MIE479 Final Report

Joshua Ha Rim Kim 1004391339
Aidan Jones 1003961340
James Gemmell 1004459631

December 8, 2021

Contents

1 Problem Statement

Financial markets have become increasingly divided between Main Street (retail investors) and Wall/Bay Street (institutional investors) [1]. The craze around meme stocks such as GME and AMC brought to light a number of different disadvantages that everyday people face when participating in capital markets [1].

A number of platforms have already been created to address this gap, mainly robo-advisors such as Wealthsimple [2], and accessible online brokers such as Questrade [3], however we believe there still exists a large gap in available resources between complex institutions and existing solutions. Say, for example, a retail investor wishes leverage Wealthsimple to gain access simple mean-variance optimization at low cost [2]. This individual has very little input, only being able to set risk tolerance and some additional options such as Halal and Eco-friendly investing.

If the investor seeks more customization, they can chose online brokers such as Questrade, and either select single stocks or ETFs [3]. This customization comes at the cost of lacking the portfolio optimization that Wealthsimple provides however.

We aim to fill this gap, by providing users with a state-of-the-art robo-advisor while not sacrificing customizability, allowing investors to still take positions on the specific assets they chose, while handling as much risk mitigation as we can with our portfolio optimization techniques. Our app, Bridge, aims to bridge the gap that retail inventors face when competing with large institutions.

2 High Level Overview

2.1 Stakeholders

For the scope of the project, we identify five major stakeholders.

- users/clients
- design team
- teaching team
- financial regulators and policymakers
- general finance & trading community

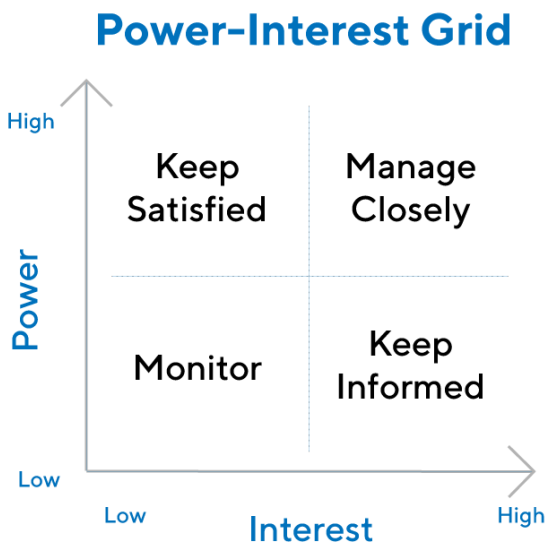


Figure 1: *Power-Interest grid used in stakeholder analysis*

The stakeholders are analyzed under power-interest framework, illustrated in *Figure 1*.

- high power, high interest: design team, teaching team
  - The design team is the supplier of the service and is fully engaged in the project by developing the software and conducting research. The team has complete authority over adding/modifying features of

the product. The team must consider all the features it wishes to implement and judge them with no bias on their merits.

  - The teaching team is in close touch with the design team by assisting them on the project. The team is also responsible for structuring the project requirements and criteria and evaluating the overall grade.

- low power, high interest: users/clients
  - Users and clients of the product are interested as they are the end receiver of the service. The performance of the project can significantly affect their investment strategies. However they do not have direct authority over how the project is carried out but can only influence it by providing user feedback and suggestions. Nevertheless, they are a key stakeholder group since the success of the product closely lies with how clients rate their experience using it. As a financial advisory service, we owe the users of our application the highest degree of diligence and must provide them with trust.
- high power, low interest: financial regulators and policymakers
  - While regulators and policymakers are not closely engaged with the project, it is important to keep them satisfied since they have judicial and legislative power to impose penalties provided that the product violates laws or guidelines.
- low power, low interest: general finance & trading community
  - It is beneficial to maintain a good reputation within the finance and trading industry since they can be our potential collaborators if we decide to scale the product to a commercial level.

To expand on the users/clients analysis, we tailored our service for a more experienced group of retail investors who have decent knowledge enough to have confidence selecting their own stocks. A detailed overview of this can be found in the section *Product Overview*.

2.2 Objectives

The high-level objectives are as follows:

- Users can easily navigate the website with friendly input/output features.

- The product should suggest a ETF-based portfolio that mitigates risk given the user's stock selection and preferences and provide friendly and intuitive analysis of the investment recommendation.
- Integration between front-end and back-end is smooth and the database can be accessed quickly.
- The website should be hosted online with robust security.

The objectives are derived directly from the course rubric constructed by the teaching team. Detailed objectives are discussed under their corresponding section.

2.3 Constraints

The following are a list of the overall project constraints:

- The final project must be accessible as a web application.
- Data are stored in a database on the cloud.
- The portfolio consists of only ETFs.
- Multi-period framework is used to generate portfolio recommendation.
- The model allows user inputs (target return, time horizon, etc.)

The constraints are derived directly from the project requirements constructed by the teaching team. Detailed constraints are discussed under their corresponding section.

3 Product Overview

As mentioned in our problem statement, we seek to provide users with a state-of-the-art robo-advisor which does not sacrifice customizability. This will be achieved through a web-application much like Wealthsimple, but with the key added feature of allowing users to "pick" stocks.

User will be able to select stocks present in both S&PTSX60 and S&P500, along with minimum and maximum constraints on each of these stocks, called "picks". These picks, along with a user's target returns, risk tolerance, investment horizon, and a metric we believe to be brand-new to robo-advising, portfolio control, will be fed into state-of-the art multi-period portfolio optimization models to create a customized basket consisting of the users stock picks and optimal ETF hedges.

This provides a tailored solution not for the most novice of investors, but the ever-growing number of retail investors who pick stocks and currently have no guidance on managing the associated risks that their stock picks might introduce.

Our vision is for Bridge to become the go-to site for investors looking to traverse from the freedom of Main Street into quantitative world of Bay Street, without compromise.

4 Front-End

4.1 Objectives

As the Front-End is the only piece of the project that the user will directly interact with, it is important that this process goes smoothly. The objective of the front-end and associated UI/UX is to remove all bottle-necks the user may face when navigating our web-application, facilitate the use of all features in an easy and intuitive way, and ensure the users initial and final impressions of our web-application are positive. This is done through iterative design, discussed in detail in the User Testing section.

4.2 Constraints

The front-end must facilitate user inputs for both target returns and investment horizons as per the syllabus slides.

4.3 Development Tools

Through market research, we evaluated two promising front-end frameworks: React [4] and Angular [5]. The criteria we considered for evaluation these two metrics are availability of learning resources and customizability, as our team has little expertise here and high customizability minimizes the risk of needing to pivot and relearn a new front-end framework.

A comparison matrix is used to evaluate the two options across our metrics, shown below in *Table 1*.

After evaluating our comparison matrix, our team decided to use React over Angular primarily due to its customizability. Although both frameworks would likely be able to achieve all features for our web-app, there is a steep learning curve to using these frameworks and none of our team members have experience in either, therefor we chose React to minimize the likelihood of needing to switch later on.

However after some time of attempting to learn React, it proved to be too difficult, and we decided to switch to another framework tool that was already determined to be our dashboarding tool: Dash [7].

Criteria	React vs. Angular	Rank	Winner
Availability of Learning Resources	From our teams initial research, we've found extensive availability of online learning resources for both React and Angular covering all topics we've identified as required so far. As such we awarded a tie between the two options for this criteria [6]	1	Tie
Customizability	According to Brainghub, in their comparison between Angular and React, Angular gives almost no flexibility [Ref.] In a way this is good as there is less opportunity to go wrong, but for the purpose of future-proofing our selection here this gives the upper hand to React [6]	2	React

Table 1: Caption

Using Dash to solely build the front-end did not come up in our preliminary research. Once identifying that this was possible, however, we began exploring the limited learning resources available. Online resources stated the needed callbacks were difficult to implement, however, our team has past experience with Dash and we determined the learning curve was less steep than continuing to pursue React.

An additional library was used called Dash Bootstrap Core Components [8], which adds bootstrap components to Dash. No comparisons were needed to determine which bootstrapping library we should use as this is the only one which came up in preliminary research, more on this in following sections.

4.3.1 Developing Pure Dash Web Applications

As mentioned above, the callbacks and file layout for this can be quite complicated. In this section we will go over exactly how this was achieved as well as the draw-backs and benefits.

File Layout In order to handle multi-page Dash web applications, page routing logic is needed. This is handled using a dedicated python file called **index.py**. This python file imports all necessary page files, has one Div with **id = 'page-content'** and contains a single callback. This callback takes the web-

application's URL as input and returns the relevant page to the **'page-content'** Div. This allows us to write each page in separate python files leading to easily navigable and readable code, and fast implementation of new pages and features. Additionally since the URL is an it's own input, we can implement logic for page routing such as failing to log in with correct username/password combinations prevents navigating to further parts of the website. A high level overview of the front-end file layout is shown below in *Figure 2*.

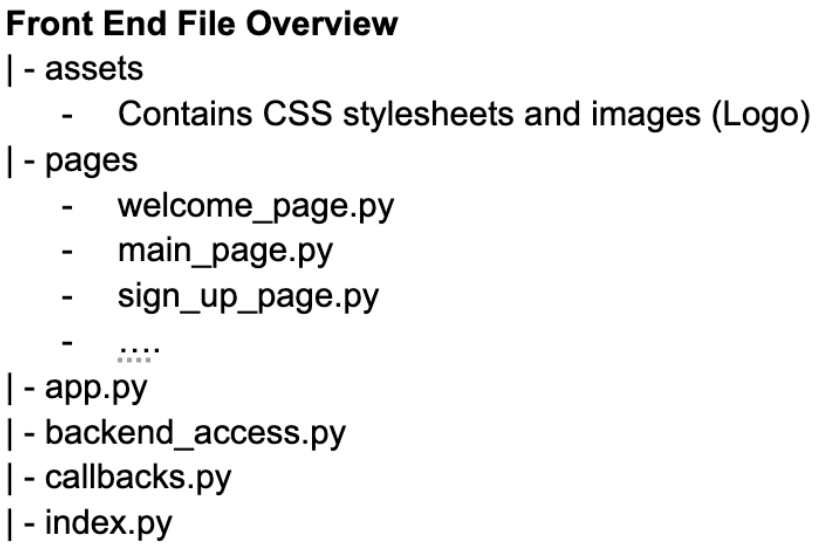


Figure 2: *Front End File Layout*

Advanced Callbacks By far the trickiest part of implementing a multi-page web is the need to communicate using callbacks between pages. Callbacks need to be stored in their own python file, **callbacks.py**, which along with index.py are imported into **app.py**. Although this may seem counter-intuitive, this structure is needed to avoid cyclical import statements. For detailed explanation please refer to the callbacks.py file in our team's github, however a brief example of how this is done follows. This is accomplished using a number of features Dash provides, the first being Dash Store Core Components, which can either contain python dictionaries or json data. An example of how this is used is with the current user's portfolio value figure. When the user signs in, logic is first used to determine if the email/password combination is valid. If it is, functions

interacting with the back-end are called to grab all information about the user, their past deposits, transaction data and portfolio data. This portfolio data is converted from a pandas dataframe to json data, then stored in a Dash Store Component. This json data is then retrieved on the main page to be used to draw out the users portfolio over time.

Finally an important piece in allowing us to implement advanced callbacks is the use of Dash Dependencies Inputs and Outputs in place of the regular Dash Inputs and Outputs. This is done since the base Dash Inputs and Outputs only allow each ID to appear once as an input to a Dash callback and once as an output. The logic needed by this web-application is too advance for this thus we needed to be able to provide IDs to multiple callbacks as both inputs and outputs, which is possible with Dash Dependencies.

Stored Data Refer to the above paragraph for details about Dash Store Components. A number of these are used to handle both passing information between pages as well as improving performance. They can mostly be avoided and instead every time a page is visited we hit the back-end for relevant information, however a better implementation is only updating the Store component when relevant information changes.

4.3.2 Dash Bootstrap Components

Bootstrapping was primarily used for aesthetics, where all plain HTML is replaced with bootstrap components that then share a common theme, in the case of our web-application the Dash Bootstrap Theme *MINTY* was used, as it matched closely with our initial wire frames.

4.4 User Testing

4.4.1 Purpose

User testing was used to refine the final layout of the web-application, as well as identify where more resources are needed to explain different features.

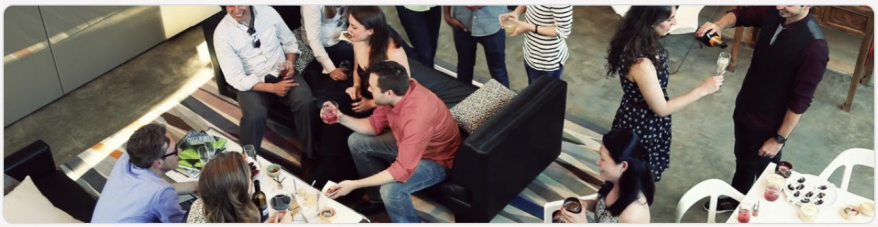
4.4.2 Methodology

A questionnaire was provided to a number of friends from Engineering Science, some in the EMSF progam and others outside this program, as well as family

members. This provides us with users with a wide range of investing expertise as well as technology experience.

We intended to have two separate questionnaires, one for account creation and another for portfolio creation, however since the portfolio creation features have not yet been finished, only the former questionnaire has results, the latter will be completed for the final report.

The account creation questionnaire (in the the format of a Google Form) (*Figures 3 - 5*) was given to 9 individuals, along with the most up to date version of the locally hosted web-application. The results of which are summarized below. Note not all questions were mandatory as some may not be accessible depending on how far through the web-application the user gets.



Bridge User Experience Form - Account Creation

Please complete this form in one go, and be as honest as possible, don't hold back with the criticism!

aidanjones55@gmail.com (not shared) Switch account

* Required

What is your first impressions of the Welcome Page? *

Your answer

Could you navigate to the sign-in page? *

- ◯ Yes, I found it.
- ◯ No, I could not find it.
- ◯ Other:

Figure 3: *Account Creation Questionnaire - Part 1*

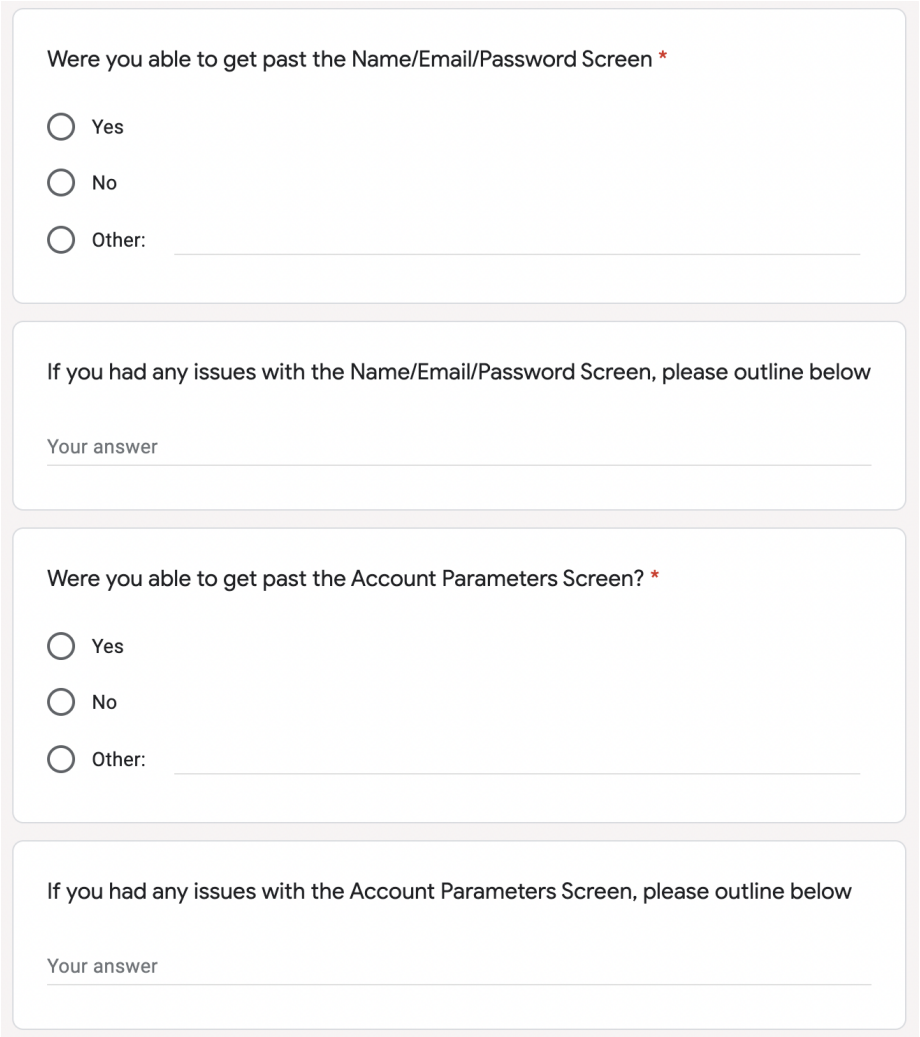


Figure 4: *Account Creation Questionnaire - Part 2*

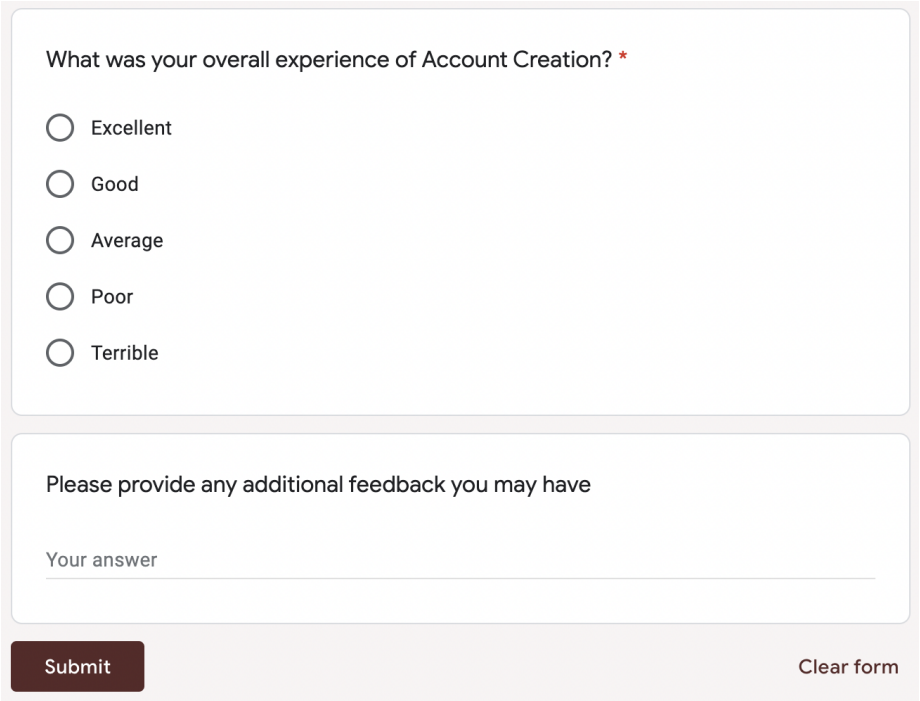


Figure 5: *Account Creation Questionnaire - Part 3*

4.4.3 Results

In total 9 Account Creation forms were receive. Some notable results are outlined below.

In terms of the Welcome Page, all user responses were either positive or neutral. The only critique we could draw from it is that the welcome page feels like a mobile app instead of a web application (*Figure 6*).

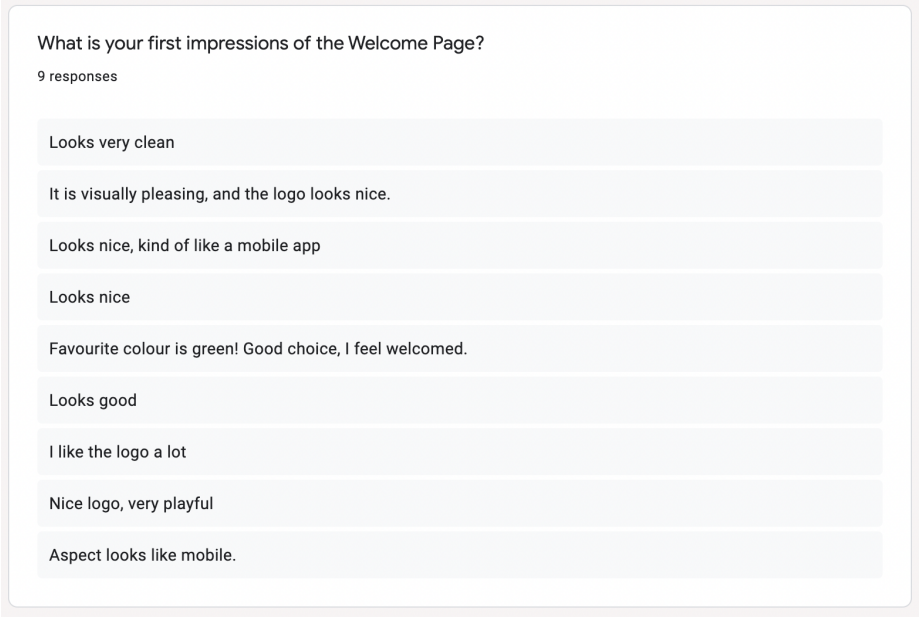


Figure 6: *Welcome Page Questionnaire Results*

In terms of the Name/Email/Password Page, all users were able to navigate past it. The only real critique we can draw from the results here is one user complained that the error message doesn't distinguish between the Emails not matching and the Passwords not matching (*Figure 7*).

