on the left. The graph was a bar-line chart which easily highlighted which stocks gave the greatest returns for a particular strategy, and which strategy picked the best (highest returning) stocks.

Evaluation with Learning Points

This section analyses how the defined objectives of this project have been met, the strengths, weaknesses, and external feedback on the software for this project. Learning points are included within this section that detail the skills and knowledge acquired throughout the project, highlights in the project and improvements/ further work that would be conducted to overcome any pitfalls or weaknesses of the project.

The aims of this project and their status of completion is evaluated below:

1. Implementing both the epsilon and QMS pre-defined strategies using Python code

As witnessed in the implementation section, both strategies were successfully implemented according to their original designs. Using Jupyter Notebook, python code was formed in a way that was easily readable by viewers and easily manipulatable by the editor. One weakness of using Jupyter Notebook was the limited power of unit tests when it came to testing functions. Using an environment such as VS Code would have enabled more rigorous and efficient testing of, specifically, the different combinations of epsilon and target values for the epsilon strategy. Conducting unit tests to find the best combinations would have saved time and, potentially, found a more optimal combination of parameters. However, the necessary computations needed to create functions and data frames were still present regardless of the environment used. Strengths in the implementation of the two strategies lay in their functions themselves, which were concise and powerful. They reduced the volume of code as opposed to if the strategies were implemented outside of functions, and the use of functions also allow datasets containing stocks in other indexes to be used within the same functions.

2. Test both strategies via 'back testing' to make evaluations based on return percentages

The aim to back test both strategies was met successfully and was, perhaps, the biggest strength of this project. The back test function was implemented exactly according to its original design, with a user-friendly method of selecting the data and amount of money to be back tested with. Understanding certain calculations for back test function was difficult at first, as the source that this code was followed from did not clearly explain the code. However, sufficient time was allocated to understand the mathematical procedures in the code which realised a successful adaptation of the back test code to this project's unique problem. The output of the back test function gave insightful, numerical indications as to which stocks gave greatest investment returns that the back tested strategy had picked. It also showed the number of stocks that were invested in per stock, giving even more information on the investment value of a selected stock. An improvement for this section would be to include error handling when inputting parameters into the back test function. For example, if a user inputted a string, rather than a number, as the investment amount parameter, an error prompting the user to specifically enter a number would show up. This would improve user interactivity with the software.

3. Evaluate both sets of results to conclude the effectiveness of the epsilon strategy against the commonly used and well-reviewed QMS

This section firstly analyses the technique used to evaluate the results from back testing both strategies, before documenting the actual results and comparing the sets of results from the two strategies to reveal the effectiveness of both (and specifically, the epsilon strategy).

Firstly, the idea of using a bar-line plot to visualise return data from the back test results of the two strategies worked well. Different coloured bars, representing different stocks, allowed for easy comparison of returns given by investing equal amounts into each of the stocks. The addition of the lines highlighted the peaks of the bars well, but evaluating its effectiveness overall, the lines did not provide any useful information. If the x-axis of the graphs indicated some period, then the lines would reveal a clear trend in returns over time. However, the x-axis were simply the different selected stocks. When comparing the two graphs showing the two strategy's results, it was obvious in this case which strategy had selected the best returning stocks. However, comparing two other strategies with similar results would require some sort of statistical evaluation showing which strategy performed better overall. A weakness in this visual method of evaluation was the absence of this, and an improvement would be to calculate the average return percentage across all selected stocks for each strategy, to give a clearer indication of which strategy performed better.

