

Forecasting Agricultural Commodities using Climate Patterns

Written by Yash Hemanth Ganeshgudi

Contents

Analysis	2
A Description of my Solution	2
My Project's Stakeholders	11
Computational Methods used within my Solution	18
Research into Existing Solutions	30
Features of my Solution	43
Limitations of my Solution	48
Minimum Hardware and Software Requirements	50
Specific Success Criteria	54
Design	67
Internal Structure of my Solution	67
Proposed Screen Designs and Usability Features	79
Summary of the Processes at Each Stage	98
Specific Algorithms we plan to Implement	112
Suitability and Technical Details behind Algorithms	119
Key Variables, Structures and Classes	155
Validation within each Stage of the Solution	166
Identifying Test Data for Development and Alpha Testing	167
Justifying Test Data for Black Box Testing	172
Stakeholder Design Feedback and Sign Off	173
Development of the Coded Solution	175
Building an Initial Prototype	175
Iterative Design alongside the User Interface	233
Evaluation	259
Testing to Inform Evaluation	259
Usability Features and Usability Testing	286
Maintenance	286
Final Evaluation	288
Appendix	289
Code Listings	289
Bibliography	316
References	316
Disclosure	319

Analysis

A Description of my Solution

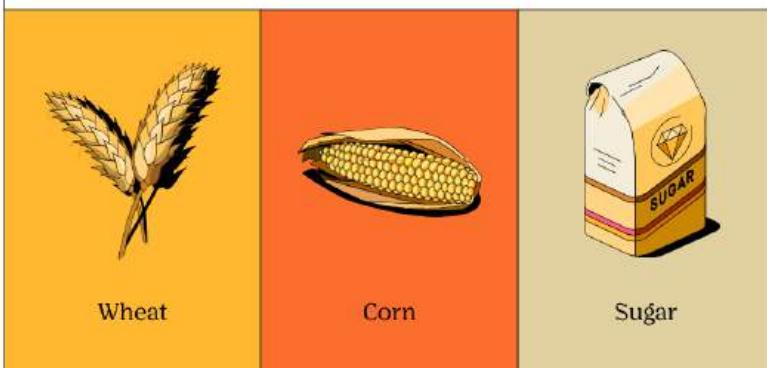
What are agricultural commodities? :

Commodities are, ‘Raw materials that can be bought or sold’. Commodities are generally classified into two categories: hard commodities and soft commodities. Hard commodities refer to resources that must be mined or extracted, such as metals, oil and natural gas. The most traded commodities globally by volume are crude oil, gold and natural gas. The global commodity market is largely dominated by hard commodities. Soft commodities on the other hand have to be grown and cared for, such as agricultural products or livestock. The global soft commodity market is mainly focused on soybean, corn, sugar and wheat.

Agricultural commodities in this case are products of agricultural activities such as corn, sugar and wheat. These products are mass produced and grown worldwide in multiple countries. Cocoa crops require a humid environment so are found in multiple countries along the equator. Multiple countries grow and consume products so the need to trade globally is required. To do this we use a single agricultural commodity representing the price per standardised unit of a crop. A standardised unit is a fixed amount of an item. For example, wheat commodities are traded in bushels, each bushel representing one million wheat kernels or approximately 27.216 kg. Cocoa commodities on the other hand are traded based on their weight using a metric tonne or 1000 kg. These standardised units of product are traded worldwide as commodities. Different independent markets in each country may exist for example the Ghana Commodity Exchange (GCX) and the London Commodity Exchange (LCE). Each market however reacts similarly to changes in a commodities price as they’re trading the same crop. Farmers and manufacturers use the price of an agricultural commodity to trade, buy and sell crops.

Agricultural commodities make up the majority of the soft commodity market due to the global dependency on food to trade large quantities of produce. There is an industrial demand for crops such as corn and soybean for biofuel and biogas also contribute to this demand. These fuels are an alternative to traditional fossil fuels like petroleum and natural gas. These demands are set to increase both due to the rapidly increasing population size and the need for cleaner alternatives to fossil fuels. Both of these factors mean there is a growing need for agricultural commodities.

Examples of Commodities



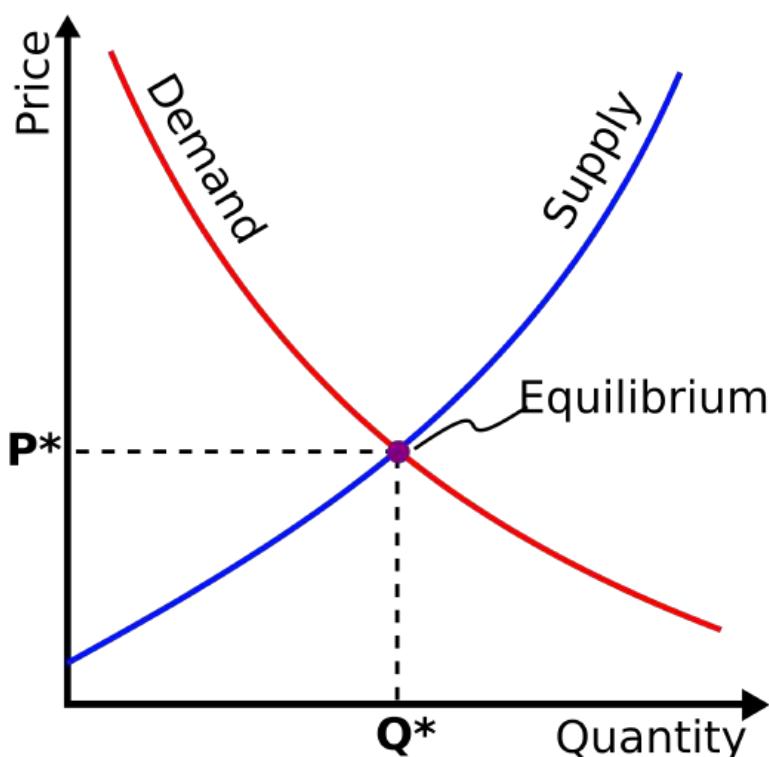
How do we forecast agricultural commodities? :

I was inspired by an article by Cristie Doug, an agricultural economist "How Weather Drives Commodity Prices". I wanted to explore the relationship between weather and agricultural

commodities. Crops are directly impacted by the weather, with over 50% of the world's produce grown outside on fields; this is still relevant today even with the rise of greenhouses and vertical farming. "An understanding and appreciation for what weather can do is a critical feature for agricultural markets". Dramatic changes in the weather in the short term and changes in the climate in the long-term can extract crucial insight into an agricultural commodities future.

To forecast an agricultural commodity we need to understand how its price is determined. Using two variables. Supply and Demand. An agricultural commodities price reflects the relationship between these two variables. Supply is generated by farmers and retailers with large supplies of these crops. Such as the government of China who stockpiles large quantities of rice, wheat and corn to help stabilise food

prices by maintaining the supply. Conversely the demand of a crop is made up of consumers located in different countries including the food needs of a population. manufacturers and industrial consumers. There is a certain equilibrium between these two variables, supply and demand, to maintain a normal price. In normal cases supply meets demand and the price remains constant. Let's say however there is an abundance of bananas due to a high yield in Costa Rica vastly above the global demand. This would mean banana commodity prices



would drop if demand is expected to remain the same. Mismatches between these two variables cause the price of an agricultural commodity to change to reflect this new change in equilibrium. An abundant supply with low demand will push the price of a crop per unit down. As there is a mismatch between supply and demand. So the price reflects this, with a lower price of the crop's commodity. To forecast an agricultural commodity we need to take this into account. We look at each variable more extensively below. The image shows how supply and demand change in response to a change in price or a change in quantity of the agricultural commodity. The goal of this system is to reach an equilibrium where the price and quantity are in equilibrium.

How do we forecast demand for an agricultural commodity? :

Demand can be thought of as the volume or quantity of product that consumers want.

A hard commodity such as lithium which is a type of metal used to make lithium-ion batteries. During 2021 the price of lithium per metric tonne reached an all time high, because of the rise of electric car investments and sales meant lithium's demand outpaced its supply being mined and refined. In this scenario the supply of the item couldn't be increased so rapidly as there was a physical limit of the amount of lithium that could be extracted from mines around the globe. However the demand in this scenario skyrocketed causing the price of lithium to do the same reaching nearly 80,000 USD compared to about 15,000 USD before it rose.

Looking at agricultural commodities on the other hand the overall demand for a crop's commodity remains relatively stable compared to hard commodities as these commodities describe raw produce which are necessities such as wheat and rice. Food is a basic human right because it is essential for survival. People must eat everyday regardless of economic conditions or conflicts. Taking this into account we can see that the primary driver of demand for food is the growing worldwide population. As mentioned beforehand industrial crop demand is also growing however at the current rate of production the infrastructure isn't able to scale so rapidly causing a scarcity of food. Bioenergy production in this case biofuels, biogas, and biodiesel only make up about 13% of global cropland.

It is true that demand does play a role in the price but these exist in trends with major fluctuations uncommon. Due to the nature of commodities being a necessity, we can expect the overall demand of an agricultural commodity to remain stable even during economic unrest. Agricultural demand growth is slowing and mostly driven by worldwide population growth. With the global demand for crops projected to grow only by 1.2% per annum over the coming decade as stated by the Organisation for Economic Co-operation and Development.

Food is an essential human resource we cannot live without. So with slow predictable growth in demand and the fact it is a necessity. We can assume the demand overall to be constant with no rapid fluctuations in price. Compared to other hard commodities such as lithium, where its demand independently controls its price. Agricultural commodities inherently can be thought of as a stable commodity. Therefore to accurately predict agricultural commodities we have to look at the other side of the coin, and independently analyse how the supply of an agricultural commodity changes.

How do we forecast the supply of an agricultural commodity? :

Supply refers to a crop's availability and abundance. So to predict the future supply of a crop we need to be able to anticipate future crop yields. Crop yield refers to the amount of the crop that is harvested. Global crop farmland is growing at a slow rate so we can expect the area of land that is capable of cultivating plants remains the same year after year.

All of this means that to forecast agricultural commodities as a whole, we will need to focus our efforts entirely on deciphering the relationship between the weather and the respective crop. As stated beforehand other causes of fluctuations can be taken into account but, their relative unpredictability and actual significance on a crop's demand and supply are difficult to predict accurately. The variables that we don't take into account are population-change, industrial demand and acres of farmland used for cultivation. What this means is that we expect as a whole the demand for crops globally to rise gradually, but at a constant predictable rate. With the only major changes being caused independently due to the weather. Therefore by looking independently at the relationship between crops and the weather our aim is to reliably predict a crop's performance. With over 50% of the world's produce grown outside on fields, we are reliant on the earth's weather systems to provide for the vast majority of our produce. All this means that historical and future weather data can be used as a detailed marker to assess and predict the performance of agricultural commodities.

Weather systems are complex so to summarise them our program relies on factors that affect a range of plants the most. These factors are also the most commonly collected values by meteorological stations all over the world making our system accessible to a wide range of locations. All of these factors affect photosynthesis which is the process by which a plant uses sunlight, water and carbon dioxide to produce oxygen and energy. This process is essential for plants to grow as it gives them energy and nutrients needed to thrive.

Sunlight	Sunlight is the main source of a plant's energy through photosynthesis. A constant source of sunlight is needed to maximise
----------	---

	a crop's growth and its ultimate yield. An area of farmland with a high cloud cover would cause a crop to receive too little sunlight hindering it from growing as large. Cloud cover is the metric we use to measure the amount of sunlight a crop receives over its lifetime. Cloud cover measured in oktas or as a percentage. An okta represents an eighth of the sky being covered by clouds, complete cloud cover is 8 oktas. Percentage cloud cover more accurately measures the percentage of the sky that is covered by clouds. Measuring the amount of cloud cover overhead means we can calculate the amount of sunlight and its intensity that the crop underneath will receive.
Water	Water is a crucial component of photosynthesis. It is also used to transport nutrients and minerals around the plant's tissues. Therefore a constant source of water is required to prevent a plant from dehydrating. Precipitation therefore is the metric used to gauge suitable water availability for crops. Suitable in the sense that extreme precipitation isn't advantageous to a crop. Droughts and heavy storms show the extremes of precipitation and weather events and will slow or even stunt a crop's growth. The fluctuations therefore of precipitation will also have to be taken into account. Fluctuations could be large-scale storms following a drought which would not be beneficial to a crop. Compared to the same amount of rainfall, but spread across a larger range. The crop would perform better in the second scenario due to being able to grow effectively. Even though in both scenarios the average rainfall across the entire period remained the same. The spread of the rainfall over a period should also be taken into account.
Temperature	Photosynthesis is a process that relies on enzymes. These enzymes are most efficient at temperatures between 25°C to 35°C. Temperatures outside of this range severely hinder the growth rate of the plants and prevent them from growing sustainably. Drought and freezing temperatures can even kill crops lowering the total yield of a crop. As stated above the spread of temperature should also be taken into account. As the average temperature over a period of time may not show sudden changes in temperature. Staying within the optimal temperature range throughout the entire growing stage is crucial to maximise a crop's yield.
Humidity	Humidity describes the amount of water vapour in the air. This can also affect photosynthesis indirectly. A constant humidity is needed to allow plants to efficiently absorb water, nutrients and minerals. Different plants require different ranges for humidity and some are

able to handle extremes. This means there is no set range and depends entirely on the crop. Humidity outside of this range prevents plants from absorbing nutrients and minerals effectively. However drastic changes in humidity are normally accompanied by changes in temperature / precipitation. So it is just another metric to take into account to understand the total impact on the crop's yield.

Dramatic changes in the weather in the short term and changes in the climate in the long-term can change the course of a crop's development and help us estimate an agricultural commodities future. Using just these four factors we can effectively model a weather system and estimate the total impact whether it be positive or negative on a crop's globally. Irregularities in climate patterns such as the previously mentioned droughts and storms, are weather events in the short term. These short-term weather events are able to wipe out a large proportion of crops extremely quickly. In contrast to long-term climate patterns such as how climate changes as a whole. If a plant's needs fall outside of the range of its environment it may not be able to yield optimally in this case hindering its results. For example sugarcane requires a high temperature and a lot of sunlight if it's grown in an area that has longer winters and a limited amount of sunlight. It is unable to thrive compared to a sugarcane crop that was placed in an ideal environment.

Each crop is unique in the amount of resources it needs to grow and its sensitivity to changes in these resources. Crops such as Wheat can be considered adequately weather-resistant because of its high tolerance to changes in temperature and precipitation. Whereas Cocoa can be considered weather-sensitive because of its need for a high humidity between 75-85% and consistent rainfall. This means that we can't use one model to cater to all crops instead when analysing and producing outputs for data we need to select a single crop and focus solely on that. The greater a crop's resistance to the weather the less sensitive its commodity is to climate events. This tells us that different crops react differently to the environment it is put in. Separating each crop and producing independent models means you can easily model the crops characteristics and how it is impacted by each individual weather factor. Having one large model that analyses weather data and the respective commodity market data. Won't be as effective as a model built from the ground up and fine tuned with its relevant dataset. Each independent crop's model will be trained on weather data that is in the regions it is grown and its own historical commodities prices.

There is also a need to understand dramatic weather events in the short term. For example a drought and its actual impact on partially weather-resistant crops. Whether a crop's commodities shows it can withstand short-term changes or not. Understanding the uniqueness of each crop is only possible when using distinct

models. Changes in the same environment will impact two crops differently so our model has to be appropriately biased on its unique effect on each crop. This bias represents the resistance of a crop to weather factors or can be thought of as the range of weather factors it can tolerate. All this means that we can create a model specific to our crop to accurately predict an outcome.

Each crop is grown across a wide range of locations all sharing similar climates. Spread across the globe each country will have independent climate events impacting their crops. This provides a level of security for a crop's commodity as there are multiple sources of the desired produce. So a dramatic climate event in one region won't have as severe of an impact on the overall price as the supply remains relatively stable. Taking this into account our program will need to factor in different regions where crops are produced and how this all impacts the future trend. Perhaps the international nature of these producers also show how this tool can reduce the fluctuations in food prices globally. In theory, dramatic climate events are predicted and their negative impact on the crop's market price can be quantified. Identifying an approximate impact on the international supply of that specific crop. Regional farmers across the globe can counter this price increase by increasing their yield output of this crop. Their incentive being the higher profitability of the worldwide low supply. The big picture is that the market overall will be resilient to changes in climate as farmers can react effectively and instantly. This would mean that rapid price increases are dampened as supply fluctuates less. Perhaps this tool can be seen as a solution to market volatility. Volatility refers to the variation or fluctuation of the price of something over time. A volatile market means prices change dramatically in a short period of time. A less volatile global supply would be one where food prices and global supply of food is kept secure. A farmer's ability to predict short and long term demand can secure this dream. Climate events don't often produce an immediate effect on global markets as there is a lag time between growth and harvest. This means that the time before global supply runs low can be exploited to prevent a global shortage of food.

Justification of implementing my solution and broader implications

A system that takes an algorithmic approach to predicting agricultural commodities can be justified with the added benefit it provides in the following scenarios.

Problem	Solution
Crop's are extremely price volatile due to there being constant mismatches in supply and demand.	Accurate forecasting will mean the volatility of the price is reduced so ultimately agricultural markets are more stable. This also mitigates speculation within the market as there is a specially

	designed tool to address the needs of users who want to track how the market reacts to weather events.
Increased food security as farmers and producers across the globe can anticipate and mitigate the impacts of extreme climate events.	Supply and Demand can be closely controlled as it encourages forecasting future yield of crops using this application. Impacting how locally and globally there is a consistent supply of food to feed a population.
Economic benefits as our solution are that it encourages trade globally.	Democratizing and open sourcing the link between the climate and agricultural commodities encourages farmers, traders and governments to make informed decisions about agricultural produce.
Climate Change has increased the frequency and intensity of extreme weather events across the globe along with the increasing population agricultural security has been impacted.	Our solution focuses on using real-time data to forecast and spot trends extremely quickly compared to traditional methods which would involve humans manually analysing data. This bridges the technological gap in agricultural forecasting as it allows farmers and traders to have access to a free program that is able to use advanced forecasting tools.

Ultimately our solution addresses multiple global challenges within the agricultural commodity market and allows for both global supply and demand to be managed by democratizing the data and making it free and open source for anyone to access.

