OCR H446 Component 3 Programming Project

Candidate Name: Alexander Lanecki Candidate Number: 4422

Salesian School Chertsey

64898

# Contents

[Contents 1](#_Toc206683809)

[1. Analysis of the problem (10 marks) 2](#_Toc206683810)

[1.1 Computational Methods 2](#_Toc206683811)

[1.2 Stakeholders 5](#_Toc206683812)

[1.3 Research of existing solutions 6](#_Toc206683813)

[1.4 Essential Features 12](#_Toc206683814)

[1.5 Limitations 14](#_Toc206683815)

[1.6 Hardware and software requirements 16](#_Toc206683816)

[1.7 Success criteria 18](#_Toc206683817)

[2. Design (15 marks) 20](#_Toc206683818)

[3. Iterative Development (15 marks) 22](#_Toc206683819)

[4. Iterative Testing (10 marks) 24](#_Toc206683820)

[5. Post-devlopment Testing (5 marks) 26](#_Toc206683821)

[6. Evaluation (15 marks) 28](#_Toc206683822)

# 1. Analysis of the problem (10 marks)

## 1.1 Computational Methods

*Problem Identification:*

There already exist many existing websites, apps, and assistants to provide financial advice based on market trends, incorporating also breaking news and external factors which may affect how a stock’s price may fluctuate in the near future. However, they are almost always blocked by a huge paywall. Furthermore, even if a client is willing to pay for these services, tools are uncomprehensible by those with no technical knowledge or expertise in the financial sector simply appearing as jargon. They are optimised for institutional investors with large funds and advanced understanding. And if these tools are free or cheaper, this is possible as they oversimplify data or provide generic, vague advice similar to those of a fortune teller.

For a beginner, a useful app should be intuitive to use and possibly explain what each tool does, showing examples of its use and rather than assuming the client already possesses knowledge of the system, teaches them how to use it effectively and efficiently. This is beneficial as it would not only consolidate a client’s knowledge, it would also save the client time in the future, rather than always needing to figure out what a function of the app does, they would already know how to use it fully. It would also improve the usability and make the client more satisfied with the product as they could bring out more potential from the app.

This makes this a computational problem as it involves analysing massive datasets which are dynamically changing in tune with live market data. Manually calculating trends would be infeasible due to time constraints and cognitive limits, along with the redundancy of information even when it is comprehended due to the frequent movement of prices. Automating this using algorithms enables predictions to be made quickly and consistently.

*Computational methods:*

**Decomposition –** defined as “breaking down a complex problem or system into smaller parts that are more manageable and easier to understand” (BBC) This will be used to separate the problem into back-end and front-end components, with sub-programs to control and solve individual systems. For example, there would be modules such as a data collector (to retrieve financial data), preprocessor (to sift through the data and clean it), predicter (runs the ML model), and a visualiser (takes the final data and renders the graphic for the user) This modular approach would then be simpler to test, debug and replace individual components without affecting the whole system.

**Abstraction –** defined as “the process of filtering out – ignoring - the characteristics of patterns that we don't need in order to concentrate on those that we do” (BBC) The algorithm that I will employ is based on a complex system of interlinked sub-programs, and the detailed workings of it is abstracted away from the end user. The front-end interface will only show the necessary inputs the user can make to select an algorithm depending on the time period desired for the prediction, however the specifics of changing the machine’s learning dataset and performing the prediction will be hidden away. However, more advanced users may have the option to fine-tune hyperparameters of certain models such as α (alpha), which controls the learning rate of the model (how quickly it adapts to new data), γ (gamma) the discount factor which controls how much future reward matters and ε (epsilon) for controlling exploration e.g. how often to take a random action (explore) compared to a best known action (exploit) to retrain a model. This keeps it relevant even for more advanced users while still abstraction most of the workings of the model such as the actual process the learning algorithm goes through to find the best action based on information it is given. Also each layer of the graph and its interactivity will be abstracted apart from the very basic visible end which the client can view and interact with to edit some parameters. This along with a few more simplifications of inputs the user can make makes sure they don’t feel overwhelmed by financial and ML terms like “gradient descent” or “activation function”. Instead, they simply choose a stock, date range and prediction type.

**Iteration –** defined as “Iteration in programming means repeating steps, or *instructions*, over and over again” (BBC) This will be used to train the model while subtly adjusting weights using gradient descent over multiple epochs to become as accurate as possible and minimise the loss function. It may stop early if the accuracy exceeds a minimum threshold (usually around 80-90%)   
  
Additionally it should repeat its prediction multiple times in isolated instances and take a weighted average to find whether the selected stock is most likely to rise or fall in price by the end of the prediction time period. Without iteration this would be unoptimized and take much more space and memory than it should which is made lighter by using iteration to not repeat identical code. It could also make predictions less reliable or include anomalies without checking properly due to lacking analysis produced. Especially for my project which uses machine learning to give predictions of the future rather than a definitive answer to a query, it needs iteration to remove these possible anomalies in results so as to minimize loss and maximise benefit.

**Modelling –** defined as “the use of computers to simulate and study complex systems using mathematics, physics and computer science” (National Institute of Biomedical Imaging and Bioengineering (NIBIB)) I will use this as I need to simulate the environment of the real world financial market and the selected stock’s movement. The ML algorithm then also needs to play multiple possible futures and which is the most likely outcome based off previous trends which is a model of the actual market. This makes the assumption that past trends have correlation with the future. It ignores macroeconomic events such as political turmoil, policy changes, or sudden corporate restructuring unless reflected in recent short-term price movement. This is left out in order to simplify the algorithm as trying to also compute natural language news reports would be overly ambitious for my project.

**Visualisation –** defined as “the graphical representation of information and data” (Tableau) This is a core component of my proposed application as the user must have some sort of graph displayed to them, for them to be able to visualise the overwhelming amounts of data such as close prices, open prices, daily average etc. Part of this will include an option to choose between a normal line graph, and a candlestick chart to provide alternative ways of viewing the data. Candlestick graphs are commonly used in financial trading as they display more information (e.g. open/close price, quartiles and ranges) so may be more suitable for amateurs and can help beginners learn financial literacy. But the option of a simple line graph is still made available to provide an at-a-glance visualisation of larger market trends simple to understand for both novices and intermediates. This may allow the user to use intuition and possibly question the ML algorithm from seeing the data displayed in a comprehensible way, as the model is not guaranteed to be accurate 100% of the time, which is imperative as a mistaken prediction may cause great losses.

**Data mining –** defined as “the process of sorting through large data sets to identify patterns and relationships that can help solve business problems through data analysis” (TechTarget) I will use this to extract large quantities of data from different financial display websites such as Yahoo finance, for ease of access to all popular stock’s data. This will mostly be achieved through python modules such as yfinance, which integrates the yahoo finance database into your project for ease of access to historical market data. This can then be processed to identify trends such as steady incline, steady decline, stagged peak incline, stagged peak decline, which represent different ways a stock’s price may fluctuate. Using these, trends for a specific stock can then be used to try and fit its past performance with its present performance to estimate the trend it is currently following and will continue to follow. This can also compensate for the few flaws in input data that may cause inaccuracies in ML predictions – poor-quality data and overfitting. Furthermore, related stocks such as those from the same sector, those whose companies are partnered or working in similar situations (such as energy supply companies Vistra, Constellation energy, NextEra) can be paired together. This allows them to be used to predict each other – for example if NextEra rises, it is likely Vistra or Constellation will also rise depending on the circumstance for which NextEra rose.

**Machine learning –** defined as “a branch of artificial intelligence (AI) focused on enabling computers and machines to imitate the way that humans learn, to perform tasks autonomously, and to improve their performance and accuracy through experience and exposure to more data.” (IBM) I will train a selection of different network algorithms for the user to select one based on the time period they wish to predict for, as they differ in accuracy for long/short term based on their specialty. Some of these include: linear regression, decision trees and random forests, support vector machines, artificial neural networks and reinforcement learning. However some of these may be less suitable for my purpose such as the artificial neural networks which could be overly-complex for my project in comparison to the very slight possible improvement in the accuracy of its prediction. This method will allow recommendations to become more accurate to the stock, rather than being based of arbitrary values and wrongly extrapolated to use the same standard for every stock. By using multiple learning techniques iteratively, a linked recommendation can be made specific for the stock.

**Heuristics –** defined as “methods used to find quick and effective solutions when dealing with complex issues. They don't always guarantee the optimal solution, but they often lead to satisfactory results within a reasonable time frame” (Soft Journ) My program will use this as it needs to have a balance of accuracy, certainty and time-efficient. Due to the nature of the market, the predictions need to process relatively quickly so that they do not become outdated as soon as they are released. But as the consequences of incorrect predictions are grave, they must also employ a great sense of certainty that what it estimates will happen with a specified probability and if not, they should not be very far off. This makes heuristics essential to give relatively accurate advice in a quick time frame with reasonable certainty that it will in fact happen as said. A general consensus for accurate predictions can be taken from the range of being correct in future movement 80-90% of the time. It should also not take more than 30 seconds for one computation – if one is taking too long it may be skipped and included in a log bin to ensure any redundancies are recorded to avoid such in the future or fix it while developing. If the model hits this target of being correct within this range of accuracy and time then it can be considered “good enough” to be efficient while not compromising on quality.

However, my solution will not be needing **concurrence** as all parts of the program must happen sequentially. First the ML algorithm must be trained to a suitable degree of accuracy, then it can be fed live data for it make a prediction, then multiple isolated versions predict similar outcomes, another algorithm must check the error interval between each prediction so as to make sure no obvious anomalies are present, and if they are to remove them. Then it should use a weighted average to calculate the final prediction and feed it back into the main program which finally runs all the other subprocesses and displays the processed data in a user-friendly way to the client.

## 1.2 Stakeholders

**Client:**

TBD

**Novice end-user:**

These are normal people who have no professional experience – or otherwise – in the financial sector or working with financial tools to analyse the market and its trends. These may be people from any background and job, who simply wish to earn more interest or profit from their unused savings which likely sit in low-yield savings accounts or government bonds. They have no prior experience trading or investing and are satisfied with any small increase in their compounded increase in savings. This would make them interested in my project as it will be much more intuitive to use for these types of people, giving predictions for them to base their portfolios on without needing to do much actual analysis themselves on trends and identifying possible growing stocks. The interface will include info markers and tooltips for terms such as “volatility” or “divergence” and a short pre-use tutorial to teach them why the model may be predicting a general trend in a certain direction. This can encourage the user to get more familiar with the movement of the market and instead of entirely depending on visual prediction rather than making manual decisions. This makes my solution more suitable compared to other apps or websites, due to this guide for use and intuitive visualisation rather than the overly complex systems that already exist – typically for use by professionals and those with existing knowledge.

**Amateur investors:**

People in this category will have some knowledge in the market – more so from prior personal investments rather than commercial expertise. These people may find other solutions satisfactory as they can understand the business jargon however they may desire a cheaper alternative that gives quick, accurate predictions. Furthermore, if desired there will be an option for deeper control over how predictions are generated. For them, a separate “advanced settings” section will be provided, where they can tweak the α, γ and ε values for reinforcement learning-based models. These can be adjusted using sliders or number inputs and the effect can be observed during retraining. This level of control supports transparency and experimentation while allowing users to customise behaviours of the predictive model to suit their needs which ensures the prevailing relevancy of my app.

They can also still perform their own analysis afterwards rather than blindly following the predictions and come to their own conclusions based on factors other than just open and close price, such as a moving average. They could also be deeply in touch with corporate news so events that may cause sudden unpredictable changes in price can be identified by the user, which would not be known to the algorithm as it only looks at past performance.

**Data subjects:**

This includes companies whose stocks are being analysed and possibly influenced by the predictions made by the app. If the algorithm recommends selling a certain stock – e.g. Apple – then the prediction spreads and many people start selling, the price may decrease due to an increase in supply with a decrease in demand. This gives companies a vested interested into the predictions made to ensure sudden demand doesn’t change. However, the scale of the project and its customer base size, are insignificant compared to the giant megacompanies so realistically these stakeholders have very little stake in the project and are likely to not even consider it. Despite this, I must still consider legal and ethical implications of encouraging mass behaviour, so a disclaimer will clarify that t he tool is for educational purposes and does not provide guaranteed financial returns, to avoid legal or ethical issues in case of inaccurate predictions.

## 1.3 Research of Existing Solutions

**ProRealTime**

A screen shot of a computer

AI-generated content may be incorrect.**(**<https://www.prorealtime.com/en/web>)

ProRealTime is a company that offers trading and charting services. The charting webapp interface is shown above and it provides powerful tools for selecting and analysing different stocks. One of their tools is a linear trendline drawer, which allows the user to estimate the general trend a selected stock may follow and even has an AI guided autotrend tool, however this is very generalised and only provides a brief view into a few possibilities as to how the stock may perform. I will include a similar tool in my solution as it will show its thought process on the graph on where it predicts the most likely movement will be in the near future.

There are also a few more basic tool such as a rectangle tool, note sticker and ruler to annotate the graph as to how the stock is performing based on your own analysis. Although all these tools are manual and require much more careful analysis on the user’s front end to generate an accurate prediction. However I will not add tools like this as they can be found on free versions of many other analytical tools which do not provide much help, especially in my scenario as the analysis of past performance is done by the algorithm.

It also has an “Indicators” tab that allows the user to select and add to the graph any one of multiple analysis tools such as the moving average, volume, vortex and linear regression. However, with the exception of a few simple self-explanatory ones, they are hard to understand for an average user and beginner. This already heavily limits the ease of use for a normal user and severely limits the actual functionality for them. My solution will have a few of these once again as the algorithm itself will use some of them so it can easily be reflected onto the graph as its thought process. These will mainly include the moving average, sale/buy volume, and linear regression line.

Overall, this interface is unintuitive, the tools are too complex to use for an average person, yet not sufficient for people with expertise, and the additional tools that would be more helpful in analysis are blocked behind the premium subscription.