The results from back testing both strategies are documented below (Figure 5 showing QMS results, and Figure 6 showing epsilon strategy results):

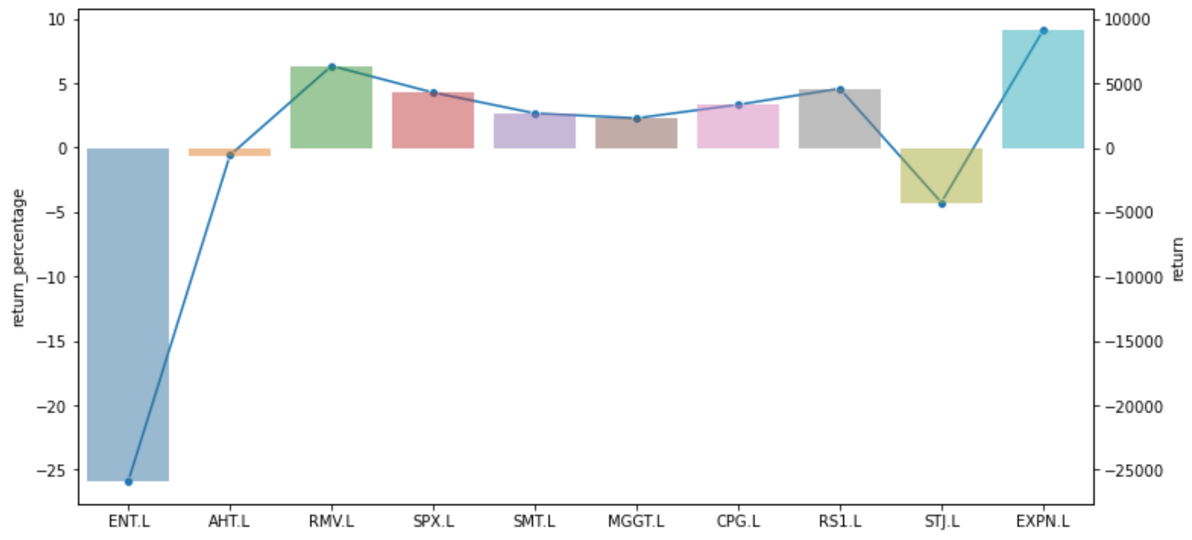


Figure 6 – QMS Results

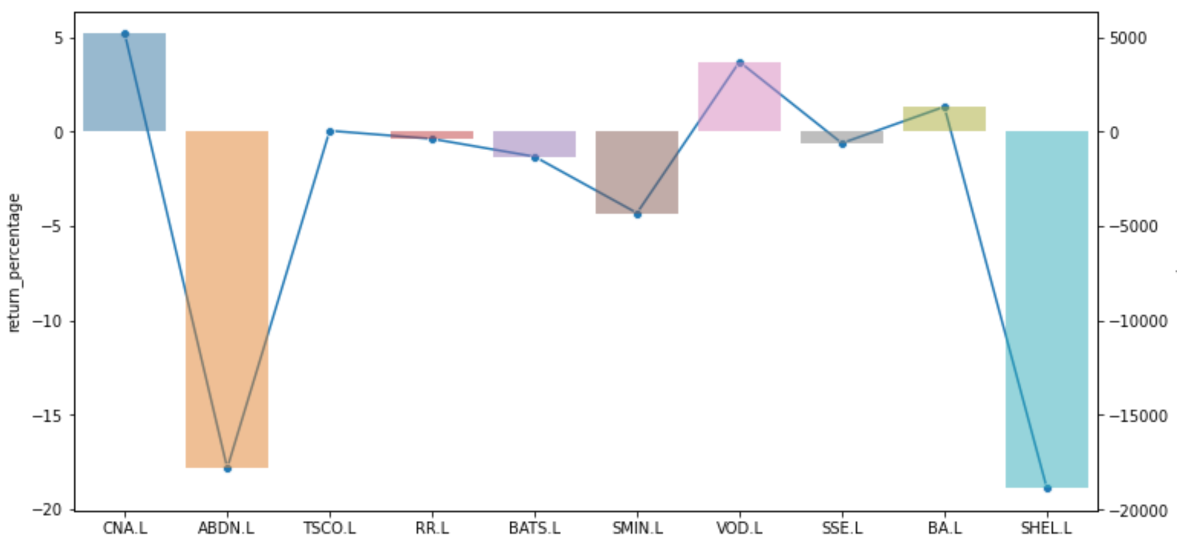


Figure 7 – epsilon strategy results

When analysing the QMS results, it is seen that most selected stocks gave positive returns. The majority of these averaged around 5% return and given a 10% return was researched to be 'good', these results were deemed to be 'okay'. There were three stocks that gave negative returns and investing in 'ENT.L' produced a massive loss in comparison to investing in other stocks. An improvement to this project is to explore the reason behind anomalies, such as this, as this result may not give meaningful information on the strategy's effectiveness. Overall, these results were expected, with no super high returns and many stocks giving positive returns.