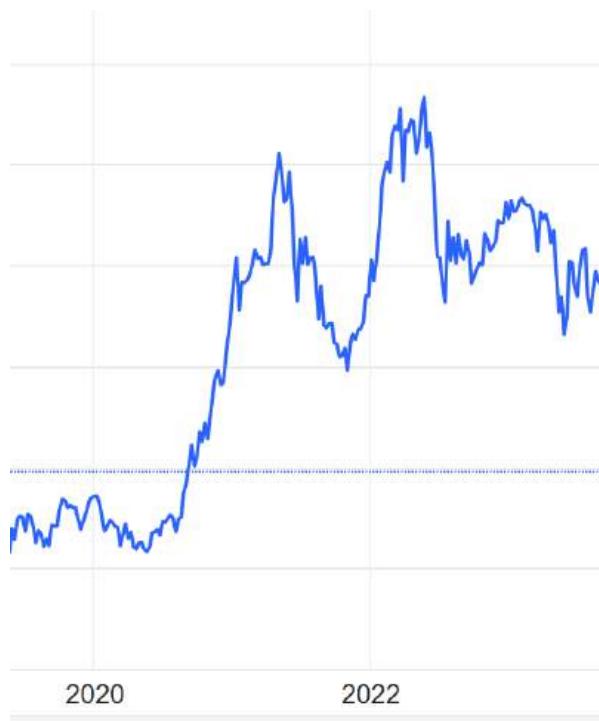
Examples of when the weather has directly impacted commodity prices :

	Price		Day	Month	Year
Soybeans	1,002.33	▼	-6.69	-0.66%	-1.06%
Cocoa	9,236.14	▲	983.15	11.91%	4.73%

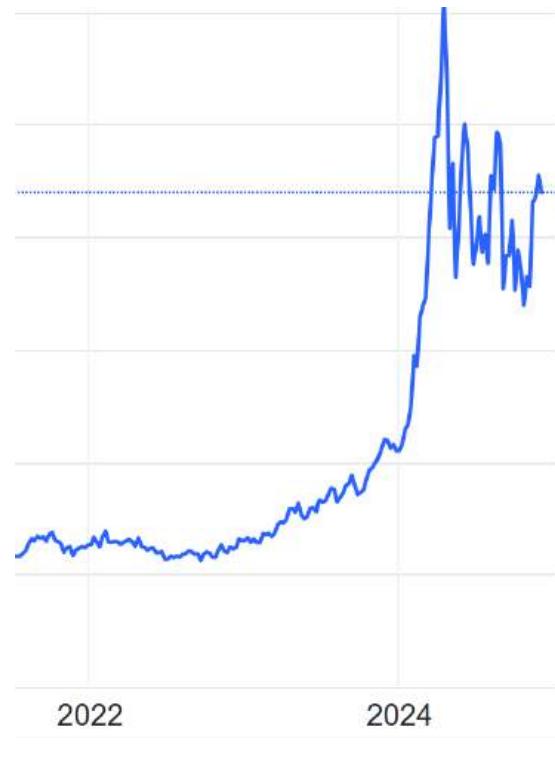
Image shows changes in Soybean and Cocoa Commodity Prices (06/09/24)

Soybean	Cocoa
---------	-------

Soybeans are grown all over the world, the majority located in Argentina and the US. Soybean's require a lot of water so need a constant supply of rainfall. During Q2 of 2022 A lack of rainfall paired with a sudden drought in these regions disrupted soybean production and caused the price to increase to \$1700 per unit and only now dropping to a normal price of \$1000 per unit during 2024. In this case other countries stabilised the price partially increasing production of Soybean, however it was only able to partially relieve the mismatch between supply and demand. This specific example shows the variation in crop yields due to extreme changes in the weather. The fact that international producers were not even able to meet the required demand shows the inherent fragility of farming crops. The image below shows the change in price of soybean during 2022.



Cocoa beans are grown primarily in West Africa due to the cocoa bean's need for an environment with a high humidity. During the first quarter of 2024 Cocoa bean's hit a record high of 9500 USD a metric tonne compared to its average price during 2023 of 4000 USD a metric tonne. This is shown in the yearly percentage change during 2024 of 158.50%. The cause, extreme droughts in West Africa causing a lower cocoa yield. In this scenario even with the extremely high price, global cocoa demand remained relatively the same with only an estimated 5% decrease. This fact reiterates that agricultural commodities are immune to changes in demand. As we can see even with the price reaching an all time high of over double the normal price, global cocoa bean consumption didn't fall short. The image below shows the change in price of cocoa during 2024.



We can account both these surges in prices directly to changes in the climate which caused a dramatic impact on the production of these crops. In the examples above we can see the lack of supply due to the climate being the sole determinant of the price. Perhaps even climate change is also a factor as it increases the amount of extreme climate events. These 2 scenarios emphasise the need to use climate and market data to forecast the future. Our aim explicitly is to build a tool that applies this theory. We will use historical price and weather data to forecast agricultural commodities. These scenarios shown above present a clear application of this concept into both commodity markets and as a crop yield forecasting tool. Catering both to farmers and your average commodity trader. We explore how a range of different people hope to use our system below where we will introduce you to the stakeholders of this project.

This tool in summary takes the complexity out of predicting agricultural commodities. Using historical market and climate data to predict the future price. The price of a commodity is reflected by the relationship between supply and demand. As food is essential, demand for agricultural products remains stable. Whereas supply is volatile due to the changing climate which impacts a crops growth. Wider scope and proof of this concept can be seen in the surge in price of Cocoa and Soybean due to a drought. Forecasting a commodity can be done by inputting future climate predictions into a trained model which then outputs a future trend. Key metrics within the climate are cloud-cover, precipitation, temperature and humidity. These metrics each affect a crop differently so you can have weather sensitive and resistant crops. Which means that you need a unique model per crop. Resilience of the market is also enabled by this tool as a decrease in supply due to a climate event can be rapidly reacted to by international producers keeping global supply secure.

My Project's Stakeholders

Identified suitable stakeholders for the project and described them explaining how they will make use of the proposed solution, why it is appropriate to their needs and their requirements.

Mrs Poornima Ganeshgudi

Teacher of Geography

Who are they?

Mrs Poornima has decades of experience teaching physical geography concepts such as climate change and how weather systems work. She's a keen investor and has a basic understanding of market movements and commodities. She's particularly interested in how climate change directly impacts prices in everyday foods from cereals to chocolate. Specifically how the wider agricultural commodity market and

climate interact and how crops are priced in conditions that are changing day-to-day.

Challenges they face?

She's unable to compile terabytes of data to make simple forecasts so she is led to reading day to day information that is broad and spans limited data points. These include financial newspapers and other sources so it is a purely qualitative approach looking at overall trends. In weather and other factors such as geopolitics. In light of this she is unable to make exact quantitative or numerical predictions. Instead making estimations of how different factors affect a crop's yield to approximate the future market's reaction. One key detail she highlighted was the futility of keeping up to date with crop yields across different topics and being able to look back at what happened in the past to guess how a crop's yield will be affected. After conducting an interview with her she said she believes her own human error and biases would negatively impact the predictions she makes.

Impact of our solution and their requirements?

Taking into account human error she feels that an automated unbiased system which will automatically retrieve, process and output a prediction will be a game-changer. Our tool automatically predicts the future of a commodity taking into account daily weather trends. The proposed solution is suited directly to her needs as it processes multiple metrics and handles formatting and processing easily. One other human aspect that this program eliminates is bias as it is mathematically unable to look at each data point differently; instead it captures all the data points and tries to minimise the error of its predictions. She wishes to use this solution as an indicator for future market performance. She believes in simplicity, so creating an easy to use tool with a specific application in the real world caters to an average trader investing in markets impacted by the climate. She requires an easy to use interface and clear indicators as to what is being processed as why. As she is a new user to price prediction tools she doesn't want to be overwhelmed with information. Another requirement is her need for reputable and up-to-date data sources as she wants predictions to be both accurate and up-to-date.

Mr Malakannavar "Manu"

Organic Cotton and Wheat Farmer

Who are they?

Manu is an organic cotton farmer that mass produces cotton under the Better Cotton Initiative. His farmland is in Khargone, Madhya Pradesh. He is a keen supporter of new and emerging technologies and interested in insight into future market movements of his crops. His farm supplies produce internationally under the Better Cotton Initiative so a broader picture of market performance will directly impact the

profitability of his produce and his activities as a day to day farmer. Being an organic farmer selling crops at such a large scale, detailed insight will enable him to make well informed and critical decisions in a small amount of time, with the support of a large amount of data. Processing data specifically for his crop's use case and its international market value.

Challenges they face?

Cotton is a commodity that fluctuates a lot so this means the price is changing rapidly. To keep up with competing in price with international markets our proposed solution fits the description of an industrial stakeholder for our product. Manu is the intended industrial stakeholder for our product. As a business owner, profitability remains a key focus and by minimising and managing risk using our system he wants to be able to focus only on the future market value of his goods. Especially due to climate change and other factors causing global supply to stagnate. In the last year alone, during 2024 worldwide cotton production fell by 3%. So the supply is set to decrease he projects that prices will increase to compensate for this. So he's not only looking at the cotton market ,but how the wider crop market will change. These unpredictable fluctuations in price with crops is something he wants to tackle and our solution directly addresses that.

Impact of our solution and their requirements?

Our tool will enable him to also maximise the productivity of his farmland by farming crops that will sell for higher prices. As our tool will allow looking at a range of crop's commodities farmers like Manu can be strategic and invest their time into more lucrative crops. This program incentivises having a globally resilient supply as other regions' farmers can quickly detect shortfalls and strategically increase the supply of the affected crops. Crop prices are time dependent and profitability needs to be assumed months before harvest so short-term insights can keep farmers informed on how the market moves without complex statistical knowledge. A commercial stakeholder will want more control over the final solution so advanced settings were suggested. This is contrary to the simple and almost minimalist approach suggested for an average trader. As the suggestions of our program may be too broad to suggest something. Industrial and commercial users would want to have the ability to edit and look at the finer detail. Advanced settings to allow you to control the hyperparameters were advised all of this will mean the program is more authentic to the customer's requirements. In summary Manu wants a well rounded and developed solution that allows him to have an edge over his competitors, by being able to look into the future to estimate the profitability of the agricultural commodity market.

Mr Shreeharsh Rao

Horticulture Society Leader and Teacher of Biology

Who are they?

Being Horticulture Societies lead Mr Rao is suited directly as an end user. His experience with plant growth and patterns particularly at the local level suit this product's as being a personal stakeholder. He's especially interested in how plants are directly impacted by biological factors which makes his insight valuable in this project. Projects like this provide a suitable research opportunity into climate change and the current schools Horticulture productivity based upon its crop yield and overall success.

Challenges they face?

As a beginner, he wants to see how global weather patterns impact the price of food in a supermarket. He also implied at the wider use-case of this tool looking at how the climate has impacted crop prices over the past decade. For example how a recent hurricane over Florida in the US might have impacted global orange prices. A significant aspect he was most interested in was seeing how you can algorithmically decode the environment where the crop is grown and see how it impacts the total crops yield and the overall price. With his strong biology background and interest in statistics he is looking forward to helping build a tool that caters both to his interests and also how changes in the wider world impact food pricing at the local level.

Impact of our solution and their requirements?

Instead of an investment point of view he is looking to use it as a research tool into how global markets work and how they change in response to different scenarios. He wants to see an easy to use, understandable interface and be able to recognize and understand what is happening behind the scenes at all times. Providing him an introduction about both agricultural commodity markets and how the global markets move in response to weather events. He represents the average everyday person and the range of uses that is possible using our tool. A requirement he wants is frequent real-time data updates. He wants a system that is always up-to-date and retrieves the most recent and relevant data without the need for human intervention. Accurately representing current market conditions.

Sharath Nagendran

IT Consultant and Hard Commodity Trader

Who are they?

Sharath is currently a quantitative trader in hard commodities such as crude oil and natural gas. He has looked into trading with agricultural commodities, but due to the limited information about the subject and the seemingly randomness of how the price fluctuates he hasn't invested any money into them. He is also a software developer as a day job so understands the underlying complexity beneath algorithms and data analysis. He is interested in the intersection between

technology and agriculture and how this tool can be used to predict the price of an agricultural commodity just using weather data and machine learning.

Challenges they face?

He struggles to try to decipher the market and see how the market reacts to different changes over time. From geopolitics to weather data he doesn't have the time to read and analyse the market's situation and how the prices fluctuate. He doesn't have enough time to scour the web and make predictions based upon basic news articles. Markets change daily so a constantly updated system is needed to make critical decisions in critical times. Therefore a data analysis that takes the global supply-chain into account would provide a competitive edge over purely quantitative traders which trade based upon only mathematical patterns and algorithms in the price of a commodity.

Impact of our solution and their requirements?

Our solution quantifies supply and demand data and saves Sharath time from looking at a range of resources which provide mixed returns. Instead decisions are made with the weight of a large amount of datasets in the background without the need for human intervention adding to the efficiency and lack of human error of our proposed solution. Our solution simplifies his workflow by effectively cutting out the need to scope out different sources and provides him with one definitive piece of evidence to base his predictions on. Being used in conjunction with other sources such as financial newspaper reports he can make decisions quicker and more accurately. Requirements he wanted are the completeness of datasets. As we're taking in so much data inputs and producing a single output, the completeness of data means that missing data points or anomalous results should be handled effectively. This means that anomalies such as incorrect temperature values or a missing cloud cover value shouldn't cause the program not to output a valid suggestion. Instead it should be robust enough to handle a range of inputs.

Stakeholders come from a wide array of backgrounds summarising concisely the range of end-users this tool caters to. From beginners to advanced users it should be accessible to everyone from investors and farmers to novice traders. A challenge may be catering to a wide range of requests from stakeholders with different areas of expertise. As the solution should be able to be used by multiple different people from different backgrounds. Developing features that are going to be used in a range of different ways. Perhaps instead of an issue it can be tackled as an advantage of this tool as it provides for a range of people and not just advanced commodity traders. Helping to build a well-rounded system with multiple use cases for multiple different types of people. Farmers for instance can reliably judge the suitability of planting different crops in their fields taking into account both their yield and the predicted market price. Helping to optimise planting decisions and negotiating better prices. Manufacturers could work out when to buy crops to maximise their profits and make sure their supply is secure.

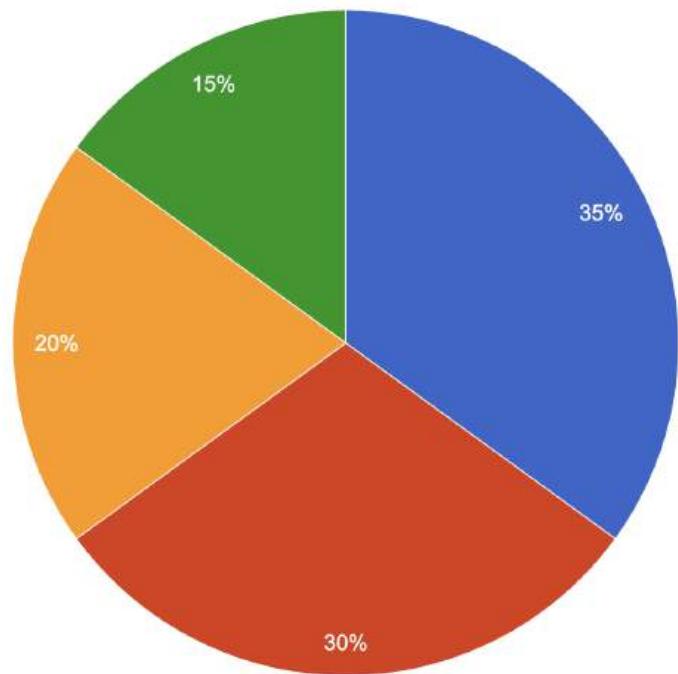
Customers could have a deeper understanding of agricultural markets and how they are impacted by the climate. All of these examples show different end-results ,but stemming from extracting the same conclusions from the same climate and market dataset. To do this I've looked at using a modular approach. Splitting the program into multiple separate functions which can be tested and developed independently. This would allow tailoring the tool to match clients specific needs by selecting and combining a range of modules. Overall the stakeholders of our tool has a diverse range of applications which is reflected by the selection of our stakeholders.

Initial Stakeholder Engagement Research

To understand and engage with their specific requirements I have used a survey which my stakeholders completed to gauge their expectations and how they plan to use the solution. This will allow me to cater directly to the end-user and directly lay out the initial plans for my solution.

Features in my solution that I should prioritize.

- Real-time Data
- Ease of Use
- Accuracy
- Customization

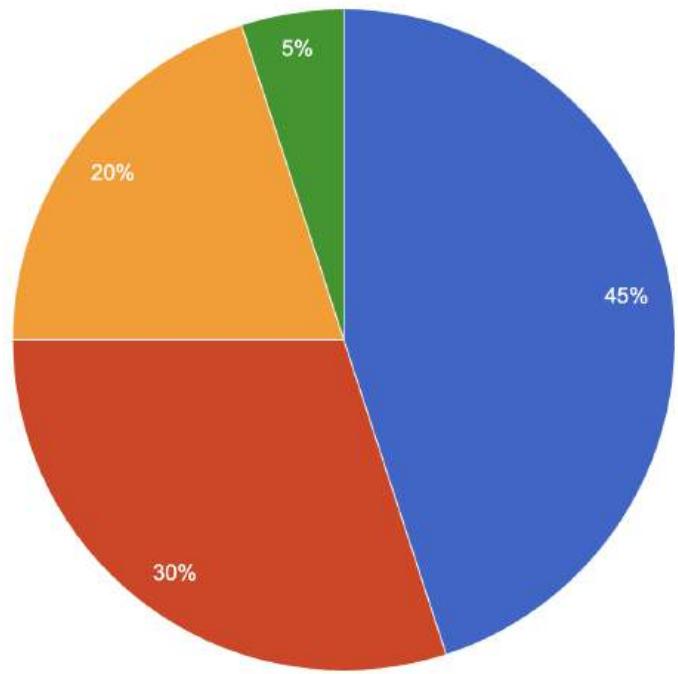


Surveying general features I should focus on the majority of stakeholders wanted a piece of software that is both easy to use and able to take into account real time data. Additionally, focusing on utilising only up-to-date data means that accuracy is maintained as we are using data representative of current market data and not old inaccurate data to make extremely accurate predictions as to future outcomes of the agricultural commodity market. Customization is a feature with the least engagement perhaps because it has limited application into the requirements of our solution to make predictions on the agricultural commodity market. Instead customization could be part of a movement to keep our solution both easy-to use

and accessible to the diverse range of stakeholders and therefore end-users that we have to cater to.

Expected time spent per session

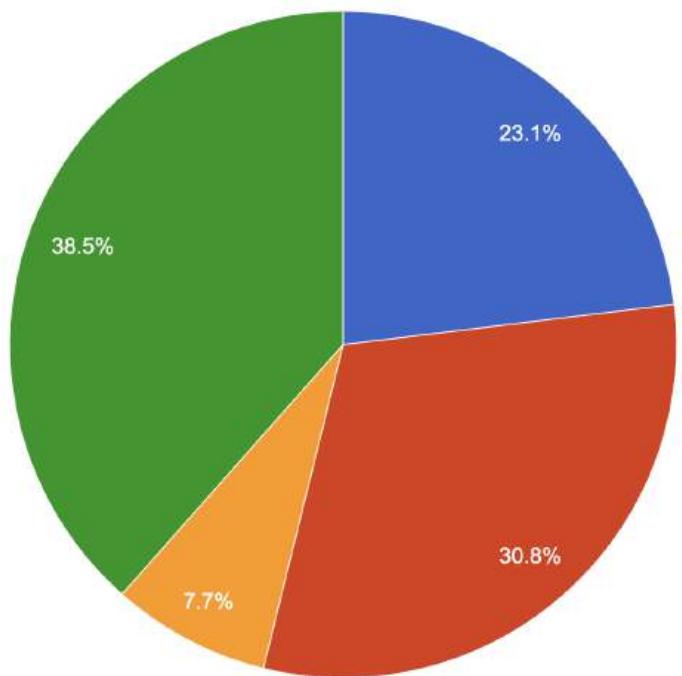
- Under 5 Minutes
- 5-10 Minutes
- 10-20 Minutes
- 20+ Minutes



The expected time per person gives us a guideline as to the general use case of our item and allows us to compare our solution with the wide array of alternative software. In this case we can see stakeholders want a seamless tool that is able to quickly produce predictions with the majority of stakeholders spending less than 10 minutes on our solution. This means that we are building a casual tool that is able to quickly produce predictions. In this case focusing on a minimalist system that allows the user to quickly produce the data they want.