A screenshot of a computer

AI-generated content may be incorrect.Different indicators the user can select to display on or next to the graph:

**Trading Central**

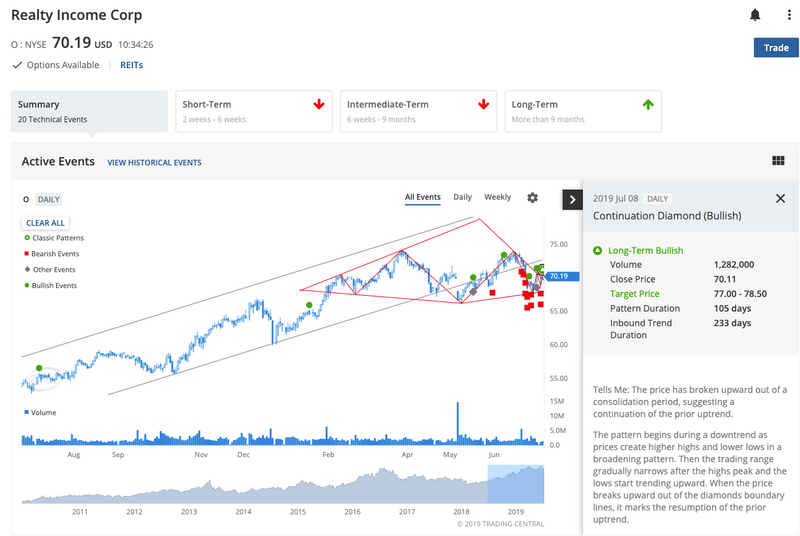
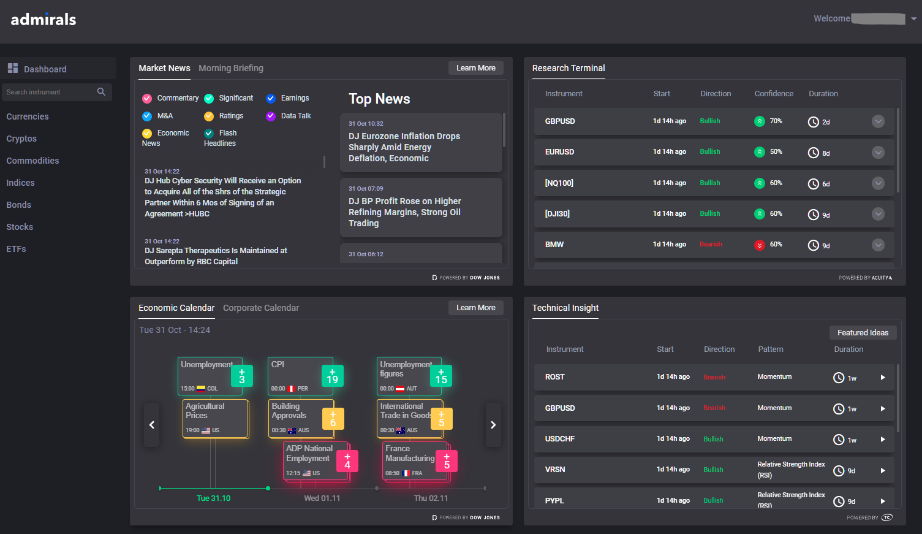
(<https://www.tradingcentral.com/tc-products/tc-economic-insight>)

Trading Central is an analytics platform which provides investment strategies using a blend of AI and expert analysis. It provides detailed charting and sentiment tools, but they are often embedded into their trading platforms rather than being standalone and while their predictions and technical indicators are quite advanced, they are full of jargon and offer little contextual guidance for users.

Similar to ProRealTime, the interface is designed for users who are already familiar with strategies and analysis involved with trading stocks. They offer a service called “Technical Insight” which analyses a stock’s price movement and gives a written prediction along with confidence levels, a trend direction and supporting evidence. This is very helpful for users with or without technical expertise as they are provided explanations and insight into why the AI made such a decision for them to then make their own decision whether there is sufficient evidence to bet on the stock’s movement in a particular direction.

A particularly strong aspect of Trading Central is that it provides simple, understandable summaries of the movement, making analysis more approachable. However, even these summaries are often littered with technical language and jargon such as “breakout resistance” or “momentum oscillator divergence”, which could alienate a truly novice user and unnecessarily overcomplicate summaries for intermediates. The interface is also heavily data-dense, with multiple widgets, graphs, and textual summaries appearing on screen at once, which may lead to information overload.

I plan to take inspiration from their use of textual explanations alongside graphs and include something similar in my project – a simple sentence summary of why the model is predicting what it is. However, I will strip away all the unnecessary technical wording and keep the focus on clarity rather than overwhelming the user with jargon. My solution will also be a separate app with all its features contained within itself rather than embedding it into third party platforms. Lastly, I would ensure only the most relevant predictions and options are shown rather than having multiple data panels cluttering the screen and user’s attention.



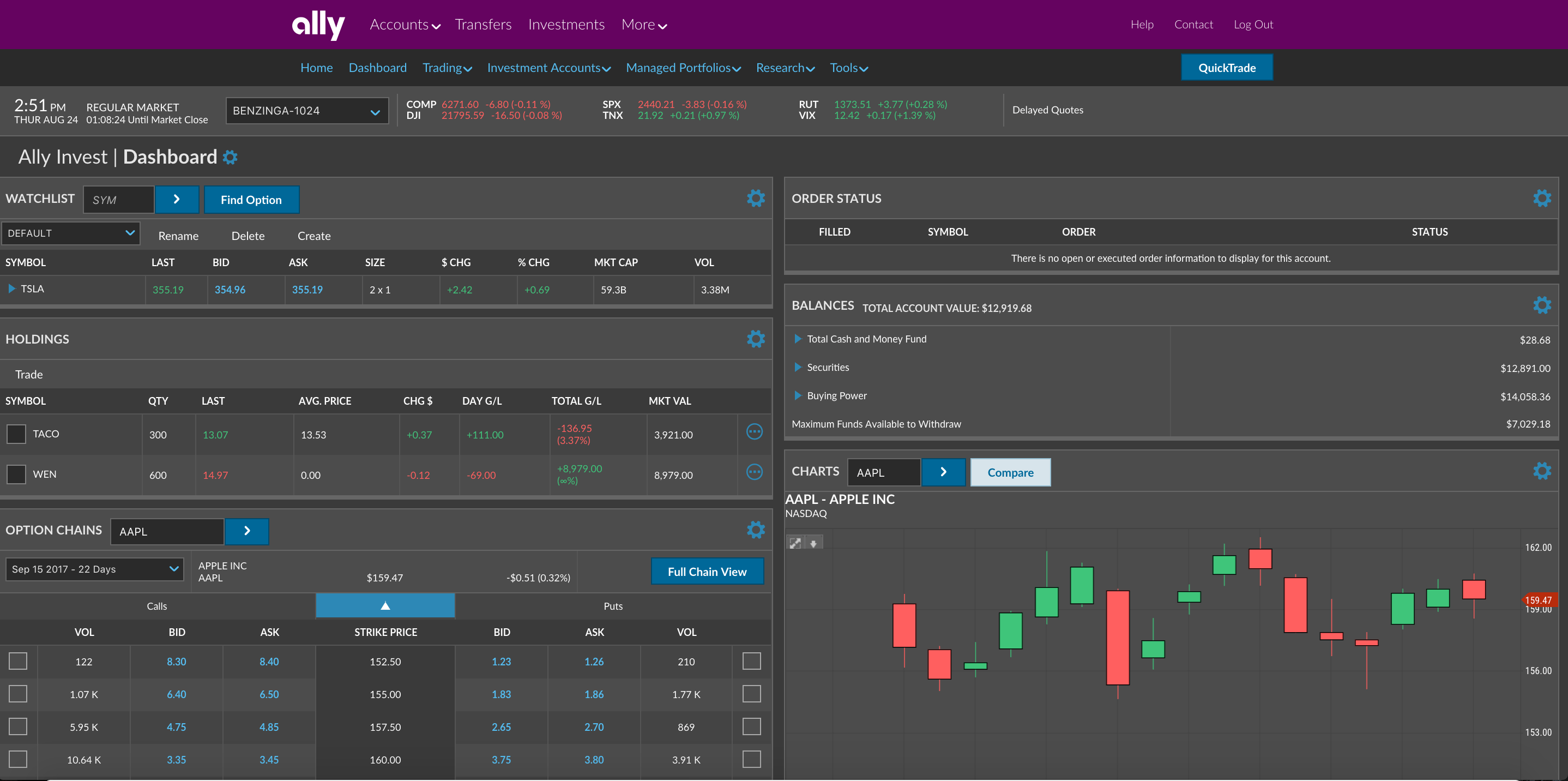
**Ally Invest**

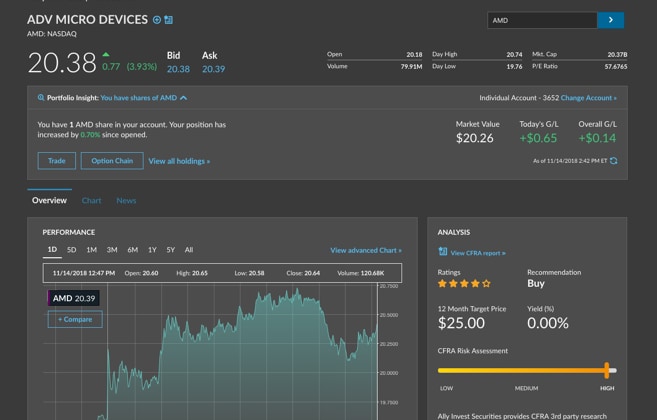
( <https://www.ally.com/invest/> )

Ally Invest is an American brokerage that attracts new and seasoned investors alike with low-cost trading and a few analytical tools. It has a stock charting feature, some basic indicators like simple moving average, and articles elaborating on investing themes. It also has a feature - the "What's Trending" tab – which highlights very popular or trending stocks in terms of recent user trades or activity in the overall market.

But Ally Invest's prediction features are limited. It has its "trending" lists, which are more of a reaction to short-term popularity, not predictions – they might not be trustworthy or even be incorrect for long-term investors. The website avoids showing detailed graphs with notes or descriptions, and although there is a clean UI, there's little or no interactivity or advanced prediction feature.

Additionally, although it tries to be novice-friendly, it does most of this in extensive text-based articles, which might be boring or confusing for those who prefer interactive programs or pictures-based learning.

I would borrow Ally's clean and modern interface design, as it is nicer than ProRealTime's or other systems' messier UIs. But I would add to this and make my predictions interactive and visible, not static or list-based. I would also avoid entirely using text-only tutorials and include step-by-step walkthroughs, hover tips, and preset learning modes. Finally, I would make sure that top-performing stock indicators, where used, are based on genuine model intelligence – not fame.



**Charles Schwab**

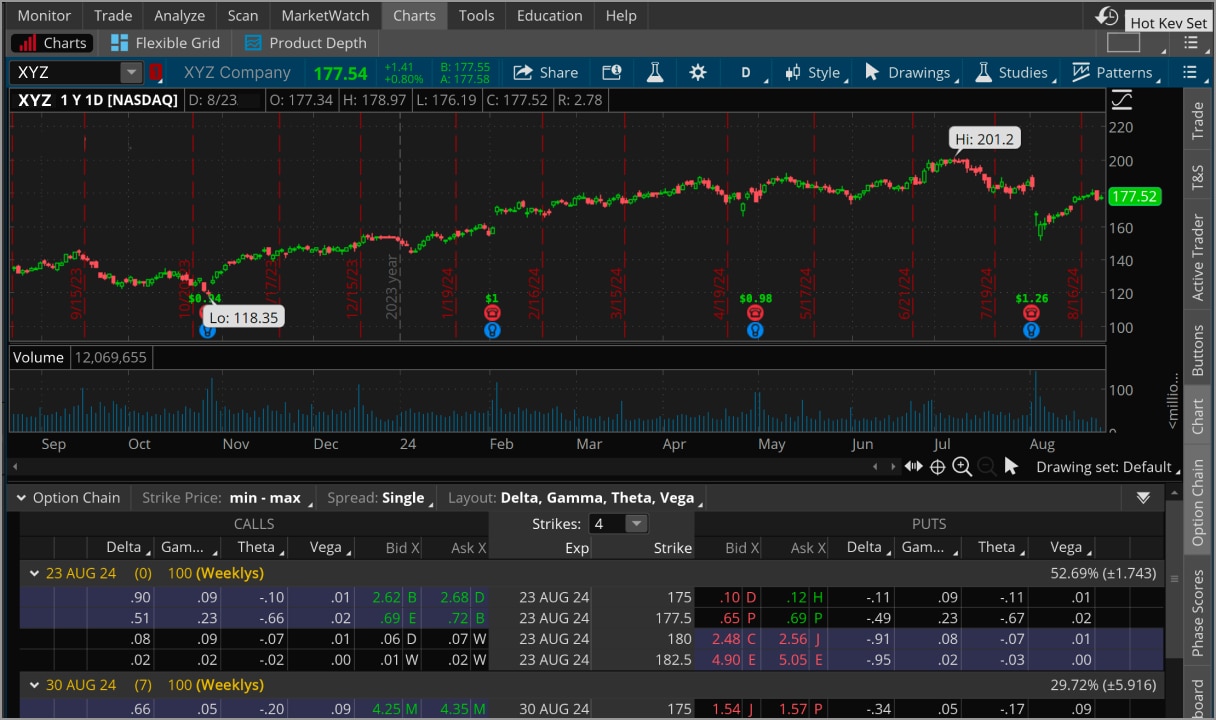
StreetSmart Edge, from Charles Schwab, is a downloadable trading platform aimed at professional traders and serious investors. Unlike many browser-based applications designed for the occasional user, StreetSmart Edge offers professional-level analysis features with the focus on customization, real-time monitoring of markets, and extensive data visualization.