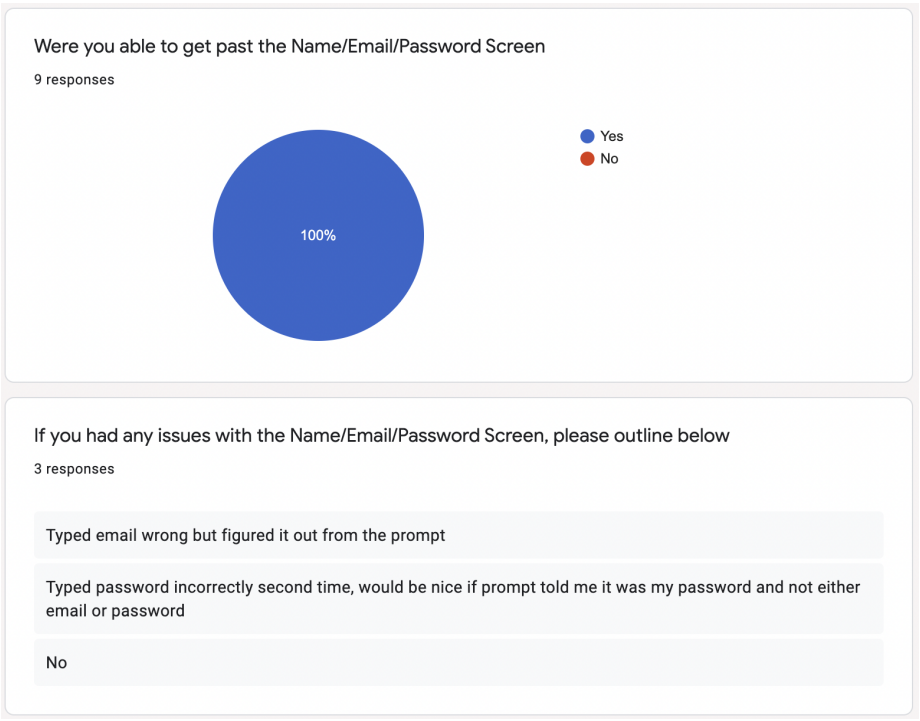


Figure 7: *Name/Email/Password Questionnaire Results*

In terms of the Account Parameters Page, all but one user were able to navigate past this, however all expressed confusion with this page and exactly what the parameters were/did. Additionally some commented on being unsure if these parameters were then fixed or could be changed later (*Figure 8*).

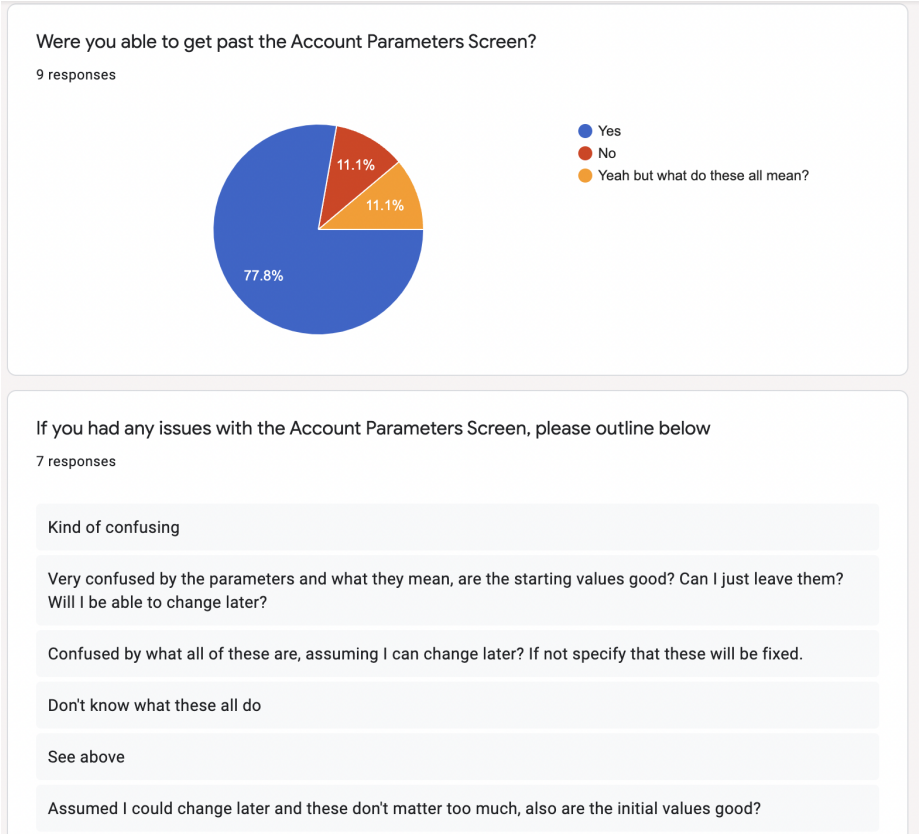


Figure 8: *Account Parameters Questionnaire Results*

Overall experience was mixed, with 4 respondents either having a Good or Excellent overall experience, 4 having an Average overall experience, and 1 having a Poor overall experience. Comments in this section point to the aesthetics being nice however the Account Parameter confusion is reiterated here (*Figure 9*).

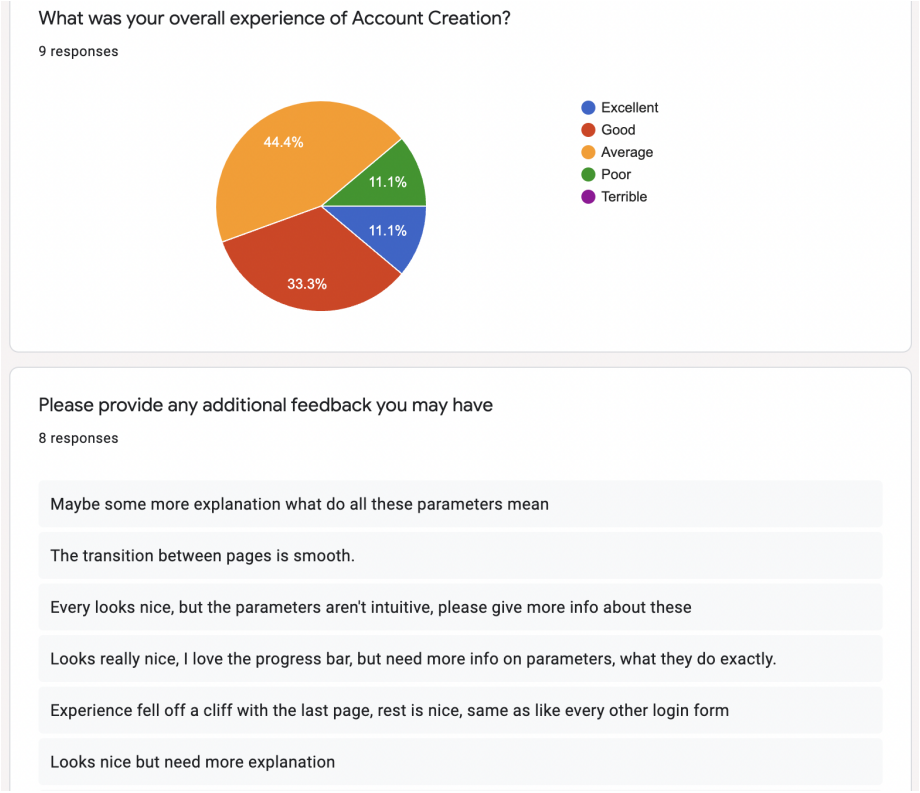


Figure 9: *Overall Questionnaire Results*

4.4.4 Conclusion

The major takeaway from the Account Creation Feedback Form was that the Account Parameters are confusing, and users were unsure if they would be able to change these parameters after navigating past this page. Popups and a card have been implemented to clarify, contents of which are discussed later.

4.5 Pages and Features

Pages were designed to be as close as possible to our wire frames while taking user feedback into account. All pages and their features are outlined in the following sections.

4.5.1 Welcome Page

This page contains welcome information as well as the option to log in, or sign up (*Figure 10*). Clicking "Login" reveals a hidden Div containing the login form (*Figure 11*), where the user can input an email/password combination. This is then checked against the client table in the back-end, if it is invalid a message appears notifying to retry (*Figure 12*), else it continues to the main page. Integration and how this login information is stored/retrieved is included in the integration section.

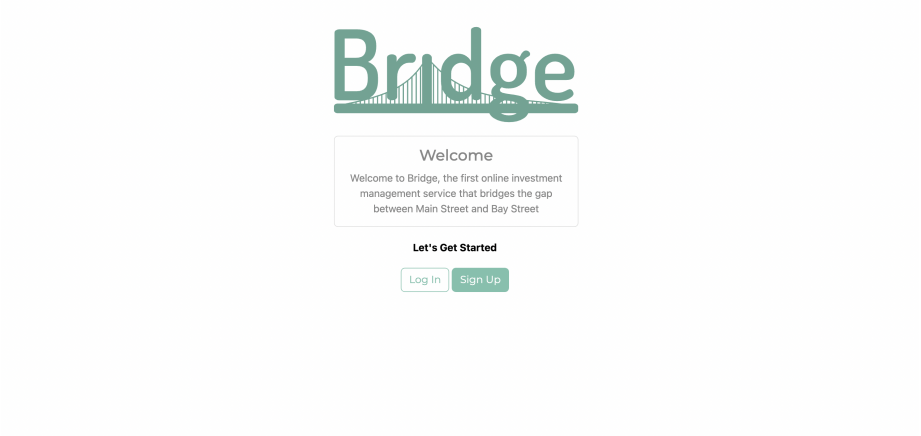


Figure 10: *Welcome Page*

Figure 11: *Welcome Page, Login Revealed*

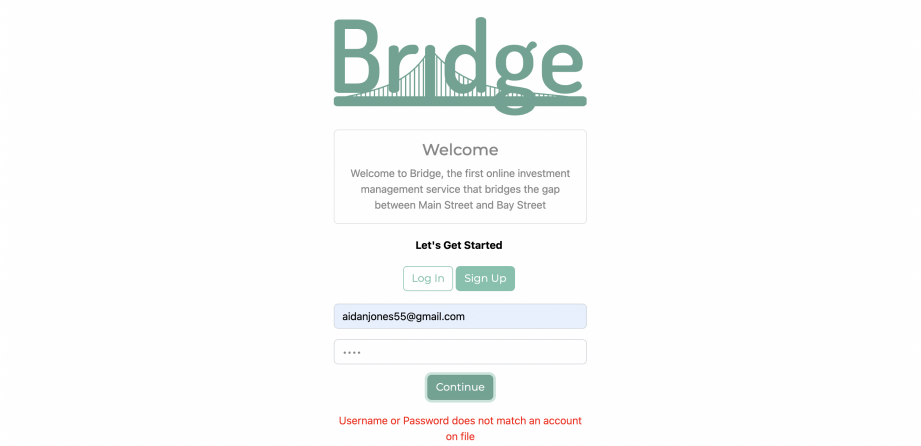


Figure 12: *Welcome Page, Login Error*

4.5.2 Sign Up Page

If the user instead selects "Sign Up", they are redirected to the sign up page, which first asks for account information (name, email, password) (*Figure 13*) and then for investment preferences (*Figure 14*).

These investment preferences were identified as the most difficult part in understanding our web-application in user testing, as such a number of features were implemented to help with explain ability. The first being a card a the top of the screen notifying users they can change these at later times, and the default values are recommended. Additionally if the user hovers over each parameter information about it appears (*Figure 15*). Although this solution did not fully eliminate the user's difficulty with this section, it did alleviate some concerns.

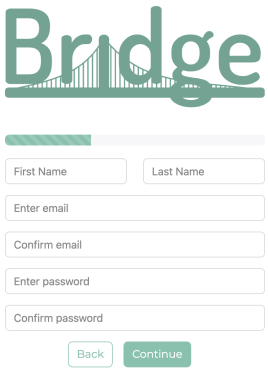


Figure 13: *Sign Up*

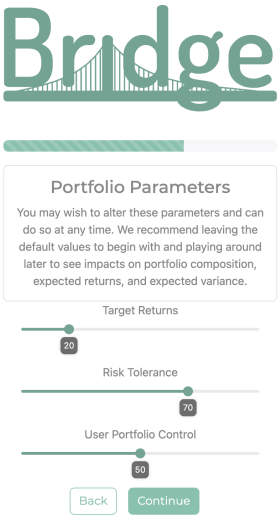


Figure 14: *Sign Up - Part 1*

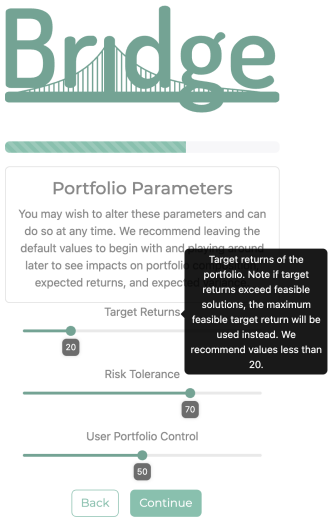


Figure 15: *Sign Up - Part 2*

4.5.3 Main Page

After successfully signing in or signing up, the user is then directed to the main page (*Figure 16*), which provides the current value of their portfolio in a message, the historical value of their portfolio in the form of a graph, as well as daily deposit information. Not shown in the image is another feature that has yet to be implemented: an overview of their portfolio composition.

From this page the user can then navigate to other pages, as well as sign out to return to the welcome page.

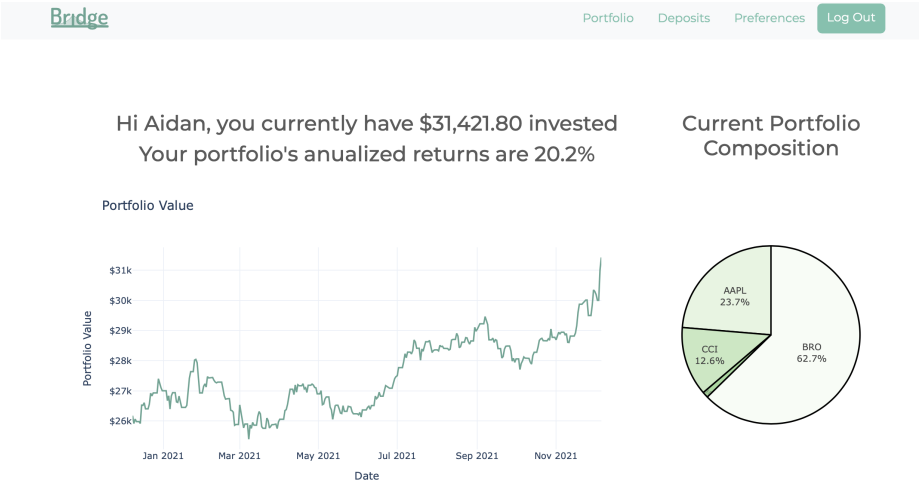


Figure 16: *Main Page*

4.5.4 Deposits Page

Navigating to the Deposits Page, the user can view the total current cash component of their account, all past transaction, as well as make a deposit and withdrawal (*Figure 17*). Clicking either Withdraw or Deposit reveals a hidden form for both (*Figure 18*), where the user can then enter an amount to either withdraw or deposit, respectively, and confirm. This action is then immediately updated in the transactions history with text describing exactly what happened, for example when the user deposits $100 a corresponding message and timestamp will appear.

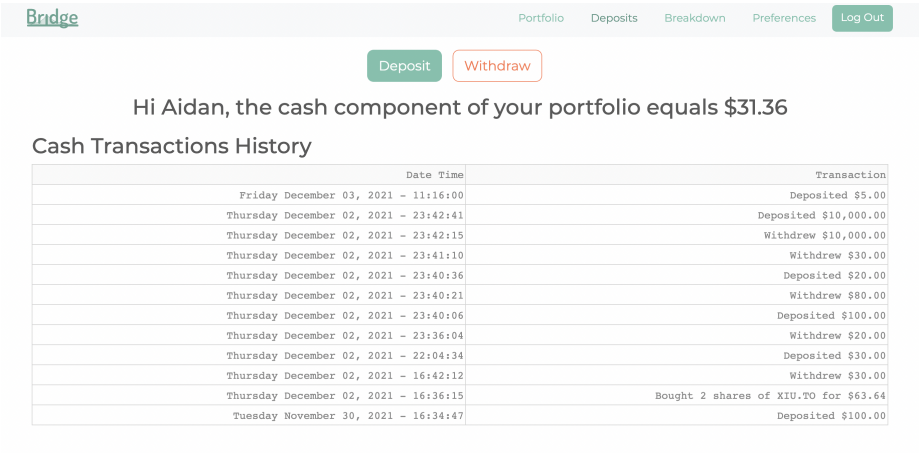


Figure 17: *Deposits Page*

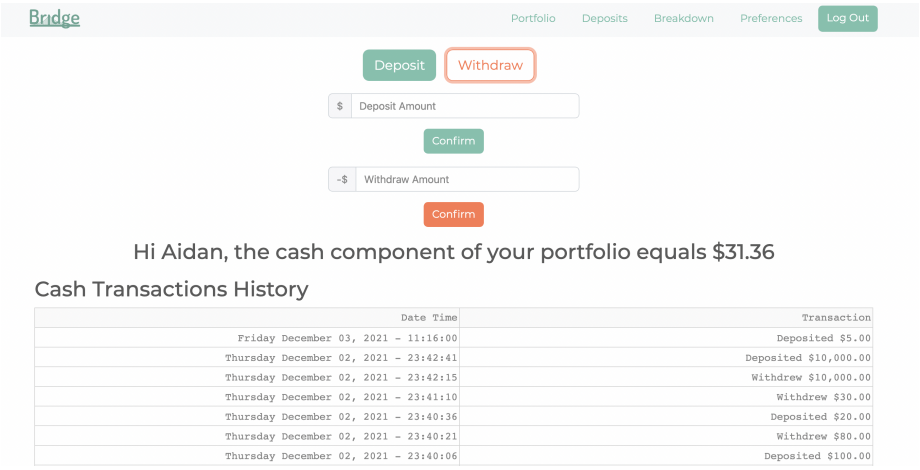


Figure 18: *Deposits Page - Revealed*

4.5.5 Preferences Page

This page handles the stock picking component of our web-application as well as allows the user to play around with previously set return, risk, cardinality, horizon and portfolio control parameters (*Figure 19*). The user can add as many stock selection components as they wish, search through all available single stocks as well as set the minimum and maximum values they want this stock to have in their portfolio (*Figure 20*).

These stock picks and min/max values are passed to the business logic and implemented as linear constraints on the portfolio, as such it is possible to get impossible solutions (say 3 stocks have minimum values set to 50%). Error checking occurs here when the user hits confirm, and messages are displayed to handle infeasible constraints.

If all inputs are accepted, a hypothetical portfolio over time is then shown on the screen (*Figure 21*), with back tested and future expected returns shown.

The user can iterate through selecting different parameters and seeing results of these on screen, as well as a breakdown of the value of the new portfolio (*Figure 22*). If they decide to act on this they hit the **Make Trades for New Portfolio** button, confirming that this is the portfolio they want, these changes are then written into the transactions table, cementing the new portfolio. Once the transactions are written into the back-end, a message appears notifying the user that the trades have gone through (*Figure 23*).

These changes are immediately reflected in both the deposits (*Figure 24*) and main page (*Figure 25*).

Note that pop-ups explaining all parameters mimic what is seen in the sign-up page to help with usability.