Potential Areas with Limited Application

- Overwhelming UI elements
- Excessive data usage
- Data processing transparency
- Slow performance



On the other side I surveyed areas that I would take away from a successful solution. Specifically features that should be either avoided or streamline the entire user experience. The main requirement from stakeholders was to avoid a slow system and all the entire operations should execute extremely fast. Additionally stakeholders didn't want to be overwhelmed by information. Our solution as well should focus on data minimalism trying to produce an output without requiring a large quantity of data. This links with overloading the user with data as if we only focus on essential data points then we can streamline the entire solution. Additionally stakeholders expected users to want the program to be transparent about processes occurring in the background.

Computational Methods used within my Solution

Identified, described and justified the features that make the problem solvable by computational methods, explaining why it is amenable to a computational approach.

Abstraction :

1a - Identification and description of how to simplify a weather system :

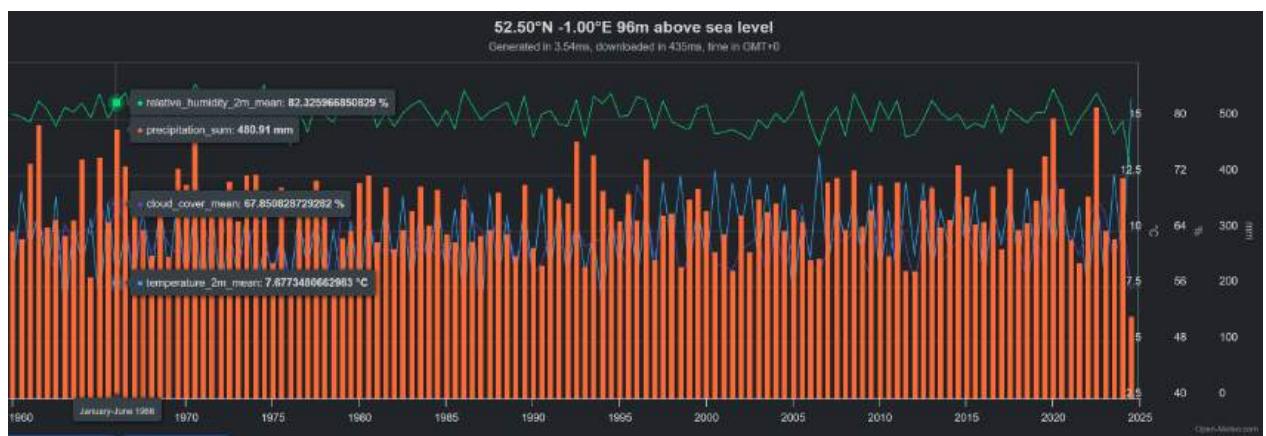
Any model of a real-life problem can be described as an abstraction. One example of this is how a complex weather system with multiple variables and effects is simplified into a collection of relevant data points. Abstracting unnecessary details meant we are only left with the simplified defining terms. These four variables are Sunlight, Precipitation, Temperature and Humidity. These four variables were picked because of the availability of values in countries across the world. As it is not complicated to measure the temperature or humidity data our datasets should be relatively complete with a minimal amount of missing values. Also these variables had the highest impact on a crop's yield so would have had the most significant impact when trying to predict that. Simplifying the weather means that all together the real environment where the crop is located is described appropriately and effectively.

1b - Explanation and justification of how to simplify a weather system :

Constraining data to 4 fundamental variables prevents complexity. As when all the data is processed only the required weather variables with the most impact on a plant's growth are analysed. Climate datasets that are created therefore remain concise and provide a valid representation of the climate over time without utilising too much storage space. Using more variables such as wind speed, dew point and atmospheric pressure in theory would add to the accuracy as the model will be able to draw upon a wider range of data points. So theoretically we will be able to make more accurate estimates. However this would overload the amount of data stored in

datasets and vastly increase the amount of processing time. As we are running on a personal computer there is a limit to the amount of storage used and the processing power we have at hand to compute results. Ultimately we would hit diminishing returns when allocating more processing time and disk space to variables with little to no impacts on a crop yield. Focusing directly on the ones with the most impact keeps our program running both responsively and accurately while also respecting the storage and processing constraints we are working with.

Computer's are best suited to this task because of the sheer volume of data that we are handling. Finding the correlation between multiple weather variables and commodity prices over a long time period. Humans are unable to do that both due to the amount of data that we can compute at a time and the complex relationships between these variables. Computer's have the ability to perform millions of calculations a second and identify patterns that would be impossible for a human to decipher manually. So simplifying a weather system from a fluid almost random moving object that you describe with qualitative words, instead into a quantitative collection of numbers which represent the environment. Perhaps this even adds to the scalability of this project as it can with enough time, data and processing power statistically analyse and model how different agricultural commodity markets react in real-time. Stakeholders during their initial interviews wanted an easy-to-use solution. This means that our program should be responsive and have the ability to handle tasks quickly.



In the image above we can see weather data that displays trends from 1960 to 2024. This includes the four main weather metrics and shows how the values change from year-to-year across decades. Abstraction in this regard allows for the efficient computation and to only keep the key elements of a complex weather system. Making it easier to analyze segments and understand it .

Stakeholder Input:

Stakeholder	Feedback

Mrs Ganeshgudi	<p>Simplifying weather data seemed like a good first step at approaching the problem, but looking at how it is graphed it looks as if all the data is random and there isn't a clear structure to it even in the simplified form.</p> <p>Mrs Ganeshgudi found this step straightforward and understood the reason for this, but pointed out the seemingly random nature of it as we are trying to correlate subtle patterns between two different datasets. In this case a weather and a commodity price dataset.</p>
Mr Manu	<p>Relying on a limited subset of data might pose a problem to industrial users such as Manu. This is inevitable as we are analysing data with a fewer amount of data points such as instead of taking multiple other weather metrics into account we just use the 4 specific ones. Interviewing Manu with this issue his main area of concern was the lack of "scope in the weather variables we have chosen. In a sense that using only four metrics may work in the majority of cases, but is it enough to correlate the price and weather data accurately and make a straightforward prediction."</p> <p>Addressing these concerns we looked into how Mr Manu currently forecasts the crop yield, specifically using a similar range of metrics. This involved a limited amount of data points as he used a similar tool bi-monthly to forecast a crop's yield. In this case he had a limited amount of both weather data points and crop yield data points so in comparison to our tool which works with daily data points in detail meant even with a limited amount of scope of our data we are able to perform directly based on the volume of quality data we have access to.</p>
Mr Shreeharsh	<p>Researching how weather impacts agricultural commodities was a key point of Mr Shreeharsh's interest in the tool. He felt as a biology teacher that our tool catered directly into his interests as focusing on only the key weather variables meant there is no need to take into account unnecessary and convoluted data processing and manipulation, which adding multiple more weather variables will mean.</p> <p>Contrary to this adding more data points may mean he is unable to successfully run it at his convenience as he may not always have access to powerful hardware. Having the ability to retrieve and process data on a range of hardware was a key idea of our tool.</p>
Mr Nagendran	<p>"My opinion as a trader is that if the accuracy of the final prediction is unaffected then it will be a good choice to make. It looks like a clear tug of war between the quality or the volume of data, but in both cases machine learning models are able to perform well. Specialising and making these choices now prevents</p>

	<p>hitting roadblocks in future development stages so in my opinion you should go for it as the model. The only necessity in the end is an accurate prediction.”</p> <p>Mr Nagendran relies on solely a performance based model as there is a constant need to produce an accurate prediction. This means that by focusing fully on the quantity of data and using a high quality selection of weather metrics which have been proven to have the most impact on a plant’s growth we can have a suitable trade off between the quality and the quantity of our training datasets.</p>
--	---

2a - Identification and description of how to abstract datasets :

Abstracting data in datasets would mean focusing only on essential information that is vital to the analysis of our data. Removing unnecessary details and being able to handle anomalous results in both climate and weather data is key to training our model efficiently. This is because streamlining datasets to include up-to-date and actually accurate data means that when we refer back to the data while it is being processed it won’t cause errors within a dataset to cascade into predictions that are less accurate. Data could also be simplified by using a technique called abstraction by generalisation to do this we can generalise hourly market prices or weather data values into weekly or monthly averages. This would still be relevant and be of a high degree of accuracy, while being considerably smaller in size. Both commodity and weather data would have the same sampling frequency as they will be processed together.

2b - Explanation and justification of how to abstract datasets :

An enhanced accuracy of a dataset as a whole will mean that there is little to no noise and errors in the data. A more accurate dataset is one without anomalies and missing values. An anomaly is a data point that deviates significantly from the expected range. To do this we will need to statistically detect errors within the dataset. So adding a pre-processing stage dedicated entirely to formatting and cleaning data will need to be taken into account. Ultimately this extra step would lead to more accurate predictions as a whole. Computers are suited to handling structured and accurate data. So abstraction of this data will mean for the more efficient applications of algorithms and the ability to easily scale our solution to handle even larger datasets. This is because there isn’t as high of a requirement to understand the data if it is clear and concise.

Plants are not that sensitive to hourly changes in the weather, but weekly and monthly cycles. The weather is constantly changing, so generalising and removing anomalous values would prevent users from hitting storage capacity constraints.

Daily averages will still retain the same trends mentioned ,but just at a lower quality. This is because instead of seeing multiple temperatures throughout the day we are looking at a larger range such as a week. So the big picture will show the average values of independent weeks across the year. Overall the impact of this will be a still clear trend over the same time period ,but in a more summarised form. However, by reducing the amount of data we are processing in the same time period. We can exploit this by processing a larger amount of data. This would mean looking at more historical data as each data point spans a longer period. Overall this would mean we can both retain accuracy and have a larger subset of data we are able to look back on. However, this may only be an issue due to the time constraints of a user which prevents us from maximising the amount of data we can process overall. These time constraints mean there is only a limited volume of data that a computer system is able to handle. Using all our data points would mean we aren't effectively using our limited resources. This also adds to the accessibility of our program as it will be able to run on a range of different hardware. Without the need for specialist hardware such as powerful GPUs which we explore when discussing our hardware requirements below.

The main justification of abstracting data would be for computational efficiency as if we limit the number of variables the effective computational load is reduced enabling faster processing of data in bulk. This means that we are able to run on low-power consumer grade computers that have limited hardware instead of relying on either powerful hardware or outsourcing processing activities onto external devices such as in the cloud. Additionally doing so we make practical trade-offs as the complexity is not justified for the additional overhead that it produces. In this case a marginally improved accuracy for predicting the future commodities price.

Thinking Ahead :

1a - Identification and description of Inputs and Outputs :

Our program as it is mainly a data analysis tool relies on a vast input of data with the ultimate prediction being the output. The prediction in this case being whether the crop's commodity will increase or decrease in price and by what amount. An API is an application programming interface that allows us to access a range of data from the internet. An API we use to retrieve weather data is OpenMeteo, an open-source weather API that stores historical and future forecasts of weather. OpenMeteo stores all 4 required weather variables in various countries across the globe and across a large timescale. Historical market data can be retrieved from a database that I have downloaded from Kaggle, a machine learning community, historic and current prices not included in this database can be retrieved from Alpha Vantage

who provides up-to-date market data. Commodity prices could include average daily, weekly and monthly prices and opening/closing prices.

The output of our solution should include a short-term prediction as to how the market will react to changes in prices. This will be indicated not by an exact price, but a trend that can be used to gauge how successful a commodity will be in the near and far future. This output will be calculated using the provided inputs after their pre-processing which includes the cleaning and formatting of data. After correlating both historical price and market data. This same model is applied onto future forecasted weather data. This means that we can produce an approximate price which we use to estimate the future of an agricultural commodity.

1b - Explanation and justification of Inputs and Outputs :

This means that my entire program can be kept small in size as it only contains the necessary code used to run the program. APIs allow users to selectively access terabytes of data without having to store it all on their own devices. This means we have a standardised source of data which we can use to efficiently access external data sources. External data sources that are up-to-date and quickly accessed. Having access to real-time updates is important to make sure your predictions are based on the latest weather and market movements. APIs also allow the program to easily provide personal results as you are only accessing specific datasets required for your analysis. Instead of storing unnecessary information on your own machine you can access high-quality data with a large volume. Relying fully upon a third-party source for data means that you can retrieve more information than if you were fully dependent on data stored on your own system. An internet connection will also be required otherwise you won't be able to receive the requested information. Talking to Mr Nagendran, a commodity-trader, he said having real-time up to date information is what gives him a competitive edge.

Climate models 1 / 7

- CMCC_CM2_VHR4 (30 km)
- FGOALS_f3_H (28 km)
- HiRAM_SIT_HR (25 km)
- MRI_AGCM3_2_S (20 km)
- EC_Earth3P_HR (29 km)
- MPI_ESM1_2_XR (51 km)
- NICAM16_8S (31 km)

Question : Why does having real-time data give you such an advantage?

Answer : The markets move too fast to analyse so we need to make split-second decisions in such a fast-moving market. I don't want to be misled by outdated numbers!

This led me to pursuing this approach relying purely on third-party databases to retrieve data from. However this means that there is an added step of formatting and cleaning

data, as you can't trust the data you retrieve is valid and accurate.

Data Sources :

<https://www.kaggle.com/> - Kaggle (Price and Climate databases)

<https://open-meteo.com/> - Open Meteo (Climate Data API).

<https://www.alphavantage.co/> - Alpha Vantage (Historical and real-time market Data API)

In the image we can see a range of climate models which contain both historical and future weather data and covers the entire world in 51 km squared segments.

1a - Identification and description of caching :

As Sharath said beforehand speed is vital to the success of this solution. Therefore I plan to implement caching into this process. The price of a commodity is retrieved multiple times so storing a dataset of just commodity prices will help speed up the process. This means that we only rely on the API for the current commodity price and not historical data. This historical data when retrieved can update the dataset we have stored on the device. Instead of accessing the data through the API and then formatting and cleansing it. Data is ready to go on our device reducing the overhead of data we need to handle.

1b - Explanation and justification of caching :

This will significantly reduce the total load time waiting for megabytes of data to reach our device and be correctly processed. Thinking ahead as to future inputs by caching helps reduce the total latency to produce our final solution. Prioritising efficiency and speed while also keeping results accurate. Caching may only be applicable to commodity data as there is a finite amount of commodities that traders are interested in. Weather data on the other hand is larger and would have to accommodate multiple countries so would put too much strain on the users storage. As part of a computational approach this would significantly reduce the amount of processing power required helping to conserve resources and dedicate them entirely to tasks such as handling the retrieval of weather data.

Question : Do you think a smoother, more seamless experience would be a benefit for this tool.

Answer : As a new trader that is constantly learning. A system that requires lots of processing power and storage requirements may overwhelm them if they were in my shoes. I want to be able to run the application easily and quickly, not limited by my own hardware.

Caching would mean our wide range of stakeholders are satisfied as the overall footprint of our tool is smaller. Only data we use constantly is cached and only specific current price and weather data is retrieved. I wanted to focus on keeping

this project accessible to multiple people. Not just focusing on a single stakeholder who has access to a powerful machine. Accessibility means that our tool should not require complex hardware such as GPUs and a large amount of storage and instead be able to handle a range of devices and different hardware.

There is an accuracy advantage with utilising real-time up-to-date data for the latest market and weather trends. Additionally fetching high-quality external data allows storage to be used effectively without needing to store databases of information. Therefore as the system is reliant on separate sources for weather and market data it allows for the system to easily scale so multiple users can use our solution at the same time. Building on this concept, implementing caching with commodity price data allows only required data to be fetched, speeding up the overall process in generating accurate predictions of agricultural commodities.

Thinking Procedurally and Decomposition :

1a - Identification and description of implementing procedure and decomposition:

Procedure involves breaking down the entire problem into a sequence of steps. Each step is smaller and more manageable. Our entire solution is made up of a collection of microservices that data cascades through and produces an output. The sequence of steps that the program follows is shown below. Data follows a logical linear sequence with each step having a clear product of what will pass into the next step. The sequence provides a framework that the program follows to get to the solution.

1 Input - Retrieve data from datasets on our device and from our APIs. This process would also include formatting and cleansing the data.

2 Processing - Correlating historical climate and market data to create a suitable model. This section is where the patterns are recognised hidden inside our datasets.

3 Prediction - Pair the model we just created with future forecasted climate data the output of this model should be the expected future price of the commodity.

4 Output - Visualisation of the predicted trend. The forecast is overlaid on top of an interactive time-series graph. This output step gives the user their desired trend.

Within each layer we can use decomposition to produce smaller independent modules which are clearly organised. These independent modules show the finer details within each step. These steps and modules fit into the overall movement and processing of data across our solution and how data should flow through the program. This is explored in more detail during the project's design phase.

1b - Explanation and justification of implementing procedure and decomposition:

Procedure of the program allows us to tackle the problem one step at a time and visualise the sequence as data passes through each step within the solution. It ensures each step is executed correctly as the framework we have planned enables this. All this promotes efficiency and allows errors to be correctly identified. Each layer remains independent of each other and data can only pass downwards from one layer to another. Using decomposition to split up each layer into multiple independent modules means that each module can be debugged and created separately. This simplifies the development and maintainability of our program. Each separate module can be connected together to structure the entire program. Segmenting our entire program links to how computers execute data in sequence and in parallel. Separate functions that make up the program enable this, allowing instructions to be executed sequentially and then when required in parallel with each other. Procedure and decomposition both in the organisation and code of our program reflects how the program will be implemented.

Maintaining a procedure along with decomposing the solution into multiple independent components allow for adaptability to new requirements along with easy debugging of the entire system without needing to restructure and develop components of the program. Additionally this allows for the implementation of an error system as there are clear inputs and outputs for each component which can be suitably validated to make sure the system is both responsive and processing data correctly. Maintaining a modular framework for the entire program allows for components to easily be replaced or debugged.

Logic Thinking :**1a - Identification and description of examples of logical thinking :**

Logical thinking involves strategically identifying places in our program where we will need to handle multiple decisions by branching and repetition by using loops. At the start of the program where we are retrieving and compiling our inputs. There is a need to validate that our data is accurate. This is part of the formatting and cleansing step. During this we are looping through our current dataset while also accommodating new data. Throughout this entire process data will need to be appropriately formatted and if it is anomalous in this case referring to values outside the range and a missing value. The code will stop executing and branch to a function which is able to handle and amend the mistakes in the dataset. Looping through and branching when a mistake is detected helps to refind the dataset and make sure it is of high quality.

Our project requires using machine learning to unlock the patterns between multiple variables. A machine learning model I am planning to implement is an LSTM(Long

Short-Term Memory) network. There is a need to repeatedly iterate over the training data and change parameters within the model to minimise the overall error and maximise the models accuracy. An epoch is defined as a complete pass through the given dataset that it is trained on. In our case the model will have to complete multiple epochs to effectively understand the dataset for the given crop.

1b - Explanation and justification of examples of logical thinking :

Using looping and branching this way in the retrieval step allows the program to remain fluid and able to handle a wide range of inputs in the datasets as we are working with external and unknown data.

Both these factors add to the adaptability of the program as it is a computer statistically analysing the data removing inaccuracies adds to the accuracy of the produced model. The dataset should accurately represent the environment the crop is grown in and the corresponding demand for that crop. Using these two techniques allows for the application in the real-world to be both practical and useful. Branching in this scenario allocates the exact data point to a function that can appropriately fix this mistake. After it is corrected, the edited value is pasted back into our main program and it continues looping through the rest of the dataset.