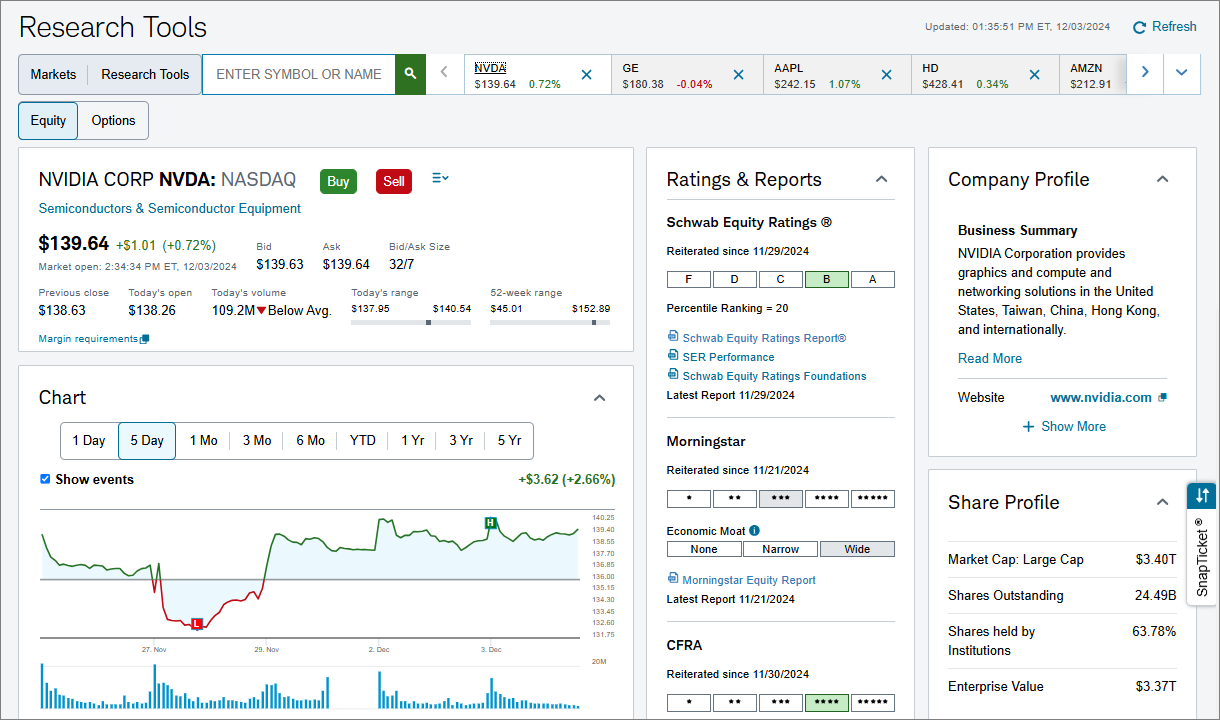
The platform includes an extensive suite of advanced features, such as real-time quotes, earnings calendars, in-platform news feeds, watchlists, and trading heatmaps, all integrated into a highly modular and customizable dashboard. Perhaps its most illustrious feature is the sector correlation map, which allows users to monitor the behaviour of groups of similar stocks – usually within the same sector or industry – in relation to each other. For instance, if the energy sector as a whole is in a strong uptrend, the map may be used to visually identify which individual companies within the group are either outperforming or underperforming relative to their peers. This tool is priceless in understanding market dynamics and is particularly useful in determining sector-wide momentum or divergence, a strategy often utilized by many institutional traders as a starting point for investment decisions.

However, while these features are effective, they come with a considerable disadvantage: overwhelming complexity. The interface of the platform is full of panels, graphs, tabs, buttons, and filters, making it visually and mentally intimidating for newcomers. Successfully operating the system typically requires not just technical financial expertise but also some knowledge of stock trading software beforehand. It also assumes that the users have some familiarity with thematic areas like beta weighting, implied volatility, moving average convergence divergence (MACD), and volume-weighted average price (VWAP)—aspects that are possibly esoteric or confusing for beginners or casual stock market participants.

Another major limitation is that several of StreetSmart Edge's more valuable features are available only to users who have a verified Schwab brokerage account, thus creating an entry barrier. Users must undergo registration, identification verification, and funding linking before they can gain access to critical tools. This requirement makes the platform less than ideal for those who want to quickly test strategies or research market dynamics without a corresponding financial commitment.

Whereas I acknowledge the strength and value of StreetSmart Edge's analytical functions, such as its power to correlate stock movement across sectors, I deliberately refrain from producing a similarly complex solution. My user base is comprised of amateur and beginning investors, many of whom might have little or no familiarity with traditional technical indicators, much less advanced trading software. My solution will apply only to the basic principle of sector analysis – for example, the tendency of stocks within a similar category (e.g., renewable energy) to move in tandem – and will explain this principle through a simplified feature known as “related movement.” Instead of depicting real-time maps or unadorned correlation coefficients, my program will detect and highlight relationships between stocks that share similar attributes and will consolidate the results into a clear and user-friendly presentation. These suggestions will be presented visually through optional overlays on charts or straightforward labels next to recommended stocks, thus allowing users to make more informed choices without requiring the interpretation of several interrelated streams of data. However, in order to retain some more advanced users I will include some more niche features that require in-depth knowledge of markets and technical skills.





## 1.4 Essential Features

My solution aims to create an interactive and accurate program that can also educate novice users into having sufficient knowledge to use the app to its fullest potential.

**Saving/Loaded Stock Data**

When the app is first installed or when a stock is loaded for the first time, the program will need to download its data (which will be using YFinance python module to extract the live data from Yahoo Finance) This will be stored in the programs memory temporarily, but that would result in the data being lost each time the program is closed and reopened, costing time in needing to redownload the data every time. To counter this, there should also be a “Commonly Analysed Stocks” tab which allows a user to save the data to a local file which prevents the need to download the same information again next time they load the app. It should also save any trends the AI algorithm identified to once again save time when the stock data is next needed to be analysed.

This is essential as it can majorly increase the algorithm’s efficiency, reducing redundancy and improving its consistency in predictions. Removing the need to redownload stock data every time the algorithm is run, can free up time to instead be used during analysis which is much more resource-heavy and every second longer it has to analyse may increase its accuracy. This also is more convenient for the user as it can reduce bandwidth usage for them and decrease waiting time to receive a prediction for education or aide in analysis. Therefore, it is essential for optimising the algorithm and being viable for quick, consistent usage.

**Refresh Data**

Alongside being able to save and load downloaded stock data to hasten future analysis, the program should also make periodic consolidations of its data. This would include, for example, every 3 hours to download the previous times’ stock movement. This ensures the data is always up to date so its predictions are never outdated or inaccurate as even small inconsistencies in immediate data could cause losses, but mostly for short term trading. Therefore, to ensure data integrity and maximum prediction accuracy, this is an essential feature.

**Trading Profile**

Another core feature of the app will be the ability to create a local trading profile, which will allow users to build a profile of their current trades (if any), trade prospects to keep an eye on for major movement, and a recommendations tab which categorises all their trades to suggest other stocks in similar markets (that have already been downloaded), or with similar performance. Different users will have varying tolerance levels for volatility and the time period over which they want to make money on, so this can also store whether to prefer long term or short-term gains and a limit for how certain the AI should be before making a prediction/recommendation.

This makes it essential as it broadens the options the user has and can take into consideration to trade. Instead of being limited to companies they choose themselves and request an analysis of, the program will routinely download new stock data within the same market segment to identify market trends and can then give more accurate predictions. This will help the user further as newer users will have less work to do in actively finding more prospective companies in the market to invest in as well.

**Predictive Algorithm**

An irreplaceable tool in the solution should be the main predictive algorithm which analyses a selected stock’s historical data. It does this by training a simplified, blank clone of the main algorithm to be familiarised with the company’s performance and any trends that have been seen. It then discloses what it learnt about the stock to the main program, which can make a few possible predictions using the trends, past turbulence and any other patterns in the data the clone identified. Finally, it determines which is most likely to occur, using other indicators, such as identifying where possible bubbles will burst and feeds the final prediction to back to parent program to be displayed on the interactive chart.

Predicting a stock’s performance using AI is needed in this solution as it is the main focus point to compel users to choose my solution over other existing solutions. Without this element, it would be left a bare-bones application for selecting and displaying historical stock data, which already has plentiful solutions for free, like Yahoo Finance which will source the data my program will be using. Furthermore, this feature is what advises novice users on which stocks may potentially increase short or long term, and aides advanced users in their own analysis. Therefore, it must be included to complete the solution as intended.

**Adjusting Prediction Timeframe**

Included in the user’s trading profile, they will have a preferred timeframe within which they want to make money in from one stock, e.g. “Make >5% return within a week”. This should be adjustable at any time and affects how the AI makes a prediction by changing the “minimum pattern width” that determines how many times a trend must occur before it is considered as a likely outcome in the future. This will have sizable effects on which stock will be recommended and predicted to give high returns within the specified time. It will also allow users to differentiate themselves between those seeking short-term returns, and those looking for a slow growth of their wealth over years or more.

As such, this makes it an essential feature to the program’s functionality to control the tolerability toward volatility, and allow the user to request analysis for returns within any timeframe.

**Graphical Interface**

The solution needs to include an interactive graph, with different indicators that display as the user’s mouse hovers over different sections. For example, when over data bars for time periods during which the selected stock’s price has increased to close, it should offer the time high, low, average, open and close. It should also offer a total movement and percentage change which will be the same for period where it has decreased instead, but then it would show the total decrease instead. There should also be a display indicator along both axis of price (y-axis) and time (x-axis) precisely where the mouse is hovering to show the price at that exact time. There would also be a bar along the left side for other interactive tools that can be placed on the graph like a line tool, or a difference measure.

This is essential for my program as it allows users to view the data in a visual way, not limited to a list of incomprehensible numbers, so market trends can also be identified by the user. As AI is not *yet* perfect, the predictive algorithm cannot be completely trusted and will be wrong a certain amount of the time, due to many other external factors as well. This makes the graph imperative for the user to, at a glance, spot whether the AI could be incorrect.

## 1.5 Limitations

My system will not be able to update with live stock data. I considered using the python module “Ascyncio” to make the program download live stock data, preferably every minute or half a minute or to stop processing to allow the script to redownload new information every minute. However, this will not be an option as this would either take much more coding from me which would complicate this project even further, but I am not confident in learning one more new module, especially one as complex as this. And the other option is also not viable as this could greatly slow down the predictive algorithm, needing to stop every minute, which is already expected to take quite some time. This could still possibly be implemented, but the impact of this feature would be minimal, if not completely irrelevant due to other complications described next so the slight extra data received, will not be able to be considered anyway. The user will also not notice this as the core feature will be the prediction rather than the data the user has access to, as there are already free alternatives such as Yahoo Finance for this.

The app will also not be able to consider any stocks that have never been downloaded before. One idea was that it had access to a pre-downloaded dataframe of the most common stocks, and their corresponding market segments, market caps and max shares on the market. However, upon research I did not find a freely available and downloadable data dump of this desired data and trying to manually create such a document would take much more time than I would like to devote to a non-essential feature. As a compromise, I will add a feature that, when a stock is selected to analyse, a prompt will ask the user if they would like to additionally analyse other companies in the market. This will require the user to perform their own research and provide the ticker symbol (unique code that identifies a stock) for any companies they would like the AI to additionally consider and analyse in accordance with the main one. Although this is not an ideal solution, it makes the feature implementable within a reasonable time and effort.

One more limitation is that the predictive algorithm will only consider down to a minimum of 1-hour intervals between close and open price data. An essential feature of the program is that the user can select the resolution that the algorithm uses to identify trends and patterns, in order to optimise the accuracy for differing terms of desired returns. In respect to this, an upper limit was considered and decided that it should only mark open and close prices for a maximum of 1 day apart, in order to still have plenty of data nodes to consider. More difficulty was found, however, in selecting a lower limit to analyse based on a short period’s open and close price, while still ensuring the program will finish within a reasonable amount of time. While it could theoretically consider down to minute-sized intervals, this would exponentially increase the amount of data nodes and time it would take for the program to complete. Not to mention, the added complexity for deciding whether a price is a reasonable average for the few minutes, during which a stock position could be changed, or whether it was a speculative price. Therefore, I determined limiting the minimum time interval to 1 hour would still allow accurate pattern recognition for short term gains while still optimising the efficiency of the AI predictive algorithm.

Another consideration I had to make due to the lack of time is simplification of data. Stocks have several attributes and, as such, considering every part would dramatically increase the analysis time, making the program unviable. Some of these attributes include open/close price, max price, min price, sales volume, and more. Naturally, the open and close price will be the main focus point for the predictive algorithm, heavily relying on these to spot patterns and make decisions. I considered also including the max and min prices within the time interval in the pattern recognition as these could maximize gains, more in short term where the user wants to sell at the highest price possible as quickly as possible. However, this would also require taking into account the sales volume made at each price point and the time at which the stock held at that price. This is because prices are highly speculative in volatile stocks so although a stock may have a max price at, for example, £110, it may have only momentarily approached that value with very few trades made at that price – most would have been made at the stable £105 where the stock was at on average in the past hours or days. This means that in order to consider any more than just open and close price, an additional 3 attributes must be included, exponentially increasing analysis time. I did also explore the possibility of including them more rarely, such as every 3 or 4 other time period where the main open/close price is used. Though, this would not be viable either as it could skew the data analysis to those few times where the extras were included or it would over generalise data, not being faithful to the actual time period the user chose. With all this in mind, even if I included this feature, it would have limited impact on most of the program, only realistically impacting short-term predictions, and even then, would minimally increase the profit on a stock.

A major difference my app would have, as compared to professional tools in the market, is that it will not have any human input to oversee the AI, such as employees to proof it before release. This means it is up to the user’s discretion to trust the AI, which to more inexperienced users, could be a cause of anxiety and distrust. It also increases the chance of an incorrect prediction being given to the user, especially due to the fact the AI only considers historical market data, with no reflection of current news. As such, a prediction that could have been culled by a human who learned of news that a new bill could sanction the company, instead is put forward with complete confidence by the AI, possibly leading to great losses. My program will be a self-hosted algorithm, which is why it will not be possible to monitor the AI, not to mention this is not a commercial project so I cannot realistically monitor the AI constantly. In light of this, I contemplated including an accurate “confidence” metre for the AI’s predictions so that if it has a low confidence, the user could be more cautious and do their own research. However, as I was already deciding to include a multitude of complex features, I determined to instead include a disclaimer whenever the user uses the app and when they load it that encourages them to not completely trust the prediction and do their own research into global news related to the company.

## 1.6 Hardware and Software Requirements

|  |  |
| --- | --- |
| **Hardware** | **Software** |
| * Processor clock speed of at least 2GHz * 4GB RAM * 16GB free hard disk space * 4GB VRAM (more is preferrable) * Display capable of 1080 x 768 pixels * Standard UK-layout keyboard * Two-button mouse | * VS code (for development) * Python modules including: TBD (for development) * Windows 10 or later OS * Internet access |

*Hardware:*

The machine learning and predictive algorithms will be quite computationally demanding, but mostly for the GPU rather than other requirements. Therefore, the processor itself only needs to be 2GHz, which is below the average amount modern devices have, enough to sufficiently render the graph and plot different analysis tools. This is important since the user needs to be able to interact with the graph to streamline the process of viewing the AI’s prediction and checking whether it has no obvious flaws such as ignoring present news. While less processing power would still be adequate, it would be laggy and painfully slow, so the user may get frustrated without the sufficient computing power.