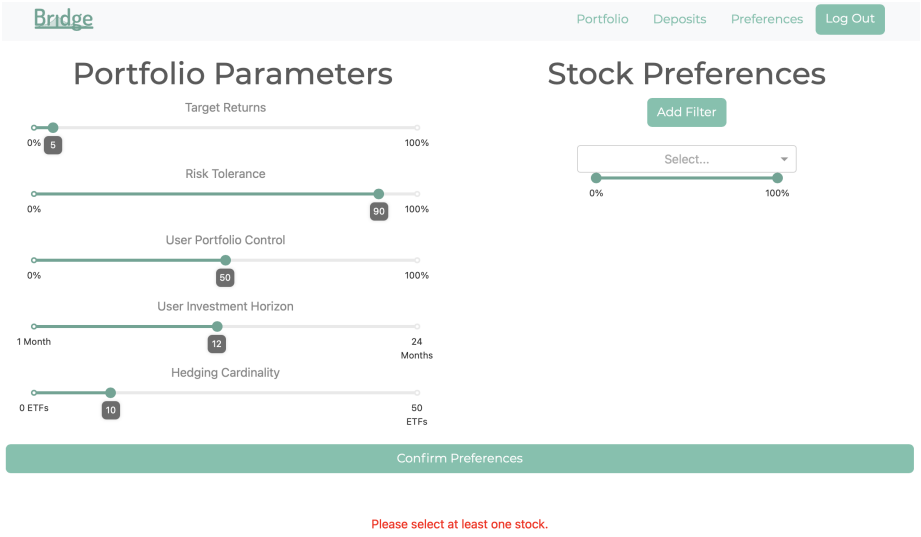


Figure 19: *Stock Picks Page*

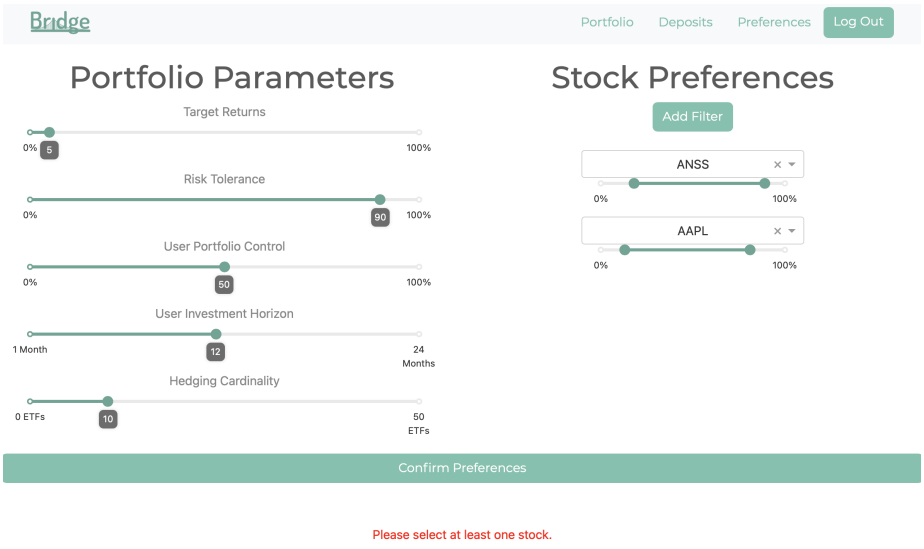


Figure 20: *Stock Picking Inputs*

Figure 21: *New Portfolio - Expected Returns and Backtest*

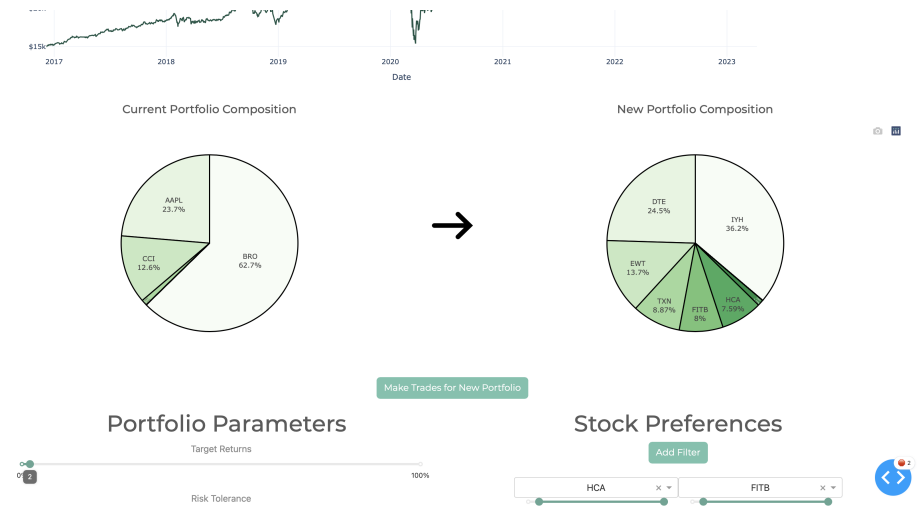


Figure 22: *Portfolio Change & Confirm Trade*

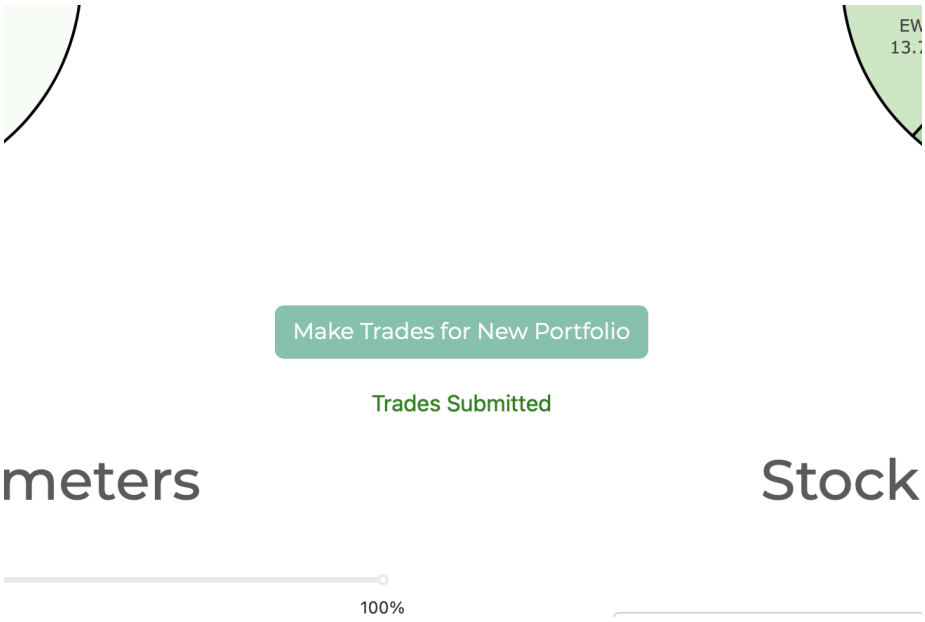


Figure 23: *Trades Submitted Notification*

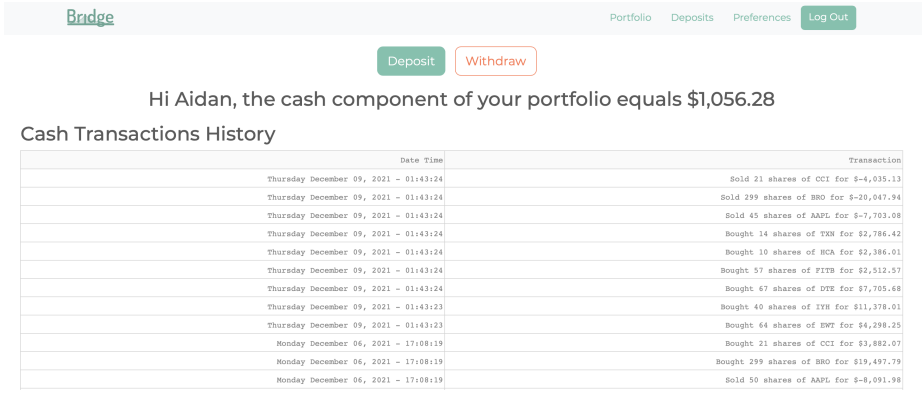


Figure 24: *Trades in Deposits*

Figure 25: *New Main Page*

4.6 Missing Features

4.6.1 Breakdown Page

Initially we wanted to show an extensive breakdown of the users portfolio on a separate page, however all functionality was cleanly incorporated into both the main page, as well as the preferences page. This meant there was no longer the need to burden the user with additional pages that may confuse them.

4.6.2 Online Deployment

Multiple attempts at deploying the web-application to AWS, Google App Engine and Heroku rendered little success. This is due to the complexity of the file layout needed to get the multi-page Dash application functioning. With more time we believe it would be possible to deploy the app and intend to continue exploring and pursuing the previously mentioned options as well as new ones.

4.7 Integration

A major benefit of implementing a pure Dash web-application is that python APIs could be implemented to handle all back-end and business logic integration with the front-end. All functions that interface with either of these are contained in the *backend_access.py* file, which can be accessed by any callback.

Initially we tried to do as much data manipulation as possible in PostgreSQL, however this proved to be difficult and lead to chained views statements that became too complex. Instead we opted to limit our PostgreSQL querries to either simple chained views or single SELECT statements, and handle most of the data manimpulation in Pandas.

All *backend_access.py* functions take the database connection as input (explained in the Back-End section) as well as the current signed-in user's email (retrieved from an account-info Dash Store Component that is instantiated when the user signs in).

As explained above, all retreived data from the backend is then stored in a Dash Store Component, either as a python dictionary or jsonified into json data (which can then be un-jasonified back into Pandas dataframes to be used in tables or graphs). As mentioned previously, the main objective of integration between back-end/business logic and front-end is the minimize unnecessary operations

to maximize performance, see the previous section on Dash Store Components to see how this was achieved.

5 Back-End

The backend component of Bridge Robo-advisor will be responsible for the data, the database management system, and the hosting of our database. To elaborate, we must first choose a source of raw historical data to input into our models. Second, select a database system that can handle all the historical, computed, and user-inputted data. Finally, we must decide on a cloud hosting service for our web application (must host) and our database (optional hosting).

5.1 High-Level Design Goals

To begin the design process we will outline the high-level goals associated with the back-end. It must be stated that the emphasis of Bridge is to generate robust, investment-grade portfolios for our users. Therefore, we must build an infrastructure that can appropriately handle the needs of our users. This means the back-end must be robust enough so that users can interact with our application with quick loading/wait times. Next, it must be stated that the design team is not composed of software engineers by trade and most have little experience with back-end implementation. Hence, we require all of the components of our back-end to be easily integrable with python and will try to select tools that we have some experience with. This will streamline the design and integration of our back-end and reward the design team with more time to focus on the portfolio models.

5.2 Back-End Subsystem Selection

5.2.1 Choice of Data Set

Through a preliminary search, we have identified 7 possible candidates to select a data source from. These data sources are yfinance, Polygon.io, Finnhub API, Alpha Vantage, Tiingo, Google Finance, and Xigite. Below, we will define the constraints and metrics used to eliminate and rank the various different data sets respectively. This will enable the design team to converge towards the best data source for the Bridge application.

5.2.2 Data Set Selection Constraints

- We require at least 20 years of historical data, focusing the search on large data sets will ensure that we have enough historical data to input into the

portfolio models.

- We require that the data set has more than just U.S. ETFs, preferably having all international ETFs.
- We require that the data set is integrable with Python, this is the program where the data set will be analyzed, thus we limit our selection to sets that are compatible with python.

5.2.3 Data Set Selection Metrics

- Familiarity, if any of the design members are familiar with the data source this will reduce implementation time and makes the availability of supporting documentation less important.
- Cost of Operation, lower cost is better.
- Availability of Supporting Documentation, the more documentation the better.
- Data Latency, will the data-source slow the application down when we request new information?
- Data Type. Specifically, the delay, time horizon, number of API calls allowed of the data set.

Data Sources	yfinance [9]	Polygon.io [10]	Finnhub [11]	Alpha Vantage [12]
Familiarity	High	None	None	None
Cost	Free	Free	150/mo	Free
Supporting Doc.	Available	Almost None	Available	Available
Data Latency	Unsure	Almost None	Almost None	Almost None
Data Type	10 minute delay 20 years unsure	End-of-Day delay 2 years 5 Calls/minute	Real-Time 1 year 300 Calls/min	Real-Time 20 years 500 Calls/day

Table 2: Data Set Features

Data Sources	Tiingo [13]	Google Finance [14]	Xignite [15]
Familiarity	None	None	None
Cost	Free	Free	500/mo
Supporting Doc.	Available	Almost None	Almost None
Data Latency	Almost None	Unsure	Almost None
Data Type	Real-Time 30 years 500 Calls/hour	10 minute delay 20 years unsure	Real-Time 20 years Unlimited Calls

Table 3: Data Set Features Continued

Immediately, we can disregard Google Finance. It was found that Google Finance has stopped supporting Python use, thus, violating a data set selection constraint. Furthermore, Polygon.io's free tier and Finnhub do not have at least twenty years of historical data. Hence, they fail the requirements and we will remove them as candidates. The remaining data set options have been placed in a decision matrixs (Table 5) to aid in the convergence process.

	Weights	yfinance	Alpha Vantage	Tiingo	Xignite
Familiarity	2	++	0	0	0
Cost	1	+	+	+	0
Supporting Doc.	1	+	+	+	0
Data Latency	2	?	+	+	+
Data Type	2	+	++	++	+
Total	-	8	8	8	4

Table 4: Data Set Decision Matrix

From the table above, it is evident that yFinance, Alpha Vantage and Tiingo all rank similar. We will disregard Xignite going forward because our lack of familiarity, its high cost, and its low supporting documentation. Next, we compare the advantages of yFinance, Alpha Vantage and Tiingo. Tiingo and Alpha Vantage rank very similarly in all regards and slightly outrank yFinance because their data is over a longer time horizon and is real-time [9]. yFinance is the only data source the design team is familiar with however, we are unable to determine whether it will limit the speed capabilities of the application. Thus, through thorough analysis we will select yFinance as a data source because of our familiarity. If we discover yFinance is infact a source of slower application response times we have two other quality options in Tiingo and Alpha Vantage.

5.2.4 Choice of Database Management System

5.2.5 Database Selection Constraints

- We require the database system is compatible with Python.
- We require the database system is compatible with Cloud Hosting Services.

5.2.6 Database Selection Metrics

- Type of Database Management System, either relational or non-relational.
- Speed of Database Management System, will the chosen database system improve the speed of our application?
- Familiarity, if any of the design members are familiar with the database system this will reduce implementation time and makes the availability of supporting documentation less important.
- Availability of Supporting Documentation, the more documentation the better.

Database System	PostgreSQL [16]	MySQL [17]	MongoDB [18]
Type	Object-Relational	Relational	Non-Relational
Speed	Improves Speed	Improves Speed	Improves Speed
Familiarity	None	Some	None
Supporting Doc	Plenty	Available	Available

Table 5: Database Management System Features

None of the database management systems violate either of the python or cloud hosting services requirements. They will all proceed to the ranking stage (below).

	Weights	PostgreSQL	MySQL	MongoDB
Type	1	+	+	+
Speed	1	+	+	+
Familiarity	2	0	+	0
Supporting Doc	1	++	+	+
Total	-	4	5	4

Table 6: Database Management System Decision Matrix

The result of the decision making matrix above suggest that all the database management systems are strong candidates. Adding a database to the Bridge Portfolio should improve the speed of the application by handling the data more efficiently. This makes the choice of relational/non-relational less important. Additionally, PostgreSQL has only slightly better supporting documentation. However, the design team is slightly more familiar with MySQL and relational databases so we will select MySQL.

5.2.7 Choice of Cloud Hosting Service

5.2.8 Cloud Hosting Service Selection Constraints

- We require the selection of a cloud hosting service that meets all compatibility constraints, chooses a service that works with the selected data-set and database.
- We require the cloud hosting service to be compatible with Plotly-DASH and React, ensures we chose a hosting service that is integrable with the front end.

5.2.9 Cloud Hosting Service Selection Metrics

- Services offered, does the hosting service have full functionality for databases and web applications?
- will the chosen host improve the speed of our application?
- Familiarity, if any of the design members are familiar with the hosting service this will reduce implementation time and makes the availability of supporting documentation less important.
- Cost, lower cost is better.

- Availability of Supporting Documentation, the more documentation the better.

Cloud Hosting Service	Heroku [19]	AWS [20]	Azure [21]	Google Cloud [22]
Services	Web and DataBase	Web and DataBase	Web and DataBase	Web and DataBase
Speed	Improves Speed	Improves Speed	Improves Speed	Improves Speed
Familiarity	Some	None	None	Some
Cost	Free	Free (12 mo.)	Free (12 mo.)	(0.003cent/min)
Supporting Doc	Available	Available	Available	Available

Table 7: Cloud Hosting Service Features

None of the cloud hosting services violate either of the python or database system requirements. Additionally, they are all compatible with the selected front-end. They will all proceed to the ranking stage (below).

	Weights	Heroku	AWS	Azure	Google Cloud
Services	1	+	+	+	+
Speed	1	+	+	+	+
Familiarity	2	+	0	0	+
Cost	1	+	+	+	0
Supporting Doc	1	+	+	+	+
Total	-	6	5	5	5

Table 8: Cloud Hosting Service Decision Matrix

Similar to with the choice of a database, the cloud hosting services all rank well in the decision matrix (above). Hosting the database on the cloud should improve the speed of the web application so they all do well in that regard. Google Cloud Platform also has only a slightly higher cost at only 0.0053 cents per minute. Additionally, there is supporting documentation available for all four but the design team is only familiar with Heroku and GCP. We will select Heroku and only in failure of implementation will we resort to GCP.

5.3 Deviations from Initial Selections

During the implementation of this project some of the selected technologies were changed because it was discovered that they either failed to meet some constraints or due to difficulties associated with the technology. In this section we will outline these deviations.

5.3.1 Data Sources

In the progress report, we noted that yfinance could be a possible source of latency with regard to downloading and updating data. Through implementation, this was not discovered to be an issue. There is high latency involved with downloading from yfinance however, this is only an issue for the initial bulk download of data. For our application, we required 20 years of historical data on securities from the TSX/60 Index, the S P 500 Index, and a large collection of Canadian and American ETFs. Although the download time was slower, we did not have to deal with capped API requests so we could bulk download the majority of the data overnight. Additionally, yfinance was fast enough to complete overnight updates so we continued to use it as the primary data source.

Through implementation, some other datasets were discovered to be necessary and were acquired when relevant. First, we expanded our universe considered securities since the project proposal to include Canadian and American Securities. Thus, we retrieved currency pair data from USD to CAD over the entire investment horizon to be able to keep track of the value of American securities in Canadian dollars for our Canadian clients [23]. Furthermore, we needed factor data for the Fama-French Five-Factor Model to be able to input into our factor estimation models. The North American five Fama-French factors were obtained on a daily and monthly basis from the following resource [24].

5.4 Database Management System

In the progress report, it was noted that MySQL would be used as the database management system however the design team pivoted to use PostgreSQL quickly in the implementation process. For all the same reasons above, PostgreSQL is convenient for the design team because of our familiarity with relational databases. Additionally, the supporting documentation for PostgreSQL was found to be in fact much superior not just slighlty. This was the primary reason for the change. For example, a step-by-step guide for hosting a PostgreSQL database on the cloud was found. The plethora of supporting documentation surrounding PostgreSQL made implementing the backend of an application much easier for a design team with limited knowledge of database management systems.

5.4.1 Cloud Hosting

In the progress report, we noted that all of the considered cloud hosting services met our requirements, ranked similarly and that we would initially use Heroku. However, since we expanded our investment horizon to include American securities as well, Heroku no longer met the requirements. Heroku Cloud Databases have a free tier limit of 10,000 rows and these additional securities pushed us over the limit. Left with two remaining two options: Amazon Azure and Google Cloud, we selected Amazon Azure because of their one-year free trial period. Additionally, the supporting documentation regarding mounting a PostgreSQL database on the cloud was very strong as mentioned above.

5.5 ERD Diagram and Program Flow

Here, we have provide an entity relational diagram below to outline the flow and logic of our application. Important tables have been given green headers, business logic models are highlighted in yellow and front-end outputs are described by pink boxes.

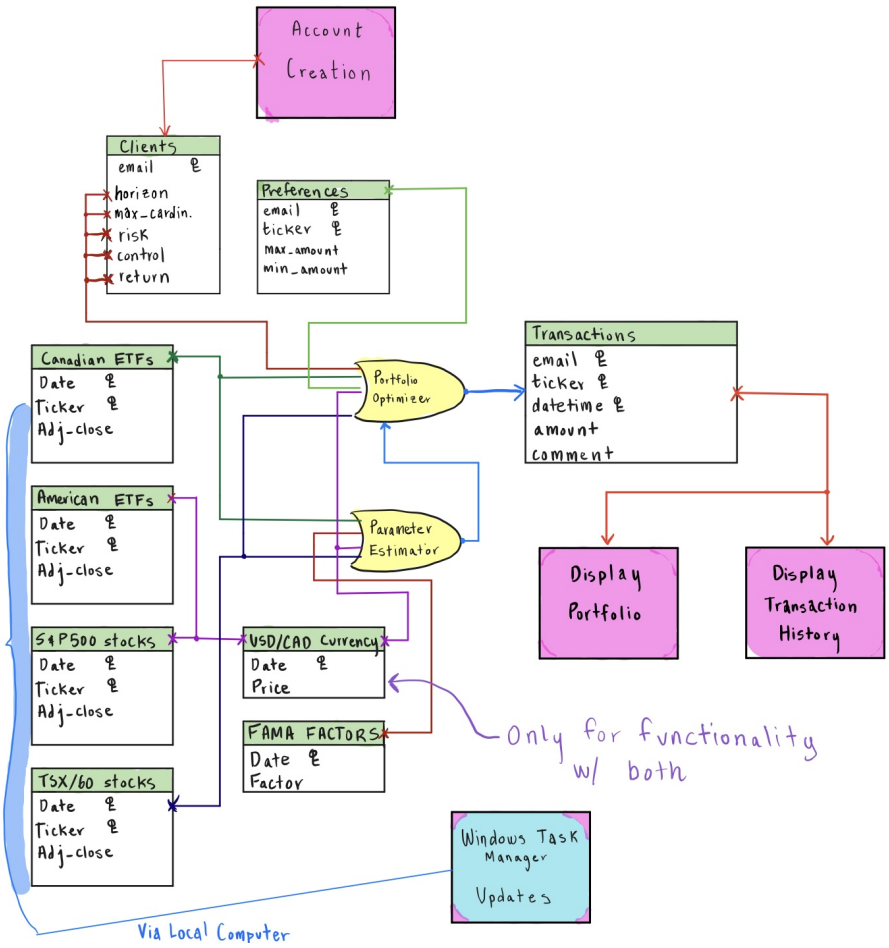


Figure 26: *Bridge Robo-Advisor ERD Diagram*

Next the important entities of the program will be outlined in greater depth and summaries of their inputs and outputs will be given.

Parameter Estimator The parameter estimator is a business logic model, that will take in historical data as its inputs and return the parameters used in our portfolio optimizer business model. All of the historical data taken in as an input must be appropriately keyed by date, so that the parameter estimator model can specifically train parameters in the desired time horizons. The Fama-French historical factor return data is unique so it does not require another primary key. However, the data for securities is

only unique by ticker and date so it requires two primary keys. After, the parameter estimation model has been trained, the outputted parameters will be used indefinitely so they will be manually coded into the portfolio optimizers and will not need to be stored in a table.

Portfolio Optimizer The portfolio optimizer is another business logic model, it deals with the initial construction of our portfolios and re-balancing that will take place throughout time. The portfolio optimizer will take in three inputs: the universe of securities considered, the portfolio targets and, the parameters obtained from the parameter estimation results. For our application, the universe of stocks considered consists of all American and Canadian ETFs along with the stocks specifically chosen by our users which will be obtained from the 'preferences' table. The portfolio targets, are the target risk, control and return parameters specific to each client. The portfolio optimizer will then output a set of weights of the securities to be purchased when we re-balance. These transactions will then be updated on our transactions table.

Transactions Table The design team has decided to implement a single transactions table for all of our users instead of implementing portfolio tables specific to the client. With all the transactions summarized in a single table we can query the 'transaction' table by client to produce their entire transaction history and display the evolution of their portfolio throughout time. The transactions table along with the securities price data can be then used to create all of the displays and elements of our application. We achieve this by querying the client table by client, the day of the transaction, and the ticker. Then query the prices table for current price of the specific securities.

5.6 Data Schemas, Dictionaries and Constraints

5.6.1 Data Schemas

Now, we outline the database schema to represent the configuration and connections between the tables and elements of our relational database. A database schema has been provided below.