Refining our model by maximising the amount of epochs that it is able to complete will mean that the trends between changes in weather data and the price is well defined and understood by the model. As each crop has a separate model there is a need to adapt to different conditions so looping through the data multiple times by utilising a large amount of epochs guarantees an accurate system.

Validation is a key element of logical thinking as it ensures that across the entire solution at each decision point only valid data passes through the system. Additionally in the real-world logical thinking allows the solution to foreshadow the discrepancies within real world databases. This builds the systems resilience and adaptability as it is able to both debug and handle a range of inputs as data passes through decision points instead of an error cascading through the entire solution. Ultimately all these systems work in harmony with each other to scaffold the system as it can resolve errors independently and using techniques such as capturing missing data and imputing them with valid substitutions instead of allowing incomplete datasets to be included within solutions reference material it uses to make predictions.

Concurrency :

1a - Identification and description of concurrency :

Training machine learning models require editing the numerical parameters of the network individually. This process is called backpropagation and involves calculating

the error and individually adjusting the weights of connections between neurons in the network. Doing this over multiple epochs allows us to produce a model with a high accuracy in predicting the actual price of a commodity in the future. As our tool relies on processing a large amount of calculations this can be sped up by using a GPU. This is a special-purpose piece of hardware that has an extremely parallel architecture. This means it has a large amount of cores that can independently process data making it suited for tasks that require parallel computation.

Concurrency can also be used when using cached data as mentioned above. We plan to use a cached database stored on the client's device for commodity price data. This will mean that when the database is being fed into the main program, looping through multiple values, at the same time data can be retrieved from the required API. This use case of concurrency allows databases to be checked and updated simultaneously reducing the overall time needed to process all the relevant data.

1b - Explanation and justification of concurrency :

As we are dealing with large datasets with multiple variables using concurrency to speed up calculations would be extremely effective. An improved training speed as multiple calculations can be executed at once at the same time. This however is entirely dependent on the GPUs hardware on their computer and most modern hardware would show a significant improvement in processing speed compared to calculating sequentially on a CPU. This is because the calculations that are being performed are not that complicated ,but instead there is a large volume of them as there may be millions of nodes and weights between them. Computational methods play a role in this as there is a need to handle such large datasets and extract insights quickly. Automating complex tasks like correlating weather and market data effectively. A concurrent approach is valid in this scenario as it adds an extra layer of efficiency to our system. This significantly improves the speed of pre-processing data. Put into production, concurrency maximises the usage of the devices hardware as components are not allowed to remain idle.

Stakeholder Feedback:

Stakeholder	Feedback
Mrs Ganeshgudi	<p>“I want to have up-to-date data in front of me. This is because I want to be able to look at both relevant data and have the model trained on all the data that is available to it no matter the recency.”</p> <p>To do this our program will need to be able to accommodate retrieving data and being able to suitably process it. What this means is that stakeholders will be able to use up-to-date information on the specific commodities and not be limited to old irrelevant data.</p>

Mr Manu	<p>"This might be technical, but using branching techniques and loops helps to streamline the process. This opens up many other opportunities as the program can react to change so you can implement tasks such as validation to verify that the data that passes through the model is valid and relevant."</p> <p>Manu as an advanced user may be dependent on this single tool for producing commodity market predictions. This would mean there needs to be an added form of redundancies to optimise and make sure all the components work suitably on the model without flaws. Using validation means that the program can branch if it detects an error or an anomaly, fix it and carry on with the rest of its processes. This means that Mr Manu can rely on the data and the internal components of our solution as at relevant stages everything is checked to be correct and only then does operations continue with the program.</p>
Mr Shreeharsh	<p>"I would want the solution to use all available resources on my computer if it means I can minimise the time it takes to produce a prediction. This is a main point if we are using low-power hardware as not utilising all the hardware may mean predictions take extremely long to be generated."</p> <p>Shreeharsh wants to maximise the utilisation of his hardware. Particularly using concurrency to train the model in parallel comes to light. Using parallel processing we are able to efficiently process the data all instead of needing to process it on a single thread or core sequentially. This is because parallel processing allows using multiple cores to process information independently so the model can be trained all at once.</p>
Mr Nagendran	<p>"As a trader there is a constant need to perform at lightning speed. What this means is that we can assume the market reacts instantly to weather events. For instance if we can predict that the global crop yield will have a 5% reduction we can make bets within the commodity market that the price will go up. However a key component of this is speed . As a trader you are competing with other traders. This means that a solution that runs and produces predictions quickly is crucial."</p> <p>For Mr Nagendran the implementation of a cached database was crucial due to the nature of the commodity markets and the need for speed. This speed meant that commodity markets will need to produce predictions fast and outperform other traders. Relying on an external API for activities such as retrieving the current price would have negatively affected it. Fixing this we can use a cached database that stores data and can be quickly accessed instead of relying on an external source which would take longer to retrieve and process.</p>

Concurrency improves speed and efficiency while also maximising hardware utilisation of multiple cores by processing data in parallel. Additionally adds an added performance aspect as data can be processed faster in parallel than sequentially. Maximising hardware utilisation also means that the model can be trained quicker and data activities such as retrieval and processing can occur quicker. This translates to enhanced responsiveness and speed for the stakeholder as processing power is not left to idle.

Using computational methods, I can see clearly how this data analysis tool should function. The backend architecture should be able to efficiently process and extract insights. Data is integral to this system as it is a broker of future trends that need to take into account multiple data points. In the background, abstraction helps to simplify the issue at hand by focusing on only 4 key weather variables. Simplifying a complex weather system into a series of numbers that a computer can easily understand. Identifying inputs and outputs helped me to see how data should be handled throughout the entire system. Using a large amount of information sourced from third-party APIs meant we had to follow clear guidelines to format and cleanse data. The data within the system had to also be efficiently accessed using caching to store frequently used data on their device and only refer to external APIs when we need to. Breaking down a problem into a series of steps and then breaking each layer further into independent modules will help focus development and understand how the system operates as a whole. Logic can be seen throughout the program specifically at points where they have to make a decision such as branching when we are analysing how correct weather data is. Repetitive tasks such as training the model on training data is done by looping through a large number of epochs. Concurrency can be implemented alongside a GPUs largely parallel architecture speeding up the huge number of computations that occurs in a sense maximising the total throughput of our system as we are not bottlenecked by slower hardware such as a CPUs sequential processing. Ultimately computational methods ensure the solution is built to a high standard which includes efficiency and accuracy.

Research into Existing Solutions

Researched the problem in depth looking at existing solutions to similar problems. Identifying, describing and explaining, justifying suitable approaches in depth based on this research.

Initial Stakeholder Research

Stakeholder	Feedback

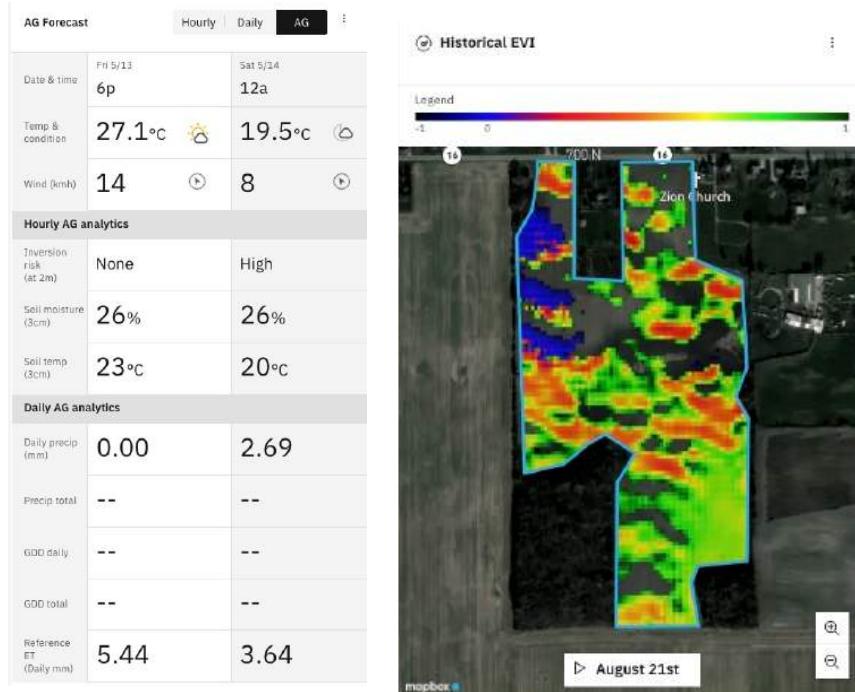
Mrs Ganeshgudi	<p>"Predicting agricultural commodities using weather patterns is a complex task. So existing solutions that do similar predictions need to produce complex predictions in a concise manner to easily interpret the underlying trend. This may mean focusing on essential information and essential user elements within the user interface to prevent overwhelming the users with an excessive amount of data and features."</p> <p>Ensuring the user has a seamless experience when using all stages of a solution is important and should be a main point that solutions are trying to accomplish. Novice users like my stakeholder want to be guided through the entire process so a clear user interface and prediction system is essential to a successful existing solution.</p>
Mr Shreeharsh	<p>"Existing solution should have a user interface that is clear as to how weather events impact the prices of crops or the respective crop's yield. This means that even novice beginner user's like myself should be able to both access the software, but also use it effectively."</p> <p>Stakeholders want detailed insights that can be accessible using a simple user interface without the need of advanced background knowledge. These insights should be both easily visualised through their interface and easy to produce without the need to understand complex theory or the need to access advanced settings. The progress of the entire system should be straightforward in creating a final prediction consistently.</p>
Mr Manu	<p>"My main initial ideas for similar solutions would be the ability to look into the future crop's commodities prices or other relevant variables such as a forecast of the crop's yield. Being strategic with these decisions the solution should be easy to use and not need any previous knowledge to use; this also should be translated in the user interface as it should provide enough detail to make important decisions."</p> <p>This is because farmers specifically want detailed insights into their crops to make decisions quickly. With rapid fluctuations throughout crop prices this means that farmers currently and increasingly in the future use technology to get critical insights to optimise their processes. Detail was another important factor as it meant the system should give extreme detail as to why it shouldn't be overly simplified as a cotton farmer for example should be able to make important decisions without the need to analyse more data.</p>
Mr Nagendran	"A competitive edge is the main aspect I am looking into similar solutions as I want to be able to quickly open and run the

application without needing to understand the underlying information. I want a user interface that provides me information quickly and allows me to navigate and flow through the entire program without any friction.”

A quantitative trader wants to just extract the background insights behind how the weather affects the commodities price without having to worry about the specifics behind how these predictions are made and why. An autonomous system is what Mr Nagendran desires, to be able to gain a slight edge over other traders who are making decisions based on qualitative opinions such as financial newspapers instead of relying on numerical predictions. Numerical predictions give an exact estimate of the future crops commodities movements.

IBM Environmental Intelligence :

Image below shows an example use case of IBM's Environmental Suite.



Identifying features :
IBM Provides a detailed climate analysis tool for farmers and commodity traders which includes a suite of tools. These tools include complex weather analysis and forecasting that can be catered to your chosen crop. IBM collates and visualises a vast range of metrics from Max Wind Gusts to the spread of rainfall as we can see in figure 3. This extremely accurate defines IBM's main goal

and detailed data

working to summarise short-term weather events accurately. Dependence on accuracy for farmers also means that long term predictions are unavailable. IBM's end-user seems to value precise weekly and seasonal forecasts contrary to longer annual predictions. Their aim is to create a tool that represents current climate events in depth. Weather Metrics range from standard ones such as Precipitation and Cloud cover to specialised metrics such as soil temperature and the temperature difference between soil and air temperature. IBM acts as a specialised data broker with some support for crop health tracking based on real-time weather events over smaller areas of land such as fields. Crop health tracking remains detailed and useful to respond to short-term events rapidly.

Descriptions of these approaches :

My stakeholder Mr Malakannavar trialed IBM's suite of tools on his cotton farm. Stakeholders were impressed by the detail ,but emphasised that it had limited impact as the extreme detail produced had little use to actual physical changes in how he farmed. He also said he enjoyed the customization of this resource as it allowed easy integration with a range of moisture sensors and other sources of data. Allowing him to get a clear picture on how his crops are performing. However this resource had no integration with current agricultural commodity price data and could only be seen as an indicator of a crop's success in the short term. Moreover, it was inaccessible to lots of users because of its complexity and lack of actual insight specific to a crops commodities value. As IBM acts a broker of data for acting rapidly on short term trends a use-case not suited for average people or traders. Farmers are likely to be able to react effectively to long term subtle changes in climate so IBM's focus on short term forecasts and regional analysis seems to be effective for its target end-users.

Anticipating changes in short term weather patterns not over long term climate forecasts also allows for a broader range of metrics. In Figure 3 we see precise crop yield analysis for a farm based in Lucerne, Indiana. The system can easily estimate a crop's yield as we can see in the Historical EVI chart above. Gauging how multiple variables interact with the final crop. Specialist metrics included soil moisture and temperature that normal weather sources did not collect. This was in addition to metrics like rainfall patterns and intensity over an area with a detailed hourly forecast. IBM's narrow scope allows it's integration of many more climate variables and exploring them in depth. Figure 3 shows how successfully a crop is growing across the farmland and accessible metrics at a glance. This is presented seamlessly to the end-user and all on one page. IBM's solution only compiles climate data and processes it to provide detailed crop yield metrics in the short-term. In comparison to our tool which takes long-term trends into account, aggregating data across days and months. IBM's advantage is in their abundance of both computational power and high quality data sources. Large servers able to process terabytes of data quickly along with custom sensors and weather stations reporting data in real time. IBM's facilities allow them to specialise entirely into short term forecasts and dominate in this field. Environmental intelligence end-users value high quality data in large quantities. However handling this amount of data may have diminishing returns as the same conclusion could be gathered from a smaller subset of data. As less data could reach the same conclusion such as extreme rainfall ,negative impact on crops. However the practicality for IBM as they have a vast amount of resources

they can handle. Reassures their users of their data produced. Conclusions of which it would not be feasible or sustainable for their average user to attain using smaller datasets and limited computational power. This emphasis on an extremely detailed weather event forecasting application is IBM's main selling point.

My stakeholder said the main dashboard page was "clear and easy to use". Building around the initial dashboard meant any user could look at all the metrics at once. Retrieving more specific information is as easy as clicking on a widget and expanding, to show more information. Having just one main menu and sub-pages within this main menu made it straightforward to interact with. Another useful element was reasoning behind conclusions it outputted. Such as when predicting short term crop loss. IBM's system highlighted factors such as higher than average rainfall and low humidity specific to a crops required environment. That meant farmers could easily understand the impacts and mitigate them. Referring to an interview I conducted with my stakeholder he said that "for predicting a specific commodities price. IBM's technology was mediocre at best, but as a simple and detailed agricultural tool it was amazing". Taking forward initial ideas for my solution, IBM's clear and concise approach and ease of use should be reproduced.

Justification of these features :

Focusing on detail means they are limited to smaller ,but more accurate forecasts as they are predicting with extreme accuracy instead of speculation. Whereas our system works with longer annual climate patterns and not day-to-day weather events. Over larger periods of time we can see a clearer trend and how this impacts the commodity market over a larger range of time. Commodity Traders are unlikely to be a main focus of this service. Due to its complex nature such as in analysing multiple constantly updating variables along with a vast number of specific agricultural features. Such as connecting soil moisture sensors in a farmer's field and complex crop yield predictions. IBM's main role is to process lots of data and produce an easily understood dashboard with regular updates. Its focus however is not to look far in advance and predict the trends and how it will have an impact on markets. Looking at Figure 3 IBM provides an array of data with limited usefulness to try and predict the price of a future commodity. We see no indication of how precipitation will have a negative impact on the crops yield and its price internationally.Covering only a small region we can only guess as to the wider impact to the market.

Data is updated to the minute ,the image shows live impact of climate events on an area something which a smaller dataset and a lack of computational power is unable to do.

Ultimately, IBM focuses on quality over quantity.

Similar approached I would Implement :

IBM's system has a clear user interface to convey information effectively. Using a simple dashboard with summaries of all current weather metrics. Figure 3 emphasised this as similar information was grouped together and only the final conclusion was outputted.

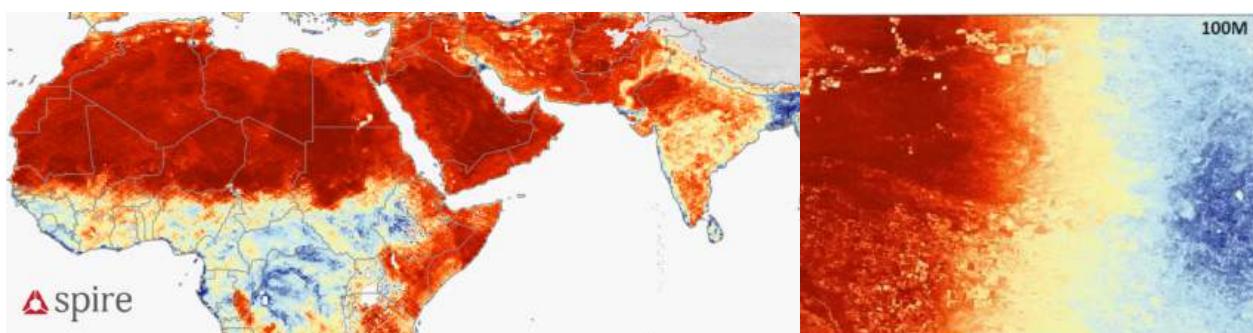


Using boxed widgets to effectively present information made outputting so much information easy to interact with. Stakeholder's didn't like the amount of information IBM output so in my tool I refined this by dedicating the majority of the space to a current and historical price chart for the agricultural commodity as seen in a mock-up design above. With the prediction being clearly overlaid on top of this. The rest of the space is clearly

organised into widget tiles and will contain other data that the user requires. Ultimately the above design a framework for how the user will interact with the piece of software after all the data has been compiled and processed.

Spire Global

Image below shows soil moisture over a small region and over continents.



Identifying features :

Spire global is a data analytics company that is specialised in using space. They have over 100 satellites orbiting the earth. Spire Global recently launched a space-driven high-resolution weather forecast for commodity traders. Spire Global is another leader for forecasting commodity markets. Providing weather insights more in line with agricultural commodity traders use-case. Their focus relies on using low orbit satellites to make well informed decisions about agricultural commodities with aerial data and live climate data. 'Spire's High-Resolution Weather Forecast equips commodity traders with crucial insights needed to anticipate the weather's influence on the market. Spire like IBM focuses on precise insights with extreme accuracy. This is paired with the fact they have access to a vast array of data from satellites that cover everywhere in the globe with each point being analysed to 1km blocks around every point across the globe. Unlike IBM, Spire's use case is focused on commodity traders, farmers and average people. Similar to our products target end-users..

Descriptions of these approaches :

Instead of covering smaller regions like IBM, Spire has access to satellites that map the entire globe. All this means is that users can utilise Spire's global coverage to analyse huge areas of land and extract crop performance. Satellite data is updated regularly and allows for metrics such as soil moisture and image data to be used for analysis. Specifically this tool is designed for agricultural commodity traders so customization and specific needs are easily requested. So you're able to monitor multiple crop's performance with ease. They use their own proprietary satellite data along with public meteorological data from organisations on the ground operated by NOAA, EUMETSAT and ECMWF. All these data points provide a clear picture for Spire customers and for the accuracy of this tool. The use of high quality up to date data that covers the globe allows them to forecast and predict weather events with extreme accuracy. Satellite information furthermore is expensive to obtain, but at the scale Spire is operating they are able to offer this service affordably. Stakeholders tested a free-trial of their service and said the application they provided was very specific and Spire thoroughly processed and outputted the data. There was no need to further analyse any of the data as the final product produced was a simple prediction of the future price of the agricultural commodity.