Furthermore, 4GB of RAM if needed to store the app’s interface, the algorithm’s data and the target stock’s historical performance. Thought, this amount is below the minimum amount pcs usually have which guarantees the user will have sufficient specs to run the app smoothly. This is important as one of the key features is to be able to store a stock’s data after the first analysis, so it needs to keep this data in memory, and later write it to secondary storage. While the algorithm runs, the app will also need to store profile data, another key feature, and actively write to it. Therefore, depending on how many stocks the user is loading, it may not meet the limit of this amount, but for its guaranteed smooth performance, the hosting device should have this amount as a minimum.

Then to store the profile and stock data once the app is closed, it needs to save it to a permanent file on the user’s drive, which requires at least 16 GB free space. If this amount is not available, there will be USB ports available to substitute in an external storage device like a USB stick to store this data. This is important to the app as it needs to save this data as one of its essential features, which it cannot do without writing to secondary storage. This amount has been chosen to leave sufficient room for as many stocks the user wants to analyse and profiles they want to create, along with space to store the ‘memory’ of the algorithm. While realistically, it should not reach this amount of storage required, people would like to be affirmed they will not run out of space for it.

The most important requirement is the amount of VRAM available in the user’s graphics card. I have set this as 4 GB as this is my pc’s spec. This is not available for integrated graphics and requires a more recent graphics card model, but it is necessary to run the learning algorithm. A key essential feature is the predictive algorithm which can recommend whether to buy or sell the stock, and for this algorithm to run, it needs plenty of power to analyse, therefore needing the GPU heavily. As with the processing power, having less would still allow the program to run, however it would increase the wait time for a result exponentially. According to my research, people would not like to wait much more than several minutes for a prediction, therefore this amount of VRAM is necessary.

For peripherals, a display is needed of at least 1920 x 1080 pixels. This is a standard for office computers and will be available as a minimum on all machines expected to use this app. This is important to ensure the graphical data is displayed in high quality, and the user can seamlessly interact with the graph to perform their own analysis.

A standard UK-layout keyboard and two-button mouse will be required to control parameters and select stocks. The keyboard is needed to enter the ticker symbol of the desired stock to analyse. Then the mouse will be used to navigate the interface including, but not limited to, the type of predictive algorithm, time frame to analyse by and graphical interactive tools such as the line tool. These are important as without them the user would not be able to interact with and control the program.

*Software:*

For development, VS Code will be needed along with the necessary python interpreters and IDE to run and debug the python script. It has wide support and has powerful extensions for debugging, Git integration and smart autocomplete. This is necessary as it allows for easy modularisation and file management, key for an application like this one, which will include many different files for saving data and holding different algorithms. VS Code is also compatible with all major platforms and runs smoothly on devices that meet the hardware requirements listed above, making it ideal for development.

The app itself will be designed to run on Windows 10 or later, as it is the most commonly used OS which ensures compatibility with the majority of users. Windows 10 gives access for all necessary permissions, memory allocation and display settings to run this complex app with graphical output and data persistence. It also supports the necessary driver support for CUDA-enabled GPUs which is necessary for my program which will be using PyTorch for machine learning.

Finally, the user will be required to maintain an internet connection while installing and using the app. This is needed to download the relevant python dependencies if not already installed such as yfinance, pandas and matplotlib. During use, an internet connection is essential to fetch real-time stock data using the yfinance API to download relevant data from yahoofinance.com. Since on of the app’s key features is the ability to analyse live market trends, a stable internet connection is critical for being able to analyse new stocks and recent movement updates. Without it, the user would be limited to outdated cached data, reducing the accuracy and reliability of the predictions.

## 1.7 Success Criteria

1. I should have a user-friendly interface. This includes being easily navigable by new and inexperienced users, and all tools being accessible to them. There should also be tooltips to briefly explain what each tools does. This will be measured via giving stakeholders a questionnaire after they have used it, where they will be asked to score its usability and accessibility out of 10, with a target score of at least 8.

**User-friendliness score > 7**

1. The prediction should have at least 80% accuracy. Over a period of time (around 1 month) I will actively use the app, note the prediction it produces and check after whether it was correct. I will do this for all the different types of predictive algorithms and over multiple different time periods (with a max of 1 week’s return due to time constraints). Then I will record whether the AI was correct within the same direction e.g. if it predicted it goes up 5% and it goes up even a few percentages it will be a success. Then I will calculate how often this occurs and if the success rate is at least 80% it will be considered a success.

**Prediction accuracy > 80%**

1. The model should produce a prediction within 2 minutes on a device with the minimum specifications. Over the period I will be testing the app, I will also measure how long it takes to complete the prediction. If the average amount is clear of 2 minutes, it will be considered successful. This ensures the user doesn’t get frustrated waiting for the analysis to complete.

**Time to predict < 2 minutes**

1. The app must support at least 3 different machine learning models (e.g. linear regression, decision trees and random forests) The app should have a drop-down menu somewhere in the interface allowing the user to choose which model they want to use to predict. This allows more advanced users to select models they are more familiar with and possibly they believe are more accurate than others. The models will also differ in accuracy depending on the time frame for the prediction so the user should be recommended which to use depending on how fast they want to see returns in. If there are 3 working choices, each also fulfilling the above 80% accuracy criteria, this will be considered achieved.

**3 different ML models**

1. The program should be able to display stock data on either a line graph or a candlestick graph. Line graphs are useful for brief analysis and viewing market trends over a longer period of time. They offer the user a way to be able to spot patterns at a glance, which could be helpful to spot potential inaccuracies in the prediction. While candlestick graphs are much more detailed showing open and close price, local minimum and maximum sale prices. For this criterion to be achieved the user should be able to toggle which graph is displayed and the transition should be rapid.

**Toggle between candlestick and line graphs**

1. The app must allow the user to analyse the history of as many different stocks as they would want. This means there should be no limit on how much stock data is stored with the app’s profile on their computer system. I will test this by trying to analyse over 100 different stocks over the testing period, and if at the end all of them still work and have maintained data integrity compared to the actual stock values, it will be successful. This fulfils the essential feature of being able to save stock data to save time from downloading the data using the yfinance API each time.

**Save and load stock data locally**

1. It should also take no more than a few seconds to extract and load the data from the saved file ready to be analysed. This requires an efficient reading algorithm to understand contextual data within to know which parts need to be fed to the prediction algorithm, displayed to the user, or to request additional data from the yfinance module. If it consistently takes just moments for the predictive algorithm to be ready to run after loading data from a save file, this criteria will be met. It is another point where the app needs maximum efficiency to not frustrate the user.

**Load local data within moments**

1. Advanced users must be able to adjust hyperparameters to retrain and fine tune the model if they have more experience, and may have greater familiarity with a more/less risk-taking algorithm. Within the sidebar or settings, there should be an option for “advanced settings” with a warning that tampering without proper knowledge may make the model less accurate. At the very minimum, the user should be able to change the alpha, beta and gamma hyperparameters. While the base settings will be as optimized as possible to me, a more advanced or experienced user might rather change it to suit their securities. This would improve retention on a greater range of users, and could even help intermediates learn how these affect the outcome prediction.

**Advanced settings - adjust hyperparameters**

1. There will be capabilities to create local profiles with different attributes that slightly altercate the program’s process. These features include their risk tolerance, stock prospects, categorised downloaded stocks, real portfolio, preference on long/short term gains. The user should also be able to create multiple difference profiles and switch between them within the app and in parallel the algorithm will change to fit these personalities. This further personalises the app to the user, allowing more flexibility to change as they see fit and greater control over the program.

**User profiles**

1. A recommendations system should analyse the user’s profile including their downloaded stocks and portfolio to find similar stocks. This will include mostly companies within the same market sector such as energy or AI, and will search for stocks using yfinance integration, to locate similar performing stocks. These will then be displayed in a “recommendations” tab on their profile with an option to download and predict its future movement. This will all be done in the background automatically for user convenience with easy access.

**Recommendation system**

# 2. Design (15 marks)

## 2.1 Problem Decomposition

A diagram of a algorithm

AI-generated content may be incorrect.

**Manage, save profiles:**

Within the app interface, there will be a button to open a user profile section. Within this new popup app space, all subfunctions to do with the user profile will be controlled. This could have been done within the same app space as the rest of the program; however, I felt that with all the different features the user profile has, it would clutter the space where the focus is on showing the graph and predictions.

Here they will be able to most importantly create and delete their accounts, and all data will be saved in an encrypted file within the same file location to maintain user data protection. Due to technical limitations, it is not possible to add a way of recovering an account with a forgotten password unless I, the algorithm designer, decrypt the data to access login credentials.

A main feature is also the ability for the user to add/remove stocks to build their portfolio. This will be easily accessible, being the main visible feature upon opening the window, showing a basic overview of the first few stocks in their portfolio. This will be customizable and stored within the user’s profile data.

Lastly, there will be a section on the right for toggles and sliders to do with the user’s preferences, such as their country, sector, and risk tolerance. These will be accessed by the main algorithm to slightly edit the way it performs predictions. I considered including this as part of the algorithm pane; however, it seemed inconvenient to make the user re-enter their preferences before each prediction. Despite this, there will still be two sliders before the main prediction is made for risk tolerance and time period in case the user desires a change for a specific prediction.

**Interactive graph:**

The main app window will contain the graph and tools to edit the graph. The user will be able to search for a stock via its ticker symbol, and the algorithm will find or download the data to display it on the graph. This is done so that the user can visualize what the predictive algorithm is outputting, to spot possibly flaws in the processed data. This works as a type of validation for the data, using the user so that they can be more confident in what the program is advising them will happen in the future.

The interactivity of the graph includes many features such as hover tooltips, line drawing tools, and annotations. These will allow the user to perform their own analysis on the performance data and draw their own conclusions to confer with the AI analysis about. I contemplated adding more features, however this is not the main focus of the application, as well as the fact that these simple features should be plenty to satisfy most users which research showed will be intermediates or beginners. Along with this, scroll of the mouse will perform zoom in/out functions to view the graph for longer or shorter periods of time. This can reveal a greater general trend in price movements that the AI may have missed if it was focused on short term gains.

Lastly, there will be a menu selection along the top to manage the graph. The user should be able to change the focused stock, add multiple with an overlay, show AI analytical specifics and save the chart layout. These are basic features that compliment the interactivity without lacking extra details, not needed by the necessary interactive elements.

**Make predictions:**

The central part of my app is its ability to use multiple trained AI models to analyse a stocks data and perform a calculated prediction according to a variety of parameters as to the future movement of that stock. The 3 different models I will use are Linear regression, Random Forest and Reinforcement learning. When the user queries the algorithm to perform, a drop-down list will allow the user to select which model they would like to use. Advanced users will also have the option to manually edit hyperparameters of the models within a hidden section before requesting the prediction.

The algorithm will then access the relevant model’s saved data, and load all the stock data, edited hyperparameters based on preferences, and time period to predict for. After analysing the data and comparing to its memory, it will produce 3 different predictions, with confidence levels based on how likely it believes that outcome to occur. Ideally there should be 1 prediction with the majority of confidence (>70%).

**Web hook:**Using the python module yFinance, the algorithm will connect to Yahoo Finance and download the historical stock data of those that are being queried. It will also be used to fill holes in data or update stored data with newer movement. It should only be used once at a stock’s first initialisation, after which the data will be stored locally and accessed from the local save file.

**Notification system:**

Every time the program does an automatic update of data, it will check the percentage change to the new close price. If there is significant change (>2% in an hour) it will push this notification. This will also be adjustable in the user’s settings/preferences where they can change the percentage significance or ignore certain stocks entirely.

*I have broken down the problem by significant processes, rather than which of them occur concurrently/interactively with each other as it lends itself to be simpler to code and test. This will allow me to test each system separately without relying on another making it easier to debug and test each module.*

## 2.2 Structure of the Solution

A diagram of a display chart

AI-generated content may be incorrect.

A diagram of a profile

AI-generated content may be incorrect.

A chart with yellow rectangular shapes

AI-generated content may be incorrect.

A diagram of a company

AI-generated content may be incorrect.

## 2.3 Algorithm Design

**Profile:**

[Sidebar Title]

***find\_ticker\_symbol* –** This function takes the input from an interactive box and searches yahoo finance for the correct link to the desired stock and returns it to the main program. This could have been part of a larger **download\_data** funtion, however then the code would have to be repeated for the **update\_data** function or would make a large **data\_consilidation** function. This would make debugging harder and make one function to complex. My solution on the other hand also lends itself well to a validation check to make sure yahoo finance is responding properly, possibly saving time in debugging where the problem isn’t the code but the website.

***add\_stock / remove\_stock*** – These are part of the **user\_portfolio** class and let the user edit their portfolio. This simply appends or removes a stock’s ticker symbol from the array **self.ticker\_symbols** a class attribute, when removing no validation required as they select from a list shown by the algorithm that it knows exists within the user’s database.

***portfolio\_performance*** – This will display the user’s portfolio stocks on the graph after saving the current state of the graph as a temporary file preventing the user from losing unsaved work. Then, for every ticker symbol in the portfolio, the corresponding JSON file is read nad passed into **load\_mulitple**. I decided to implement this procedure seperatelyt rahter than merging it into a general **load\_data** function to ensure the logic wouldn’t become bloated with checks on how many and which stocks to display. By having a dedicated function, the debugging and readability is enhanced, lending to expansion and test as well.

*Portfolio:*

# Following will be class functions of “User\_Portfolio” inherits attributes from parent “User\_profile” class unless specified otherwise

# upon click of “add” button will call the function with the search query parameter

Function find\_ticker\_symbol(input\_code): # default function

If input\_code is None:

display pop\_up “Must enter code containing characers”

return False, None

connect yfinance

ticker\_symbol = yfinance query matched result

If ticker\_symbol is None:  
 display pop\_up “That ticker symbol does not exist”

return False, None

return True, ticker\_symbol

Endfunction

Procedure add\_stock(self, search\_query):

exists, ticker\_symbol = find\_ticker\_symbol(search\_query)

if not exists:

return

self.ticker\_symbol.append(ticker\_symbol)

UPDATE\_SAVE\_FILE(self)

return

Endprocedure

# upon click of “remove” button with a selected portfolio item

Procedure remove\_stock(self, selected\_item):

self.ticker\_symbols.remove(selected\_item)

UPDATE\_SAVE\_FILE(self)

return

Endprocedure

Procedure portfolio\_performance(self, Graph: class):

If Graph.is\_live(): # checks if graph already has loaded stock

Graph.save\_state(“temporary”) # stores the current state of the graph temporarily

stocks = []

for ticker\_symbol in self.ticker\_symbols:

stock\_data = file.read(f“\saved\_stocks\{ticker\_symbol}.json”)

stocks.append([ticker\_symbol, stock\_data])

Graph.load\_multiple(stocks)

return

Endprocedure

Procedure create\_save(self, profile\_name):

[Sidebar Title]

***create\_save* –** This creates a new user profile and saves it as a JSON file for later retrieval if the username is unique, else it will display a warning and return without saving anything. If the name is valid, is gathers all the relevant attributes from the user object and writes to a file as a dictionary. This ensures smooth reading and writing of user data for the future. I chose to keep this logic separate from profile management and saving to once again smoothen testing and debugging.