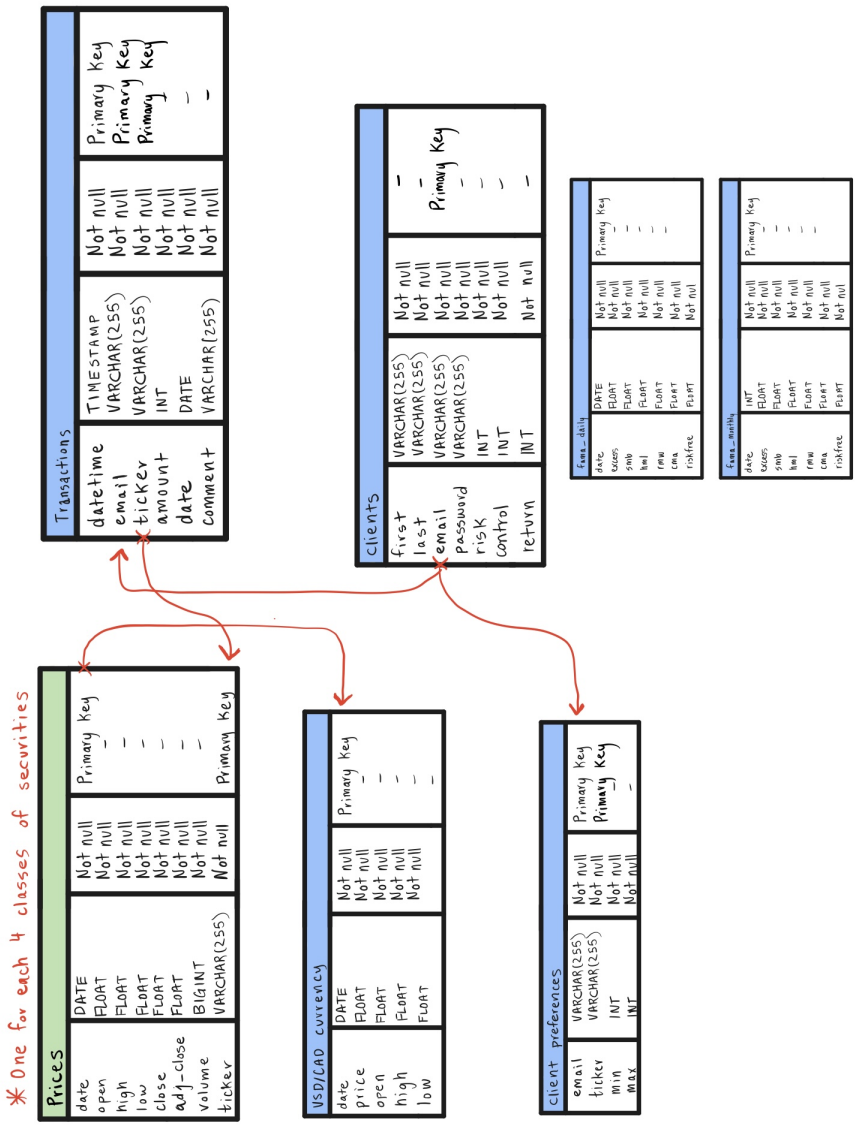
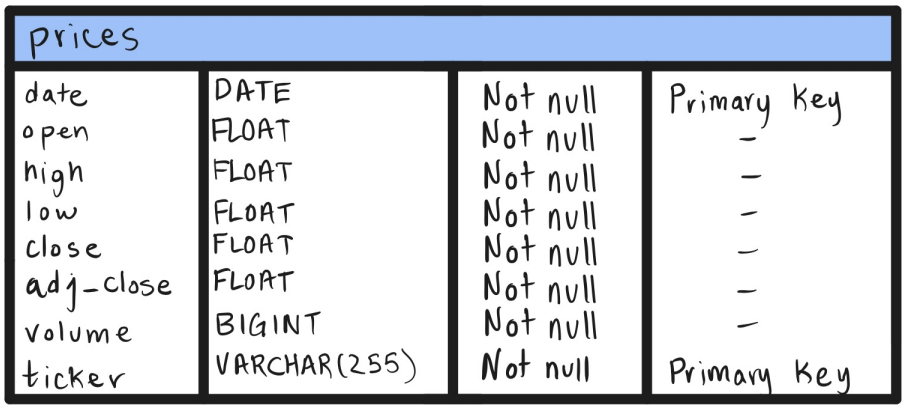


Figure 27: *Bridge Robo-Advisor Database Schema*

1. For the American securities we must adjust for the currency conversion, this relation can be achieved by querying the currency table by date.
2. All of our clients will have unique emails, we can the relate the client table to the transactions and preferences table by their emails.
3. Next, we can relate the transactions table to the securities table by their unique tickers.
4. Finally, the Fama-French factor tables will have no relation to any other table in the database.

5.6.2 Table Dictionaries and Constraints

All of the tables in our database will conform to the same set of constraints. First, all columns are constrained to be not null and second the primary keys were selected as the unique identifiers for each table.



prices			
date	DATE	Not null	Primary Key
open	FLOAT	Not null	-
high	FLOAT	Not null	-
low	FLOAT	Not null	-
close	FLOAT	Not null	-
adj_close	FLOAT	Not null	-
volume	BIGINT	Not null	-
ticker	VARCHAR(255)	Not null	Primary Key

Figure 28: *Prices Table's dictionary and constraints.*

USD/CAD currency			
date	DATE	Not null	Primary Key
price	FLOAT	Not null	–
open	FLOAT	Not null	–
high	FLOAT	Not null	–
low	FLOAT	Not null	–

Figure 29: *USD/CAD Currency Table's dictionary and constraints.*

clients			
first	VARCHAR(255)	Not null	–
last	VARCHAR(255)	Not null	–
email	VARCHAR(255)	Not null	Primary Key
password	VARCHAR(255)	Not null	–
horizon	INT	Not null	–
MAX_card	INT	Not null	–
risk	INT	Not null	–
control	INT	Not null	–
return	INT	Not null	–

Figure 30: *Client Table's dictionary and constraints.*

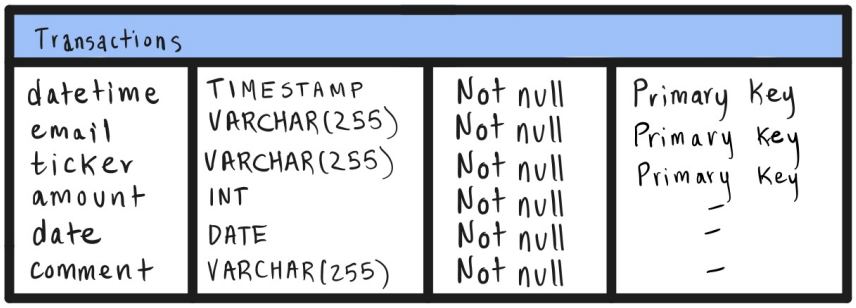


Figure 31: *Transaction Table's dictionary and constraints.*

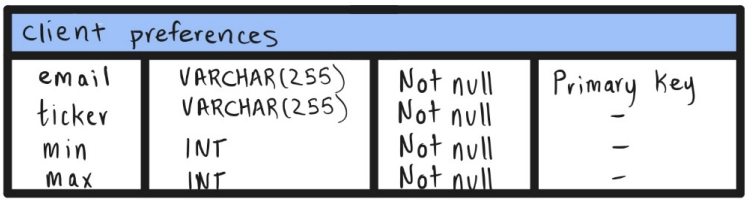


Figure 32: *Client Preferences Table's dictionary and constraints.*

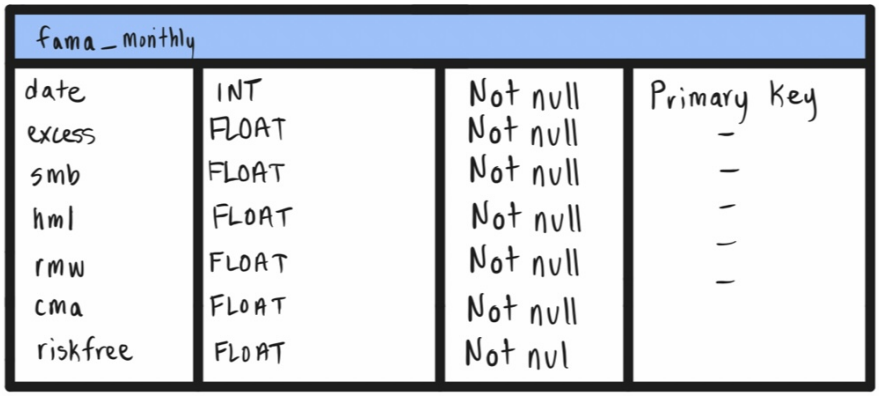


Figure 33: *Fama-French Monthly Table's dictionary and constraints.*

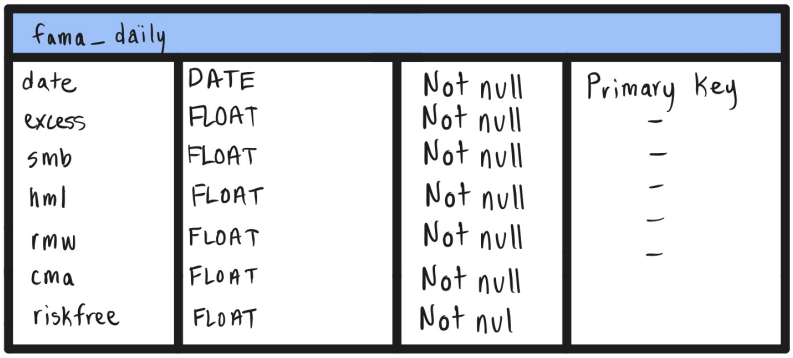


Figure 34: *Fama-French Daily Table's dictionary and constraints.*

5.7 Cloud Hosting and Authentication

As mentioned above, a very strong piece of supporting documentation was found regarding creating cloud PostgreSQL instances and uploading databases to the cloud. As mentioned in the subsystem selection section we decided to use Amazon Azure RDS to be the cloud host for the Bridge Robo-Advisor application.

The process of moving the database from our local machines to the cloud was quite simple. First, we created a cloud PostgreSQL database and replicate all of the table definitions from our local database. Next we can migrate the entire contents of the database via the backup and restore method within PGAdmin4 the application for PostgreSQL databases.

It must be noted that since our investment universe is quite large (Roughly 1500 securities over 20 years) we initially thought that it would be beneficial to load the data straight into a cloud database to save storage on our local machines. This proved to be very slow for bulk data import. This is probably because of the network latency involved with multiple connections (yfinance to local to cloud database). So instead, the bulk of the data was first downloaded locally and then pushed to the cloud, however data updates will be made straight to the cloud.

The credentials and authentication requirements for our cloud database have been included below for those that would like to inspect the data.

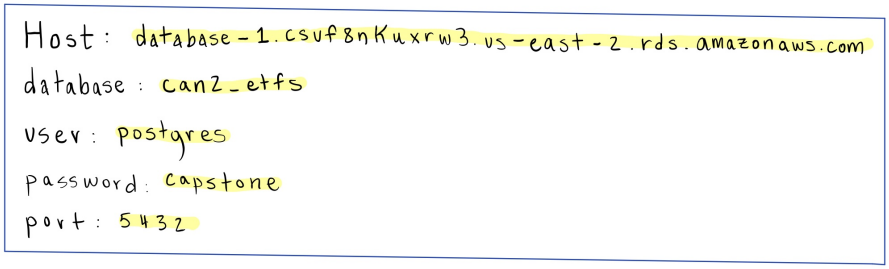


Figure 35: *Amazon RDS database credentials*

5.8 Front and Back End Integration

The psycopg2 [25] python library proved to be a keystone for the front and back-end integration. Psycopg2 is a library specifically for adapting PostgreSQL databases in python. By using the psycopg2 library to connect to the database we can now query the data using SQL commands in python. This enabled the integrated with the front end done in dash which requires python.

5.9 Back End Functionality

The Back End of the current iteration of the Bridge Robo-Advisor application has the following functionalities:

- Automatic Updates. Currently automatic updates are being completed by a local computer running a python script via windows task manager. The file has been included with this report. This task is set to complete every

night at 1 a.m. and and updates the prices of all included and American securities.

- Functionality for only American securities. The current iteration only supports functionality for American Securities (SP500 stocks and US ETFs). This decision was ultimately made to streamline the user interface, by limiting the number of securities.

6 Business Logic

The aim of the project is to deliver a user-friendly service to clients searching for ETF-based portfolio. The business logic is responsible for generating the portfolio from a set of ETFs that meets or exceeds the client's target return with minimum risk. In this section, we introduce a number of portfolio optimization models along with their corresponding pros and cons and plans to backtest the models using performances metrics. Since all models use some form of mean returns and volatility measures, we also discuss several parameter estimation methods as well as how to evaluate them.

6.1 Portfolio Optimization

6.1.1 Multi-Period Framework

Throughout the report, we will assume the following:

- portfolio of n assets;
- a cash account to handle trading costs (i.e. transaction costs, holding costs, etc.);
- over the finite time horizon, it is divided into discrete time periods labelled $t = 1, 2, \dots T$; and
- portfolio optimized at time t

Single-period optimization (SPO) only looks forward 1 immediate time step, i.e. using information from time $t = 1, 2, \dots t-1$, we attempt to create an optimal portfolio trading strategy at time t. In general, the optimization problem can be written as follows:

$$\begin{aligned} \max_{z_t} \quad & \hat{R}_t^p(w_t + z_t) - \gamma^{risk}\psi_t(w_t + z_t) \\ \text{s.t.} \quad & z_t \in Z_t, \quad w_t + z_t \in W_t \\ & \mathbf{1}^\top z_t = 0 \end{aligned} \tag{1}$$

where z_t, the variable we are solving for, is the normalized trades at time t, w_t is the normalized portfolio weights at time t, $\hat{R}_t^p(w_t + z_t)$ is the estimated portfolio return, $\psi_t(w_t + z_t$ is the risk measure, $\gamma^{risk} > 0$ is the risk-aversion parameter, Z_t is the set of trade constraints, and W_t is the set of holding constraints. Here the objective is to maximize the risk-adjusted estimated net return un-

der the self-financing constraint and possibly additional trade/hold constraints. Note that using convex objective function, including return and risk measures, and constraints is encouraged as it guarantees global optimality and allows the problem to be solved quickly. To illustrate, the exact self-financing constraint is $\mathbf{1}^\top z_t + \hat{\phi}_t(w_t, z_t) = 0$, where $\hat{\phi}_t(w_t, z_t)$ is the estimate total trading costs at time t, but it can be relaxed into $\mathbf{1}^\top z_t = 0$ for small trading costs to make the problem simpler. We will see throughout the section where we transform optimization problems that are originally non-convex into convex ones.

The problem with the single-period model is that it does not take future estimations into account. If the optimal portfolio determined at some time period t differs a lot from the one at time $t+1$, this leads to portfolio turnover and ultimately accumulating a considerable amount of transaction costs. Moreover, SPO is not capable of handling multiple, possibly conflicting return estimates on different time scales. For example, suppose we predict that a return will be positive over a short period, but over a longer period it will be negative. In a SPO framework, it is not clear how to account for the different time scales when blending the return predictions.

These motivates the use of multi-period optimization (MPO) where we attempt to create an optimal portfolio trading strategy at time t by solving an optimization problem over a planning horizon that extends H time periods into the future [23]. In general, the optimization problem can be written as follows:

$$\begin{aligned} \max_{z_\tau} \quad & \sum_{\tau=t}^{t+H-1} (\hat{R}^p_{\tau|t}(w_\tau + z_\tau) - \gamma^{risk}\psi_\tau(w_\tau + z_\tau)) \\ \text{s.t.} \quad & z_\tau \in Z_\tau, \quad w_\tau + z_\tau \in W_\tau \\ & \mathbf{1}^\top z_\tau = 0 \\ & w_{\tau+1} = w_\tau + z_\tau \\ & \tau = t, \ldots, t+H-1 \end{aligned} \tag{2}$$

where $\hat{R}^p_{\tau|t}$ is the estimated portfolio return at time τ given information up to time t. Note that the recursive constraint $w_{\tau+1} = w_\tau + z_\tau$ is added to satisfy the self-financing condition between two successive time periods. Similar to SPO, MPO encourages convex objective and constraints for quick computation and global optimality.

For the advantages that it offers, MPO framework is adopted throughout the

project and will be a part of our design constraints.

6.1.2 Design Decisions

While there are many researched models, it is not feasible to investigate all of them within the given time constraint of the project. Thus we will frame our discussion to models we have studied throughout Engineering Science EMSF curriculum and their variants. After thorough evaluation, if none of the models are deemed successful we may expand our scope to consider a wider range of models.

When making design decisions, we should consider the following objectives of a portfolio optimization model:

- the generated portfolio meets or exceeds the client's target return; and
- it should minimize the risk.

The following constraints also must be satisfied:

- the portfolio consists of only ETFs;
- multi-period framework is used;
- the model allows user inputs (target return, time horizon, etc.); and
- the model should not hinder user-friendliness of the design.

The objectives and constraints are directly deduced from the project requirement. We tried not to include other objectives and constraints beyond the project requirement because we believe the model should be chosen based on their performance and the selection process should not be influenced by external reasons. Although not explicitly stated above, we should also be aware of time constraint (one semester) of the project and feasibility of implementing the model.

After trimming the list of possible candidates using the specified constraints, we will use metrics constructed based on the objectives to evaluate each model. Details about this is discussed in the subsection *Metrics & Evaluation* under the section *Portfolio Optimization*.

6.1.3 Efficient Frontier Models

Efficient frontier models use convex optimization techniques to solve problems where their objective and constraints are some combination of portfolio performance and risk. Provided that the objective and constraints are convex, efficient frontier framework is a good starting point to develop a MPO for its simple structure. Thus they will serve as baselines for other models we will discuss later in the section.

Mean-Variance Optimization (MVO) The classical approach, first devised by Henry Markowitz in 1952, uses mean returns and standard deviations as performance and risk measures, respectively. The problem can be formulated in multiple ways, which are all mathematically equivalent, but we will list two of them below:

$$\begin{aligned} \min_{z_\tau} \quad & \sum_{\tau=t}^{t+H-1} \Big((w_\tau + z_\tau)^\top Q_\tau (w_\tau + z_\tau) + \gamma^{trade} \hat{\phi}_\tau (w_\tau, z_\tau) \Big) \\ \text{s.t.} \quad & \mu_\tau^\top (w_\tau + z_\tau) \geq R \\ & \mathbf{1}^\top z_\tau = 0 \\ & w_{\tau+1} = w_\tau + z_\tau \\ & \tau = t, \ldots, t+H-1 \end{aligned} \tag{3}$$

$$\begin{aligned} \max_{z_\tau} \quad & \sum_{\tau=t}^{t+H-1} \Big(\mu_\tau^\top (w_\tau + z_\tau) \\ & \qquad - \lambda (w_\tau + z_\tau)^\top Q_\tau (w_\tau + z_\tau) - \gamma^{trade} \hat{\phi}_\tau (w_\tau, z_\tau) \Big) \\ \text{s.t.} \quad & \mathbf{1}^\top z_\tau = 0 \\ & w_{\tau+1} = w_\tau + z_\tau \\ & \tau = t, \ldots, t+H-1 \end{aligned} \tag{4}$$

where R is the target returns given by the client, μ_τ is the mean asset returns, Q_τ is the asset covariance matrix, $\hat{\phi}_\tau(w_\tau, z_\tau)$ is the trading costs, λ is the risk-aversion parameter, and γ^{trade} is the trading costs coefficient.

The former explicitly states the goal of the project, i.e. minimize the risk and transaction costs while meeting or exceeding the target returns, while the latter

maximizes the risk-adjusted returns and provides the flexibility of tuning the risk-aversion parameter λ based on the client's risk tolerance. Both formulations are convex since a covariance matrix Q is always positive semi-definite (PSD) by construction and the mean returns μ is linear.

MVO has been popularized for its intuitive appeal and theoretical property as the pareto-optimal in-sample allocation [27]. Despite its wide application, it faces the following criticisms:

- Known as Markowitz's curse, it is numerically instable due to sensitivity to estimation error [28].
- It only considers the first two moments of the returns distribution from means and variances. It naively assumes the distribution is symmetric.
- It often produces portfolios that are over-concentrated.

Conditional Value-at-Risk (CVaR) Optimization MVO seek to minimize both up-side and down-side risk, however, intuitively speaking, we are not adverse to returns higher than our expected return. Therefore we attempt to implement an optimization model which minimizes solely down-side risk. We can define a metric value-at-risk (VaR_α) as follows:

$$VaR_\alpha(\boldsymbol{x}) = min\{\gamma \in \mathbb{R}^{n\times n} : \Psi(\boldsymbol{x}, \gamma) \geq \alpha\}$$

where the cumulative distribution function, $\Psi(\boldsymbol{x}, \gamma)$ is $\Psi(\boldsymbol{x}, \gamma) = \int_{f(\boldsymbol{x},\boldsymbol{r})<\gamma} p(\boldsymbol{r})d\boldsymbol{r}$. Note that $f(\boldsymbol{x}, \boldsymbol{r})$ is the loss in the event of our random asset return $\boldsymbol{r}$, and γ is the loss. Additionally, $p(\boldsymbol{r})$ is the probability density function of our vector of random asset returns.
VaR_α measures the minimum loss that we will match or exceed with probability $1 - \alpha$. However, minimizing VaR_α is a non-convex activity, i.e. it is a difficult optimization problem. Hence we introduce a new variable conditional value-at-risk, defined below:

$$CVaR_\alpha(\boldsymbol{x}) = \frac{1}{1-\alpha} \int_{f(\boldsymbol{x},\boldsymbol{r})\geq VaR_\alpha(\boldsymbol{x})} f(\boldsymbol{x}, \boldsymbol{r})p(\boldsymbol{r})d\boldsymbol{r}$$

This new variable measures the expected value of the losses that are greater than or equal to VaR_α. Some problems arise, however, as if we try to minimize $CVaR_\alpha(\boldsymbol{x})$, it still contains the $VaR_\alpha(\boldsymbol{x})$ term which is non-convex. In order to

amend this, we introduce the auxiliary variable γ and function $F_\alpha(\boldsymbol{x}, \gamma)$, defined below:

$$F_\alpha(\boldsymbol{x}, \gamma) = \gamma + \frac{1}{1-\alpha} \int (f(\boldsymbol{x}, \boldsymbol{r}) - \gamma)^+ \, p(\boldsymbol{r} d\boldsymbol{r}$$

Where $(f(\boldsymbol{x}, \boldsymbol{r}) - \gamma)^+ = max\{0, f(\boldsymbol{x}, \boldsymbol{r}) - \gamma\}$, which can be replaced with the auxiliary variable y_s, defined below:

$$\begin{aligned} y_s &\geq 0, \quad s = 1, .., S \\ y_s &\geq f(\boldsymbol{x}, \boldsymbol{r_s}) - \gamma, \quad s = 1, .., S \end{aligned}$$

Finally, since $p(\boldsymbol{x})$ is difficult to solve for analytically, we will implement a scenario based approach using Monte Carlo simulations. These scenarios, $\hat{\boldsymbol{r}}_{\boldsymbol{s}}$ for $s = 1, .., S$, are determined to be equally likely, therefore we can simplify the auxiliary function $F_\alpha(\boldsymbol{x}, \gamma)$ as the estimation $\hat{F}_\alpha(\boldsymbol{x}, \gamma)$, and with z_s introduced, it is defined below:

$$\hat{F}_\alpha(\boldsymbol{x}, \gamma) = \gamma + \frac{1}{(1-\alpha)S} \sum_{s=1}^{S} y_s$$

Rendering our final $CVaR_\alpha(\boldsymbol{x})$ optimization model under MPO framework, we have the following:

$$\begin{aligned} \min_{z_\tau, y_{\tau,s}, \gamma_\tau} \quad & \sum_{\tau=t}^{t+H-1} \left(\gamma_\tau + \frac{1}{(1-\alpha)S} \sum_{s=1}^{S} y_{\tau,s} + \gamma^{trade} \hat{\phi}_\tau(w_\tau, z_\tau) \right) \\ \text{s.t.} \quad & \mu_\tau^\top (w_\tau + z_\tau) \geq R \\ & \mathbf{1}^\top z_\tau = 0 \\ & w_{\tau+1} = w_\tau + z_\tau \\ & y_{\tau,s} \geq 0 \\ & y_{\tau,s} \geq f(\boldsymbol{x}, \boldsymbol{r_s}) - \gamma \\ & \tau = t, \ldots, t+H-1 \\ & s = 1, \ldots, S \end{aligned} \tag{5}$$

While CVaR optimization is similar to MVO, it addresses the issue of MVO that it naively assumes the returns distribution symmetry by using a tail-based risk measure. CVaR technique can especially come in handy when the client is

interested in preventing worst-case scenarios.