Spire Global as a data analysis firm clearly values precise and reliable data. Understanding this we see Spire's limited reliance on algorithmic analysis for forecasting tasks instead using an expert team of meteorologists and scientists. All this means is that data has been processed and clarified appropriately so you can easily understand steps forward. In figure 4 we see a soil moisture report showing differences across a region. Spire has the ability to zoom in using their satellites into a 100m x 100m area. This specific application can help differentiate the impact on a crop effectively as you can measure exactly how negatively impacted an area is. All this means is that you can gauge the productivity of a crop over a wide region without the need to individually process weather data. Spire's global coverage allows their users to easily access huge areas of land and to analyse and extract

information easily, something IBM and other providers lack the capability to do. Stakeholder's said however a potential downside to Spire's solution is their limit of 6 days to forecast prices. This may be because they're operating over a shorter period and don't take into account yearly trends and instead focus on accuracy in the short term. Their end user seems to be more in line with our product however their focus on extreme preciseness is the main differentiator.

Spire has a reliance on large datasets with a lot of metrics. From proprietary satellite data to public meteorological data sets them apart from the competition. As when making long term predictions with large amounts of data it is just not feasible while also trying to make accurate predictions. Similar to IBM, Spire believes in the same short-term forecasting philosophy.

While researching Spire's system it was evident it was a corporate tool and not advertised to the average person. However the ease of use of this application was clear as its focus was around producing insight and not outputting huge amounts of data. Spire's strategic use of human experts to verify data meant a climates impact on the commodity market was clearly understood. Spire's other tools include supply chain tracking and analysis for other commodities such as energy and oil. Making them specialised in this field. A comparison of this tool with IBM's showed how Spire's use case was focused on commodity traders and farmers instead of a broader agricultural audience. This system was more in line with our solution's main premise. To forecast agricultural commodities using climate data. On one hand Spire's solution used humans to produce the impacts of the weather on the price of a commodity whereas our solution used algorithms to do the same. This meant Spire's time range for how far in the future it can reliably predict drops as it uses a large amount of data. Our use case however, requires running natively on a less powerful computer so we use more specific weather metrics. Such as Cloud cover, humidity, precipitation and temperature. Allowing our solution to forecast further into the future as we use less data points and with greater accuracy as we are looking independently at the impact on the agricultural commodity.

Justification of these features :

Spire's user interface reflected its specialist nature. It included a suite of advanced features and customization of parameters such as climate sensitivity and advanced graphing of weather across large regions. Stakeholders that had experience with Spire said it was a complex tool and largely focused on corporate users such as a commodity trader. Making it unaffordable and therefore inaccessible to the average person. Furthermore its complexity meant there was a steep learning curve to understanding and using the product effectively. Looking into Spire's system we can see they support a range of advanced tools from third-party program connection into their service and indicators such as a moving average and other technical analysis tools. This shows Spire's persistence with providing the user all the data

possible. This makes it infeasible to try and hide information so a focus on detail makes the solution seem somewhat bloated ,but usable for experts who need to access advanced features easily.

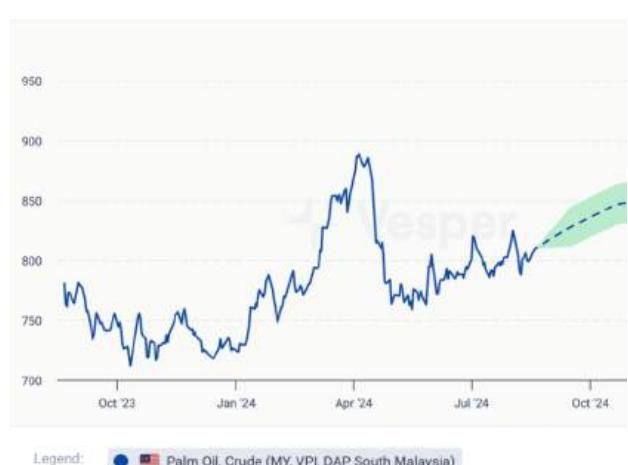
Computational Power seems to be Spire's main bottleneck as with so much data it is just too time consuming and complex to make future commodity forecasts reliably. Longer term trends are often too complex and to make precise predictions over the long term is just not possible. Whereas our system uses a smaller amount of specific data and specified to particular regions. Furthermore our solution uses algorithmic analysis to determine trends between climate and commodity data which can be done efficiently using computers. However Spire itself like IBM acts as a information broker collating specific weather data with deeper analysis and understanding relied on to the commodity trader themself. The data shows trends about what could happen ,but how this impacts specific agricultural commodities prices is left to the commodity trader to piece together manually. This still means that while better than IBM's system for the average person it still remains both prohibitively expensive and complex.

Similar approached I would Implement :

Stakeholder's wanted to see long-term predictions. Our tool, unlike IBM's and Spire's, would want to make similar accurate predictions with a large amount of data in the background ,but over a longer term. Focusing on months instead of weekly predictions as the price is unlikely going to be impacted as much in the short-term and instead over the long-term. This would mean we would have to focus entirely on an algorithmic setup in contrast to Spire's which used both humans and algorithms for predictions. Due to the sheer amount of data they are processing, a human element makes them fundamentally flawed as humans are unable to make balanced decisions quickly, compared to a purely statistical and algorithmic approach.

Vesper

Image below shows a Palm Oil commodity forecast in a Malaysian Market.



Identifying features :

Vesper is a data forecasting tool. Vesper uses both quantitative and qualitative analysis techniques. This means they use quantitative tools such as algorithms and statistical models to predict future trends. Specifically Vesper has partnered with CropGPT a firm that uses meteorological data and demand data to predict future yield and the agricultural commodity's price

they independently use similar statistical techniques to analyse data. Alongside this, qualitative methods such as supply chain and economic analysis of a region are undertaken. Vesper gathers all this data from CropGPT and external sources to inform a forecasted trend. Stakeholders said they were amazed at Vesper's 'easily understood interface' and clear focus on producing an easily understood forecast. However this meant that detail was often neglected or misinterpreted. Complex tools such as customization of how data was analysed and definitions for why the price would change was not included. Vesper acted as a simple indicator which anybody has access to and can use easily. Looking at figure 5 we can see Vesper's forecast for the Palm Oil market in South Malaysia. Vesper is simple as it takes into account all the background data and shows clients a simple trend. We can see the historic prices indicated by the solid blue line and the future projection indicated by the dashed blue line.

Descriptions of these approaches :

Vesper's simple nature allowed us to remove the need to take supply and demand into account as the tool reacts to market conditions from the input and outputs a single answer. A simple up and down trend proved to forecast accuracy. Perhaps a feature of Vesper's infrastructure focusing on a wide array of data allows it to take into account lots of factors and its impact. Compared to the computational and time limitations to what an average trader is capable of. This is because Vesper's main selling point is its accuracy and this comes with a rigorous analysis of multiple data points. From changes in the weather to smaller variables such as changes in demand and geopolitics. All this forms a tool that is unbiased and accurate.

Justification of these features :

Comparing Vesper with our tool we can see Vesper's requirement for a wide range of different pieces of data such as the weather, supply chain stability and also the economic security of regions. Over the short-term Vesper's solutions seem to be extremely accurate as they can make price estimates quickly on the impact in the near future. However over the long-term the lack of detailed data and multiple variables, Vesper's main selling point becomes its crux as it's unable to make long-term predictions. This is because using lots of different independent factors makes predicting with accuracy become almost random as contradicting variables and a difference in impact skew the results. This is reflected in Vesper's dependency in forecasting in the short term and limited support for long-term solutions.

Realistically our tool will outperform this as we focus on the main underlying variable the weather. This means that even though there is an impact from external variables such as demand and geopolitics. These exist as random fluctuations which indeed do play an impact but a much smaller one in comparison to any major changes in the crop's yield and therefore the weather.

Looking at the User Interface of Vesper we see the same message conveyed. Vesper as a simple forecasting tool. It's clear with the main historical market price and a trend line indicating future market performance. Their product is available as a web-based application and a mobile application. Stakeholder's tested their products and liked the clear structured approach of their interface. Information was separated within 3 headings. Market outlook, current-news and updates.

Market outlooks provide future projections of the market and is the first page the user sees. Current-news provides a general overview of your specific commodities and updates provide real-time impacts of events so you know how and why the market may be reacting that way. As illustrated beforehand stakeholders really enjoyed the clear structure of the user interface. As it was easy to navigate and understand at the basic level what is occurring in the market. However for advanced users Vesper's system is insufficient as it has a one size fits all approach.

Vesper's output is a simple trend like our solution. However Vesper in this case takes into account so many factors that require both algorithms and human analysis. Simplifying supply and demand to try to get insight about the final relative price is difficult as so many things may impact both seemingly randomly. Vespers main focus is to provide a general overview based upon a large quantity of different data from lots of sources. So trader's using this tool can believe that they aren't missing any information as even seemingly irrelevant information has been taken into account.

Similar approached I would Implement :



Vesper focuses on a clear UI. On all applications that stakeholders tested they said they appreciated a design that is modular. Implementing a modular design by using smaller widgets as mentioned in Spire's system simplified the user experience providing a clear starting point for user's to navigate around the application. Vespers specific

implementation of separating information into 3 distinct headings. Market outlook, current-news and updates. Meant user's knew how to get around the program from the start page. Alongside that the consistency of the user interface between devices was also highlighted by stakeholders as a positive as it meant there wasn't a steep learning curve to interact with the product such as when using a commercial application like Spire's solution.

Stakeholder Feedback:

Stakeholder	Feedback
Mrs Ganeshgudi	<p>"After interacting with Vesper's platform I found they're entire system easy to handle and use. The main components within the solution were clearly placed and I could easily navigate across the entire solution. This system was well designed and effectively embedded the predictions within the graphs of historical prices."</p> <p>Vesper's platform was easy to use and effectively developed a seamless experience. The main advantage of vesper was its clear focus on providing accurate predictions clearly overlaid on top of the relevant graph. Its interface and accessibility meant it was easy to use and produce results. It is clear that it was made for users with all levels of experience.</p>
Mr Shreeharsh	<p>"I also trialed Vesper and found the experience mixed. The main feature was the overall accuracy of the tool, as it was able to sustainably predict agricultural commodities with a high accuracy. However one potential issue was the lack of detailed explanations. Instead all the analysis seemed to be happening in the background and only a single prediction was being produced which didn't show what actually happened behind the scenes."</p> <p>Vesper aimed to be a tool that correctly performed in all categories. This meant that it had to sacrifice detail in order to be accessible to a wider range of traders. However they're underlying accuracy as suggested by Shreeharsh was high due to their emphasis on quantitative and qualitative analysis paired with research organisations such as CropGPT. All of this made Vesper's tool specialised in their field of producing simple predictions of a range of commodities.</p>
Mr Manu	<p>"IBM's environmental suite was a solution I looked into and hoped to gain detailed insight from. Detailed weather analysis paired with extremely detailed forecasts proved useful as a tool to predict weather with extreme accuracy. In this case added features included a focus on tracking the health of crops online. As an existing solution this was useful in getting accurate insights into a crop's growth, but had a limited amount of integration for both long term predictions and the respective commodities price."</p> <p>This tool was extremely useful due to its focus on both detail and accuracy. IBM's main purpose with this tool was to provide a broker of high quality weather information to farmers which it successfully did. However the main issues arose from the lack of agricultural commodity information and limited long-term forecasts. Ultimately IBMs main focus seems to be about creating an easy to use tool to forecast the weather and the</p>

	<p>yield of the crop with limited suitability in forecasting agricultural commodities.</p>
Mr Nagendran	<p>“Spire’s global coverage and advanced predictions suit my workflow exactly and is exactly the tool I would try to implement and aim for. They have access to lots of data due to their proprietary satellites stationed across the globe which collect lots of advanced weather metrics across continents paired with a focus on accuracy. This means that Spire Global is a leader in their field by giving users sometimes advanced tools which can generate suitably advanced predictions.”</p> <p>Mr Nagendran trialed Spire after wanting to look into a solution that allowed advanced users to access advanced results. His main points that he took away was the complexity behind a program which is what commercial users would want. Our solution looked at implementing Spire’s system differently by using longer term predictions instead of Spire’s inherently limited selection of a weekly prediction. By doing so our solution can cater to commercial users as well who want access to similar tools to predict the price of agricultural commodities.</p>

Comparative evaluation of existing solutions:

IBM	IBM focuses on building a short-term climate forecasting service for mainly farmers. This allows them to implement an extremely local and small scale system focusing on catering to farmers. Ultimately this limits the overall value that can be provided to agricultural commodity traders as IBM cannot handle detailed analysis over large regions such as the ones our system is covering. Along with this there is a limited ability to link it into making market predictions as their system is primarily focused on making low-level crop-yield predictions instead of direct commodity market predictions.
Spire	Spire is a highly specialised commercial application that leverages their own high-quality satellite data to make short-term predictions along with direct short-term predictions about the agricultural commodity market. In comparison to IBM this is because of their ability to cover and analyse a large region instead of focusing on a small scale area. However this comes with limitations as to the quantity of data they can analyse as a large area has a respective large volume of data produced. This means that they focus on extremely short-term predictions about the commodity market looking at time ranges of 6-days.
Vesper	Vesper utilised techniques directly from both IBM and Spire focussing on both extreme accuracy and looking at short term predictions. Additionally their tool also looks at long-term predictions taking into account both algorithmic predictions along with human analysis. Vesper

aims to create a tool that takes a holistic view to making agricultural commodity market predictions. Taking into account multiple separate data points and how they all interact for instance geopolitics and algorithmic predictions. In comparison to IBM and Spire which focus on a quantitative data-based approach, Vesper takes into account multiple variables.

Specialisation seems to be the main outlook for these products. Each product specialised into a specific subfield. Catering to different types of users. They all have similar end-goals but approach it differently. This may be due to their need to differentiate themselves from each other. Dedicating all their resources into providing specific accurate insights. IBM specialises in direct agricultural insights using an array of data sources. Their platform is built from the ground up for farmers instead of commodity traders. Stakeholders acknowledged this ,but stated the information had limited insight as it forecasted with high detail in the short term and limited data points over the long term. Furthermore it had no third-party plugin support. So there was no way to correlate agricultural-commodity data and climate data. However crop yield features were provided so the effectiveness of this tool was based upon the end-users direct needs. Traders could use the tool ,but it seemed almost as an afterthought instead of a headlining feature. Spire Global is a commercial tool that is extremely specialised in this field. Using satellites covering continents to make extremely accurate short-term predictions. Using human expertise and a very specific target audience Spire is able to monopolise this market and directly focus on the needs of advanced users. In contrast to what we hope our tool to be we want to focus on long-term predictions and an easier to use interface that doesn't have as steep of a learning curve. Vesper prioritises a wide tool that is simple and accessible to all users. It implements both quantitative and qualitative analysis using both algorithms and human experts respectively. It excels however in short-term forecasting which meets and accommodates the needs of a range of users.

Features of my Solution

Identify, describe and explain essential features of the proposed computational solution.

Stakeholder Questions :

- Question: What are the main features you expect from a market forecasting software?

Poornima said ‘Simplicity in the interface I don’t want to be overwhelmed with information. This was because overloading a user with information meant that they were unable to ’

Stakeholders want an easy to use system that doesn’t overwhelm the user. Much of the complexities such therefore happen hidden in the backend and only information necessary for critical decisions should be shown

Answer: ‘I want to be able to customise variables and be able to be in control of what is happening particularly when predictions are being made directly from the data.’ Stakeholders want a system that is able to cater to both advanced users and basic users. Perhaps the use of hidden advanced menus can give control to users that want it. These menus could allow changing the sensitivity of climate variables and the sensitivity of the climates impact of a market.

- Question : What deem unnecessary

Answer : ‘The more data too much things on my confused.’

Stakeholders want information they want to Everything else in the friction to producing the prediction. Only has to take to produce the final prediction should be focused on. Automation of steps was suggested to help minimise the amount of information the program requires from the user. This includes taking steps such as retrieving weather and commodity price data from external APIs automatically to processing it and automatically training the model.



are features you and why?

the better right, but screen and I'll get

clarity in the see the trend. program just adds final output decisions the user

- Question : What differences in the graphical user interface for historical and current commodity prices should be made when compared to existing products such as Trading212.

Trading212 is a commodity trading application which graphically shows stock performance. These commodity trading applications allow users to easily manage, buy and sell commodities or stocks.

Image below shows an example screenshot of how Trading212 graphs current and historical market prices.

Answer : 'Stocks are in essence just numbers. So graphing it differently wouldn't make much of a difference. However similar applications give users the option to change the time frame. Looking at Trading212's UI, all users would want to quickly see different parts of the graph without having to zoom or pan . So giving users the ability to select between a weekly, monthly and yearly view would be helpful.'

Stakeholders want similar tools to that of similar trading apps and how they graph their predictions. Features that also should be included are the ability to quickly switch between different time periods. Furthermore the graphs of the historical and current price with trend data suitably overlaid on top of it.

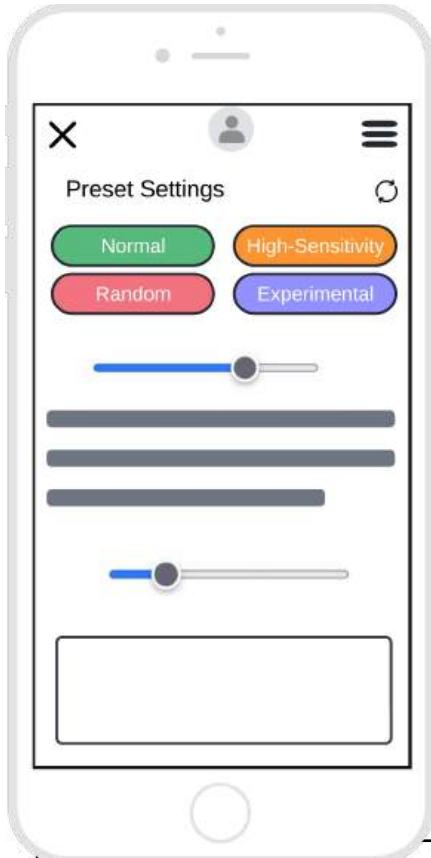
To propose this to my stakeholders I refined the above graphing interface into a potential mobile application. These mock-up designs contained a potential output page where the data and the trend is shown and another page which had a potential advanced settings page which was requested to cater to power-users of our tool. After an interview with Mrs Venn who describes herself as a beginner commodity trader. She understood what the graph represented and the clear trend that is overlaid on top of it. Improvements she suggested however was just overlaying the trend line and removing the colours from the graph. This would make it more simple and easily understood as there would be less information on the screen.

A clear consensus was made after talking with Mr Malakannavar who said he does not want an extremely complex outputted graph as it may detract from the applications suitability to a range of different users. His only requirement was the ability to finetune the model that predicts the future commodities price. He describes himself as an advanced user and advanced settings would directly give users, if required, more granular control over the system and training of the model. The mock-up designs he said ticked all his criteria for a feature that should be added he also added that the design was extremely touch friendly as he could easily navigate and use the application on his phone. Including preset settings was also another idea as it allowed users to quickly change settings without needing to understand each specific element on the page.