Analysing the epsilon strategy's results was more difficult and raised many more queries about the strategy itself. Most stocks gave no returns or negative returns, with two stocks ('ABDN.L' and 'SHEL.L') performing very poorly in returns. Again, these cases looked anomalous and further analysis of these stocks is required to determine the reasons for why these stocks were picked. However, the fact that only three stocks ('CNA.L', 'VOD.L' and 'BA.L') gave positive returns indicated that overall, this strategy did not perform well. This was apparent as the QMS had returned stocks that gave mostly positive returns, whereas relying on the epsilon strategy to make investments would realise a significant loss of money in real life.

Overall, the aim to evaluate the results was met to a certain degree. The evaluation did give meaningful information on how the strategies reacted with real-life stock data and how effective they would be when it comes to investing in stocks in real-life. However, the evaluation could have been deeper, including analysis on outliers and statistical analysis on strategy performance (calculating average returns).

4. Meeting supplementary aims

The supplementary aims of this project are evaluated here, and firstly, it is noted that the improvement of project planning and execution, and presentation skills have been met. Having stuck to the specification of this project and the gantt chart timetable, project management and planning skills have been developed greatly. The successful completion of two presentations during this project has improved confidence and effectiveness in verbal communication of large-scale tasks. The aim to improve python skills has been met to a certain degree; whilst the data handling and certain mathematical coding skills have been developed via data frames and implementation of the two strategies, the absence of unit/auto-test experimentation marks the opportunity for further development of coding skills within this project. AI-related finance sector knowledge and skills have been increased, and a basic software implementation of a stock selection strategy and benchmark testing method have been recognised and completed. This project has increased interest in this sector, and further development of related knowledge and skills is desired in future work. Finally, this project has successfully shown how software can execute a difficult set of human tasks in an efficient and renewable manner. The use of functions shortened code required and made application of these functions renewable in the sense that the same processes within the functions can be applied with different parameters to adapt to different scenarios (for example, working on datasets containing different stocks or different variables).

5. Other key evaluations

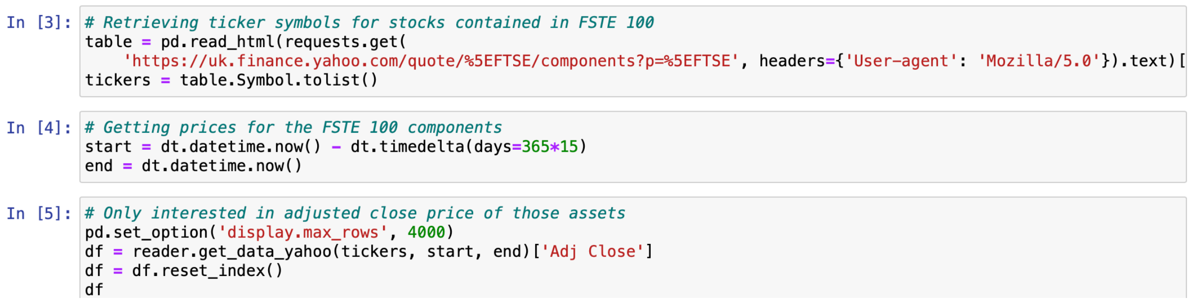
It is noted that many parameter combinations for the epsilon strategy realised no investments for any stock. Investments were only made when the epsilon and target values were both high, and investments increased the higher they got. Also, as the latest stock data is used upon program execution, the results may differ upon each program run.