6.1.4 Leveraged Models

Sharpe Ratio (SR) Optimization Sharpe ratio is a performance metric that measures the excess rate of return per unit of risk for a given asset or portfolio, formally defined as:

$$SR_p = \frac{\mathbb{E}[r_i] - r_f}{\sqrt{\text{var}(r_i - r_f)}} = \frac{\boldsymbol{\mu}^\top \boldsymbol{x} - r_f}{\sqrt{\boldsymbol{x}^\top \boldsymbol{Q} \boldsymbol{x}}}$$

This function, however, is not linear, and a transformation must be made to amend this which is as follows:

$$\kappa = \frac{1}{(\boldsymbol{\mu} - \overline{\boldsymbol{r}}_f)^\top \boldsymbol{x}}, \quad \boldsymbol{y} = \kappa \boldsymbol{x}$$

Substituting this into the vector notation of SR_p yields:

$$\begin{aligned} SR_p &= \frac{(\boldsymbol{\mu} - \overline{\boldsymbol{r}}_f)^\top \boldsymbol{x}}{\frac{1}{\kappa}\sqrt{\boldsymbol{y}^\top \boldsymbol{Q} \boldsymbol{y}}} \\ &= \frac{1}{\sqrt{\boldsymbol{y}^\top \boldsymbol{Q} \boldsymbol{y}}} \end{aligned}$$

We can now maximize the Sharpe Ratio by minimizing $\boldsymbol{y}^\top \boldsymbol{Q} \boldsymbol{y}$ over $\boldsymbol{y}, \kappa$, and the constraints become:

$$(\boldsymbol{\mu} - \overline{\boldsymbol{r}}_f)^\top \boldsymbol{y} = 1, \mathbf{1}^\top \boldsymbol{y} = \kappa$$

Rending the following optimization problem under MPO, we have the following:

$$\begin{aligned} \min_{y_\tau, \kappa} \quad & \sum_{\tau=t}^{t+H-1} \left(y_\tau^\top Q_\tau y_\tau + \gamma^{trade} \hat{\phi}_\tau(y_{\tau-1}, y_\tau) \right) \\ \text{s.t.} \quad & (\mu - \overline{r}_f)^\top y_\tau = 1 \\ & \mathbf{1}^\top y_\tau = \kappa \\ & \kappa \geq 0 \\ & \tau = t, \ldots, t + H - 1 \end{aligned} \tag{6}$$

Finally, the optimal portfolio weights may be recovered as:

$$z_i^* = \frac{y_i^*}{\kappa^*} = \frac{y_i^*}{\sum_{j=1}^{n} y_j^*}$$

Note that SR optimization model still yields a solution on the efficient frontier, but the one that has the greatest Sharpe ratio. The solution is sometimes referred to as the tangency portfolio as it is the tangent point between the capital allocation line (CAL) and the efficient frontier [29]. *Figure 36* illustrates this.

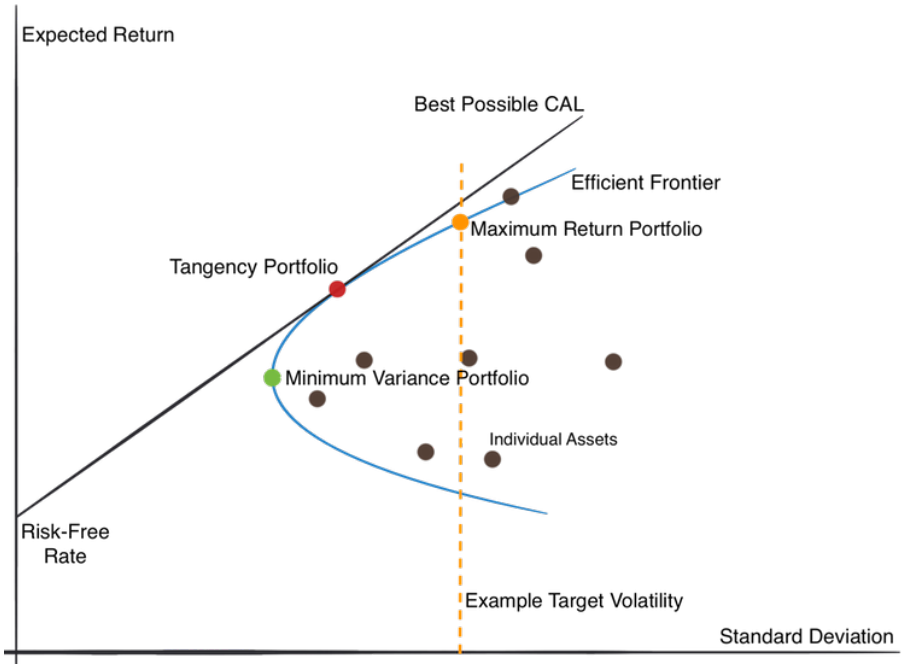


Figure 36: *Efficient frontier and tangency portfolio.*

Risk Parity Optimization The purpose of risk parity optimization, also known as equal risk contribution, is to generate a portfolio with equally distributed risk by allocating capital such that the contributions of risk from any one asset is equal to the contributions of another.

One of the key advantages of this optimization technique is that our portfolio is not subject to error in the estimations of expected returns, as the model minimizes solely based on the covariance matrix $\boldsymbol{Q}$. This optimization method

looks to find a portfolio such that $\boldsymbol{R}_i = \boldsymbol{R}_j \forall i, j$, where $\boldsymbol{R}_i$ is the individual risk contribution of asset i, and $\boldsymbol{R}_i = \boldsymbol{x}^\top \boldsymbol{A}_i \boldsymbol{x}$. $\boldsymbol{A}_i \in \mathbb{R}^{n \times n}$ is the individual risk contribution of asset i derived from $\boldsymbol{Q}$, defined below:

$$\boldsymbol{A}_i = \begin{bmatrix} 0 & \cdots & 0 & \frac{1}{2}\sigma_{1i} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \vdots & 0 & \cdots & 0 \\ \frac{1}{2}\sigma_{i1} & \cdots & \cdots & \frac{1}{2}\sigma_{ii} & \cdots & \cdots & \frac{1}{2}\sigma_{in} \\ 0 & \cdots & 0 & \vdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \frac{1}{2}\sigma_{ni} & 0 & \cdots & 0 \end{bmatrix},$$

An issue arises, however, as each $\boldsymbol{A}_i$ is indefinite, having its own positive eigenvalue and negative eigenvalue, with all other eigenvalues being zero, leading to non-convexity in minimization function, shown below, and many local minimum.

$$\min_{\mathbf{x}} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i^\top \boldsymbol{A}_i x_i) - (x_j^\top \boldsymbol{A}_j x_j))^2$$

One workaround to this is to disallow short selling, resulting in the new optimization problem that is well-behaved within this smaller feasible region:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i^\top \boldsymbol{A}_i x_i) - (x_j^\top \boldsymbol{A}_j x_j))^2 \\ s.t. \quad & \mathbf{1}^\top \boldsymbol{x} = 1 \\ & \boldsymbol{x} \geq \mathbf{0} \end{aligned}$$

Some issues still remain, such as the 4^{th} degree polynomial being extremely non-linear. One solution to this, and what will be employed in this project, is rendering a convex model using some decision variable $\boldsymbol{y} \in \mathbb{R}^n$. Now define the function of this decision variable as:

$$f(\boldsymbol{y}) = \frac{1}{2} \boldsymbol{y}^\top \boldsymbol{Q} \boldsymbol{y} - c \sum_{i=1}^{n} \ln y_i$$

where c is some arbitrary positive scalar. Since $\boldsymbol{Q}$ is PSD and natural logarithmic function is strictly concave, $f(\boldsymbol{y}$ is a strictly convex function, allowing us to

take gradient, set it to zero and achieve a few desirable properties:

$$\bigtriangledown f(\boldsymbol{y}) = \boldsymbol{Q}\boldsymbol{y} - c\boldsymbol{y}_i^{-1} = 0$$

Where $\boldsymbol{y}^{-1} = [\frac{1}{y_1}, ..., \frac{1}{y_n}]$. Therefore:

$$\begin{aligned}(\boldsymbol{Q}y)_i &= \frac{c}{y_i} \quad \forall i \\ (y_i\boldsymbol{Q}y)_i &= c \quad \forall i \\ (y_i\boldsymbol{Q}y)_i &= (y_j\boldsymbol{Q}y)_j \quad \forall i, j\end{aligned}$$

which achieves our desired outcome. Prohibiting short selling, and recovering the optimal portfolio weights after minimizing over our decision variable yields the risk parity optimization problem under MPO framework like the following:

$$\begin{aligned}\min_{y_\tau} \quad & \sum_{\tau=t}^{t+H-1} \left(\frac{1}{2} y_\tau^\top Q_\tau y_\tau - c \sum_{i=1}^{n} \ln y_{\tau,i} + \gamma^{trade} \hat{\phi}_\tau(y_{\tau-1}, y_\tau) \right) \\ \text{s.t.} \quad & y_\tau \geq 0 \\ & \tau = t, \ldots, t+H-1 \\ & \therefore z_i^* = \frac{y_i^*}{\sum_{i=1}^n y_i^*} \quad \forall i\end{aligned} \tag{7}$$

Risk parity optimization method addresses the issue of MVO that it often produces over-concentrated portfolios by intentionally distributing equal amount of risk contribution to each asset. Hence solutions tend to be very diversified. It also has an interesting property that the projected volatility of RP portfolio is between that of minimum-variance portfolio and equally-weighted portfolio, i.e. $\sigma_{MV} \leq \sigma_{RP} \leq \sigma_{EW}$.

Leveraging Both models are not capable of controlling the projected portfolio returns, which violates one of the project constraints. However the two-fund separation theorem states that any efficient portfolio can be replicated as a linear combination of two efficient portfolios. Hence we can leverage the risk parity (assuming it is approximately close to the efficient frontier) or SR-optimized portfolio with a portfolio generated using efficient frontier models.

Say x_τ is the risk parity or SR-optimized portfolio and y_τ is the efficient portfolio. Then the leveraged portfolio can be written as $z_\tau = \delta x_\tau + (1-\delta) y_\tau$ for some $0 \leq \delta \leq 1$. Since we want to include the risk parity or SR-optimized portfolio as much as possible, we want to maximize δ while meeting the target returns constraint. It can be formulated as an optimization problem like the following:

$$\begin{aligned} \max_{\delta} \quad & \delta \\ \text{s.t.} \quad & \mu_\tau^\top (\delta x_\tau + (1-\delta) y_\tau) \geq R \\ & 0 \leq \delta \leq 1 \\ & \tau = t, \ldots, t+H-1 \end{aligned} \tag{8}$$

where R is the client's target return.

To take this even further, if we have access to a money-market ETF, we can designate it as a risk-free asset and apply one-fund separation theorem, which states that any efficient portfolio can be constructed as a linear combination of the tangency portfolio, i.e. SR-optimized portfolio, and a risk-free asset. In this case, we can leverage the SR-optimized portfolio with the money-market ETF such that the target return is achieved.

6.1.5 Robust Models

The fundamental limitation of estimating a reliable expected returns, referred to as mean blur, along with the difficulty to forecast accurate mean returns is one of the key challenges of the efficient frontier approach. The resulting portfolios from the MVO are usually very sensitive to parameter estimation. To overcome this, the robust counterpart of MVO has been proposed.

Box Uncertainty If we assume the true expected returns lie within a box uncertainty set:

$$\mu^{true} \in \{\mu^{true} \in \mathbb{R}^n : |\mu_i^{true} - \mu_i| \leq \delta_i, i = 1, \ldots, n\} \tag{9}$$

where μ^{true} is the true expected returns and δ_i is the maximum distance we assume exists between our estimate and the true expected returns, then the target constraint becomes:

$$\mu_\tau^\top (w_\tau + z_\tau) - \delta^\top |w_\tau + z_\tau| \geq R \tag{10}$$

Ellipsoidal Uncertainty Realizations in the corners of the box constraints might be too extreme. Instead we may want to create an ellipsoidal uncertainty set like the following:

$$\mu^{true} \in \{\mu^{true} \in \mathbb{R}^n : (\mu_i^{true} - \mu_i)^\top \Theta^{-1} (\mu_i^{true} - \mu_i) \leq \epsilon_2^2, i = 1, \ldots, n\} \quad (11)$$

where μ^{true} is the true expected returns and ϵ_2^2 is the maximum confidence interval we assume exists between our estimate and the true expected returns, then the target constraint becomes:

$$\mu_\tau^\top (w_\tau + z_\tau) - \epsilon_2 \|\Theta^{1/2}(w_\tau + z_\tau)_2\| \geq R \quad (12)$$

6.1.6 Hierarchical Clustering Framework

Hierarchical clustering asset allocation is a framework to cluster assets by measuring similarity between assets [30]. Once clustered, we can perform one of the known portfolio optimization techniques within each cluster. The clustering occurs through three stages: tree clustering, quasi-diagonalization, and recursive bisection. A commonly used metric to examine the similarity of assets is correlation coefficients between stock returns. The intuitive appeal of hierarchical clustering is the ability to allocate capital to groups of similar assets.

While this framework brings many merits, two major advantages are robustness and flexibility. Since the assets are initially diversified through clustering process, the portfolio tends to be diversified and the weights are stable against estimation errors. The flexibility comes from the fact that any of the known portfolio optimization methods can be employed once the clusters have been formed.

6.1.7 Transaction Costs

Throughout the discussion on portfolio optimization models, we assumed a convex function representing the total estimated trading costs $\hat{\phi}^{trade}$. There are primarily two types of trading costs: transaction costs and holding costs. For the purpose of this project, we will assume the holding costs $\hat{\phi}^{hold} = 0$, hence $\hat{\phi}^{trade} = \hat{\phi}^{transaction}$.

A common approach to modelling normalized transaction costs, by Boyd *et al.*,

is:

$$\hat{\phi}^{transaction} = a|z| + b\sigma\frac{|z|^{3/2}}{(V/v)^{1/2}} + cz \tag{13}$$

where V is the trading volume, v is the portfolio value, and a, b, c are parameters [26]. Note that this function is convex, adhering to MPO framework requirement.

6.1.8 Metrics & Evaluation

While in-sample analysis can provide useful insights, we essentially want the models to perform well out-of-sample. Hence we will only consider out-of-sample metrics for evaluating our models. Here are the list of metrics to evaluate portfolio optimization models:

- portfolio return R^p: we need to check whether the generated portfolio meets the target return specified by the client
- portfolio volatility σ^p: we prefer a portfolio with lower risk than the one with higher risk; here we measure risk with standard deviation
- Sortino ratio $S = \frac{R^p - T}{\sigma_d}$, where T is the target return and σ_d is the downside risk that the portfolio does not meet the target [31]

While the choice of using portfolio return and volatility as the metrics is obvious - it aligns with our objectives - Sortino ratio requires some justification. Although the objectives state that we should prefer the portfolio with the lowest risk that meets the target, such preference can lead into counterintuitive situations. For example, say the target return is $R = 10\%$ and we have a portfolio of 10% return with 5% risk and another portfolio of 30% return with 10% risk. If our preference is to select the portfolio with minimum risk as long as it meets the target, the former portfolio is chosen. However a reasonable investor would pick the latter since it provides an additional 20% return for only 5% additional risk. Hence we adopt Sortino ratio, which is a ratio between excess return beyond the target and downside risk that the asset is below the target, as our metric.

Downside risk can be formulated as follows: $\sigma_d = \sqrt{\int_{-\infty}^{T} (T - r)^2 f(r) dr}$, where T is the target return, r is the random variable for the distribution of returns, and $f(r)$ is the distribution for the returns. If we assume $f(r) \sim exp(\mathcal{N})$, i.e returns are log-normally distributed, we can numerically integrate it given that

we know the first and second moments of the normal distribution.

6.2 Parameter Estimation

Before portfolio optimization can take place, one must estimate the input parameters - the asset expected returns and covariance matrix - hence the importance of designing an accurate parameter estimation model cannot be emphasized enough. Contrary to the fundamental analysis approach, the quantitative technique utilizes historical data along with economic information to forecast assets' future behaviour.

6.2.1 Design Decisions

Similar to portfolio optimization model selection, there are too many parameter estimation models to investigate within the given time constraint of the project. Thus we will frame our discussion to models we have studied throughout Engineering Science EMSF curriculum and their variants. After thorough evaluation, if none of the models are deemed successful we may expand our scope to consider a wider range of models.

When making design decisions, we should consider the following objective of a parameter estimation model:

- the model can accurately estimate future returns (or other measures i.e. expected returns or risks) in a consistent manner

The following constraints also must be satisfied:

- the model must work for ETFs; and
- the model should not hinder user-friendliness of the design.

The objective and constraints are directly deduced from the project requirement. We tried not to include other objective and constraints beyond the project requirement because we believe the model should be chosen based on their performance and the selection process should not be influenced by external reasons. Although not explicitly stated above, we should also be aware of time constraint (one semester) of the project and feasibility of implementing the model.

After trimming the list of possible candidates using the specified constraints, we will use metrics constructed based on the objectives to evaluate each model.

Details about this is discussed in the subsection *Metrics & Evaluation* under the section *Parameter Estimation*.

6.2.2 Historical Average

The simplest way to estimate mean return and volatility is to take the historical statistics. In other words, we set the mean return and volatility at time $T+1$ and onward equal to the historical mean return and volatility from $t \in [1, T]$ respectively. Formally,

$$\begin{aligned}
\mu_{t,i} &= \left(\prod_{k=1}^{T} (1 + r_{k,i})^{1/T} \right) - 1 \\
\sigma^2_{t,i} &= \frac{1}{T-1} \sum_{k=1}^{T} (r_{k,i} - \overline{r}_i)^2 \\
\sigma_{t,ij} &= \frac{1}{T-1} \sum_{k=1}^{T} (r_{k,i} - \overline{r}_i)(r_{k,j} - \overline{r}_j) \\
t &= T+1, T+2, \ldots
\end{aligned} \tag{14}$$

However this approach has a major flaw - it is only valid if we assume the market is static and the future returns are identically distributed to the past. There are numerous economic research that indicate the financial market is dynamic.

6.2.3 Factor Models

An improvement we can make from the historical average approach is to estimate the asset returns as a linear model, i.e. return can be written as a linear combination of factors. Factor models are based on the following assumptions of an ideal factor environment:

- $cov(f_i, \epsilon_j) = 0$ for all i, j
- $cov(\epsilon_i, \epsilon_j) = 0$ for all $i \neq j$
- $cov(f_i, f_j) = 0$ for all $i \neq j$ (this can be ignored for single-factor models)

Note that the not all factor models respect the ideal environment, as some factors can be correlated. For example, size factor and value factor, both being subsets of the market, are correlated.

Under the general framework of factor model, the return of asset i can be described by the following relationship:

$$r_i - r_f = \alpha_i + \sum_{k=1}^{M} \beta_{ik} f_k + \epsilon_i \tag{15}$$

where r_i is the return of asset i, r_f is the risk free rate, f_k is the the return of factor k, α_i and β_{ik} are the intercept from regression and corresponding factor loading, respectively, M is the number of factors, and ϵ_i is the stochastic error term of the asset. If $M = 1$, it becomes a single-factor model.

We then calibrate the model by choosing a set of optimal regression coefficients that satisfy the minimization problem seen below.

$$\begin{aligned} \min_{\mathbf{B}_i} \quad \|r_i - XB_i\|^2 &= \min_{\mathbf{B}_i} \quad (r_i - XB_i)^\top (r_i - XB_i) \\ &= \min_{\mathbf{B}_i} \quad r_i^\top ri - 2r_i^\top XB_i + B_i^t opXB_i + B_i^\top X^\top XB_i \\ &= \min_{\mathbf{B}_i} \quad f(B_i) \end{aligned} \tag{16}$$

This is a convex quadratic unconstrained minimization problem, thus only the first order necessary conditions must be satisfied to ensure a global minimum. Therefore, the optimal regression coefficients will satisfy the following system of equations.

$$\begin{aligned} \bigtriangledown f(B_i) = -2X^\top r_i + 2X^\top XB_i &= 0 \\ 2X^\top XB_i &= 2X^\top r_i \\ B_i &= (2X^\top X)^{-1} X^\top r_i \end{aligned} \tag{17}$$

This means a closed form solution to the factor model is possible with optimal regression coefficients B^* given by,

$$\therefore B^* = (X^\top X)^{-1} X^\top r \tag{18}$$

Factor model is flexible as the number and types of factors to employ to the model can be arbitrary and choosing the appropriate one is a design problem. While a wide range of factor models have been investigated in the industry and academia, we will discuss three notable kinds - CAPM, Fama-French, and sparse

models.