Sharath said 'Amazing work, this meets my expectations and requirements. Why not add customization features such as colour and different graphs.' Stakeholders want to build on top of simple features and add ability for complex structures such as other graphs. These features could be hidden within



advanced settings as normal users may want to use only default settings. These complex features will be a piece of our solution however stakeholders understand the need to focus solely on making a piece of software that caters to everyone's needs. Taking that into consideration we plan to add customization options as an advanced feature.



As produced beforehand when we decomposed our problem into 4 distinct steps. Showing how data passes and cascades through each step in the program one by one. All the features of our solution can be shown as a collection of functions within each individual layer or step. The diagram represents how our program functions from a simplified view.

1 Input - Retrieve data from datasets on our device and from our APIs. This process would also include formatting and cleansing the data.

2 Processing - Correlating historical climate and market data to create a suitable model. This section is where hidden patterns are recognised within our datasets.

3 Prediction - Pair the model we just created with future forecasted climate data the output of this model should be the expected future price of the commodity.

4 Output - Visualisation of the predicted trend. The forecast is overlaid on top of an interactive time-series graph. This output step gives the user their desired trend.

We outline in more detail the precise main features that are included in each step and why they occur. These features are inherently broad and will be made up of multiple smaller independent modules when we compile the entire program.

Layer	Feature
Input	During the input layer there are two main features. The user needs to be able to easily input the exact agricultural commodity that they want to analyse and provide a clear start-up page so clients feel welcomed and informed about how to use the product. In the background the system should retrieve data and appropriately format and cleanse it before passing onto the processing stage. Advanced users at this stage may be allowed to go into more detail by accessing an advanced settings page.

Processing	The main feature within this stage is training a specialised model with the data we retrieved in the previous layer. This step is referred to as processing as the model is trained several times on the processing data to help extract insight and decipher hidden patterns between the weather data which contains multiple separate metrics and the crop's commodity price. These correlations will be embedded within the model and then passed directly onto the next step. This layer involves extremely heavy data traversal as it is retrieving and processing data at an extremely fast rate. Complexities such as processing are hidden in the program's backend and the user is unaware about what happens in the backend.
Prediction	This layer receives the predictive model and needs to apply this onto future forecasted data. Forecasted weather data in the short and long-term will be retrieved. This data is then directly inputted into the model to produce a range of outputs which are processed to make a prediction into the long-term of the future price of the desired commodity.
Output	The user is then presented with a clear laid out screen with the historical and current commodity market price data. Alongside the predicted future trend that is based on all processed weather data.

Energy and Precious/Industrial Metals Commodities may be referenced in this paper, but is not a main focus as any climate correlation remains subtle and not a major factor in predicting the price. However as we are building a data analysis tool as suggested by stakeholders customization of inputs remains accessible to advanced users even if the output is unclear. This is unclear because hard commodities such as these will predictably be less impacted by the weather and more impacted by other unpredictable variables.

This tool in summary takes the complexity out of predicting agricultural commodities.

The user is first able to select exactly the precise crop they want to analyse and look into further. Using historical market and climate data to predict the future price. To do this data collection will occur where historical weather datasets that contain the main four metrics are retrieved and the respective commodity prices are also retrieved.

Once all the data is retrieved it can be fed directly and processed by a machine learning model which aims to learn the underlying correlations between

everything as it is trained to identify patterns at correlations as to how the weather impacts the price of an agricultural commodity.

Then after all this takes place the finalised trained model can be used to predict the future change in price of an agricultural commodity. This is done by retrieving forecasted weather data and then inputting it into the model to then produce predictions as to the future price of the agricultural commodity which is delivered to the user.

Across the entire program our main aim is to allow the user to easily interact with our solution to produce the prediction of an agricultural commodity using climate data. With repetitive and fatiguing data formatting, manipulation and processing tasks hidden in the background and giving the users a simple program which they can use to produce long-term predictions for the movement in price of their chosen crops commodity.

Limitations of my Solution

Identified and explained with justification any limitations of the proposed solution.

Our solution is specific to only forecasting the price of an agricultural commodity using patterns in the weather. This means that it doesn't take into account other factors such as geopolitics and other events that can't be quantified within the climate. These climate data points are extremely precise and we have focused our tool to only use the following. Cloud-cover, precipitation, temperature and humidity. These 4 were selected as they had the highest impact on a plant's growth. In comparison other systems such as Vesper which take multiple variables into account such as global-supply and political events. This may mean that it is more accurate to some extent as they are able to reproduce specific future scenarios more accurately such as tariffs on foods which will impact the price. This however has meant they require more resources such as larger computers to store and process data while adding human analysis to this process. All of this means that even though they are more accurate they don't rely on a purely algorithmic approach as we have followed through with. In comparison our tool, unlike Vesper, focuses on maximising the insight that we can gain just by processing and taking into account a smaller range of variables. This means that we can process a larger time-scale of data at potentially the same level of accuracy. Vesper on the other hand focuses on extreme accuracy and detail so a wide array of data is necessary. This has meant that it is unable to make long-term predictions instead focusing on short-term ones. Climate patterns happen over seasonal trends so our tool is at the forefront of uncovering climate impacts specifically. This will mean however we neglect the impact of other factors on the price of an agricultural commodity.

Another limitation may be processing power constraints our solution should be able to handle to provide valid predictions. As our solution is running on consumer hardware like laptops and mobile devices, there is a maximum amount of processing power the system has access to. This is in comparison to large-scale corporations such as IBM who have the capacity to process much more data faster. This may be considered a limitation as our hardware is unable to, to any extent, match the performance of large servers running in parallel. Instead we need to look at the differences in the data we process between the two. IBM particularly dedicates lots of resources to processing data due to the sheer amount of data they have to process as they have countless customers around the world and each with a large amount of data. Our system however doesn't require as much data as we are focused on only 4 key weather metrics. IBM also provides a service to customers so extreme accuracy has to be guaranteed, this means that they are required to have the capacity to deal with large amounts of data while also producing in depth results. Our tool on the other hand produces less data to be processed and doesn't need to be at extreme accuracy this means that while technically being a limitation the exact scenario differs. However user's with specialist hardware will be able to maximise the accuracy of our tool as they can process more data within the given time constraint.

Ultimately our tool may be considered limited, however this means that it is specialised in doing the tasks it requires. For example as described above our tool doesn't take anything other than weather trends into account. This means that it is focused and can master just analysing and producing predictions for weather data. This specialisation means it can be built from the ground up as one niche tool that directly meets its listed success criteria. Another limitation that was described was the lack of processing power on consumer-hardware. This would mean that in comparison to large corporations that provide similar services our prediction's may not be as accurate. We agreed with this however our specialisation into producing solutions that are accurate instead of predictions that are extremely accurate meant our requirements differed. Corporations like IBM have large amounts of resources as they need very accurate future forecasts. This requires them to process past diminishing returns as they need to be certain of accuracy. Our solution on the other hand doesn't require as rigorous training and predicting so results produced just need to be representative of both market conditions and the crop's respective environment. This means we have a certain extended margin of error for our tool.

Minimum Hardware and Software Requirements

Specified and justified the requirements for the solution including (as appropriate) any hardware and software requirements.

Minimum Requirements

Secondary Storage: **5GB** minimum available storage

CPU Requirements: A CPU with a minimum clock speed of **1GHz**.

GPU Requirements: A GPU with a minimum clock speed of **1.5GHz**.

Primary Storage/RAM Requirements: **8GB** total RAM on the device.

Minimum Software Requirement: **Python 3.9.0**

Compatibility

Different devices have a range of different hardware. This means that features may run differently due to hardware bottlenecks. Such as a lack of CPU power which handles data caching and retrieval being a ‘bottleneck’ to the processing speed and therefore throughput. This is because even though the GPU is able to handle data in parallel effectively, the hardware bottleneck of the CPU limits the overall speed and therefore throughput of the application. To combat this our program highlights its adaptability as a key feature. Responsiveness is a main idea conveyed by stakeholders as we want our application to cater to multiple end-users with different intentions. We built on responsiveness by setting a creating a time constraint. The time constraint defines the maximum amount of time the application can use to process all the data. This means that our system will still be usable on low power systems such as a laptop. While also catering to advanced users who have access to the maximum amount of processing power. A time constraint means that the amount of time that the program takes to process data is relatively constant across all devices, but the amount of data that is processed ranges from how much processing power the device at hand has.

GPU

A main component of this program is as a data analysis tool. A key role therefore is in the throughput of data that can be processed. Our solution will need to have the capacity to process larger datasets that include climate and market data. The trends between them will need to be correlated by producing a model. This model is relied upon to make future predictions using future weather forecasts. Past market performance therefore is needed to make future market predictions. These weather and commodity datasets are extremely large and need appropriate computational power to process them and train the respective model. Acknowledging this the Graphical Processing Unit plays a significant role in data analysis. GPU's have a parallel structure with a large number of low power cores. In comparison a CPU has

fewer, but more powerful cores. The average mid-range GPU has several thousands of cores compared to an average CPU with 8-16 cores. Therefore GPUs are specialised on performing simple tasks on large datasets. This means that a GPU plays a key role in handling and coordinating data analysis activities. So a powerful GPU is integral to retaining accuracy when models are compiled in smaller time frames as we will need to process more data in a smaller amount of time. A GPU's throughput of data it can process is directly dependent on its speed. Speed is dependent on the GPU's clock speed. It is measured in Hz (instructions executed per second). A slower GPU will take a longer time to process the same dataset therefore the limited throughput of a slow GPU will add to the time constraints. The time constraint is the amount of time the program takes to output a solution. These time constraints will have to limit the amount of data that can be processed and therefore the accuracy of data that will need to be processed so maximising the amount of data that a given GPU can process per second is necessary.

We also need to take into account that this tool is for the average user so the requirements should be appropriate for a range of computers. To calculate this we have looked at the range of GPU clock speeds of an average laptop's GPU. As most GPUs are integrated within the laptop CPU. This means they are slower and are at the lower end of commercially available GPU, therefore it is extremely feasible and accessible for all users. Most integrated GPUs are able to reach clock speeds of 1.5GHz and above so therefore we have decided that to be the baseline clock speed.

CPU

CPU performance and amount of memory will play a limited role in the practicality of a system when running this application. This is because the program's data processing is directly handled by the GPU which algorithmically analyses the data. However minimum amounts are still listed as to allow both the computer and application to run responsively and not be a resource strain. Data analysis will be handled in Python 3.9 due to its practicality in analysis of big data and its compatibility with machine learning libraries such as TensorFlow and SciKit. Python 3.9 is extremely portable so is able to run on a multitude of systems effectively however processing large datasets brings in time constraints as computation might be infeasible on significantly slow systems. Most modern processors are 2GHz and above. For exact reference this means most users will have an Intel i3 or above or any other similar hardware. CPU Requirements just take into account a single individual cores clock speed, but systems with a CPU with a high clock speed will have added stability and responsiveness. As the CPU is not used extensively within the system and instead tasks that require power are dedicated to the GPU. The CPU's hardware doesn't have as strict of a guideline for a system that works effectively. A CPU with a clock speed of 1GHz is given as any computer with a capable GPU will be able to effectively run both our program and other applications smoothly.

Primary Storage / Random Access Memory

This is to allow both an application to run and the OS and other applications to run effectively. Memory or also called Random Access Memory is a computer's short-term storage; this space temporarily stores data that has to be accessed quickly by the hardware. RAM is extremely fast and quicker than retrieving data from secondary storage or even retrieving data from third-party sources such as through an API. To specify requirements we need to estimate the amount of data the system should need to handle. Assuming we are processing data while also requiring the system and other applications to run smoothly we need to dedicate at least 5GB of RAM solely to the system as windows and other OS use 0-4GB of RAM, while open applications have 1GB of potential RAM. This means we can dedicate 3GB of RAM directly to our program that is running. The amount of data that the system requires in RAM is dependent on the GPUs clock speed as a GPU with a higher clock speed will require more information quickly. A small amount of RAM paired with a fast GPU will create a bottleneck as the system is able to process more data than it receives. The minimum requirement therefore is set at 8GB. Anything under this amount will significantly slow down performance of both the application and your device. Using hardware that is above this requirement is recommended to provide a buffer between your used RAM and the amount of RAM you require.

Python

Python 3.9.0 was released in 2020 and is currently a STABLE build with constant security updates till 2025. Making it a suitable minimum program requirement. This is because our tool is developed in python in Visual Studio Code. Newer versions of Python are also compatible and recommended giving further stability and more security updates. Python is used as its a popular data analysis language with compatibility with multiple systems APIs. These include the APIs that weather and commodity price data is retrieved from and also has compatibility with web scraper API to scrape the current price of an agricultural commodity if needed. Alongside this version of python is compatible with Tensorflow, Keras and Scikit-Learn which are machine learning modules which can help test and develop my machine learning model. The use of python also adds to its portability with use on a range of platforms and OS which is required to keep our solution accessible to all.

Storage

Data will be retrieved from the API processed and then promptly deleted. Data will not be saved and take up hard drive space. So realistically only 1-2GB storage for the program and other saved datasets. Taking into account data that is retrieved we assume we will need an average of 3GB of buffer memory for temporary dataset storage. Reliance on external datasets means that our software package remains relatively small and the only required disk space has necessary program files and databases. This means user's only retrieve the data they need and do not need all

databases stored on the file. Therefore all of this means that our total secondary storage requirement is 5GB.

Stakeholder Feedback:

Stakeholder	Feedback
Mrs Ganeshgudi	<p>“Looking at your project as a data analysis tool it looks like there is a huge amount of data that will need to be processed, how does your requirements accommodate this.”</p> <p>This is true as our main purpose is to both retrieve and process data so having access to large databases is key. We use a dynamic dataset storage solution by using cached data alongside APIs to retrieve the data. This means that we can minimise the amount of data stored within the device so only 5GB is required. Weather data points are collected from all across the world so storing decades of historical weather data would be inefficient. This also means that we need to process a large quantity of data which we do by having a slightly higher GPU requirement which means that we are able to process information in parallel across multiple separate cores.</p>
Mr Manu	<p>“As a farmer, I need a tool that can easily run on a range of devices I have around without having to purchase a new one.”</p> <p>Accessibility is a key component of this solution as we are supporting multiple devices with the full implementation done in python. This means that most personal computers and other devices should be able to run the program without needing extra modules or porting, instead python is widely supported on all devices and can run natively on the device.</p>
Mr Shreeharsh	<p>“I want a tool that can run quickly and easily on my own laptop without crashing. It shouldn’t take up too much storage space and I don’t want to be limited by my own hardware.”</p> <p>The hardware requirements help to not restrict users if they don’t have access to high power hardware and instead can run the entire program on a range of devices. This means that a low CPU clock speed of 1GHz can handle the program. This is mainly due to the planned implementation of a time constraint which means that there is a limited total amount of time for processing to occur</p>

	<p>so the entire program is responsive. This means that high power hardware can process more data within the set time segment and low-power hardware would inevitably process less, but both should come up with a relatively similar solution.</p>
Mr Nagendrann	<p>“Quantitative traders such as myself need to be able to easily run complex financial models and simulations to get accurate predictions. There is a discrepancy between the processing power of a device and the hardware requirements. This in one way makes the entire solution more accessible as anyone is able to run it, but also means accurate predictions may also suffer.”</p> <p>There is a clear trade-off between accessibility and accuracy as discussed beforehand. Our hardware requirements were designed to be as lenient as possible to accommodate a wide range of users and the collection of stakeholders we have represents that. The clear implementation of caching, parallel processing and algorithms should mean that even hardware limited requirements are able to produce a suitable output which is acceptable for a wider range of users.</p>

Ultimately the minimum requirements seek to be as lenient as possible to maximise the amount of potential users that can use this system. As our program is focused on accessibility to users with a range of hardware. This is because we are not processing a lot of data compared to tools that require specialist hardware such as a powerful GPU. There is also a need to build efficiently for everyday hardware. This can be implemented easily due to our tool being specific to finding the link between agricultural commodities and climate patterns.

Specific Success Criteria

Identified and justified measurable success criteria for the proposed solution.

No.	Identified Success Criteria	Explanation and Justification	How is it measured ?
1	A User Interface with clear differentiable operations.	The User Interface should be clearly structured. A minimalist approach should be applied. This means that our end-users should be able to	The user should be able to easily understand graphics on screen and what options they can change. This can be measured by

No.	Identified Success Criteria	Explanation and Justification	How is it measured ?
		navigate across the entire solution without needing extra support or a tutorial to do so.	interviewing stakeholders and asking them to define what is happening on each page.
2	A maximum of 2 decision points on each page. A decision point is a point that the user decides which page to next access.	A decision point is a point within a program in which the user has the option to make a decision. Limiting the amount of interactable options keeps pages concise and retains what is happening on each page.	Each page has decision points. This there will have to be only a maximum of 2 branches to other pages in the program. So there can only be a maximum of 2 interactable elements that link to another page.
3	Clear colour differentiation between interactive objects and different objects such as trend lines and market data.	Making independent pieces of information differentiated using colour allows you to easily distinguish between different functions. Ultimately linking back to our criteria for a Clear UI.	Separate elements should be clearly differentiated on a page. This can be tested by testing stakeholders on the readability of a page.
4	Timescale scaling. Allow the ability to zoom in and pan across the overlaid trend data.	Allow the user to interact with historical and current data. Allowing them to zoom and pan around the graph. Zooming and panning allows users to examine closely all the data.	Adjust and zoom into different cross sections of the graph and confirm it zooms and pans x,y axis automatically
5	Allow for the specific time scale selection of graph data. For instance looking at data across individual days to months and years.	This means users can easily select predetermined pieces of the graph such as year-to-date, monthly and weekly data.	The program should automatically be able to zoom and pan into specific timescales while also having the predictions align correctly over the new time segment.
6	Customizable colour schemes across all the front-end pages. Aiding the accessibility of our system.	Accommodate different colour schemes, to allow for accessibility for conditions like colour blindness.	Clients should be able to customise the colour schemes of the program. A measurable criteria would be a selection of different themes for the graph.

No.	Identified Success Criteria	Explanation and Justification	How is it measured ?
7	Compatibility with different devices of different screen dimensions.	The program should be able to scale correctly to screens that are of different dimensions.	Run the solution on various devices. Including smartphones, tablets and a laptop. Make sure there are no overlapping or missing elements and everything displays correctly.
8	Allow for resizing on the device while you are using the software.	The program should be able to handle resizing and how elements and objects scale on the screen.	While using the piece of software resize it. Make sure there are no overlapping or missing elements and everything displays correctly.
9	Clean data retrieved by the API. Using a function that initially checks for missing or anomalous values and then imputes them with valid values.	Retrieved data may contain errors such as missing values or anomalous results outside of the given range. The program should be able to handle this and still function correctly.	Test the program with a range of data with known errors and anomalies and make sure data is thoroughly cleansed. This means that the program should be able to identify anomalies and errors within a dataset and substitute the value accordingly.
10	Format data retrieved by the API. To successfully pass onto the processing stage.	Retrieved data should be formatted correctly for the next stage where it will be processed. The program should be able to format data effectively and pass it onto the processing stage.	Test the program with data of different formats such as excel files which are in an XML format . Make sure the program is able to handle all files produced by the API effectively and is able to pass it onto the next stage of the program.
11	Keeping API keys secure and not accessible to the user directly. Such as using encryption techniques to store them safely.	The system should implement security for the API keys to prevent misuse. Secure API keys should be stored encrypted so only the program can use it.	Locate API keys and make sure they are not stored in plain-text and are appropriately encrypted so only users with actual access can use them.