Conclusions

This project has demonstrated the importance of AI in stock selection through the creation of fluid and efficient software that was able to implement and back test stock selection strategies. The power of programming, and specifically Python code, has been highlighted in in-built and created functions serving a range of purposes, whilst also in data handling.

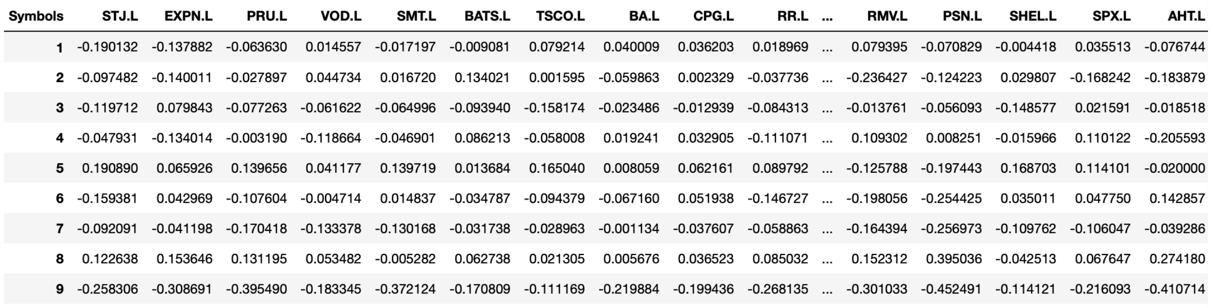
The focus of this project was to evaluate and report the performance of the epsilon strategy. Whilst implementation and application of its algorithm was successful, the strategy itself was found to produce low-quality results on FSTE 100 stock data within the 15 year time period. This could have been due to several factors, but it was thought that a longer period than 15 years was needed for the epsilon strategy to work better. Further experimentation with this strategy is required, with a longer period and maybe some changes to the formula itself (using -S instead of S for example). The QMS was implemented well and served as a useful benchmark for comparison against the epsilon strategy. It is predicted that with the right development, the epsilon strategy could meet or exceed the performance of the QMS.

Appendices



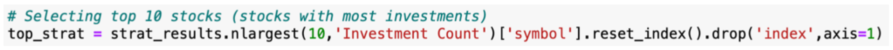
```
In [3]: # Retrieving ticker symbols for stocks contained in FSTE 100
        table = pd.read_html(requests.get(
            'https://uk.finance.yahoo.com/quote/%5EFTSE/components?p=%5EFTSE', headers={'User-agent': 'Mozilla/5.0'}).text)[
        tickers = table.Symbol.tolist()

In [4]: # Getting prices for the FSTE 100 components
        start = dt.datetime.now() - dt.timedelta(days=365*15)
        end = dt.datetime.now()

In [5]: # Only interested in adjusted close price of those assets
        pd.set_option('display.max_rows', 4000)
        df = reader.get_data_yahoo(tickers, start, end)['Adj Close']
        df = df.reset_index()
        df
```

Appendix 1 – code to load required data from Yahoo Finance into a pandas data frame