***load\_save* –** similar to the previous function however it only saves to the currently logged in user’s profile. It will be called whenever the user performs new calculations e.g. adding/removing stocks to their portfolio, changing preferences etc.

***prefer\_exchange* –** this saves the users preferred country to trade from – e.g. NYSE, London stock exchange etc. This will have minimal impact, but when the program finds stock data it will either prioritise rates from one or another stock exchange.

***prefer\_sector* –** similarily, this has minimal impact on the algorithm, however it can help guide the algorithm in finding the correct stock and when making recommendations on new stocks, it will have an increased chance of recommending those from this sector. It validates from a JSON of existing sectors to prevent invalid inputs and ensures the algorithm is only working with recognised sectors.

***prefer\_risk\_tolerence* –** Based on a scale of 1 to 10 from the ui, it will be used later in the algorithm when a prediction is made. This will tweak hyperparameters to give weight toward safer or riskier investments.

***prefer\_target\_gain* –** This allows the user to set a target to aim for with their investing. It will aid the algorithm further with accuracy as if their target is slightly above or below what would be expected by their risk tolerance. This allows the algorithm to make more accurate predictions the user desires. The function also checks to make sure it doesn’t deviate too far from their risk tolerance to prevent unrealistic expectations.

for file in file.read(“\user\_profiles”):

if file[“name”] == profile\_name:

display pop\_up “Profile name already in use”

return

profile\_data = {

“name”: profile\_name,

“ticker\_symbols”: self. ticker\_symbols

…

}

file.write(f”\user\_profiles\{profile\_name}.json”, profile\_data)

return

Endprocedure

Procedure load\_save(self, profile\_name):

If profile\_name not in “\user\_profiles\”: return “Invalid name”

load\_file = file.read(f”\user\_profiles\{profile\_name}.json”)

self.stock\_links = load\_file[“stock\_links”]

…

return

Endprocedure

*Prefferences:*

# Within new window, user has the profile displayed and can edit certain information

# Within “User\_profile” class

Procedure prefer\_exchange(self, exchange):

self.preferences[“exchange”] = exchange

return

Endprocedure

Procedure prefer\_sector(self, sector):

sectors = file.read(“existing\_sectors.json”)

if sector not in sectors:

display pop\_up “That is not a valid sector”

return

self.preferences[“sector”] = sector

return

Endprocedure

Procedure prefer\_risk\_tolerence(self, value: int): # value will be taken from a slider in account section from 1 to 10

self.preferences[“risk\_tolerence”] = value

return

Endprocedure

Procedure prefer\_target\_gain\_loss(self, percentage: float):

deviation = percentage – (self.preferences[“risk\_tolerence”] \* 3 )

if | deviation | > 10:

display pop\_up “Your target gain varies too greatly compared to your risk tolerence”

return

self.preferences[“target\_gain”] = percentage

return

Endprocedure

# time\_period will be inputted via a drop down box in the user profile page so no validation needed

Procedure prefer\_immediate\_returns(self, time\_period: string):

match time\_period:

case “short-term”:

self.preferences[“immediate\_return\_scale”] = 3

case “medium-term”:

self.preferences["immediate\_return\_scale”] = 2

case “long-term”:

self.preferences["immediate\_return\_scale”] = 1

return

Endprocedure

***prefer\_immediate\_returns* –** This sets whether the user is looking for short-term, medium-term or long-term returns. It will map to a numerical return scale which can be later used by the algorithm when calculating predictions. This further makes sure the algorithm produces the most helpful prediction for the user as stocks are volatile, going up in short term but may go down after a year or more.

*These functions are separated into small, modular procedures because each preference represents a distinct and independent aspect of the user’s investing style. By keeping them separate, the code is easier to maintain, test, and expand later (e.g., adding new preferences without rewriting a large, combined function). This approach also makes the user profile highly flexible, as each part can be adjusted individually without affecting the others, which directly supports the app’s goal of being intuitive and customisable.*

**Display chart:**

[Sidebar Title]

***fetch\_stock\_data* –** This function handles retrieving data for a specific stock query. If the ticker symbol doesn’t exist, it safely exits, preventing errors from invalid queries. If the ticker exists, the function then checks whether data for that stock has already been saved locally. If not, it connects to Yahoo Finance and downloads the data in real time. If the stock is already stored, it simply returns the cached data from the local files. This design balances efficiency and reliability: fetching live data ensures accuracy, while checking local storage prevents unnecessary repeated downloads. It also reduces dependence on an internet connection for every action, which is important for performance.

***display\_graph* –** This procedure is responsible for actually plotting the data on screen in either candlestick or line graph format. Once the relevant stock data is retrieved via **fetch\_stock\_data**, it processes the raw input into a structured dataset. Then, depending on the user’s chosen graph type, it calls the appropriate plotting method from the Matplotlib library. The benefit of separating data fetching from graph rendering is that each can be tested and maintained independently. For example, improvements to graph design (like adding a moving average overlay) can be made without altering the fetching logic.

***update\_data* –** This asynchronous procedure runs in the background and periodically updates stored data every hour. It loops through all files in the **saved\_stocks** directory, re-fetches the latest data from Yahoo Finance, and then checks for differences before updating. This ensures the stored data does not become outdated, while also avoiding redundant downloads if nothing has changed. Making this an async background task is crucial because it ensures the user can still interact with the application while updates are happening, improving usability and efficiency. This completes success criteria 2: the app must update stock data automatically and on demand, ensuring data freshness without user intervention.

*Loading data:*

# yfinance module downloads new data from yahoofinance.com

# functions will be class functions of “Graph” class unless specified otherwise

# Function will be called when a stock is queried to be loaded

Function fetch\_stock\_data(search\_query): # default function

exists, ticker\_symbol = find\_ticker\_symbol(search\_query)

if not exists:

return None

files = os.read(“\saved\_stocks\”)

if ticker\_symbol not in (file.filename for file in files):

connect yfinance

download data

return data

else:

return match file data

Endfunction

Procedure display\_graph(self, stock\_query, price\_scale\_y: float, time\_scale\_x: string = ”week”):

data = fetch\_stock\_data(stock\_query)

data.process( ticker\_symbol,

date\_time,

open\_price for time\_scale\_x,

close\_price for time\_scale\_x,

max\_price for price\_scale\_y,

min\_price for price\_scale\_y )

if self.graph\_type == “candlestick”:

mplot.candlestick(data)

if self.graph\_type == “line”:

mplot.linegraph(data[ticker\_symbol, date\_time, average\_price(open, close price)])

return

Endprocedure

# Periodically updates the saved data every hour

async Procedure update\_data():

files = os.read(“\saved\_stocks\”)

for file in files:

connect yfinance

down\_data = fetch\_stock\_data(file.filename)

new\_data = file dissimilarity to downloaded data

file.write(new\_data)

return

Endprocedure

*Interactivity:*

[Sidebar Title]

***hover\_tooltip* –** This asynchronous procedure continuously checks if the user’s mouse is hovering over a data point on the graph. If so, it matches the date time to the dataset and displays the specific details where the mouse is hovering (see 2.4). This feature enhances the app’s usability by allowing the user to quickly see key information without cluttering the chart with text. It makes the app approachable to novice users while still being useful to more experienced investors who need precise values.

***draw\_line* –** This procedure allows the user to draw a straight line if the line tool is selected, which can be used to highlight trends or suppot personal annotations. By integrating with Matplotlib’s plotting tools, the feature is consistent with the graph’s style and does not disrupt the existing data display. This is a valuable addition because it provides interactivity and customisation, helping users perform their own analysis instead of relying solely on the algorithm’s predictions.

***annotate* –** This procedure lets users click a point on the graph and place a text box containing notes. For example, they could annotate “Company earnings release” or “Prediction anomaly.” The text box is drawn directly on the graph, creating a visual reminder of key events or personal insights. This functionality increases engagement and personalisation. By allowing annotations, the tool supports both educational and practical use cases, helping novice users learn and advanced users record detailed strategies.

# Most features will be from within the matplotlib library in python

# Within graph class

# continiously checking whether the mouse is hovering over a spot to show tooltip

async Procedure hover\_tooltip(graph, cursor):

x\_pos = cursor.getx()

y\_pos = cursor.gety()

if {x\_pos, y\_pos} in tooltip\_lookup\_table:

display tooltip\_lookup\_table[{x\_pos, y\_pos}]

return

return

Endprocedure

Procedure draw\_line(graph, cursor):

await click

x\_pos\_1 = cursor.getx()

y\_pos\_1 = cursor.gety()

await click

x\_pos\_2 = cursor.getx()

y\_pos\_2 = cursor.gety()

if ((x\_pos\_1, y\_pos\_1), (x\_pos\_2, y\_pos\_2)) not within graph:  
 return

mplot.line((x\_pos\_1, y\_pos\_1), (x\_pos\_2, y\_pos\_2))

return

Endprocedure

Procedure annotate(graph, cursor):

await click

x\_pos = cursor.getx()

y\_pos = cursor.gety()

box = draw.text\_box(x\_pos, y\_pos)

box.text = input()

return

Endprocedure

*These functions are separated because each represents a distinct aspect of the data pipeline: fetching data, displaying data, updating data, and interacting with the graph. If all these were bundled together, the code would be more difficult to maintain, test, and extend. By decomposing them:*

*Fetching and updating can evolve independently (e.g., switching APIs).*

*Display logic can expand (new chart types) without affecting data retrieval.*

*Interactive tools can be added modularly (e.g., new annotations, overlays).*

*This decomposition supports Mark Band 4 design criteria, as it justifies why splitting into smaller, manageable functions produces a clearer, more flexible, and more maintainable solution.*

*Resolution change:*

# Scroll mouse for zoom

A diagram of a change

AI-generated content may be incorrect.

***zoom* –** The zoom function allows the user to change the level of detail on the chart by scrolling up or down. The program awaits a scroll event, then checks the direction. If the user scrolls upwards, the chart is redrawn with an increased zoom, letting the user see smaller movements and more detail in the stock’s price data. If the user scrolls down, the chart re-plots with a decreased zoom, showing a wider time range with less granular detail.

This functionality makes the chart flexible for different types of analysis. For example, a novice may zoom out to see general trends, while an advanced user might zoom in to analyse data on day-to-day levels or note anomalies. This achieves success criteria 9: users must be able to interact with the chart (zoom and scale controls).

***change\_time\_scale* –** The **change\_time\_scale** function enables the user to stretch or compress the x-axis of the chart. The program awaits a horizontal drag event (**stretch\_x**). If the drag is away from the right end, the chart re-plots with a smaller incriments in time scale (e.g., showing fewer days but in more detail). If the drag is toward the right end, the chart re-plots with an larger incriments in time scale (e.g., showing months or years of data at once).

This function gives the user direct control over how much history is visible. For instance, they may want to look at just the past week for a short-term trade, or zoom out to several years for a long-term investment view. This matches the **prefer\_immediate\_returns** setting in the user profile, making the time frame both user-driven and profile-driven.

*These two functions are kept separate because havine both zooming and time scaling allows more control and precise analysis. Zoom focuses on magnifying into a spot on the mouse, making more notes be avaliable to write, while time scale adjusts the actual time span covered on the chart, quickly allowing a view for the differnce in trends short term and long term. Separating them improves clarity and can be more appealing to advanced users rather than just simply having a function to change the x while the y is adjusted automatically.*

**Main algorithm:**

*Prediction models:*

# Acts on class “Predictive\_algorithm”

A diagram of a model

AI-generated content may be incorrect.

***select\_model* –** The prediction system begins when the user selects a model from a dropdown menu. The program first fetches the required stock data and then branches into the chosen model: Linear Regression, Random Forest, or Reinforcement Learning. Each model processes the data differently, “learns” from patterns, and makes three separate predictions. These are compared against stored memory to assess accuracy. Finally, the predictions are returned with confidence values and saved to file for later reference.

This structure ensures that users are not limited to a single algorithm and can experiment with different approaches depending on their needs. For example, a novice might prefer Linear Regression for its simplicity, while a more advanced user could explore Reinforcement Learning for adaptive predictions.

***linear\_regression* –** This model uses a simple statistical approach to identify linear trends in stock data. It loads previous training activity data, fits a line to detect upward or downward movements, and compares this trend to memory (historical performance). After generating three separate predictions, it outputs the results with confidence values based on how closely the data fits the linear model.

This is the most transparent method, making it suitable for beginners who want easy-to-understand results.

***random\_forrest* –** Random Forest is an ensemble method that combines multiple decision trees to make predictions. It loads its training activity data, analyses the stock’s patterns, and compares them against historical memory. Because each “tree” produces a slightly different prediction, the model is able to average these results, improving accuracy and reducing the risk of overfitting.

This is more robust than linear regression and provides better accuracy when trends are non-linear. It is useful for amateur investors who want stronger predictions without needing to understand advanced machine learning.

***reinforcement\_learning* –** This model uses trial-and-error learning to improve over time. It loads past activity data and tests different strategies, storing the outcomes in memory. By comparing predictions to past successes and failures, it adapts its behaviour to maximise accuracy. The algorithm then generates three predictions, weighting them based on what has worked best historically.

This model is the most advanced, simulating a “reward system” similar to real decision-making. It gives the program adaptability, especially in volatile markets where static models might fail. For users with more technical knowledge, it provides deeper flexibility, aligning with success criterion #7: advanced users can adjust hyperparameters to retrain models.

*Each model is separated into its own branch to keep the system modular. If all were lumped together into one “mega-model,” the code would become harder to maintain, and users wouldn’t be able to clearly choose between methods. Separating them also improves testing – I can compare model performance side by side and adjust them independently.*

*Error handling:*

A diagram of a program

AI-generated content may be incorrect.