No.	Identified Success Criteria	Explanation and Justification	How is it measured ?
12	Using caching to store frequently accessed commodity price datasets on the device locally instead of relying on external sources.	To improve performance we are planning to implement computational methods such as caching. Caching allows us to reduce dependency on the API.	Retrieve historical commodity price data from the stored database instead of from third-party sources such as the API.
13	Split-screen to see two predictions at once simultaneously and being able to interact with both instances separately at the same time.	Comparative analysis describes the process of comparing 2 things side by side. Comparing two different crops over the same time period has been suggested by stakeholders for this system.	Be able to look at two predictions at once. This means that two forecasts have to be seen at once side-to-side.
14	Up-to-date data used, relying on high-quality external datasets such as for current commodity prices and weather data.	Use real-time data such as the current commodities price and weather data. New data will therefore have to be retrieved regularly.	Measured by confirming that datasets are up-to-date and not outdated. Do this by being able to always retrieve datasets from an API or from the internet to always have access to relevant data.
15	Intuitive Navigation across the entire system between pages and when outputting data.	Showing clear indicators of what page you are and actionable steps as to interact with other features.	The majority of stakeholders should be able to describe all the pages in the program and how to navigate between them.
16	Transparency with the user about what data is being used and how it is being processed	The end-user should know exactly what is occurring at each moment in time. Indicate exactly what is happening behind the scenes in the program at all times to the user.	Display the current tasks being executed within the background of the program and the stages of how data is being processed , retrieved and outputted.
17	Scalability by the system being able to handle multiple sizes of data.	The solution should be built to handle processing and compilation of a large volume of data.	The program should be able to run on a wide range of hardware irrespective of processing power. Test the system on devices with

No.	Identified Success Criteria	Explanation and Justification	How is it measured ?
			high and low processing power.
18	Responsive Design across all the pages so the user can interact with elements quickly without the system getting either stuck or taking extremely long to process a task.	Switching between pages should feel extremely responsive and intuitive. Each page should load quickly.	Navigating between pages should occur quickly taking a maximum of 5 seconds. Processing pages on the other hand, when data is being compiled and processed follow a different constraint.
19	Help and Support resources provided to the end-user so they are able to understand how to interact with the software.	An informational page when you open the project that specifies how to use it correctly.	There should be a Frequently asked questions page and information on how to use our product.
20	Efficient use of storage space such as only storing necessary databases and programs.	Optimising the stored database and not storing duplicates or incorrect values to the hard drive.	Use a minimal amount of storage so it doesn't take up more than necessary which may affect the user's other activities.
21	Consistency when processing results. With minimal variation when analysing the same dataset with the same hyperparameter variables.	Trend's should remain reproducible with the same datasets. Processing the same datasets should provide the same trends with a limited amount of variation.	A trend can be categorised into 5 different trends. Neutral, strong positive, positive, strong negative and negative. With neutral indicating the commodities is forecasted to remain the same price. Positive and negative represented by a price rise and fall respectively.
22	The program should be able to highlight values that it deems anomalous. Anomalous values are data points	Passing extremely incorrect information into the model might skew its predictions and affect the overall accuracy of our model.	Manually intercept data and pass extremely high or low values into the system that is outside the normal range and see if the program is able to correctly identify

No.	Identified Success Criteria	Explanation and Justification	How is it measured ?
	outside the normal range.		them.
23	Anomalous values that are flagged should be handled appropriately and fixed. Such as using techniques such as imputation to replace the values with valid substituted values.	Flexibility within database processing so the solution is able to handle anomalous values. This makes our program robust to a range of inputs.	Manually pass anomalous values that are too high or low that are outside the normal range, into the system. Verify the system is able to handle them by either manually correcting the values or discarding them.
24	Flexibility of the program to handle missing data points across commodity price datasets and weather datasets. As externally sourced data may contain missing values.	The database and API may contain missing values due to a range of reasons. These data points come from external sources. Fixing errors means our model receives high quality training-inputs.	Manually remove data points within our datasets to simulate an incomplete dataset. Then we can verify if the program is able to handle missing values without crashing or not being able to train the machine learning model.
25	Ability for the system to handle external errors from the API. Automating debugging processes so the system is resilient to external errors due to the API.	The system should be able to handle a range of errors internally from the external APIs.	The system should be implemented to handle all outputs from the API and not just specific data retrievals.
26	Suitable error messages generated to the end-user in case actions can't be executed.	Error codes and concise descriptions should be given to the end-user. So they understand why something isn't executing correctly.	Errors handled by the system that can't be resolved for instance a retrieval error due to a lack of internet should be shown to the user.
27	Add advanced settings for users that want more control as part of a	Allow users to change specific processing parameters to give access to users who want more granular control.	Add an advanced page that allows users to adjust parameters and other settings.

No.	Identified Success Criteria	Explanation and Justification	How is it measured ?
	custom advanced settings page.		
28	Add a start-up page to open when the user first opens the program.	A start-up page that is the landing page of our tool.	When the user opens the program they should be greeted with a start-up page.
29	The start-up page should convey the meaning of our tool effectively. With labelled pages as to future decision points within the program.	The start-up page introduces our user to the tool and therefore should convey the meaning of what our tool does. Making sure new users can easily understand the function of our tool.	Stakeholder's will help me iterate through the tool's development to verify that the end-user is able to successfully navigate the start-up page without confusion.
30	Simplicity and minimalism is kept throughout all the visuals and graphics in the application. By only outputting necessary data to the user.	Prevents overwhelming users with information. Keeping everything concise also means the meaning of a page is clearly understood.	Measured by verifying that there is not any irrelevant or advanced information on the page unless it is requested. We can do this by asking stakeholders to produce multiple specific predictions using the solution verifying users are able to understand our interface simply and effectively.
31	Multitasking capability to be able to access and predict the future commodities movements for multiple crops at once simultaneously.	Retrieving and visualise data for 2 different crops without needing to close and reopen the application.	Make a stakeholder interact with the program and execute multiple different predictions for crops in succession.
32	Appropriate Indexing of the commodity price database within the locally and externally sourced one.	Optimise storing large databases to speed up traversal. This means it is not a bottleneck for the program and there is longer to process all of the data.	Values stored in a database should be retrieved quickly. Retrieving a range of values should not act as a bottleneck for the entire program by impacting the overall time we have to train

No.	Identified Success Criteria	Explanation and Justification	How is it measured ?
			the model and make appropriate predictions.
33	Interrupting the system while processing should not corrupt the contents of the database. Adding additional file handling anti file corruption measures so data is handled appropriately.	Building reliability into a system is important to make sure the piece of software works in a range of scenarios.	This is tested by forcefully interrupting the program while it is running and making sure it works as expected after it was crashed previously.
34	Process data concurrently when available on the user's system. Such as parallel processing.	Accommodate using hardware with a range of cores. Focusing on processing data in parallel such as when training the machine learning model.	Successfully train the model on a GPU or hardware with multiple cores. Verify that calculations are processed in parallel using multiple separate cores.
35	Provide a wide range of potential commodities that the user can select and analyse. Agricultural commodities range from cocoa, arabica coffee to corn, sugar and wheat.	There is a wide range of available agricultural commodities. The number of agricultural commodities that this tool has access to exactly has not been finalised and will be confirmed when building the model.	Analyse at least 3 different commodities using the solution. In each case the system should retrieve data, process it and output a solution effectively.
36	The model takes seasonal harvest changes into account so isn't a static model it is actively able to improve its previous predictions and model.	Seasonal changes have an impact on crops and agricultural commodities as some crops such as sugar which is grown in the summer and harvested in the winter.	The program should be able to identify and apply seasonal changes to agricultural commodity prices.

No.	Identified Success Criteria	Explanation and Justification	How is it measured ?
37	Continuous refinement of the machine learning model by training it with newly accessible data to make more accurate predictions.	The model should be continually updated and refined. Recompiling models with up-to-date information retains its relevance and accuracy over time.	Models should be checked if they are up-to-date before they are used to make predictions. If not they should be remade to take into account new predictions.
38	Resource optimization so the hardware it is running on locally is fully utilised effectively and optimised to produce the most accurate and fast as possible.	The program should be effectively optimised for extremely powerful systems and low power systems to use all available hardware resources.	The tool should be scalable and able to efficiently use hardware when necessary. This means that all resources should be utilised to prevent idling when processing information. The program should run smoothly on all the stakeholder's devices without excessive lag or unexpected errors.
39	Ethical data processing by following GDPR across the entire software solution.	The user's data should be kept private and secure and not easily accessed.	User identifiers such as specific crops they analysed and settings should be kept private.
40	Transparency with the user about what data is being imported from external sources and how it is being processed.	Data that is fetched from third-party sources such as the API should be clear to the user.	Clearly documenting data retrieval tasks will need to be shown when retrieving current/historical market and climate data and other data that is necessary to be fetched from external sources.
41	Follow data exchange standards when passing exchanging data across the entire software solution.	Follow protocols such as JSON, XML, Pandas and CSV to allow compatibility with different systems.	Follow a widely used data exchange standard and implement that fully into your code.
42	Adhering to API Rate Limits to avoid abusing their	As we are using open-source third party data sources we need to respect their fair-use	To measure this we need to artificially overload our system to make sure it is

No.	Identified Success Criteria	Explanation and Justification	How is it measured ?
	system.	policies.	able to effectively block API requests over the limit.
43	Cross-Platform compatibility with multiple devices such as laptops and mobile devices irrespective of their operating system.	This system should be compatible with a range of operating systems to reach a larger audience.	Successful installation and running of the program on a range of devices such as Windows, MacOS and Linux.
44	Consistent Data sampling frequency to make sure all the datasets contain all the necessary data to generate a model.	The system when processing data should sample weather and commodity data at the same frequency. This means that the same time segment is used across both such as a weekly or yearly access to data.	Confirm both datasets have the same frequency that data is sampled from by looking at the size and dimensions of the sampled datasets and validating they are the same size.
45	Be able to effectively execute operations in the background while the user is not actively using the application.	The system should be able to run efficiently in the background without the need to take up a lot of resources affecting other applications running.	Successfully run the system in the background while actively using the computer for another task.
46	The solution should effectively be able to have a high uptime without crashing or experiencing preventable errors such as not having access to data.	Our program should be able to remain open without crashing while not actively being used.	Test the program by running it consecutively for 2 hours. It should be the only application running.
47	Implement a progress bar to show the progress of any processing activities directly to the user.	Client's want to be reminded and have knowledge on when to expect an output a progress bar allows them to see progress of training/testing the model in real-time and reassured them	When data is being processed on the front-end side of the program clearly show a progress bar so the user can anticipate the amount of time it will take.

No.	Identified Success Criteria	Explanation and Justification	How is it measured ?
48	Build using a modular design using several individual and independent reusable components.	A modular design of using independent functions for the code makes it more readable and maintainable and also allows for things such as concurrency as you can run processes in parallel.	Use independent separate functions for the solution's backend framework. Each separate function should do different tasks.
49	UI of the visualisation of the outputted graph should be consistent with other trading applications.	Keeping a familiar UI for the outputted graph makes the software easier to use and clearly caters to previous traders.	Reproduce similar graphing visualisation outputted by other commodity-trading software. Implementation of the visual cues such as that included in similar programs enhances the clear user interface we aim for.
50	Trend interface should be clearly overlaid on top of historical and present agricultural commodity data.	Trend data should be suitably shown on top of future/historic price data. The entire output of this program is a prediction. Therefore this prediction should be clear and detailed.	The user should be able to see in detail the future prediction of an agricultural commodity over the short and long term. A clear trend line should be shown as a prediction as to the future change in price of the agricultural commodity.
51	Automatic API Data retrieval timeout by being able to retry requesting data in case the API hasn't provided it within an appropriate time frame.	The system should not wait forever for an API response and instead be able to handle the error automatically and re-request the necessary packets.	Artificially cause the API to timeout and allow the system to automatically resolve the error by requesting the retransmission of packets.
52	Directly output errors that occur to the user in a clear format.	An error should be clearly outputted and presented to the user in a correct format.	Manually add errors and see how the program outputs the respective impact to the user. This is to test the functionality of an error request system to make sure it is able to clearly display errors directly to the user.

No.	Identified Success Criteria	Explanation and Justification	How is it measured ?
53	Data should be retrieved from the API and passed onto the model within 10 seconds.	APIs data should be processed quickly. This is so the program can maximise time processing the data to gather insight.	The time between when the client requests a prediction and when all the data is passed onto the model from the initial API call should be a maximum of 10 seconds.
54	The prediction should be produced within 60 sections of being executed. This includes time to retrieve and compile data.	Software retains its responsiveness and the user doesn't have to wait too long for the requested trend data. A longer time to output data would interfere with the user's experience for a marginal improvement in accuracy.	Measuring the time between when the client requests a prediction and when the program actually produces it. It should be a maximum of 60 seconds and can be measured using a unit test.
55	Automate the number of epochs the model is trained over. A single epoch is one training cycle over the training data.	The system should be able to adjust to a range of inputs and be able to adjust the amount of epochs it should train under. Under the time constraints processing should take approximately 50 seconds.	Test and minimise the difference between the predicted processing time and the actual processing time it takes for the model to be trained.
56	Display the time-period graphs for the price of an agricultural commodity and accurately.	Commodity graphs with different past and future time-scales should be appropriately visualised on the final graph.	Measure using how different ranges of commodity data is auto-scaled correctly onto a graph. It is appropriate if it scales and zooms in automatically correctly.
57	Retrieve external agricultural commodity price data and climate data using an API successfully.	External data from the API should be retrieved from the internet and passed onto the next stage of the program.	Verify this by testing 3 different agricultural commodities and if weather and current price data can be retrieved successfully.
58	Compatibility of both weather and commodity price data to be used in conjunction with each other to produce a model	Weather data and commodity should be of compatible formats to be used as data points when training the model.	The model should be able to input and then correlate both weather and price correctly without any errors that are due to the data exchange protocol of the original datasets.

No.	Identified Success Criteria	Explanation and Justification	How is it measured ?
	that can predict the future prices of agricultural commodities.		
59	Validate data types and the format of data before it is passed into the model	Training the model with invalid data will negatively impact the accuracy	Confirm data format and data type is automatically validated before it is used for training the model. Test by manually adding invalid data and see if it is flagged.
60	Create a set of pages that all users will interact with that contain all the essential functionality of the software solution.	Novice users require a simplified user interface that has clear actionable activities. In this case we may include simplified pages that only have required elements.	Implement a “normal” set of pages which cater directly to all users and stakeholders. In this case we will include only necessary functions and features.
61	Add different pages for different types of users such as advanced users who want additional functionality.	Specialising pages allows all users to effectively use our system and not be overwhelmed with the amount of information as the relevant type of end-user is accessing the relevant page.	The program should have a range of different types of pages for different types of users. This means we can implement pages with more or less information such as an advanced settings page.
62	Automatically validate all the data that is passed into the model. Through using validation checkpoints at each stage when data is exchanged.	Training the model with data with different ranges may skew the results and confuse the model making it produce valid ,but incorrect predictions. So double checking the data's validity with a different function with different parameters makes sure it is correct.	Confirm all data has the same range before it is directly sent into training the model. Test by manually adding and sending data with an invalid range and see if it is flagged.

In summary, the success criteria was centred around expanding on the specific tasks that need to be achieved to reliably and accurately forecast agricultural commodities using climate data. Each piece was identified which involved looking at what stakeholder's wanted from the research stage and necessary steps that had to be

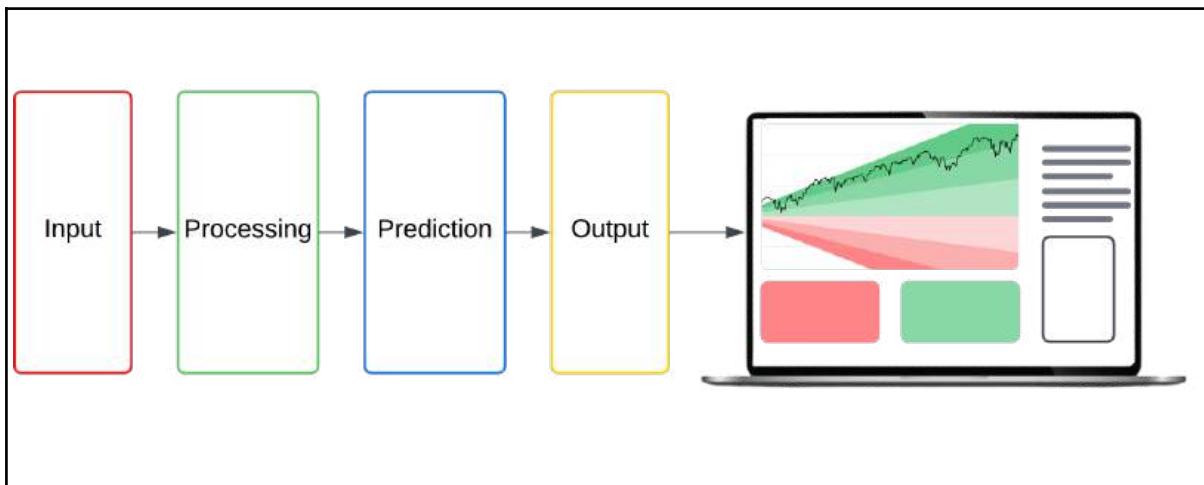
taken in order for the program to function. Additionally each success criteria is justified and explained as to why it should be implemented. Finally we also had to focus on how it is measured to gauge if the solution to a problem worked. The ultimate goal we are trying to achieve is an effective tool, all of these steps are stepping stones to reach our final destination. By following the criteria and implementing it correctly we can deliver on what the project set out to do.

Design

Internal Structure of my Solution

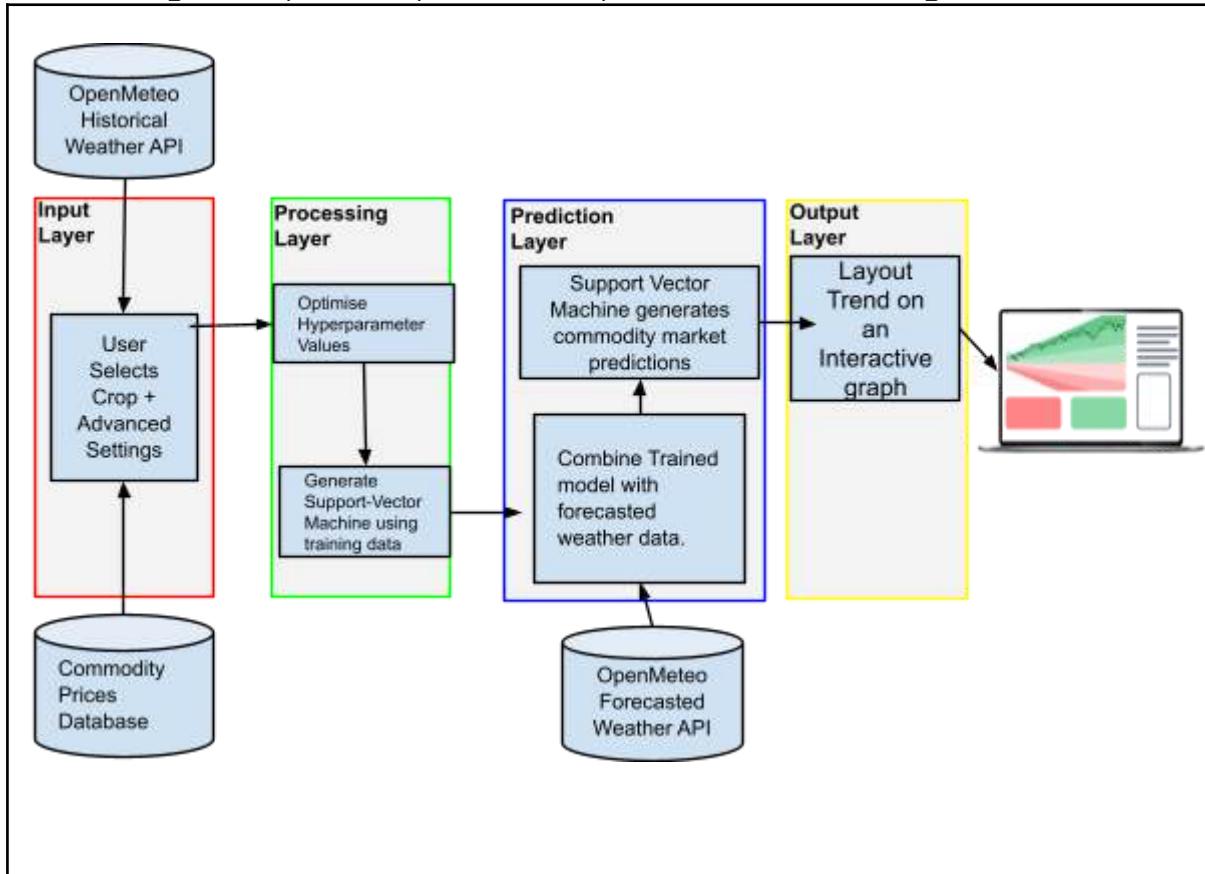
Broken the problem down systematically into a series of smaller problems suitable for computational solutions, explaining and justifying the process.
Defined in detail the structure of the solution to be developed.