Capital Asset Pricing Model (CAPM) CAPM is a single-factor model that takes exposure or sensitivity into account. Formally,

$$r_i - r_f = \alpha_i + \beta_{im}(f_m - r_f) + \epsilon_i \tag{19}$$

where $\beta_{im} = \sigma_{im}/\sigma_m^2$ is the exposure of asset i to the market and $(f_m - r_f)$ is the market excess return.

CAPM is one of the simpler factor models yet it has been used extensively in theoretical settings. However it may not be practical for several reasons. Firstly, it assumes that the risk-free rate is constant, which is rarely the case in the market (ex. volatility in short-term government-issued bond yield). Secondly, the assumption that investors can lend and borrow at a risk-free rate is unattainable in reality. Therefore, the minimum required return line might actually be less steep than the model calculates.

Fama-French Factor Models A typical Fama-French model is the Fama-French three-factor model which considers the market, size, and value factors, and the expected return for asset i is shown below:

$$r_i - r_f = \alpha_i + \beta_{im}(f_m - r_f) + \beta_{is} f_s + \beta_{iv} f_v + \epsilon_i \tag{20}$$

where r_i is the return of asset i, r_f is the risk free rate, f_m, f_s and f_v are the market, size and volume factors, respectively, α_i and β_{ik} are intercept from regression and corresponding factor loading respectively, and ϵ_i is the stochastic error term of the asset.

One does not need to stop at 3 factors and extend the model with more factors. For example, Fama-French four-factor model includes momentum as its fourth factor. Despite the fact that Fama-French models have the same issue with the risk-free rate as CAPM, it is more powerful given its additional features.

Sparse Factor Models One problem with including large pool of factors is that we may run into risk of overfitting, thus making the models very sensitive to noises generated from estimating our parameters. Additionally, it reduces the interpretability of the models. One approach to avoid this is by introducing regularization. Here we mention three regularizations:

- Best Subset Selection (BSS): $\|B_i\|_0 = 1_{\alpha_i \neq 0} + \sum_{k=1}^{p} 1_{\beta_{i,k} \neq 0} \leq K$
- LASSO: $\|B_i\|_1 = |\alpha_i| + \sum_{k=1}^{p} |\beta_{i,k}| \leq s$
- Ridge Regression: $\|B_i\|_2^2 = (\alpha_i)^2 + \sum_{k=1}^{p} (\beta_{i,k})^2 \leq s$

The first two, BSS and LASSO, promote sparsity of the model by encouraging the loading coefficients to converge to zeroes. Ridge regression is a technique we can use to prevent overfitting and decrease the complexity of the model. Limitation of the ridge regression is that it does not reduce the number of variables since it never leads to a coefficient been zero rather only minimizes it. Hence, this model is not good for feature reduction. On the other hand, while BSS and LASSO promote sparsity, if there are two or more highly collinear variables then LASSO regression select one of them randomly which is not good for the interpretation of data.

While ridge regression is convex, the original formulation of BSS and LASSO are not. Hence they require modification such that they conform to the standard quadratic programming.

Here, we implemented a penalized form of the LASSO model, seen below.

$$\min_{\mathbf{B}_i} \quad \|\mathbf{r}_i - \mathbf{X}\mathbf{B}_i\|_2^2 + \lambda \|\mathbf{B}_i\|_1$$

where $\mathbf{r}_i$ is the return of asset i, λ is our chosen parameter, and $\mathbf{B}_i$ is our vector of intercept α_i and factor β_{ik} coefficients. Note, λ will be chosen using hyperparameter grid search.

In order to accommodate the absolute value of $\mathbf{B}_i$ so that we may implement standard quadratic programming, $\mathbf{B}_i$ must be substituted as follows:

$$\mathbf{B}_i = \mathbf{B}_i^+ - \mathbf{B}_i^-$$
$$\|\mathbf{B}_i\|_1 = \mathbf{B}_i^+ + \mathbf{B}_i^-$$

The derivation with the above substitution is as follows.

$$
\begin{aligned}
& \|\mathbf{r}_i - \mathbf{X}\mathbf{B}_i\|_2^2 + \lambda\|\mathbf{B}_i\|_1 \\
= \;& (r_i - \mathbf{X}\mathbf{B}_i)^\top (r_i - \mathbf{X}\mathbf{B}_i) + \lambda\|\mathbf{B}_i\|_1 \\
= \;& r_i^\top r_i - 2r_i^\top \mathbf{X}\mathbf{B}_i + \mathbf{B}_i^\top \mathbf{X}^\top \mathbf{X}\mathbf{B}_i + \lambda\|\mathbf{B}_i\| \\
= \;& r_i^\top r_i + \left[\begin{pmatrix} -2r_i^\top \mathbf{X} \\ 2r_i^\top \mathbf{X} \end{pmatrix} + \begin{pmatrix} \lambda \\ \vdots \\ \lambda \end{pmatrix} \right]^\top \begin{bmatrix} \mathbf{B}_i^+ \\ \mathbf{B}_i^- \end{bmatrix} + \begin{bmatrix} \mathbf{B}_i^+ \\ \mathbf{B}_i^- \end{bmatrix}^\top \begin{bmatrix} \mathbf{X}^\top \mathbf{X} & -\mathbf{X}^\top \mathbf{X} \\ -\mathbf{X}^\top \mathbf{X} & \mathbf{X}^\top \mathbf{X} \end{bmatrix} \begin{bmatrix} \mathbf{B}_i^+ \\ \mathbf{B}_i^- \end{bmatrix}
\end{aligned}
$$

We now have that $\mathbf{H} = \mathbf{2} \begin{bmatrix} \mathbf{X}^\top \mathbf{X} & -\mathbf{X}^\top \mathbf{X} \\ -\mathbf{X}^\top \mathbf{X} & \mathbf{X}^\top \mathbf{X} \end{bmatrix}$, $\mathbf{c}^\top = \left[\begin{pmatrix} -2r_i^\top \mathbf{X} \\ 2r_i^\top \mathbf{X} \end{pmatrix} + \begin{pmatrix} \lambda \\ \vdots \\ \lambda \end{pmatrix} \right]^\top$.

Standard form of QuadProg is now:

$$
\begin{aligned}
min \quad & \frac{1}{2} \begin{bmatrix} \mathbf{B}_i^+ \\ \mathbf{B}_i^- \end{bmatrix}^\top \mathbf{H} \begin{bmatrix} \mathbf{B}_i^+ \\ \mathbf{B}_i^- \end{bmatrix} + \mathbf{c}^\top \begin{bmatrix} \mathbf{B}_i^+ \\ \mathbf{B}_i^- \end{bmatrix} \\
s.t. \quad & \mathbf{B}_i^+, \mathbf{B}_i^- \geq \mathbf{0}.
\end{aligned}
$$

Note that $r_i^\top r_i$ is a constant term so it can be omitted in the objective function of the minimization problem. This standard form of QuadProg gives an equivalent solution to the original quadratic programming problem with linear constraints containing absolute values, see *Solving Quadratic Programming Problem with Linear Constraints* – Jan Busa, Department of Mathematics and Theoretical Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Kosice [32].

Finally, a BSS model was implemented. This model is very similar to the LASSO method mentioned above, however we introduce a cardinality constraint instead of penalizing $\mathbf{B}_i$. The model is shown below.

$$
\begin{aligned}
\min_{\mathbf{B}_i} \quad & \|\mathbf{r}_i - \mathbf{X}\mathbf{B}_i\|_2^2 \\
s.t. \quad & \|\mathbf{B}_i\|_0 \leq K
\end{aligned}
$$

where K is our cardinality constraint, given to be 4.

Derivation of the standard QuadProg form is shown below, see *Best subset*

selection via a modern optimization lens – D. Bertsimas [33]:

$$\begin{aligned} & \|\mathbf{r}_i - \mathbf{X}\mathbf{B}_i\|_2^2 \\ = \; & (r_i - \mathbf{X}\mathbf{B}_i)^\top (r_i - \mathbf{X}\mathbf{B}_i) \\ = \; & r_i^\top r_i - 2r_i^\top \mathbf{X}\mathbf{B}_i + \mathbf{B}_i^\top \mathbf{X}^\top \mathbf{X}\mathbf{B}_i \end{aligned}$$

In standard form for quadratic programming:

$$\begin{aligned} & \min_{\mathbf{B}_i} && \|\mathbf{r}_i - \mathbf{X}\mathbf{B}_i\|_2^2 \\ & s.t. && \|\mathbf{B}_i\|_0 \leq K \\ \implies & \min_{\mathbf{B}_i, y_i} && \frac{1}{2}\|r_i - \mathbf{X}\mathbf{B}_i\|_2^2 \\ & s.t. && lb \cdot y_i \leq B_i \leq ub \cdot y_i \\ & && y_i \in \{0,1\}, \\ & && \sum y_i \leq K. \\ \implies & min && \frac{1}{2}\begin{bmatrix}\mathbf{B}_i \\ y_i\end{bmatrix}^\top \mathbf{H}\begin{bmatrix}\mathbf{B}_i \\ y_i\end{bmatrix} + \mathbf{c}^\top \begin{bmatrix}\mathbf{B}_i \\ y_i\end{bmatrix} \\ & s.t. && \mathbf{A}\begin{bmatrix}\mathbf{B}_i \\ y_i\end{bmatrix} \leq \mathbf{b} \\ & && y_i \in \{0,1\}. \end{aligned}$$

where $\frac{1}{2}\mathbf{H} = \begin{bmatrix}\mathbf{X}^\top\mathbf{X} & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix}$, $c = \begin{bmatrix}-2r_i^\top\mathbf{X} \\ \mathbf{0}\end{bmatrix}$, $\mathbf{A} = \begin{bmatrix}-\mathbf{I}_d & lb \cdot \mathbf{I}_d \\ \mathbf{I}_d & -ub \cdot \mathbf{I}_d \\ \mathbf{0}_d & \mathbf{1}_d\end{bmatrix}$, $\mathbf{b} = \begin{bmatrix}0 \\ \vdots \\ 0 \\ K\end{bmatrix}$,

$\mathbf{0}_d = \begin{bmatrix}0 \\ \vdots \\ 0\end{bmatrix}^\top$, $\mathbf{1}_d = \begin{bmatrix}1 \\ \vdots \\ 1\end{bmatrix}^\top$, $lb = -30$, $ub = 30$, and $d = \#parameters + 1$.

An upper bound and lower bound, ub and lb respectively, were chosen to be sufficiently large but do not encapsulate the noise from integer programming in Python.

6.2.4 Principal Component Analysis (PCA)

The goal of PCA is to create a set of factors intrinsic to our specific asset data set and reducing the dimensionality of the unknown feature space. Below is a detailed outline of a possible PCA implementation.

1. Compute the expected return of all assets $\boldsymbol{\alpha} = \mathbb{E}[\boldsymbol{R}]$, where $\boldsymbol{R}$ is the matrix of asset returns.
2. Compute the covariance matrix of the assets $\boldsymbol{\Sigma}$.
3. Eigendecompose the covariance matrix $\boldsymbol{\Sigma} = \boldsymbol{\Gamma}\boldsymbol{\Lambda}\boldsymbol{\Gamma}^\top$, where each column in $\boldsymbol{\Gamma}$ is an eigenvector $\boldsymbol{\gamma}_i$ normalized to 1, and $\boldsymbol{\Lambda}$ is a diagonal matrix where each diagonal element is an eigenvalue $\boldsymbol{\lambda}_i$.
4. Sort the eigenvectors by order of decreasing eigenvalue and collect them to create an ordered matrix $\boldsymbol{\Gamma}$.
5. Take the matrix of asset returns and subtract the corresponding expected return from each entry to obtain $\boldsymbol{R} - \boldsymbol{\alpha}$.
    - This ensures that each column has a mean of zero and our new matrix of asset returns is centered around the origin.
6. Compute the Principal Component matrix $\boldsymbol{p} = \boldsymbol{\Gamma}^\top(\boldsymbol{R} - \boldsymbol{\alpha})$.
    - The intuition behind this is to transform our origin-centered matrix to the eigenbasis. Since eigenvectors are orthogonal to each other, this ensures that the principal components are mutual independent. Hence the model respects the ideal environment for a factor model.
    - We have achieved the goal of identifying the principal components of our asset return matrix. Furthermore, they are sorted in order of decreasing eigenvalue. We will select a subset of the principal components as our corresponding factor loadings.
7. Select the first p principal components from the Principal Component matrix corresponding to those with the highest eigenvalues. The components with the highest corresponding eigenvalues provide more information on the distribution of the data.
8. We can estimate the asset returns from the chosen factors as follows: $\hat{\boldsymbol{r}} = \boldsymbol{\alpha} + \boldsymbol{V}^\top \boldsymbol{f} + \boldsymbol{\epsilon}$, where $\boldsymbol{V}$ is the collection of first p eigenvectors, $\boldsymbol{f}$ is the chosen p factors, and $\boldsymbol{\epsilon}$ is the stochastic error term.

The methodology of how factors are simulated is further illustrated in detail in the section *Monte Carlo Simulations*. Finally, here are the outputs of the PCA model that we obtained with the eigenbasis calculated above:

- Vector of asset expected returns, $\boldsymbol{\mu} \in \mathbb{R}^n$.
  - Note that $\boldsymbol{\mu} = \boldsymbol{\alpha} = \mathbb{E}[\boldsymbol{R}]$.
- Diagonal matrix of asset idiosyncratic variances, $\boldsymbol{D} \in \mathbb{R}^{n \times n}$.
  - For the scope of this project, we will assume that the asset residuals $\boldsymbol{\epsilon} \sim \boldsymbol{\mathcal{N}}(\boldsymbol{0}, \boldsymbol{D})$ is normally distributed and uncorrelated. From $\boldsymbol{\epsilon}_s = \boldsymbol{R}_s - (\boldsymbol{\alpha} + \boldsymbol{V}^\top \boldsymbol{f}_s)$, we can calculate the unbiased estimate of the residual variance $\boldsymbol{\sigma}^2_{\boldsymbol{\epsilon}_s} = \frac{1}{T-p-1}\|\boldsymbol{\epsilon}_s\|_2^2$, where T is the number of observations, p is the number of factors, and $T - p - 1$ is the degree of freedom. Then the diagonal matrix of idiosyncratic variances can be constructed as follows:

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{\sigma}^2_{\boldsymbol{\epsilon}_1} & 0 & \cdots & 0 \\ 0 & \boldsymbol{\sigma}^2_{\boldsymbol{\epsilon}_2} & \cdots & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \cdots & \boldsymbol{\sigma}^2_{\boldsymbol{\epsilon}_n} \end{bmatrix}$$

- Matrix of factor loadings, $\boldsymbol{V} \in \mathbb{R}^{p \times n}$ for the first p principal components.
  - This is the first p components in $\boldsymbol{p}$.
- Factor covariance matrix, $\boldsymbol{F} \in \mathbb{R}^{p \times p}$.
  - By design, this matrix is diagonal because PCA factors are uncorrelated, thus $\boldsymbol{\sigma}_{i,j} = 0$ for $i \neq j$. Since the variance $\boldsymbol{\sigma^2}_i = \boldsymbol{\lambda}_i$, we can construct the factor covariance matrix as follows:

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{\lambda}_1 & 0 & \cdots & 0 \\ 0 & \boldsymbol{\lambda}_2 & \cdots & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \cdots & \boldsymbol{\lambda}_p \end{bmatrix}$$

- Vector of factor skewness, $\boldsymbol{S} \in \mathbb{R}^p$.
  - Skewness is the 3^{rd} order standardized moment of a probability distribution. It may be thought of as how symmetrical a distribution is about its mean, where negative skewness represents a mean below

the mode, and vice-versa for positive skewness. When skewness is equal to zero, the mean, median and mode are all the same.

  - The adjusted Fisher-Pearson coefficient of skewness (used by MatLab) is calculated as $S = \frac{\sqrt{N(N-1)}}{N-2} \cdot \frac{\sum_{i=1}^{n}(Y_i - \overline{Y})^3}{N\sigma^3}$.

- Vector of factor kurtosis, $\boldsymbol{K} \in \mathbb{R}^p$.
  - Kurtosis is the 4^{th} order standardized moment of a probability distribution. Kurtosis may be thought of as how fat the tails of a distribution are, where large values of kurtosis indicate a greater number of outliers than small values.
  - Kurtosis is defined as $K = \frac{\sum_{i=1}^{n}(Y_i - \overline{Y})^4}{N\sigma^4}$.
- Asset covariance matrix, $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$
  - The asset covariance matrix can be calculated as follows: $\boldsymbol{Q} = \boldsymbol{V}^\top \boldsymbol{F} \boldsymbol{V} + \boldsymbol{D}$

As mentioned above, PCA guarantees mutual independence between the factors, which is necessary for an ideal factor model. Moreover it reduces overfitting of the data by reducing the number of factors in the model. However these advantages entail a major drawback that PCA lacks interpretability since principal components lose the intuitive meaning of the original features when they are transformed.

Monte Carlo Simulations To obtain the factors, two Monte Carlo simulation methods are considered: Gaussian process and stochastic process with higher moments.

The scenarios for the Gaussian process, $\boldsymbol{f}_s$, can be drawn from the multivariate normal distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{F})$, where $\boldsymbol{F}$ is the covariance matrix of the PCA factors. The Gaussian process is centered at $\mathbf{0}$ as the PCA factors have the mean $\boldsymbol{\mu} = 0$. The second random sampling comes from asset idiosyncratic noise, $\epsilon_s \sim \mathcal{N}(\mathbf{0}, \boldsymbol{D})$, where $\boldsymbol{D}$ is the $n \times n$ diagonal matrix of asset idiosyncratic variances, again from PCA. For the stochastic process with higher moments, skewness (3^{rd}-order moment) and kurtosis (4^{th}-order moment) will be incorporated into the generation of the factor scenarios, $\boldsymbol{f}_s$, and ϵ_s generated as before.

Both methods for Monte Carlo simulations are then used to generate the asset

returns scenarios, $\hat{\boldsymbol{r}}_s$, for $s = 1, .., S$ using the following equation:

$$\hat{\boldsymbol{r}}_s = \boldsymbol{\mu} + \boldsymbol{V}^T \boldsymbol{f}_s + \boldsymbol{\epsilon}_s$$

6.2.5 Autoregressive Models

Two major concerns with ordinary linear model, like factor models or PCA, are that they often fail to capture more complex information, such as mean-reversion or surprise earning calls, and that they assume homoskedasticity, i.e. variation of error terms in the statistical model is constant. Autoregressive models can be alternatives since they have the ability to consider their own past behaviour as inputs for the model. Namely, we will examine ARMA-GARCH model for its heteroskedastic characteristic.

ARMA-GARCH Autoregressive moving average model (ARMA), evident from its name, is a ensemble of autoregressive model, which takes past into account as a linear combination of previous terms, and moving average model, which uses linear combination of previous error terms. ARMA is a model for the realizations of a stochastic process imposing a specific structure of the conditional mean of the process. Generalized autoregressive conditional heteroskedasticity model (GARCH), very similar to ARMA, is a model for the realizations of a stochastic process imposing a specific structure of the conditional variance of the process [34]. Consider y_t that has unconditional mean zero and follows an $ARMA(p,q) - GARCH(s,r)$ process. Then:

$$\begin{aligned} & y_t \sim D(\mu_t, \sigma_t^2) \\ & \mu_t = \varphi_1 \mu_{t-1} + \ldots + \varphi_p \mu_{t-p} + (\varphi_1 + \theta_1) u_{t-1} + \ldots + (\varphi_q + \theta_q) u_{t-q} \\ & \sigma_t^2 = \omega + \alpha_1 u_{t-1}^2 + \ldots + \alpha_s u_{t-s}^2 + \beta_1 \sigma_{t-1}^2 + \ldots + \beta_r \sigma_{t-r}^2 \\ & \frac{u_t}{\sigma_t} \sim i.i.D(0,1) \end{aligned} \tag{21}$$

where $u_t = y_t - \mu_t$, $\sim D$ is some density distribution, $\varphi_i = 0$ for $i > p$, and $\theta_j = 0$ for $j > q$.

Especially, GARCH model is widely used in finance due to their effectiveness in modeling asset volatility [35]. GARCH aims to minimize errors in forecasting by accounting for errors in prior forecasting and enhancing the accuracy of ongoing predictions. GARCH models describe financial markets in which volatility can

change, becoming more volatile during periods of financial crises or world events and less volatile during periods of relative calm and steady economic growth. A simple regression model does not account for this variation in volatility exhibited in financial markets. It is not representative of the extreme events that occur more often than predicted.

6.2.6 Supervised Machine Learning

Supervised machine learning has been extensively used for predictive purposes in the field of finance. In fact, all aforementioned parameter estimation models, including factor models and autoregressive models, can be categorized as supervised learning since they are trained on labelled dataset in attempts to forecast future behaviour. In this section we will discuss more complex, non-linear machine learning techniques, contrary to the previous approaches which are linear. Two notable models worth mentioning are transformer and XGBoost.

Transformer Transformer uses self-attention mechanism to detect patterns from sequential data and is employed in many state-of-the-art models like *BERT* and *GPT-3* [36]. It has demonstrated significant improvement from traditional sequential ML models in natural language processing (NLP) and image recognition, such as RNN (recurrent neural network) and LSTM (long short-term memory network), and we hope to see similar enhancement in financial forecasting.

XGBoost XGBoost, which stands for extreme gradient boosting, is a regularized implementation of gradient boosting algorithm which is known for its performance and speed. Here are some studies where XGBoost has demonstrated promising results in financial forecasting:

- One research found a 5% increase in returns for Risk Parity models, and lowered estimation errors for 9 out of 10 indices using XGBoost to estimate asset volatilities [37].
- XGBoost has shown promise in predicting SP 500 returns. This is especially appealing given that the project involves modelling ETF returns [38].

6.2.7 Metrics & Evaluation

While in-sample analysis can provide useful insights, we essentially want the models to perform well out-of-sample. Hence we will only consider out-of-sample metrics for evaluating our models. Here are the list of metrics to evaluate parameter estimation models:

- mean-squared error $MSE = \frac{1}{n}\sum_{t=1}^{H}(r^p - \hat{r}^p)^2$
- standard deviation across test sets σ_{test}

We want to minimize the MSE to attain high accuracy of the estimates. The more inaccurate the estimate is, the greater it penalizes for its deviation from the true value. At the same time we prefer a model with low standard deviation across test sets since the parameters need to be estimated in a consistent manner.