***Handle\_errors* –** The central procedure begins by iterating through each data entry in the dataset. For each entry, it performs three checks: data holes, NA values, and anomalies. If a repair or tag is needed, it updates the entry before moving to the next index. Once all entries are processed, the fixed dataset is returned.

This ensures that all forms of data corruption are systematically handled, keeping the dataset consistent before it is fed into prediction models. Without this, the models might misinterpret missing or anomalous values as meaningful patterns, reducing prediction accuracy.

***check\_data\_holes* –** This sub-process detects blank entries in the dataset. If an entry is empty, the program reconnects to Yahoo Finance to download the missing data. If the data is still unavailable, the entry is removed entirely to avoid introducing misleading gaps.

This step ensures data completeness, which is vital for time-series analysis. Missing values could distort stock price trends and cause the model to miscalculate volatility or averages. For example, the exchanges may have been closed one day due to a festival or other and no trades would have been done.

***replace\_NA* –** Here the procedure checks whether an entry is marked as NA (not available). If so, the entry is removed, and the corrected dataset is returned. This prevents placeholder values from contaminating the algorithm’s learning process.

Although it may seem simple, explicitly handling NA values avoids silent errors where the program would otherwise treat NA as a valid number.

***find\_anomalies* –** This stage identifies entries where the price change between consecutive data points is outside the acceptable range (e.g., not between 0.95x and 1.05x of the previous value). If such anomalies are found, they are tagged as “erroneous.” Instead of deleting them, the tag preserves the data but marks it for caution.

This design choice balances accuracy with transparency: anomalies are not discarded immediately but flagged so users or models can interpret them correctly. This prevents rare but real market events (like sudden crashes) from being mistakenly deleted or ignored and interpreted as normal market movement.

*The three sub-processes are separated to isolate different types of data problems: missing entries, unavailable values, and suspicious anomalies. Each issue requires a different solution and combining them into one large function would make debugging and future expansion more difficult. By modularising error handling, it is ensured that the program can adapt – for instance, adding a fourth module in the future to check for duplicate entries.*

*Web hook:*

A diagram of a web hook

AI-generated content may be incorrect.

***download\_data* –** T

A diagram of a graph

AI-generated content may be incorrect.

## 2.4 Usability Features

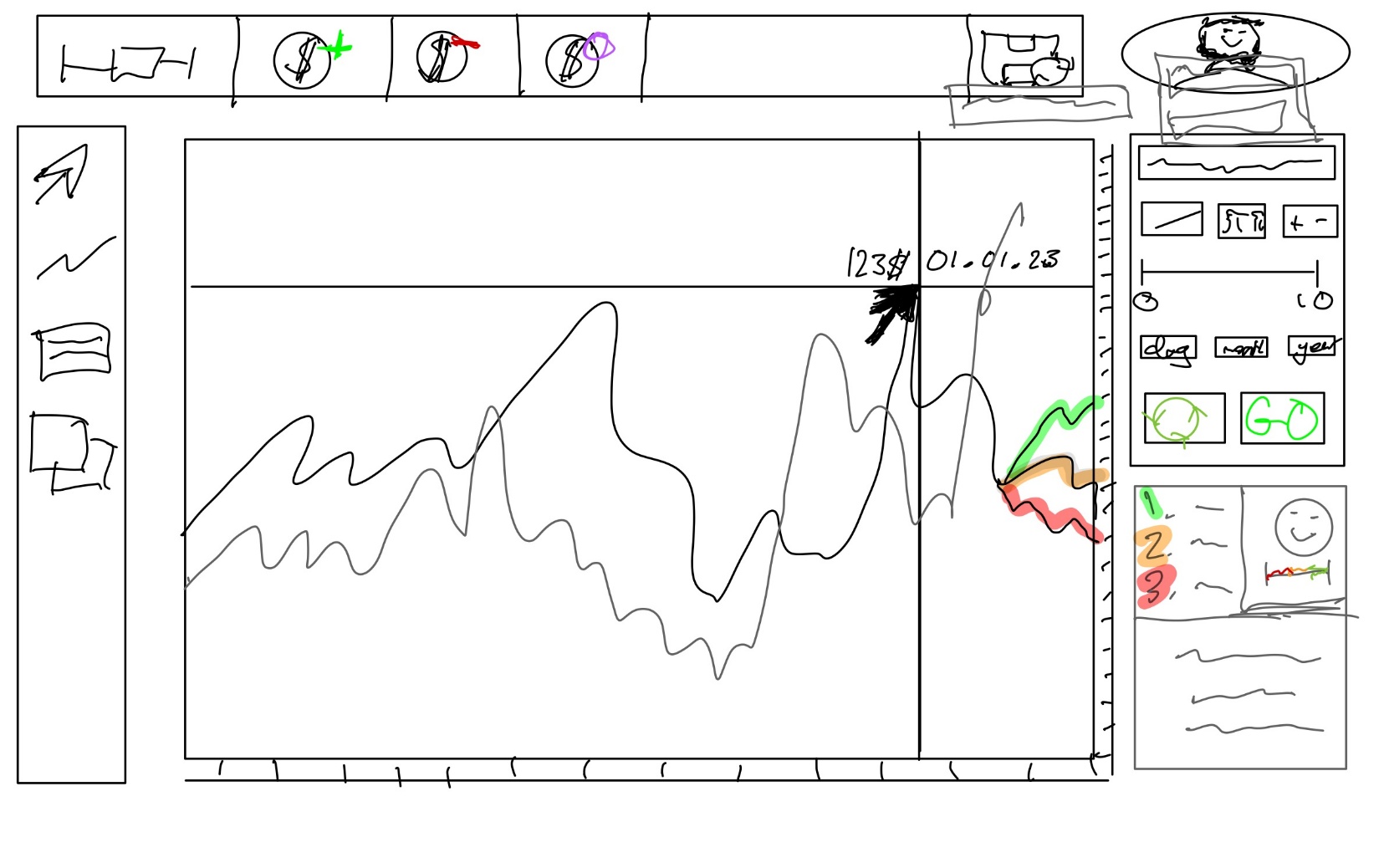
Icons to change chart layout, add/remove a stock from the chart, and load portfolio performance

Profile icon – displays options to login with username and password if not logged in, or opens a new window for profile data if logged in

Graph interactivity tools: curser tooltips (default), line tool, notes, change layer (only for multiple stocks displayed

Graph tooltips display specific details for where the mouse is

Save the current state of the graph



Options for user to edit before running prediction algorithm, e.g. risk level, time length; Or to rerun the prediction

Prediction will be drawn automatically onto the graph with different colours depending on how likely the outcome is (match right description)

Description of predictions: 3 confidence levels, general confidence and small description of patterns

Time on x axis, price on y

Can overlay multiple stock graphs over each other to compare trends

Search a stock to add to the portfolio

Show basic account information, and unique account identifier

A screenshot of a computer

AI-generated content may be incorrect.

Change since previous day open

Graph of past week movement

Ticker symbol

Chosen 4 stocks (including a search result) can be moved and selected to give brief overview of its recent performance

Simplified list of top portfolio stocks’ changes from previous day

Preferences can be edited: risk tolerance, time period, preferred stock exchange, preferred market, target gain, push notifications

A stock is selected and these options appear, to reposition, or delete stock

Searched stock is displayed at the top

Account management: create new, login to new, save, export data

## 2.5 Variables and Validation

## 2.6 Iterative Test Data

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Field** | **Data** | **Explanation** |
| Find\_ticker\_symbol() | Input\_code | “APPL”, “GOOG”, “NVDA” | This is **valid** data as they are all real ticker symbols. They only contain characters and no special symbols. |
| “MAND”, “JOOBL”, “TT”, “GH32”, “(3\*$” | This is **invalid** data as they are not valid ticker symbols. Some also contain illegal characters. YFinance should find no results and return the error that the symbol does not exist. |
| “” | This is **erroneous** as it does not meet the minimum requirement of containing characters. |
| Add\_stock() | Search\_query | Validated by Find\_ticker\_symbol() | |
| Remove\_stock() | Selected\_item | None selected | As this is not based on user input, but a selection, for the case where nothing is selected the program defaults to a None state which will be recognised to not perform any actions |
| Portfolio\_performance() | Graph | Current graph object passed | Whatever state the graph is in currently is saved with the label “temporary” to not lose any work for the user. If there is nothing loaded it will simply not save any work |
| Create\_save() | Profile\_name | “Rafsan”, “Bobthebuilder” | This is **valid** data as the user can enter whatever string they want as the profile name. This will be the name of their profile file so as long as they are all valid characters, the program accepts the data. |
| “sda.”, “asd,/”, “Tom\@” | This is **invalid** data as it contains illegal characters in the string. As this name will also be the saved file name, any characters that a name cannot take will also not be allowed in this variable. It will also not allow “.” to prevent issues with the file extension being changed. |
| “” | This is **invalid** data as there is nothing presence. There should be a presence check to make sure the user must use a name. |
| Load\_save() | Profile\_name | No validation needed as checks string against existing names which were previously validated. | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Field** | **Data** | **Explanation** |
| Prefer\_exchange() | exchange | This is autocompleted by type hinting based on an existing list of stock exchanges, so no validation needed | |
| Prefer\_sector() | sector | This is autocompleted by type hinting based on an existing list of company sectors, so no validation needed | |
| Prefer\_risk\_tolerence() | value | This will be inputted from a slider on the application window, on a scale of 1 to 10, where each mark is an integer. Therefore, no validation needed as only integers can be inputted anyway. | |
| Prefer\_target\_gain\_loss() | percentage | This is also inputted from a slider but will allow floating-point numbers to input. However, this is still from a slider so only floating-point numbers between -10 and 10 will be inputted. | |
| Prefer\_immediate-\_returns() | Time\_period | This will be based off a button the user selects, with only 3 options to choose from with default values, therefore no validation needed. | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Field** | **Data** | **Explanation** |
| Fetch\_stock\_data() | Search\_query | Validated by find\_ticker\_symbol() | |
| Display\_graph() | Stock\_query | Validated by find\_ticker\_symbol() | |
| Time\_scale\_x | Defaulted to a value of 1 year. Any changes will be a product of this based of either the zoom or changing x scale function. | |
| Price\_scale\_y | Defaulted to 1, which is equal to the max price at the top and min price at the bottom for the selected time frame. Any changes will be a product of the zoom function. | |
| Hover\_tooltip() | Cursor | The position of the mouse taken as an input, if the position does not have a tooltip available for it, nothing should happen | |
| Draw\_line() | Cursor | The position of the mouse is taken as an input, upon click after the line tool is selected. If the click is not within the graph space, it should not process the click otherwise continue with function. | |
| Annotate() | Cursor | The position of the mouse is taken as an input | |
|  |  |  |  |

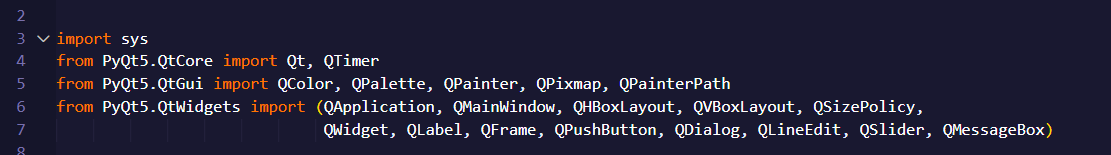
|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Field** | **Data** | **Explanation** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 3. Iterative Development (15 marks)

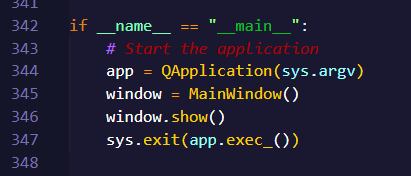
## 3.A GUI Button Response

In my solution, I will be using PyQt5 module for the front end development, which I have decided to code first. Here are the basic creation code to structure the window and allow placement of the buttons and other gui features.

Currently modules imported and used for gui



Initialisation code



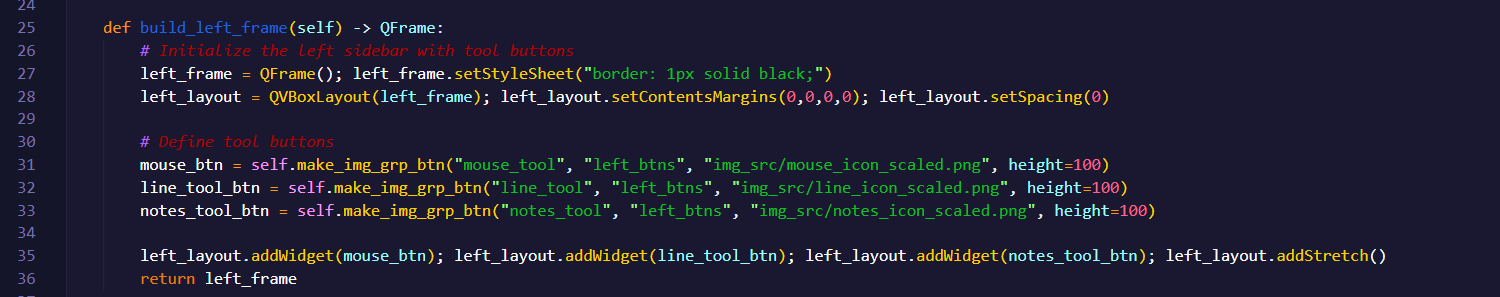
Defining function of the main pyqt5 window.



This code initialises the necessary methods and functions to build the gui application. It creates a class variable dictionary called self.btns which will be used to store group buttons. This will be used to keep track of which buttons belong to the same widget, and to control the logic of only allowing one button within the same group to be selected at once (e.g. for selecting a tool) This is a class variable so that it can be accessed later by the graph logic, for the user to interact with the graph in accordance to which tool is selected. There is also a dictionary for self.colours which simply holds the hexadecimal values of the 3 different colours that are used as a background for the buttons depending if the user is hovering, clicking or the button is in a default state.

Lines, 20,21,23 are basic PyQt5 logic to create a central widget to hold all the frames of buttons within the main window, and sets the layout. Then it adds the frames that are defined in separate class methods. Line 22 calls these for use here. This makes the code more maintainable by separating each section of the main gui into separate functions.