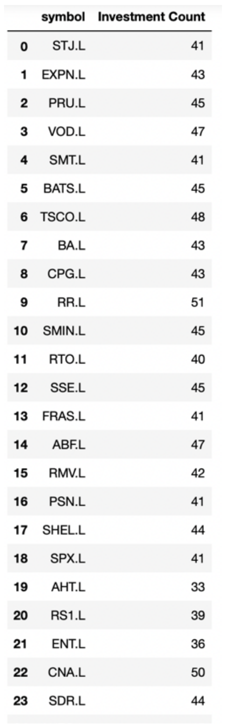
Symbols	STJ.L	EXPN.L	PRU.L	VOD.L	SMT.L	BATS.L	TSCO.L	BA.L	CPG.L	RR.L	...	RMV.L	PSN.L	SHEL.L	SPX.L	AHT.L
1	-0.190132	-0.137882	-0.063630	0.014557	-0.017197	-0.009081	0.079214	0.040009	0.036203	0.018969	...	0.079395	-0.070829	-0.004418	0.035513	-0.076744
2	-0.097482	-0.140011	-0.027897	0.044734	0.016720	0.134021	0.001595	-0.059863	0.002329	-0.037736	...	-0.236427	-0.124223	0.029807	-0.168242	-0.183879
3	-0.119712	0.079843	-0.077263	-0.061622	-0.064996	-0.093940	-0.158174	-0.023486	-0.012939	-0.084313	...	-0.013761	-0.056093	-0.148577	0.021591	-0.018518
4	-0.047931	-0.134014	-0.003190	-0.118664	-0.046901	0.086213	-0.058008	0.019241	0.032905	-0.111071	...	0.109302	0.008251	-0.015966	0.110122	-0.205593
5	0.190890	0.065926	0.139656	0.041177	0.139719	0.013684	0.165040	0.008059	0.062161	0.089792	...	-0.125788	-0.197443	0.168703	0.114101	-0.020000
6	-0.159381	0.042969	-0.107604	-0.004714	0.014837	-0.034787	-0.094379	-0.067160	0.051938	-0.146727	...	-0.198056	-0.254425	0.035011	0.047750	0.142857
7	-0.092091	-0.041198	-0.170418	-0.133378	-0.130168	-0.031738	-0.028963	-0.001134	-0.037607	-0.058863	...	-0.164394	-0.256973	-0.109762	-0.106047	-0.039286
8	0.122638	0.153646	0.131195	0.053482	-0.005282	0.062738	0.021305	0.005676	0.036523	0.085032	...	0.152312	0.395036	-0.042513	0.067647	0.274180
9	-0.258306	-0.308691	-0.395490	-0.183345	-0.372124	-0.170809	-0.111169	-0.219884	-0.199436	-0.268135	...	-0.301033	-0.452491	-0.114121	-0.216093	-0.410714

Appendix 2 – percentage changes data frame



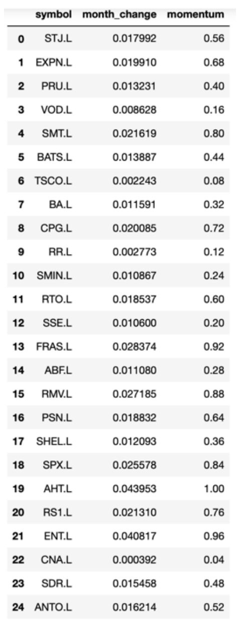
```
# Selecting top 10 stocks (stocks with most investments)
top_strat = strat_results.nlargest(10,'Investment Count')['symbol'].reset_index().drop('index',axis=1)
```

Appendix 4 – code to retrieve top 10 stocks indicated by a strategy (epsilon strategy in this example)



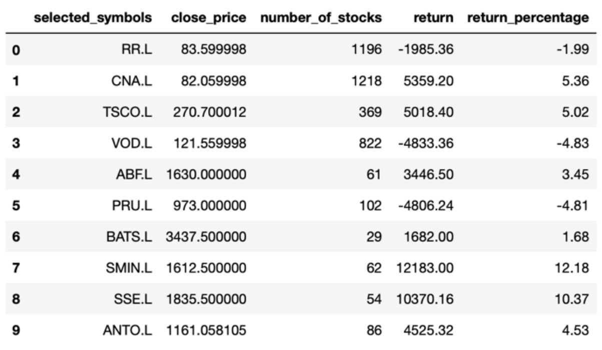
	symbol	Investment Count
0	STJ.L	41
1	EXPN.L	43
2	PRU.L	45
3	VOD.L	47
4	SMT.L	41
5	BATS.L	45
6	TSCO.L	48
7	BA.L	43
8	CPG.L	43
9	RR.L	51
10	SMIN.L	45
11	RTO.L	40
12	SSE.L	45
13	FRAS.L	41
14	ABF.L	47
15	RMV.L	42
16	PSN.L	41
17	SHEL.L	44
18	SPX.L	41
19	AHT.L	33
20	RS1.L	39
21	ENT.L	36
22	CNA.L	50
23	SDR.L	44