6.3 Backtesting & Hyperparameter Tuning

Financial models are tested using historical datasets to estimate their performance in hopes that past performance implies a similar level of performance in the future. Moreover historical datasets are useful when tuning hyperparameters for the models. To do so, we will employ k-fold cross validation approach, where $k = 5$ is chosen for this project. However the choice of the number of partition k is subjective and can be changed during the course of the project. The provided financial time series dataset is split into k partitions. Given the nature of sequential data, we eliminate or limit data points from appearing in both the training and testing sets. Furthermore the training will occur in a sliding-window fashion to preserve serially correlated information.

Backtesting and hyperparameter tuning will use grid search method, a technique that defines a search space as a grid of hyperparameter values and evaluate every position in the grid using the specified metrics. In other words, the models will be evaluated at points in the search space $R \times T \times HP$ where R, T, and HP are defined as follows:

- R: the set of target returns
- T: the set of time horizons
- HP: the set of hyperparameters, will contain different parameters depending on the model. For example, some typical ones to include are

the risk-aversion level, the trading costs coefficient, and the regularization factor.

A simple stress test will also be conducted to measure how well each model performs during an economic crisis. We will select $2008 \sim 2009$, the sub-prime mortgage crisis, and $2020 \sim 2021$, the COVID-19 pandemic, as the stress testing periods.

6.4 Implementation

6.4.1 Clustering

Our current implementation relies on 500 company stocks in S&P500 and over 1500 ETFs listed on NYSE and NASDAQ stock exchanges. Performing returns forecasts or portfolio optimizations on the entirety of these assets would be computationally very heavy, hence hindering user experience with longer wait time between portfolio generation processes. Consequently we need a way to narrow down our list of ETFs before conducting any analyses.

One approach is to cluster the ETFs based on their historical returns. We implemented k-medoids clustering method based on Euclidean distance between monthly returns of ETFs over past 5 years. k-medoids clustering is similar to k-means clusteirng in a sense that they are both clustering algorithms which attempts to minimize sum of intra-cluster metrics.

k-means clustering is an unsupervised machine learning framework that given some initialization of centroids, it iterates the following two steps:

- assignment step : assign each observation to the cluster with the closest mean;
- update step : calculate the new means to be the centroid of the observations in the cluster.

The algorithm is deemed to have converged when the assignments no longer change.

Both the k-means and k-medoids algorithms are partitional (breaking the dataset up into groups). k-means attempts to minimize the total squared error, while k-medoids minimizes the sum of dissimilarities between points labeled to be in a cluster and a point designated as the center of that cluster. In contrast to the k-means algorithm, k-medoids chooses data points as centers.

The most common realisation of k-medoids clustering is the Partitioning Around Medoids (PAM) algorithm and is as follows:

- initialization: randomly select k of the n data points as the medoids;
- assignment step: associate each data point to the closest medoid;
- update step: for each medoid m and each data point o associated to m swap m and o and compute the total cost of the configuration (that is, the average dissimilarity of o to all the data points associated to m); select the medoid o with the lowest cost of the configuration.

Repeat alternating steps 2 and 3 until there is no change in the assignments.

k-medoids brings two major benefits that k-means does not. It could be more robust to noise and outliers as compared to k-means because it minimizes a sum of general pairwise dissimilarities instead of a sum of squared Euclidean distances. Furthermore, the centroids we obtain after running the algorithm are existing data points from which the framework is trained. This has an intuitive appeal since for our purpose the existing data points would be a set of actual ETFs that we can select.

To enhance the clustering, we employed PCA to reduce dimensionality of the historical returns. Since we are considering 5-year worth of monthly returns, that leads to 60 dimensions for k-medoids algorithm to train on. High dimensions can lead to overfitting, resulting in poor out-of-sample performance. We select the optimal number of principal components using maximum likelihood estimation approach (MLE). Thus the optimal principal components would be different for every training period.

Both PCA and k-medoids algorithms rely on built-in functions from *scikit-learn*.

6.4.2 Forecasting

Parameter estimation models we are considering are not capable of predicting future returns; they simply illustrate the relationships between historical returns and factors. Hence we need a forecasting technique to predict future factors which can then be applied to the factor models.

The approach we implemented is ARIMA-GARCH from the section *Parameter Estimation*. We used *pmdarima* and *arch* Python libraries to implement the model. For every training period, past 5-year monthly returns are provided as

inputs, and the ARIMA model searches for the optimal combination of parameters that best fits the input data that minimizes Akaike information criterion (AIC), an estimator of prediction error and thereby relative quality of statistical models for a given set of data. The list of the ARIMA model parameters is as follows:

- p: order of autoregressive (AR) terms, i.e. number of lags to include in the model;
- q: order of moving average (MA) terms, i.e. number of lagged forecast errors to include in the model; and
- d: the minimum number of differencing needed to make the series stationary.

The optimal value for the parameter d is determined using Augmented Dickey–Fuller (ADF) test, which tests the null hypothesis that a unit root is present in a time series sample. The list of the GARCH model parameters is as follows:

- r: order of variance terms, i.e. number of variance lags to include in the model; and
- s: order of residual variance terms, i.e. number of lagged residual variances to include in the model.

Similarly, the optimal values are determined by AIC criterion. The package *pmdarima* comes in handy when training autoregressive time-series models and searching optimal hyperparameter, namely with its *auto.arima* functionality.

Finally after training both ARIMA and GARCH, the models can be now used to generate predictions of expected returns and estimated volatilities for a given forecasting horizon.

6.4.3 Parameter Estimation Models

Our parameter estimation models rely on *Pandas* and *NumPy* for managing data. They take stock/ETF prices, returns, and economic or factor datasets if needed as inputs in *Pandas* DataFrame, and output forecasted returns or metrics (i.e. expected returns vector or covariance matrix) in *Pandas* DataFrame as well.

For each factor model, it is trained over 5-year historical monthly factors as the independent variables and 5-year historical monthly returns as the dependent variables. The appropriate factor loadings are obtained using *scikit-learn* which

can train multiple linear regression models and is capable of generating useful test statistics for evaluation.

Once the regression is complete, the aforementioned forecasted factors can be fed into the model to yield returns predictions for a given horizon.

6.4.4 Portfolio Optimization Models

We have utilized an open-source Python package *CVXPY* to structure our portfolio optimization models. The package is capable of solving convex problems but may lead to issues for non-convex ones. Thus we employ *Gurobi* Solver to handle any non-convex optimization problems, such as integer programming, under *University of Toronto* academic license.

The package relies on *Pandas* and *NumPy* for managing data. *Pandas* implements structured data types as in-memory databases and provides a rich API for accessing and manipulating them. Through Pandas, it is easy to couple our package with database backends. We also rely on *SciPy* for complex mathematical calculations, such as clustering algorithms for hierarchical cluster asset allocation models.

Our business logic implementation provides an object-oriented framework with classes and functions representing objectives, expected returns, risk measures, transaction costs, holding constraints, trading constraints, etc. Multi-period optimization models are constructed from instances of these classes. Single-period optimization models can also be done by setting trade horizon to 1. Each instance generates *CVXPY* expressions and constraints for any given period, making it easy to combine the instances into a single convex model.

Our implementation in Python is inspired by the works of Boyd *et al.* and Martin *et al.* For more details and code of the implementation, please follow this link: https://github.com/joshuakim314/bridge-robo-advisor.

The major breakdown of the portfolio optimization model implementation is as follows:

- BaseOptimizer class: a generic, abstract class that structures portfolio optimization models;
- BaseConvexOptimizer class inheriting BaseOptimizer: a class that integrates CVXPY and Gurobi for portfolio optimization purpose; objectives,

constraints, and portfolio weights are initialized in this class;

- BaseMPO class inheriting BaseConvexOptimizer: a class that extends BaseConvexOptimizer such that it allows multi-period optimization;
- EfficientFrontier class inheriting BaseMPO: a class that constructs models using efficient frontier framework or its variants - this includes MVO, robust MVO, Sharpe ratio optimization, risk parity optimization, and CVaR optimization;
- HCAA class inheriting BaseMPO: a class that constructs hierarchical clustering framework; and
- objective class: a class with objective functions which are imported by EfficientFrontier and clustering classes; some notable ones are portfolio variance, portfolio returns, quadratic utility, and helper objective functions such as regularization terms and transaction costs.

6.5 Empirical Results & Discussion

For backtesting, a total of 15 investment periods are observed for each model, each one year long. The investment horizon begins in January 2001 and ends in December 2020, with each model having the chance to reinvest at the end of each year. To train factor models and forecasting frameworks, a trailing 5-year monthly returns look-back window was used. A linear transaction cost model, i.e. 10 basis points (bps) of the weights difference between each trading period, is employed. To mimic a typical novice investor's user preferences, the following have been imposed throughout the backtesting:

- stock selections: DIS, IBM, JPM, KO, WMT
- weight constraints: for each stock, it contributes a minimum of 5% of the portfolio and a maximum of 100% of the portfolio
- control: the total weights of user's selected stocks should be at least 50% of the portfolio at any given time
- ETF cardinality constraint: the total number of ETFs in the portfolio must not be greater than 10
- target return: it matches the average market (SPY) returns of the past 5 years from the trading time

- risk aversion parameter: this is fixed at $\gamma = 1.0$
- look-ahead horizon: this is fixed at 12 months

The first 10 periods, where the first period is from January 2001 $\sim$ December 2005 for training and from January 2006 $\sim$ December 2006 for testing and the last period is from January 2010 $\sim$ December 2015 for training and from January 2016 $\sim$ December 2016 for testing, are used for validation and hyperparameter tuning, and the last 5 periods, where the first period is from January 2011 $\sim$ December 2015 for training and from January 2016 $\sim$ December 2016 for testing and the last period is from January 2015 $\sim$ December 2019 for training and from January 2020 $\sim$ December 2020 for testing, are used for testing and final model selection.

6.5.1 Validation Results

To start off, we trained the ARIMA-GARCH model for each investment period using past 5-year monthly returns data as input to predict the next 1-year monthly expected returns and volatilities. *Table 9* shows the average out-of-sample coefficients of determination r^2 of each factor over the entire validation periods.

Factor	Excess	SMB	HML	RMW	CMA
average r^2	-0.07	0.02	0.05	0.01	-0.08
r^2 under stress case	-7.46	-5.10	-7.51	-3.94	-4.67
r^2 for first month	0.02	0.12	0.14	0.06	-0.01
r^2 for last month	-0.27	-0.15	-0.22	-0.13	-0.20

Table 9: *various average r^2 values over the validation periods*

Note that the stress case refers to 2008 subprime mortgage crisis when the overall market was extremely bear.

Table 9 illustrates that HML and SMB factors are the most predictable out of the 5 factors. At the same time we observe that excess return is very difficult to forecast given that the average r^2 is beyond the range between 0 and 1. This is expected since returns often behave stochastic under a log-normal distribution. It is important to recognize that the ARIMA-GARCH model performs very poorly under stress cases. One reason why this is the case is because the ARIMA-GARCH model learned a seasonal autoregressive pattern. Since the stock market behaved in a significantly differently manner between 2008 and

the previous 5 years, the model yielded completely-off predictions. Furthermore we see that the model estimates the first month much better than the last month since all but one r^2 values for the first month are between 0 and 1 whereas the values are all outside the bound for the last month. This suggests that the model is able to predict returns for a shorter forecast horizon than a longer one. Nevertheless, the model demonstrates weak autocorrelation as most of them have negative coefficient of determination or small positive values.

For parameter estimation models, we considered four types of Fama-French models: 3-factor model, 5-factor model, 3-factor model with Elastic Net, and 5-factor model with Elastic Net. Elastic Net is a regularization technique that leverages both LASSO and ridge regressions. The regularization term can be written as follows:

$$EN(\alpha, l_1) = a \cdot \|w\|_1 + \frac{b}{2} \cdot \|w\|_2^2 \tag{22}$$

where $\alpha = a + b$ and $l_1 = \frac{a}{a+b}$. The sets of hyperparameters to run grid search are:

- $\alpha = 0.01, 0.1, 0$
- $l_1 = 0, 0.05, 0.1$

Similar to the ARIMA-GARCH model, each factor model is trained on past 5-year monthly returns data to obtain the factor loadings and the intercept. Given that the number of all possible combinations of factor models and hyperparameters is 14 and each ticker would have its own factor loadings, it is infeasible to display all the results here in the report. However here are several in-sample and out-of-sample statistics worth noting.

	Stocks	ETFs
average r^2	0.42	0.75
r^2 under stress case	0.25	0.47

Table 10: *average in-sample r^2 for stocks and ETFs over the validation periods*

Again, note that the stress case refers to 2008 subprime mortgage crisis when the overall market was extremely bear.

Key observations we can make from *Table 11* are:

- Fama-French factor models show somewhat strong in-sample correlation between returns and factors.
- ETFs are generally much easier to predict returns than individual stocks using factor models. This is expected since ETFs are less volatile due to their diversified nature. Moreover, many ETFs in our asset universe track the stock market, e.g. SPY (S&P 500) or RUI (Russell 1000), which has high correlation with excess return factor.
- Similar to ARIMA-GARCH model, factor models perform worse under stress cases. This is likely because the stock market in 2008 behaved in a drastically different manner, deviating away from trained historical data.

Model	Average r^2	Model	Average r^2
ff5-en-0.01-0.05	-1.148	ff3-en-0.01-0.05	-0.035
ff5-en-0.0-0.0	-1.234	ff3-en-0.0-0.0	-0.057
ff5-en-0.01-0.0	-1.219	ff3-en-0.01-0.0	-0.053
ff5-en-0.0-0.05	-1.234	ff3-en-0.0-0.05	-0.057
ff5-en-0.1-0.05	-0.544	**ff3-en-0.1-0.05**	0.128
ff5-en-0.1-0.0	-1.091	ff3-en-0.1-0.0	-0.024
ff5-mlr	-1.234	ff3-mlr	-0.057
historical	-0.008		

Table 11: *average out-of-sample r^2 for each factor model over the validation periods*

The model name is written in the following convention: $\{$*model type*$\}$-$\{\alpha\}$-$\{l_1\}$. The historical model serves as a benchmark and r^2 is calculated by finding the coefficient of determination between 5-year monthly historical average returns and 1-year future monthly returns.

Here we clearly see that **ff3-en-0.1-0.05** performs the best out-of-sample out of all 3-factor models considered and **ff5-en-0.1-0.05** performs the best out-of-sample out of all 5-factor models considered. Hence the optimal hyperparameters for the factor model is $\alpha = 0.1$ and $l_1 = 0.05$. However we should interpret this with caution since while the 3-factor model with Elastic net along with the optimal hyperparameters performs the best out of all models considered, the 5-factor model with Elastic net along with the optimal hyperparameters underperforms relative to the historical benchmark. Therefore we only accept **ff3-en-0.1-0.05** as the valid parameter estimation model and carry on with our analyses.

To make additional comments on the number of factors, although it is well-established that 5-factor models estimate expected returns significantly better than CAPM or 3-factor models, these out-of-sample metrics indicate otherwise. We hypothesize this is the case since we are feeding the factor models with somewhat inaccurate, weakly-correlated factor forecasts from ARIMA-GARCH. As the 5-factor models have more degrees of freedom for them to overfit to training data points, they may not be robust to forecast errors. Furthermore we observe huge discrepancies between in-sample and out-of-sample r^2 values, most likely due to the estimation errors from our forecast technique.

Once the parameter estimation is complete, we move on to portfolio optimization models. Here we have implemented robust MVO with ellipsoidal uncertainty sets (robust MVO), CVaR optimization, leveraged Sharpe ratio optimization (SR), and leveraged risk parity optimization (RP). During the validation stage, hyperparameters will be tuned from the following sets:

- robust MVO: confidence interval can be 90%, 95%, and 99%
- CVaR optimization: α can be 0.90, 0.95, and 0.99
- both leveraged SR and RP optimization do not need to tune hyperparameters

Note that we fixed the number of scenarios for CVaR optimization to 100 due to computational constraint.

Using the forecasted expected returns and covariance matrices from trained ARIMA-GARCH and the optimized factor model, portfolio optimization is now performed to find the optimal hyperparameters. *Table 12* summarizes the results. To maximize the Sharpe ratio, the optimal hyperparameters are $conf = 95\%$ and $\alpha = 0.99$. It is interesting to note that the more risk-averse you are with hyperparameters, the out-of-sample Sharpe ratio increases but so does portfolio variance.

Model	R^p	σ^p	Sharpe ratio
Robust MVO, $conf = 90\%$	0.0040	0.0364	0.109
Robust MVO, $conf = 95\%$	0.0042	0.0378	0.112
Robust MVO, $conf = 99\%$	0.0044	0.0398	0.111
CVaR, $\alpha = 0.90$	0.0050	0.0424	0.118
CVaR, $\alpha = 0.95$	0.0052	0.0437	0.119
CVaR, $\alpha = 0.99$	0.0054	0.0443	0.122
Leveraged Risk Parity	0.0046	0.0338	0.135
Leveraged Sharpe Ratio	0.0043	0.0395	0.109

Table 12: *average portfolio returns, volatility, and Sharpe ratio over the validation periods*

6.5.2 Test Results

Same procedure is done during testing periods - clustering, forecasting, parameter estimation, then finally portfolio optimization. Since we have already tuned the hyperparameters from the validation set, portfolio optimization can be performed right away. *Table 14* and *Figure 37* summarizes the results. One can see that CVaR optimization model (with $\alpha = 0.99$) surpasses other models by a large margin of portfolio Sharpe ratio and portfolio returns. One drawback to point out is that it also has the highest volatility out of the four.

Model	R^p	σ^p	Sharpe ratio
Robust MVO	0.0050	0.0364	0.137
CVaR	0.0081	0.0407	0.200
Leveraged Risk Parity	0.0012	0.0351	0.035
Leveraged Sharpe Ratio	-0.0021	0.0303	-0.071

Table 13: *average portfolio returns, volatility, and Sharpe ratio over the validation periods*

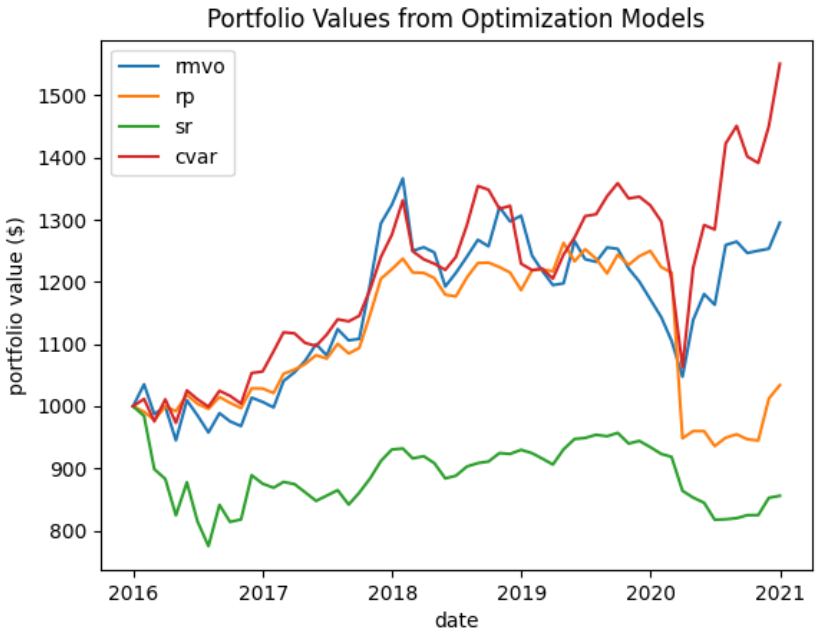


Figure 37: *portfolio values of 4 different optimization models*

6.5.3 Further Analyses

Benchmarks For comparison, we added two benchmarks: SPY, a passive index fund that tracks the U.S. market, and an equally-weighted portfolio, which is one of the simplest active fund one can construct. Augmenting two benchmarks to the table and graph yields the following:

Model	R^p	σ^p	Sharpe ratio
Robust MVO	0.0050	0.0364	0.137
CVaR	0.0081	0.0407	0.200
Leveraged Risk Parity	0.0012	0.0351	0.035
Leveraged Sharpe Ratio	-0.0021	0.0303	-0.071
Equally Weighted	0.012	0.0462	0.255
SPY	0.013	0.0437	0.291

Table 14: *average portfolio returns, volatility, and Sharpe ratio over the validation periods*

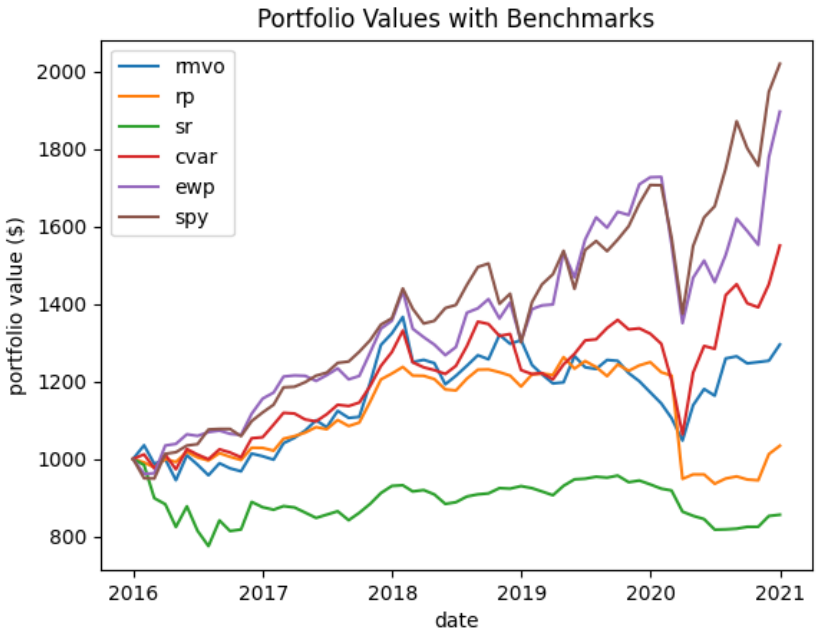


Figure 38: *portfolio values of the models and two benchmarks*

First of all, we observe that none of the optimization model-generated portfolios surpass the benchmarks. While it is widely known that beating the market is extremely difficult in long-run, the fact that they cannot equally-weighted benchmark does not necessarily indicate our models have failed. Since we wanted a set of stock tickers that have been listed on the exchange for the entire duration of validation and testing periods, we introduced "survivorship bias". Due to the nature of how the stock picks were selected, it is likely that highly-performing stocks would survive through the entire validation and testing duration (20 years). In other words, most of the poorly-performing stocks would have died out within the span of 20 years, hence did not even receive a chance to be selected. A more detailed analysis is conducted on the following sections.