Function to build the left frame

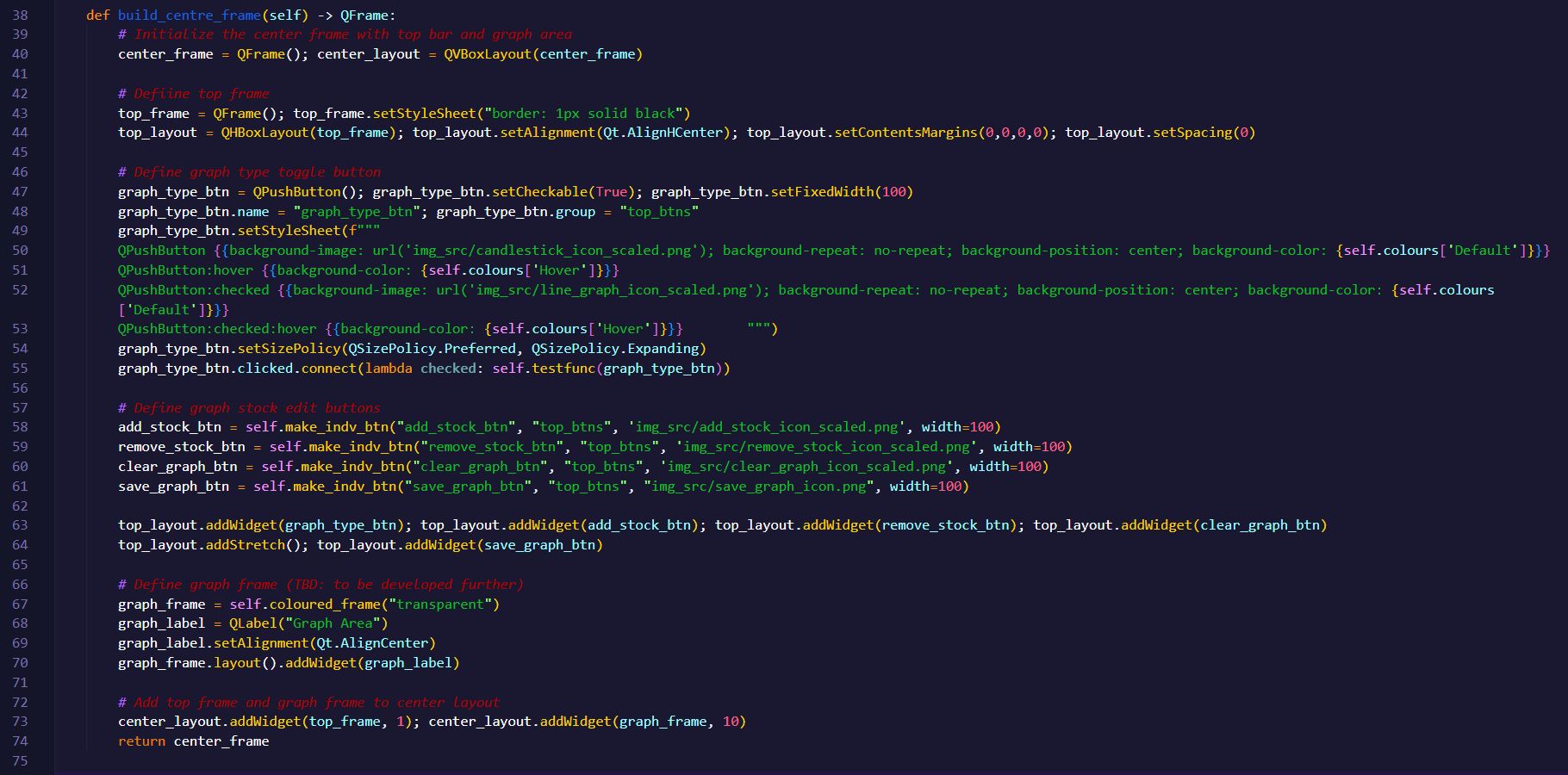


A black arrow pointing to a graph

AI-generated content may be incorrect.

This function creates the left frame which holds the toolbar buttons. These include the mouse tool, line tool and notes tool. Upon further iterations, when the graph frame is complete, these will interact with the graph logic to change the way the user interacts with the graph panel. Upon definition of the buttons in lines 31-33, they call the class function self.make\_img\_grp\_btn which I made into a function as the code is reused many times and takes several lines to simply define the button behaviour. This makes the code more maintainable and reduces the line number greatly. At the end, the code adds the buttons to the frame and returns it to the initial \_\_init\_\_ function to add to the main frame.

Function to build the middle frame

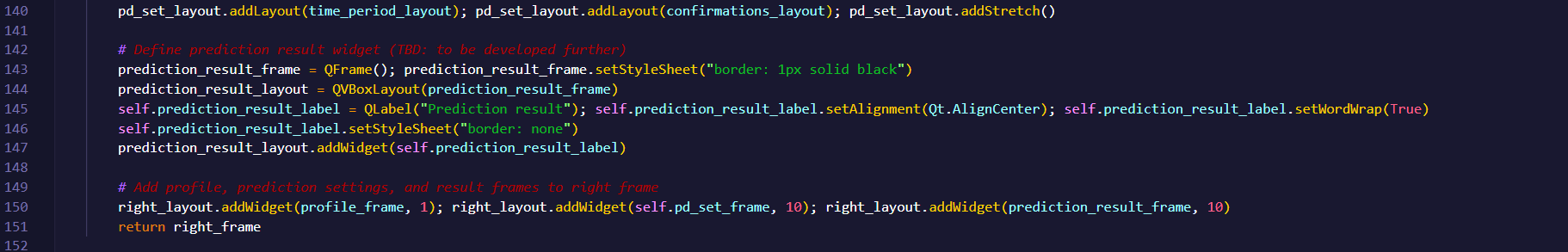
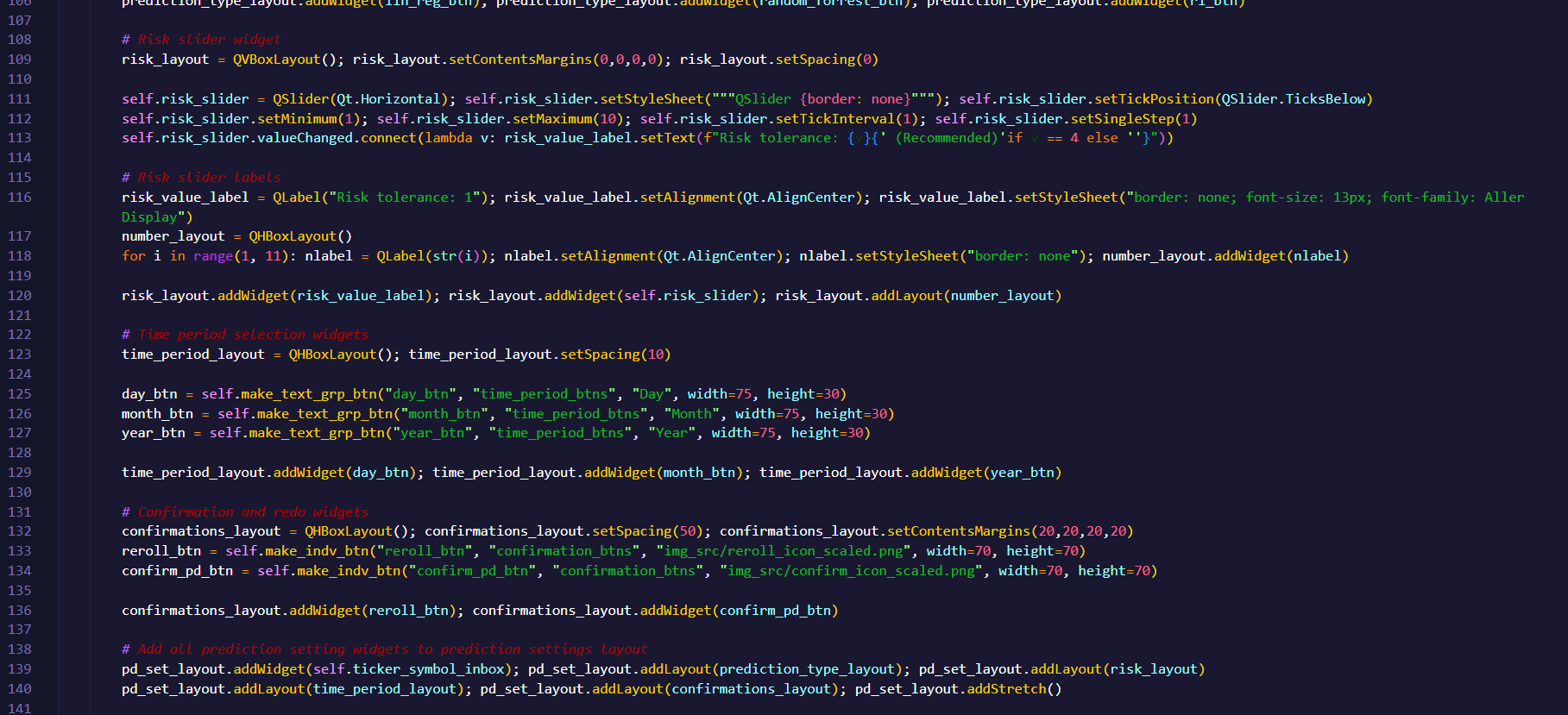


This function constructs the central area of the main window, which is primarily dedicated to the graph display and interactions to edit it. It is split vertically into a top frame for controls and the main graph frame. Lines 39-44 set up the top\_frame with a horizontal layout to arrange control buttons side-by-side. Lines 46-55 define the graph\_type\_btn. This is a special button that is checkable, allowing it to toggle between two states. This is what will change whether the graph is displayed as a line graph or candlestick graph. It uses a standard state to show the 'candlestick' icon and a checked state to show the 'line graph' icon. Lines 58-61 define the buttons for editing the graph: add\_stock\_btn, remove\_stock\_btn, clear\_graph\_btn, and save\_graph\_btn. These use the reusable self.make\_indv\_btn function for creation, improving code maintainability and reducing repetition.

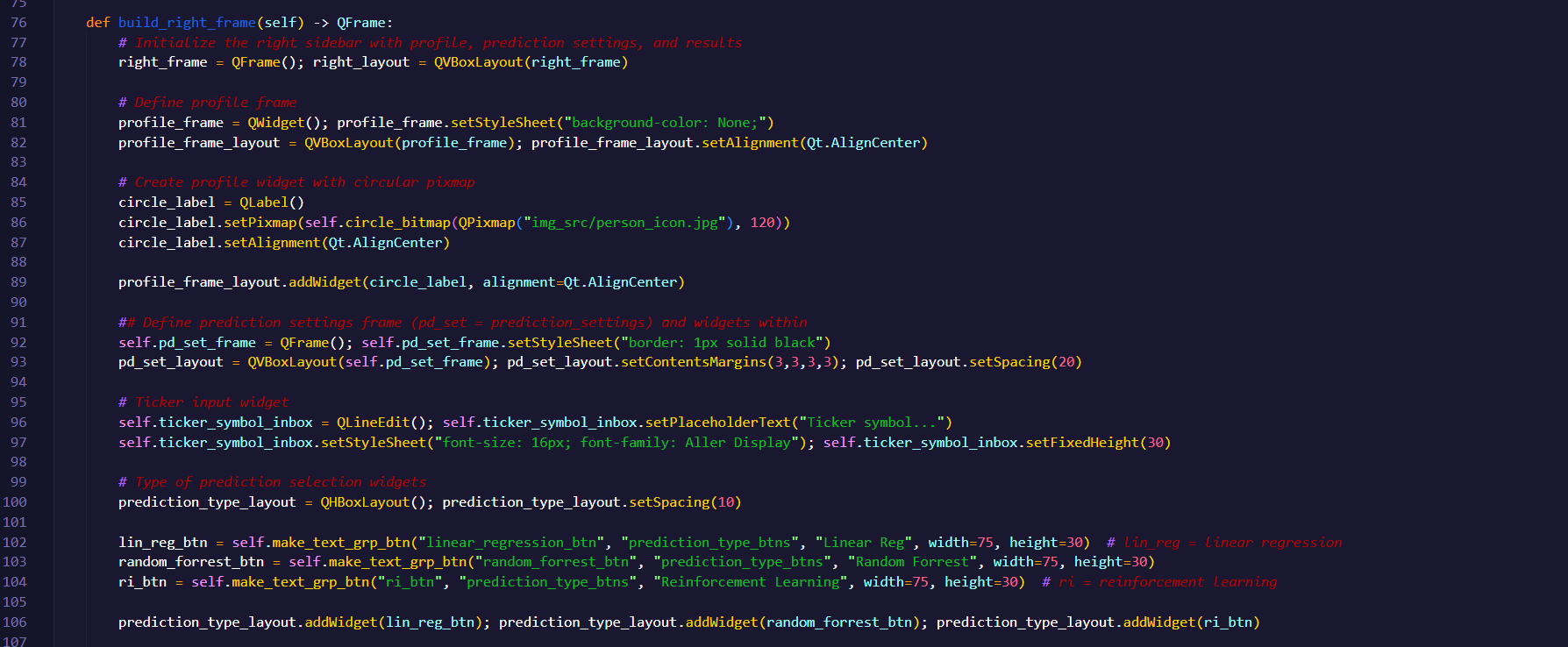
Lines 63,64 arrange these buttons along the left end of the top frame, while after the top\_layout.addStretch(), save\_graph\_btn is pushed to the right. Lines 67-70 define the main graph\_frame. It currently uses a placeholder label but is designed to be where the live, interactive stock graph will be embedded later in development. Line 73 uses center\_layout.addWidget() with stretch ratio 1 to 10 to control the proportional size of the top\_frame and the graph\_frame, making the graph area take the largest amount of space as it is the most important gui aspect of the application.

A screenshot of a computer

AI-generated content may be incorrect.



Function to build the right frame



This function builds the right sidebar, which holds user details, prediction input settings, and the prediction results. Lines 77-82 define the profile\_frame. The circle\_label displays a circular image using the self.circle\_bitmap method to mask a standard image into a circle for visual appeal. In later iterations, this will become interactive to allow the user to create, login and manage their account.