Appendix 3 – Output of epsilon strategy function



	symbol	month_change	momentum
0	STJ.L	0.017992	0.56
1	EXPN.L	0.019910	0.68
2	PRU.L	0.013231	0.40
3	VOD.L	0.008628	0.16
4	SMT.L	0.021619	0.80
5	BATS.L	0.013887	0.44
6	TSCO.L	0.002243	0.08
7	BA.L	0.011591	0.32
8	CPG.L	0.020085	0.72
9	RR.L	0.002773	0.12
10	SMIN.L	0.010867	0.24
11	RTO.L	0.018537	0.60
12	SSE.L	0.010600	0.20
13	FRAS.L	0.028374	0.92
14	ABF.L	0.011080	0.28
15	RMV.L	0.027185	0.88
16	PSN.L	0.018832	0.64
17	SHEL.L	0.012093	0.36
18	SPX.L	0.025578	0.84
19	AHT.L	0.043953	1.00
20	RS1.L	0.021310	0.76
21	ENT.L	0.040817	0.96
22	CNA.L	0.000392	0.04
23	SDR.L	0.015458	0.48
24	ANTO.L	0.016214	0.52

Appendix 5 – Output of QMS function



	selected_symbols	close_price	number_of_stocks	return	return_percentage
0	RR.L	83.599998	1196	-1985.36	-1.99
1	CNA.L	82.059998	1218	5359.20	5.36
2	TSCO.L	270.700012	369	5018.40	5.02
3	VOD.L	121.559998	822	-4833.36	-4.83
4	ABF.L	1630.000000	61	3446.50	3.45
5	PRU.L	973.000000	102	-4806.24	-4.81
6	BATS.L	3437.500000	29	1682.00	1.68
7	SMIN.L	1612.500000	62	12183.00	12.18
8	SSE.L	1835.500000	54	10370.16	10.37
9	ANTO.L	1161.058105	86	4525.32	4.53

Appendix 6 – Output of back test function (this is an example of the epsilon strategy's back tested results, and not the actual results)

Bibliography

Hussain, S., 2022. *Refinitiv Developers*. [online] Developers.refinitiv.com. Available at: <https://developers.refinitiv.com/en/article-catalog/article/quantitative-momentum-strategy-for-picking-top-ftse-stocks.> [Accessed 10 June 2022].

Finance, Y., 2022. *FSTE 100 Top 30 Components*. [online] Uk.finance.yahoo.com. Available at: <https://uk.finance.yahoo.com/quote/%5EFTSE/components?p=%5EFTSE> [Accessed 10 June 2022].

Raposa, 2021. *How to Build Your First Momentum Trading Strategy in Python — Raposa*. [online] Available at: <https://raposa.trade/blog/how-to-build-your-first-momentum-trading-strategy-in-python/> [Accessed 11 June 2022].

Youtube.com, 2020. *How to build a trading strategy [Momentum] with Python?*. [online] Available at: <https://www.youtube.com/watch?v=dnrJ4zwCADM> [Accessed 11 June 2022].

Singh, V., 2021. *Backtesting: How to Backtest, Analysis, Strategy, and More*. [online] Blog.quantinsti.com. Available at: <https://blog.quantinsti.com/backtesting/> [Accessed 14 June 2022].

Adithyan, N., 2021. *Picking Stocks with a Quantitative Momentum Strategy in Python*. [online] Medium. Available at: <https://medium.com/codex/picking-stocks-with-a-quantitative-momentum-strategy-in-python-b15ac8925ec6> [Accessed 20 June 2022].

Hargreaves Lansdown. 2021. *FTSE 100: Performance*. [online] Available at: <https://www.hl.co.uk/shares/stock-market-summary/ftse-100/performance#> [Accessed 8 July 2022].

GitHub. 2022. *"No tables found" suddenly · Issue #55 · atreadw1492/yahoo_fin*. [online] Available at: <https://github.com/atreadw1492/yahoo_fin/issues/55> [Accessed 21 September 2022].