Stress Tests Qualitatively speaking from *Figure 38*, CVaR portfolio mimics the movement of the market the best when it is bullish and drops the least when the market is bearish. This can be a good sign that CVaR model diversified the risk enough so that it prevents a huge crash. After all, CVaR optimization model's objective function seeks to minimize the "worst-case" scenarios, i.e. the

greatest loss. Now let's take a look at the statistics of these models in 2020 when COVID-19 pandemic led to a bear market followed by a quick recovery.

Table 15 summarizes how each portfolio behaved during the pandemic. As we can see, besides the market index, CVaR and robust MVO recovered the best with the largest Sharpe ratio of 0.212 where CVaR optimization is slightly more attractive for its higher portfolio returns. Particularly, CVaR portfolio actually beats the equally-weighted benchmark this time by having a higher returns and Sharpe ratio.

Model	R^p	σ^p	Sharpe ratio
Robust MVO	0.0092	0.0434	0.212
CVaR	0.016	0.0749	0.212
Leveraged Risk Parity	-0.013	0.0695	-0.188
Leveraged Sharpe Ratio	-0.0070	0.0225	-0.311
Equally Weighted	0.011	0.0787	0.135
SPY	0.016	0.0743	0.224

Table 15: *average portfolio returns, volatility, and Sharpe ratio during COVID-19 pandemic stress case*

6.6 Final Model Selection

Deducing from the reasons discussed in earlier sections, our final model selection is:

- parameter estimation: Fama-French 5-factor model with $EN(\alpha = 0.1, l_1 = 0.05)$
- portfolio optimization: CVaR optimization model with $\alpha = 0.99$

6.7 Modifications from Previous Deliverables

6.7.1 Transaction Costs

The final implementation of the transaction costs deviated away from the original proposal which was a polynomial equation with respect to traded volumes and weights, as stated in the section *Transaction Costs*. The alternative that is used is a linear model that takes 10 basis points (bps) of the weights difference between each trading period. The rationales behind this decision is as follows:

- we currently do not have a reliable data source to fetch realized transaction costs to train the polynomial model;
- the linear model is computationally advantageous compared to the polynomial model since it does not require any training; and
- given that ETFs generally have higher liquidity, transaction costs are lower; hence a complex model is not as needed as it is for individual stocks (Boyd *et al.* employs the polynomial model to represent stocks' transaction costs).

6.7.2 Forecasting Techniques

Although the preliminary proposal suggests that supervised machine learning techniques were to be implemented to forecast asset returns and factors, we decided to avoid it for the following reasons:

- we have access to limited data, mostly daily prices and factors, and neural networks require considerable amount of training data to expect adequate performance;
- training neural networks is computationally heavy compared to autoregressive models; and
- it is hard to expect a significant improvement from ARIMA-GARCH unless provided with external data source (e.g. macroeconomic data, news headlines, etc.). To exemplify this, training a RNN solely on asset returns would simply generate a lagged time-series, which is something ARIMA is capable of.

6.7.3 Evaluation Metrics

One of the evaluation metrics discussed for portfolio optimization models is Sortino ratio. Instead we had chosen Sharpe ratio to measure the model performance. Firstly, our current formulation requires numerically integrating an log-normal distribution function. This will add computational burden to the product, making our service for clients slower.

6.8 Future Considerations

6.8.1 Black-Litterman Model

Black-Litterman model, created by Fischer Black and Robert Litterman, is a portfolio construction method that overcomes the MVO's issue with highly-concentrated portfolios, input sensitivity, and estimation error maximization [39]. BL tackles this problem by a Bayesian approach to combine investors' subjective views on expected returns of one or more assets with the market equilibrium distribution. The key advantage of BL is that it can handle any number of subjective views of the investor, along with their corresponding confidence levels [40]. By introducing a priori to the framework, the model generates a more intuitive and sensible portfolio weight vector. At the same time, the disadvantage of BL is that assembling accurate, coherent subjective views can be sometimes complicated and difficult.

Although we initially considered BL model for its capability to allow clients to induce their subjective views on the market which can lead to better user experience, we decided to leave it out for now for two reasons. Firstly, it violates the MPO constraint; secondly, the model can introduce unnecessary confusion for novice investors. For example as noted in [41], there have been many misuse cases of BL where a priori and an equilibrium model conflict with one another.

There has been a recent breakthrough in research by Kwon *et al.* that proposes a novel method to implement BL model under a MPO framework [42]. Moreover, BL can still be useful if we restrict the usage to only experienced investors. Hence time permitting, we may consider BL model as one of our possible candidates.

6.8.2 Holding Costs

Throughout the report, we assumed the holding costs $\hat{\phi}^{hold} = 0$ for simplicity. However if time permitting, we may consider developing a more complex model for the holding costs, especially if we want to aggressively use short-selling.

6.8.3 Other Factor Model Regularizations

If time permitting, we may consider other regularization terms for the factor models such as elastic net, least angle regression, and partial least squares method.

7 Deployment

Multiple attempts at deploying the web-application to AWS, Google App Engine and Heroku rendered little success. This is due to the complexity of the file layout needed to get the multi-page Dash application functioning (online tutorials simply don't cover these processes beyond single Python file deployment). With more time we believe it would be possible to deploy the app and fully intend to continue exploring and pursuing the previously mentioned options. We apologize for this and in order to run the Dash application locally (the only available method for interfacing with the web-application) one must clone the Github repository, install all dependencies using *requirements.txt* and run *Main.py/Index.py*. We hope the complexity of the final web-application can be taken into consideration when evaluating this section.

8 Project Management

8.1 Roles

The major leads for all roles are shown below in (*Table 16*). Each individual was responsible for building out their respective section pre-integration as well as providing other leads with needed placeholders. During the pre-integration phase, all group members will take on a minor role in the other sections and are available for help if the major lead needs it. During integration the team will work together to debug, and when finalizing the project roles will be much less fixed.

Role	Major Lead	Minor Leads
Front-End	Aidan	James + Josh
Back-End	James	Aidan + Josh
Buisness Logic	Josh	Aidan + James

Table 16: Project Roles

8.2 Pre-Integration

All leads worked on their respective sections until the need for integration arose. The next step was the back-end lead providing both the front-end and business logic leads with API calls so they could read and write from the databases. This was accomplished with relative ease due to both the front-end and business logic being written entirely in Python, and the APIs meant integration was just simple PostrgreSQL queries implemented with PsycoPG2.

8.3 Finalization of Integration

After both the back-end was connected to the front-end and business logic, all that remained for integration was for the front-end to connect to the business logic. This only occurred when the stock picks and other portfolio parameters were passed from the preferences page to the portfolio construction. Some difficulty was experienced here as file-pathing was not straight forward, and the final decision was to refractor the business logic files to share the same directory as the Dash application.

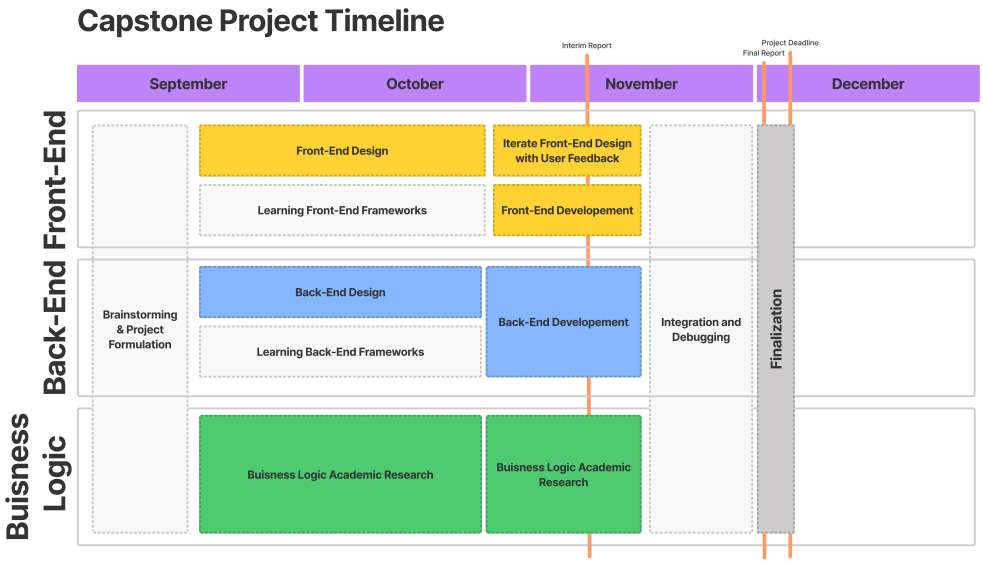


Figure 39: Gantt Chart

8.4 Summary

A Gantt chart is shown below fairly accurately reflects the actual project timeline (*Figure 39*).

9 Alternative Designs

In this section we will document all the alternative designs considered in the process of completing the Bridge Robo-Advisor application. First, we will examine the the components specific to the individual subsystems and then we will examine other considerations from the scope of the integrated application.

9.1 Front End

Without a doubt a better method for implementing the front-end is React. A number of complex workarounds were required to enable all features in Dash, leading to extreme difficulty when it came time to deploy to a web hosting service. Most if not all of these difficulties have simpler solutions with React.

Although the decision to use Dash was justified in the above section, and ultimately resulted in a front-end design the team is proud of, if we had additional time or were providing guidance to a team about to start, we would use/recommend React.

In terms of the deployment process, almost all of the time working on this was devoted to AWS. Of the three deployment methods we considered (AWS, Google App Engine and Heroku), AWS was the most complex and required Dev-Ops experience beyond any member in our group. Our team should have pivoted to Heroku sooner (something we have experience in). This decision to start with AWS stemmed from it being the most robust of the other options, however having non-robust deployment is better than no deployment at all.

9.2 Back End

First of all, to address the investable universe of securities considered by the Bridge Robo-Advisor: we first wanted to consider a universe consisting of all American and Canadian ETFs. In reality, this was too big of a set of securities to achieve a streamlined user interface. Instead we refocused the investable universe back to strictly American ETFs and the S&P500 stocks to achieve better performance. This is unfortunate however, the current Back End is built to support the original investment universe and is ready to be integrated if we had access to more computing power.

Next, to address the scheduled data updater. Originally, we considered creating a Back-End infrastructure that was completely self reliant when deployed. This meant that the data update system should be integrated with the rest of the web-application when deployed. Due to time constraints, instead we chose to use a local computer to complete the automatic updates. This is achievable because the database is mounted on the cloud and can be accessed by any computer. This current implementation is not very robust and would need to be updated if this project were to be scaled any further. This could be achieved by setting up SQL Server Integration Services which has specific support on AWS where the database and application are hosted.

Finally, the design team initially wanted to have the application to support real-time or as close as possible price data. This was not achievable with our data set selection requirements. Xignite, was identified as an API we could use. Should we implement this in the future if we decide to continue working on Bridge, we

would only use the real-time data to support UI/UX displays not the Business Model.

9.3 Business Logic

9.3.1 Clustering

Although the current implementation of clustering provides a diversified selection of ETFs, it inherits the random nature of k-medoids clustering algorithm. To be more specific, the final clusters, and their corresponding centroids, of k-medoids clustering algorithm can be different depending on the initialization. This is especially the case when there are many data points so that multiple local minima exist. An approach that we have considered is to increase the number of clusters then run the clustering algorithm several iterations. Subsequently we will only select centroids which have appears more than once throughout the iterations. While this is more robust to initialization, we were not able to implement this due to its large computation time. Not only do we need to run it for multiple iterations, we also increase the number of clusters to enhance the probability of obtaining overlaps. Since the time complexity of the algorithm is $\mathcal{O}(n^2k^2i)$, where n is the number of data points, k is the number of clusters, and i is the number of iterations, increasing both the number of clusters and iterations would make the computation time grow in a cubic fashion. For this reason we gave up on this implementation.

9.3.2 Forecasting

Although we decided to use ARIMA-GARCH model to forecast future factors, other non-linear models, namely supervised neural networks, could have been implemented as mentioned in the section *Modifications from Previous Deliverables*. While the neural network approach has the advantage that it can capture non-linear relationships, it has significant disadvantages such as a lack of transparency and limitations to the interpretability of the prediction [43]. This is prone to practical problems in terms of accountability. Furthermore if not provided with enough data points, it can lead to overfitting and poor out-of-sample performance. Finally training a neural network is very computationally heavy. For these reasons, we decided not to implement it.

9.3.3 Parameter Estimation

With regards to factor models, we have not implemented CAPM or Fama-French factor models with more than 5 factors. The former was not considered as there are abundant research from academia and industry that Fama-French 5 factor model is a significant improvement from CAPM or the 3 factor model [44]. The latter was not considered for a different rationale: firstly, incorporating more factors is not necessarily better since the model loses interpretability while it gains marginal performance; secondly, while the factors included in the 5 factor model have been thoroughly researched empirically, others have not.

9.3.4 Portfolio Optimization

Although the vanilla mean-variance optimization (MVO) serves as a good benchmark for portfolio optimization models, it was implemented for this project as the model has been proven to perform poorly out-of-sample for two reasons:

- it is numerically unstable due to sensitivity to estimation error; and
- it often produces overly concentrated portfolios.

Consequently MVO was not chosen as our candidate model but the robust MVO, which addresses the first point (numerical instability), has been instead. Similarly, only the ellipsoidal robust MVO was implemented but not the box robust MVO. This is because the realizations in the corners of the box uncertainty constraints may be too extreme. Instead, one may prefer a smoother uncertainty set such as ellipsoidal uncertainties.

9.4 High Level Design

Initially, in on preliminary report and project proposal we intended on the Bridge-Robo Advisor application to be much more of a traditional Robo-Advisor and use WealthSimple as a benchmark for our application's capabilities because it is one of the best options available in the market. However, through the process of completing this project the team identified a gap: There existed no type of Robo-Advisor product that could help clients specifically looking to for help with managing their stock picks. So we swiftly shifted, to refocus our application on this task. This will help further differentiate Bridge on the market and helps us identify a more scoped target market.

10 Appendix

10.1 References

[1] M. Phillips and T. Lorenz, "'Dumb Money' Is on GameStop, and It's Beating Wall Street at Its Own Game," The New York Times, 27-Jan-2021. [Online]. Available: https://www.nytimes.com/2021/01/27/business/gamestop-wall-street-bets.html.[Accessed:06-Nov-2021].

[2] "Do Money Right: Invest, Save, Spend & File Taxes," Wealthsimple. [Online]. Available: https://www.wealthsimple.com/en-ca/?utm_source=google&utm_medium=cpc&keyword=wealthsimple&matchtype=e&network=g&device=c&gclid=CjwKCAjwh5qLBhALEiwAioods1hpJfLkk50_s2BLxISmhp52RukpgOEUPCJMZpK9Ld-3d4L1CjHRthoCpUwQAvBwE. [Accessed: 13-Oct-2021].

[3] "Keep More of Your Money," Questrade. [Online]. Available: https://www.questrade.com/. [Accessed: 06-Nov-2021].

[4] "React – A JavaScript library for building user interfaces," – A JavaScript library for building user interfaces. [Online]. Available: https://reactjs.org/. [Accessed: 06-Nov-2021].

[5] Angular. [Online]. Available: https://angular.io/. [Accessed: 06-Nov-2021].

[6] "Angular vs React – A Complete Comparison (2021)," Brainhub. [Online]. Available: https://brainhub.eu/library/angular-vs-react-comparison/?utm_term=reactvsangular&utm_campaign=[txt]librarycontent-boosttraffic&utm_source=adwords&utm_medium=ppc&hsa_acc=3034370497&hsa_cam=12612047914&hsa_grp=124530861591&hsa_ad=518336452951&hsa_src=g&hsa_tgt=kwd-322192807431&hsa_kw=reactvsangular&hsa_mt=e&hsa_net=adwords&hsa_ver=3&gclid=CjwKCAjwz5iMBhAEEiwAMEAwGDFsABwE. [Accessed: 06-Nov-2021].

[7] "Data dashboarding tools: Streamlit v.s. Dash v.s. Shiny vs. Voila vs. Flask vs. Jupyter," RSS. [Online]. Available: https://www.datarevenue.com/en-blog/data-dashboarding-streamlit-vs-dash-vs-shiny-vs-voila. [Accessed: 06-Nov-2021].

[8] Dash Bootstrap Components, https://dash-bootstrap-components.opensource.faculty.ai/

[9] Yahoo finance api documentation. [Online]. Available: https://algotrading101.com/learn/yahoo-finance-api-guide/w.

[10] Polygon.io documentation. [Online]. Available: https://www.polygon.io/docs/#getting-started.

[11] Finnhub api documentation. [Online]. Available: https://finnhub.io/docs/api/introduction.

[12] Alpha vantage documentation. [Online]. Available: https://www.alphavantage.co/documentation/.

[13] Tiingo documentation. [Online]. Available: https://api.tiingo.com/documentation/general/overview.

[14] Google finance api documentation. [Online]. Available: https://support.google.com/docs/answer/3093281?hl=en.

[15] Xignite Financial Data Apis. [Online]. Available: https://www.xignite.com/financial-data-apis.

[16] Python PostgreSQL Tutorial Using Psycopg2. [Online]. Available: https://pynative.com/python-postgresql-tutorial/.

[17] MySQL Connector/Python Developer Guide. [Online]. Available: https://dev.mysql.com/doc/connector-python/en/

[18] How to use Python with Mongo DB. [Online]. Available: https://www.mongodb.com/languages/python.

[19] Getting started on Heroku with Python. [Online]. Available: https://devcenter.heroku.com/articles/getting-started-with-python.

[20] Boto3 Docs 1.19.11 documentation. A Sample tutorial. [Online]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/sqs.html.

[21] Use CI/CD to deploy a Python Web App to Azure App Services on Linux. [Online]. Available: https://docs.microsoft.com/en-us/azure/devops/pipelines/ecosystems/python-webapp?view=azure-devops.

[22] App Engine Standard Environment. [Online]. Available: https://cloud.google.com/appengine/docs/standard.

[23] Investing.com Canada. 2021. USD CAD Historical Data - Investing.com Canada. [online] Available at: https://ca.investing.com/currencies/usd-cad-historical-data [Accessed 27 November 2021].

[24] French, K., 2021. Kenneth R. French - Data Library. [online] Mba.tuck.dartmouth.edu. Available: https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html [Accessed 17 November 2021].

[25] PyPI. 2021. psycopg2. [online] Available at https://pypi.org/project/psycopg2/. [Accessed 17 November 2021].

[26] S. Boyd, "Multi-Period Trading via Convex Optimization," Stanford 2017.

[27] M. L. D. Prado, "Building Diversified Portfolios that Outperform Out of Sample," The Journal of Portfolio Management, vol. 42, no. 4, pp. 59–69, 2016.

[28] J. Pfitzinger N Katzke, "A Constrained Hierarchical Risk Parity Algorithm with Cluster-based Capital Allocation," Working Papers 14/2019, Stellenbosch University, Department of Economics.

[29] S. Karlsson, S. Mazur, and S. Muhinyuza, "Statistical inference for the tangency portfolio in high dimension," Statistics, vol. 55, no. 3, pp. 532–560, 2021.

[30] T. Raffinot, "Hierarchical Clustering-Based Asset Allocation", The Journal of Portfolio Management 44(2):89-99, December 2017

[31] A. Chaudhry and H. L. Johnson, "The Efficacy of the Sortino Ratio and Other Benchmarked Performance Measures Under Skewed Return Distributions," Australian Journal of Management, vol. 32, no. 3, pp. 485–502, 2008.

[32] J. Buša, "Solving quadratic programming problem with linear constraints containing absolute values," Acta Electrotechnica et Informatica, vol. 12, no. 3, 2012.

[33] D. Bertsimas, A. King, and R. Mazumder, "Best subset selection via a modern optimization lens," The Annals of Statistics, vol. 44, no. 2, 2016.

[34] V. Naimy, O. Haddad, G. Fernández-Avilés, and R. E. Khoury, "The predictive capacity of GARCH-type models in measuring the volatility of crypto and world currencies," Plos One, vol. 16, no. 1, 2021.

[35] T. Takaishi, "GARCH Parameter Estimation by Machine Learning" International Journal of Engineering and Applied Sciences (IJEAS) ISSN: 2394-3661, Volume-5, Issue-8, August 2018.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," Jun. 2017. arXiv:1706.03762v5

[cs.CL].

[37] Y. Kim, H. S. Choi, S. W. Kim, "A Study on Risk Parity Asset Allocation Model with XGBoost" Journal of Intelligence and Information Systems, Volume 26 Issue 1, Pages.135-149, 2020.

[38] L. Nevasalmi, "Forecasting Multinomial Stock Returns using Machine Learning Methods" The Journal of Finance and Data Science, Volume 6, November 2020, Pages 86-106.

[39] T. Idzorek. "A Step by Step Guide to the Black-Litterman Model", Zephyr Associates, Inc. 2013.

[40] A. Salomons, "The Black-Litterman Model Hype or Improvement?" ABP Investments.

[41] L. Chincarini, D. Kim. "Uses and Misuses of the Black-Litterman Model in Portfolio Construction", Journal of Mathematical Finance, vol.3, no.1A, 2013.

[42] R. Oprisor and R. Kwon, "Multi-Period Portfolio Optimization with Investor Views under Regime Switching," Journal of Risk and Financial Management, vol. 14, no. 1, p. 3, 2020.

[43] K. Nakagawa, T. Uchida, and T. Aoshima, "Deep Factor Model," ECML PKDD 2018 Workshops Lecture Notes in Computer Science, pp. 37–50, 2019.

[44] Y. Erdinç, "Comparison of CAPM, Three-Factor Fama-French Model and Five-Factor Fama-French Model for the Turkish Stock Market," Financial Management from an Emerging Market Perspective, 2018.