Lines 92-140 define all the buttons, sliders and other widgets relevant to selecting settings for the prediction. These include: a QLineEdit for the user to input the stock ticker symbol (e.g., 'AAPL', 'GOOG'). Three buttons for selecting the prediction model type (Linear Regression, Random Forest, Reinforcement Learning ) which use self.make\_text\_grp\_btn to ensure only one option can be selected at a time, enforcing the logic for the model choice. The Risk Slider (QSlider) allows the user to select a risk tolerance level (1 to 10), where the valueChanged.connect on line 113 updates the risk\_value\_label in real-time. The label includes logic to flag a value of '4' as (Recommended). Helper labels are added to show the numerical range (1 to 10). Three buttons for selecting the prediction time period (Day, Month, Year). These also make use of self.make\_text\_grp\_btn for mutual exclusion. Finally, the reroll\_btn and confirm\_pd\_btn provide actions to re-run a prediction or start a new one, respectively. They use self.make\_indv\_btn as they word independently of any other button.

Lines 142-147 define the Prediction Result Frame, which holds the self.prediction\_result\_label to display the outcome of the prediction (currently a placeholder). Line 150 lays out the three main components (profile\_frame, self.pd\_set\_frame, prediction\_result\_frame) in the right sidebar, using stretch ratios 1, 10, 10 to ensure the correct heights of the boxes are used.

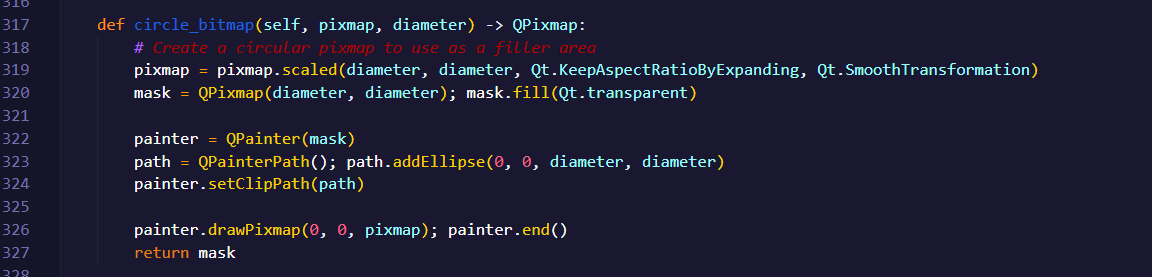
A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

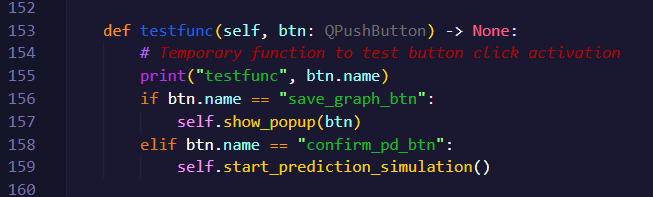
AI-generated content may be incorrect.

Function to create the circle profile icon



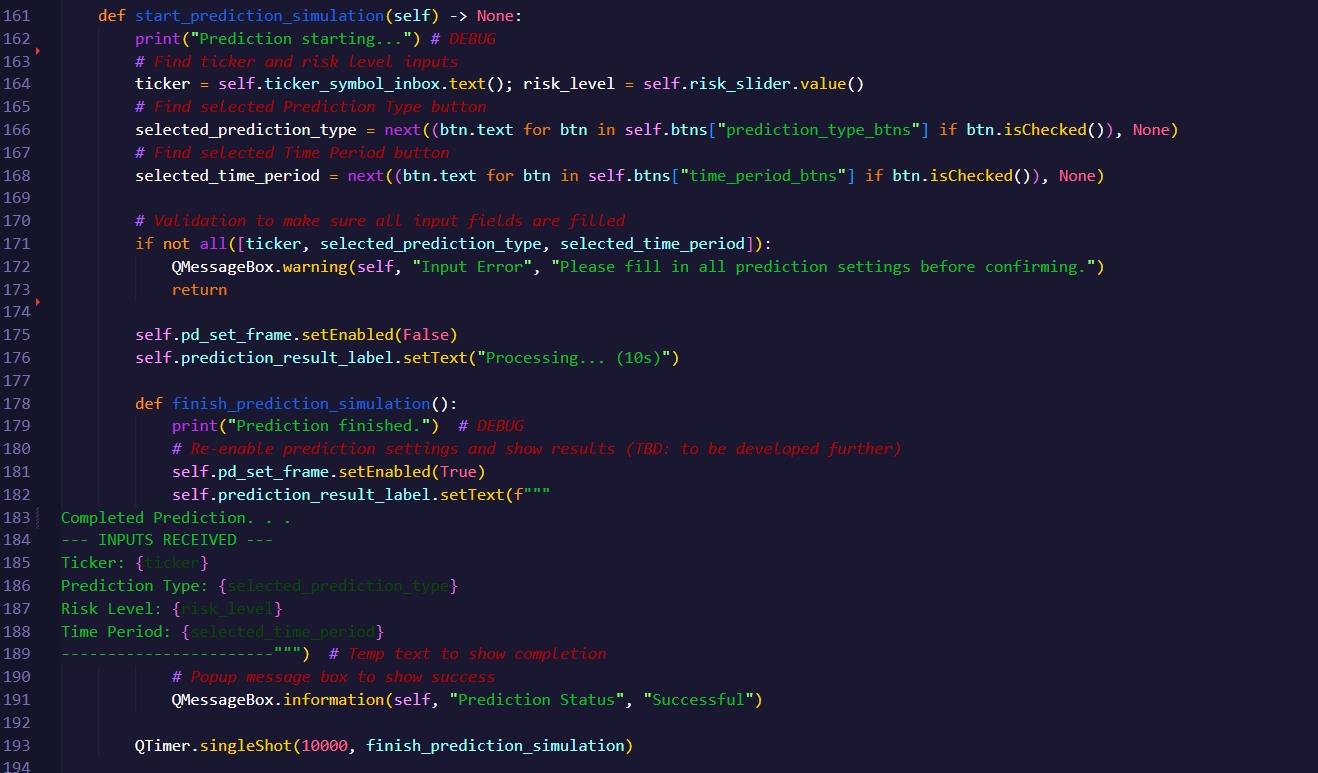
This helper function takes a square QPixmap image and transforms it into a circular image using a masking technique. This is used in the build\_right\_frame method for the circular profile picture. This will be developed in a future iteration for login and profile logic, as a button to open up a popup with login fields or to open a new window with other profile details.

Function called upon button pressed



This is a temporary handler function used to check that the buttons are correctly calling the function upon click. Line 155 prints the clicked button's name to show whether it is activated if it is meant to upon click. Line 156 checks if the clicked button is the save graph button and, if so, calls self.show\_popup() to initiate the save dialog. Line 158 checks if the clicked button is the confirm prediction button and, if so, calls self.start\_prediction\_simulation() that will later connect to the prediction script and temporarily prints the inputs it received.

Function to take and validate the prediction inputs



This function is responsible for gathering the user's prediction inputs and temporarily displaying them. First it retrieves the required input values: ticker and risk\_level are directly read from their respective widgets, the selected\_prediction\_type and selected\_time\_period use a generator expression and the next() function to find the checked button within their respective groups in the self.btns dictionary. If no button is checked, it defaults to None. Then it performs input validation to make sure all inputs exist and if not, a QMessageBox.warning is displayed to the user, and the function stops. The validation for the ticker symbol input existing will be performed within the web hook script when connecting to Yahoo Finance API, or checking if it exists in the already downloaded data.

If all the data exists, the code temporarily disables the prediction settings frame to prevent the user from changing inputs while the prediction is running, and updates the result label to show a processing message. Lines 178-191 define a localized function which is called when the prediction process completes. Making it localized prevents needing to increase the amount of class functions and privatises it, as it is only used to re-enable the settings frame after the prediction result comes through. For now, it displays the inputs, and shows a success pop-up. Before the prediction script is written, it uses a QTimer.singleShot method to simulate a 10-second delay for the prediction to run, after which it calls the finish\_prediction\_simulation function. This timer allows the GUI to remain responsive while the "calculation" is happening.

Functions to save the graph state



The save\_graph function is a placeholder for the final graph saving logic and implements a temporary visual feedback mechanism. Until the next iteration, it prints the save name to the console. It also create a small, frameless, transient message box that displays "Saved." in the center of the screen for two seconds using QTimer.singleShot(2000, msg.close), providing instant feedback to the user without interrupting the flow.

Show\_graph\_save\_popup creates and manages the pop-up dialog for naming a saved graph. It creates a modal dialog and positions it directly below the button that triggered it making the GUI context-aware; this will allow for more fluid resizing of the window when the full app is ready. Currently it also adds a label and an input\_box for the user to type in the name to save the graph state as. When the graph functionality is added in a later iteration, it will pull the data of the graph widget and save it in a file to be loaded later. At the end it connects to the save\_and\_close helper function. When the user presses Enter, it calls self.save\_graph and closes the pop-up.

Function to create an independent, checkable button



This is a factory method to create independent buttons that do not interact with other buttons in their group and have an image as the background. It accepts parameters for name, the group (e.g. left tool buttons), the image source (as a string of its relative path to the source code), and optional sizing. It also includes creating custom attributes like btn.name, btn.img and btn.group for easier access later within the click logic function (currently testfunc). Creating the functions this way allows more modularity, and will keep the decision of which function to call based on which button was pressed within the same area.

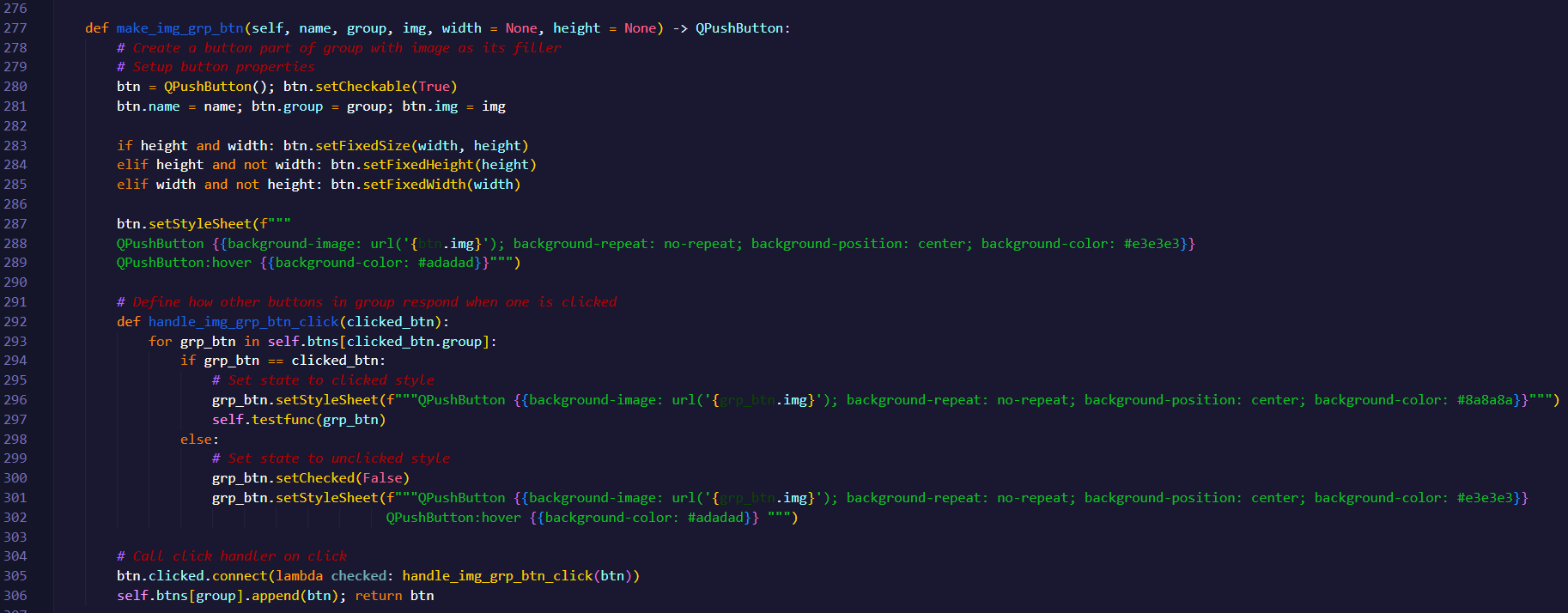
Here, the self.colours dictionary is used to ensure uniformity across all the buttons background colour based on what state it is. Line 243 adds the button to the corresponding list in the self.btns dictionary for centralized management, although it doesn't enforce mutual exclusion logic.

Function to create a button part of a group with text fill



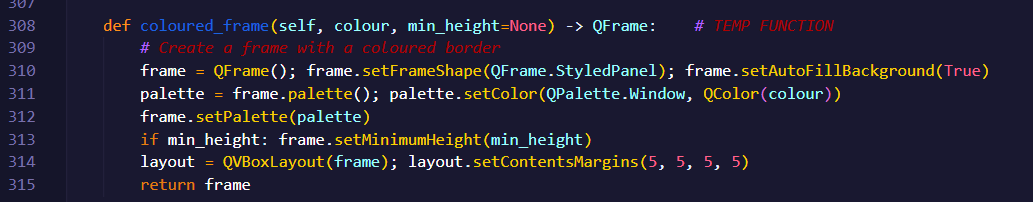
This is a factory method to create a button that belongs to a group which is passed as a parameter. This function will create mutual exclusion (only one button in the group can be checked at a time) as it is used for the prediction settings where only one option can be selected. When a button is clicked, it iterates through every button in its group and checks if it is the current button. If it is, its style is set to a darker, "clicked" colour using the self.colours dictionary, and self.testfunc is called with itself as the parameter. For all the other buttons in the group, they are unchecked to make sure only one is selected at any time. Their style is also reset to the default unclicked appearance, enforcing the one-selected-at-a-time rule.

Function to create a button part of a group with image background



This is another factory method with almost identical logic to make\_text\_grp\_btn, but it is specifically for setting an image as the background of the button. It still enforces mutual exclusion and is used for the Tool buttons in the left sidebar. The primary difference is in the style sheet manipulation, which instead of simply setting a background colour and using set.text, it sets a background using “url(‘…’)” for parsing the image directories. Elsewise, the one-selected-at-a-time logic is identical, iterating through all the buttons in its group to ensure only it is selected.

Temporary function to create a frame with border and fill



This is a temporary, reusable helper function used to quickly create a QFrame with a background colour, and border. It is temporary as its main use may be for visually separating sections during the initial GUI building phase. Later on, these coloured frames will be just a regular QFrame as the border will be seen from the widgets within it.

# 4. Iterative Testing (10 marks)

# 5. Post-devlopment Testing (5 marks)

# 6. Evaluation (15 marks)