Below is a simplified representation of my program. As part of a modular framework I have split each process into 4 separate stages. At each stage there are clear inputs and outputs.



Adding more detail to each stage we can see how data and what processes happen

at each stage and specific inputs and outputs occur at each stage.



Initially we have decomposed our solution into 4 separate layers. This decomposition highlights the flow of data throughout the entire system. Data flows directly from the start to the finish of the program interacting with each layer or step until it is all complete. This means there is a clear sequence or procedure within the framework of the program. Each layer is executed one-by-one, to produce the final solution as we can see at the end of the program. For reference we have included the definitions of what each layer aims to do and a visualisation of the series of steps.

1 Input - Retrieve data from datasets on our device and from our APIs. This process would also include formatting and cleansing the data.

2 Processing - Correlating historical climate and market data to create a suitable model. This section is where the patterns are recognised hidden inside our datasets.

3 Prediction - Pair the model we just created with future forecasted climate data the output of this model should be the expected future price of the commodity.

4 Output - Visualisation of the predicted trend. The forecast is overlaid on top of an interactive time-series graph. This output step gives the user their desired trend.

Why do we use separate layers ? :

To show the actual structure of our program we need to look individually at the structure of each individual layer. To do this we have structure diagrams for each layer. Each independent layer will have an input it receives, processes and then outputs. This means that each layer has clearly defined criteria for what should happen with the data. Using a purely systematic approach helps to easily accommodate building the system using smaller independent modules and also rapidly refining parts of the program without affecting other layers. As each layer is entirely separate, editing functions within a layer will only impact the layer it is contained in. As seen in the diagram below we plan to implement encapsulation within separate layers. Each layer is executed in a chronological order until the final output is produced. At the input stage of the layer databases and other data points can be accessed inside the layer.

Debugging specific functions within a layer is easier than testing the entire program. This means you can test each function without affecting the rest of the program. Having predefined inputs and outputs makes developing a solution easier as it is clear as to what occurs in each section of the code. Overall this makes the code more maintainable as you can easily maintain and update specific sections. This methodology reflects Object-Oriented Programming (OOP) as your focusing on organising the program into separate “objects” that perform different behaviours.

The diagram below shows the entire planned internal structure within each layer and how they are incorporated as a sequence of layers within the whole program. In the diagram below you can see the flow of data through an individual layer from the starting input block to where data is outputted, behind the output blocks. This is done to show how data is contained within each layer which is called encapsulation. Encapsulation within each layer means there is a controlled access to elements and pieces of information that the program can retrieve and use. This means that only data within each layer can be used and outside functions that haven't passed through the input stage can't be retrieved. Similarly to show the passing of data through each layer, preceding layers outputs can be inputted into the following layer.

Evidence showing a systematic approach decomposing the input layer :

Independent Action	Explanation	Justification
Prompting the user for their specific crop selection. The program asks the user specifically to select the agricultural	This is a crucial step as the program can only process one crop at a time effectively. So the retrieval and processing	By starting with crop selection, the system guarantees all the subsequent actions that occur are directly

Independent Action	Explanation	Justification
commodity they want to analyse.	of the user's choice will be based on this crop selection.	relevant to their needs. This means that the entire program is kept efficient as unnecessary data is not retrieved or processed.
Retrieval of information specific to that crop. This step occurs after the crop is selected and gets the coordinates of regions where produce is grown and the quantity they grow it at.	This action is important as understanding the geographical distribution of the crops production means we can pinpoint the exact impact of the weather on the crop. Alongside this we can estimate the relative importance of regions in the crops commodity markets.	Knowing both the area where crops are grown and the production they are grown at allows for our program to focus on relevant data points for example the weather data and the importance of each region. Instead of having to use more processing power to discern relationships through convoluted data.
Retrieving and updating commodity price data for the crops agricultural commodity guarantees up-to-date data and data points are not missing.	Commodity prices are used as a main input for the forecasting model. As the model uses historical and current price data to train the model, relevant data is necessary.	Caching also significantly improves performance by reducing the need to repeatedly fetch the same data from an API or external third party sources. Which would require more steps to guarantee the data is actually usable to train the model. This is because we will need to also validate and format data appropriately.
Merging Datasets ; We merge datasets as we are taking in data from multiple separate locations so we create a single comprehensive weather dataset.	A single representation of the crops' weather across all the major regions as there is only a single file that needs to be processed by the model.	A single file that contains all the data means that there is consistency within the training of the model as all the regions are taken into account at once.
Format and Validating is performed to make sure the file is usable and can	Data formatting ensures compatibility with the machine learning model	Clean and well formatted data is a necessity when we are building a data

Independent Action	Explanation	Justification
handle missing values or anomalies.	and validation techniques retains the quality of data and prevents errors.	analysis tool that needs to be both accurate and reliable at forecasting data.
Advanced settings are part of our solution as we allow users to access and modify advanced settings (hyperparameters) that directly control how the machine learning model works.	This provides an added level of flexibility for advanced users who want more control over the model's training process.	This means that we can easily cater to a wide array of users which are also represented through our selection of stakeholders. This means that both beginners and experts can get insight from our tool.
Tutorials and prompts are provided to help guide the user throughout the entire experience so they can use our tool effectively.	Users can understand the precise functionality of our tool if they are ever unsure what is either happening in the background or how to navigate / use our system.	Clear tutorials and prompts improve the overall user experience and as before reinforce our dedication to providing for a wider audience.
The input layer outputs the following objects. Formatted and validated weather and commodity price dataset and also advanced user settings stored appropriately in a file.	These outputs are inputs for the next layer (processing layer) within our entire system.	By clearly defining what our layers' outputs will be we can ensure a smooth transition when we decide to process data in the following layer.

Evidence showing a systematic approach decomposing the processing layer :

Independent Action	Explanation	Justification
Initializing the datasets. These are the weather and commodity price datasets.	We prepare weather and commodity price datasets for processing by ensuring it has a suitable format and it is able to be used by the machine learning model.	This sets the stage for efficient data handling and prevents us from hitting compatibility issues within the model while training. This means that data can be used effectively in the subsequent training

Independent Action	Explanation	Justification
		process.
Implementing suitable error handling	Creating an error handling function means we can quickly catch and manage errors during data processing and other stages within the whole system.	This means that the entire system is kept stable and users have a positive experience. As the system is able to handle unexpected errors while also informing the user so they can fix them.
Scaling data	Standardise the data, so the model can receive data in a suitable format.	Scaling the data helps to improve the training speed and overall accuracy of the model.
Validating data	Verifying that data meets the models requirements before training occurs prevents errors during training.	This adds an extra layer of protection against incorrect or incompatible datasets.
Training the machine learning model	Training the machine learning model involves using a suitable machine learning algorithm to analyze and correlate the weather and price data.	This is the main component within the processing layer as the system is learning the relationship between weather patterns and commodity prices.
Implementing a progress bar	A progress bar will give users real-time feedback as to the progress of tasks being executed.	Progress indicators will help keep transparency to the user as to background tasks that are occurring. Which means during computationally intensive tasks such as the processing stage the user can expect how long is required to process all the data.
Output of the processing layer	The final output of this layer should be a fully trained and optimised machine learning model capable of classifying all	The trained model is the main output for the processing layer and will be applied in the following prediction layer.

Independent Action	Explanation	Justification
	weather patterns and predicting the impact on the crops commodities price.	

Evidence showing a systematic approach decomposing the prediction layer :

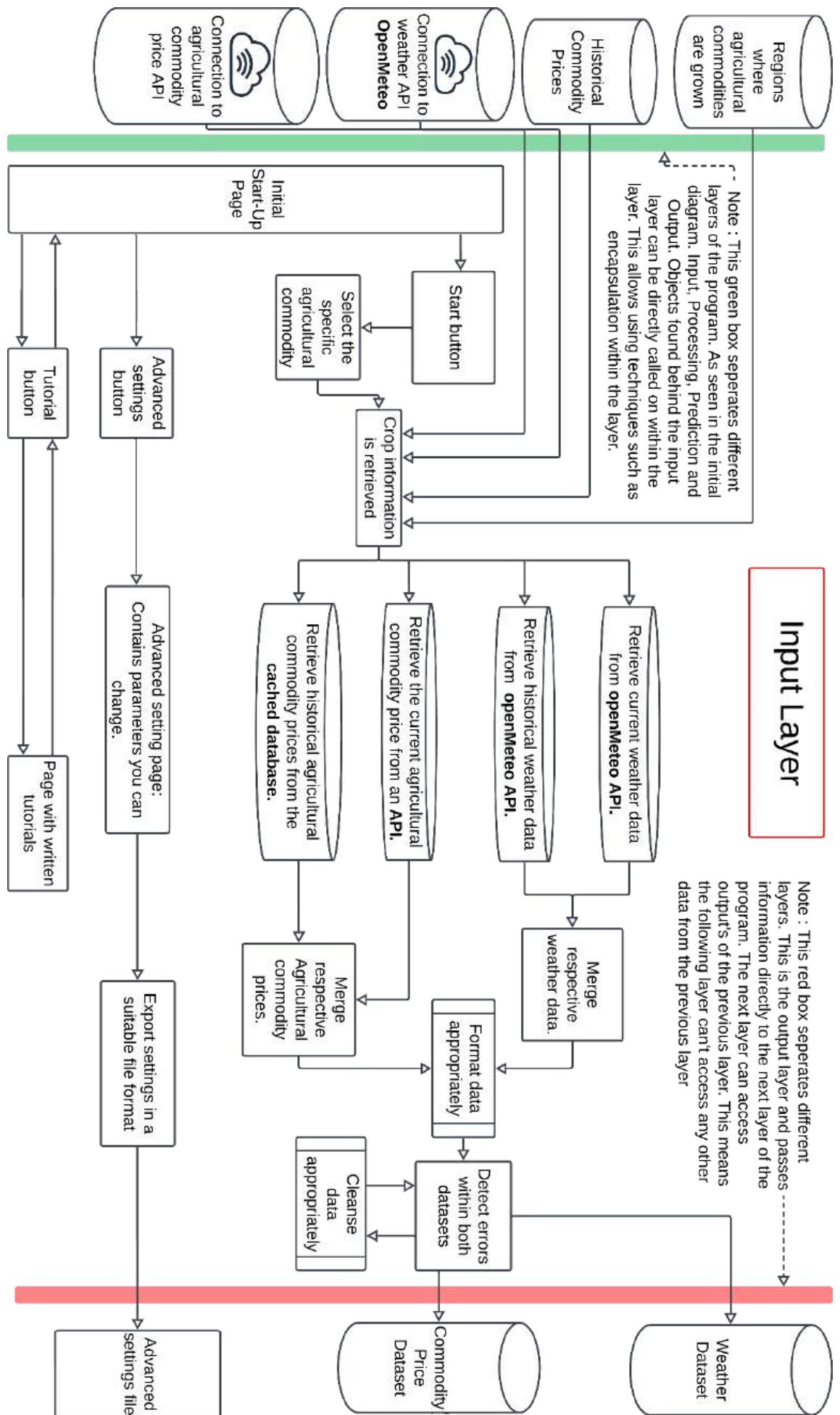
Independent Action	Explanation	Justification
Retrieve forecasted weather data	We need to fetch forecasted weather data for all 3 major crop producing regions.	We need to retrieve data for all 3 locations as when we trained data we used the same 3 locations. Accurate future weather data is also therefore crucial to making accurate predictions about commodity prices.
Retrieve crop information	Crop information includes extra information about the selected crop for instance coordinates of all the locations it is grown and the quantity that each region produces.	We use this extra information to specialise the weather data to our crop to get a more geographically accurate distribution of crops growth around the world and represent different weather systems in different countries.
Merge weather data	Combining the weather data for all three separate regions into a single dataset means we can have a single dataset that represents all weather data.	Allows us to get data in more depth and look at regional variations and weather patterns across the entire world at once.
Format and scale data	The data will need to be appropriately formatted and scaled to ensure it is compatible with the trained machine learning model.	Formatting and scaling data ensures that data is always of the same format. Training data will have been formatted and scaled in the same way so this step guaranteed reproducibility.

Independent Action	Explanation	Justification
Generate all the predictions	Using the trained machine learning model we can input the formatted weather data and generate predictions for the agricultural commodity.	We can make predictions with extreme detail and try to produce a more detailed forecast of how the commodity prices might change over time due to the changing weather events.
Create multiple separate time segments	To provide a more detailed prediction we can forecast the future commodities price in smaller chunks for instance generating the price daily and then merging all that information to estimate how the price of the crop will move over the following week.	This would mean we can see predictions in more detail and would be overall more informative for the user. As our program should be able to account for the fluctuations within the price and not just a single prediction of the commodities price.
Output prediction data as an array	We will then pass all these predictions to the next layer as an array.	An array has a clear index so we can easily structure and organise how the price changes in each time segment without needing multiple separate variables. All the data will be organised in a single file.

Evidence showing a systematic approach decomposing the output layer :

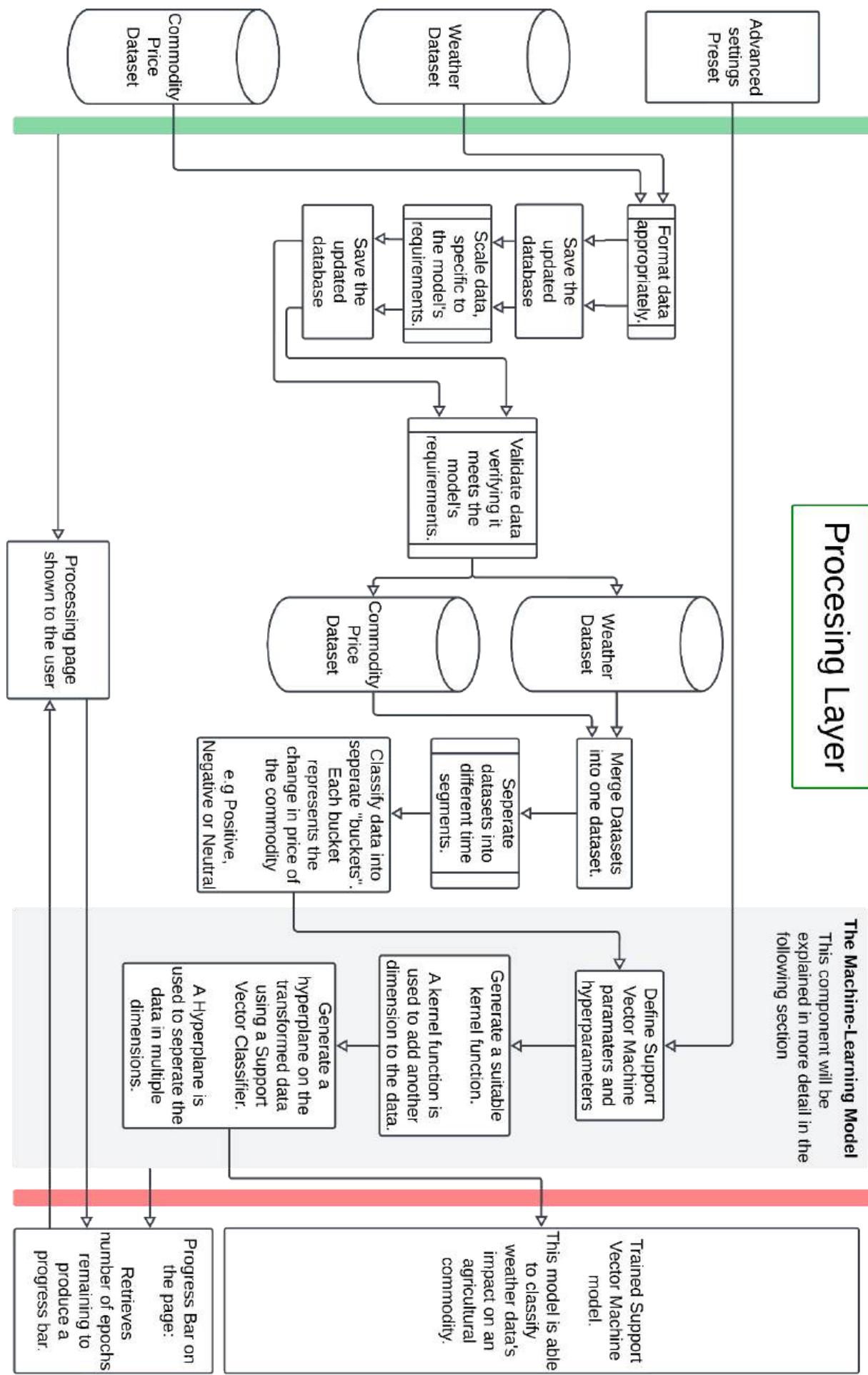
Independent Action	Explanation	Justification
Initialise the prediction data	Prepare and format the predicted time segment data ready for visualisation so it can be easily plotted on the graph.	Ensuring prediction data is a suitable format makes sure everything is functional and prevents compatibility issues such as not being able to plot the data points.
Plot all the current and historical crops	Display the current and historical commodity	This means that the user can see the context

Independent Action	Explanation	Justification
commodity price data on a graph	prices on a time series graph for the user to clearly see all the information.	behind the predictions and can understand how commodity prices change over time. All without needing to open another application instead all the data is provided to the user at once.
Overlay the predicted prices on the original graph	Overlay the predicted time segment data on top of the historical and current commodity price data.	This means that the user can clearly see both the predicted price data and also the previous commodities price. Making it even easier for the user to interpret the forecast we produced.
Enable user interaction such as easily being able to zoom / pan / scale the entire graph.	Allow the user to easily interact with the graph by zooming / panning / scaling the entire graph effectively. So the backend program will need to be able to handle changing the scale and axes.	This will mean the user can interact more in depth without graph and predicted data giving them freedom to analyze their prediction.

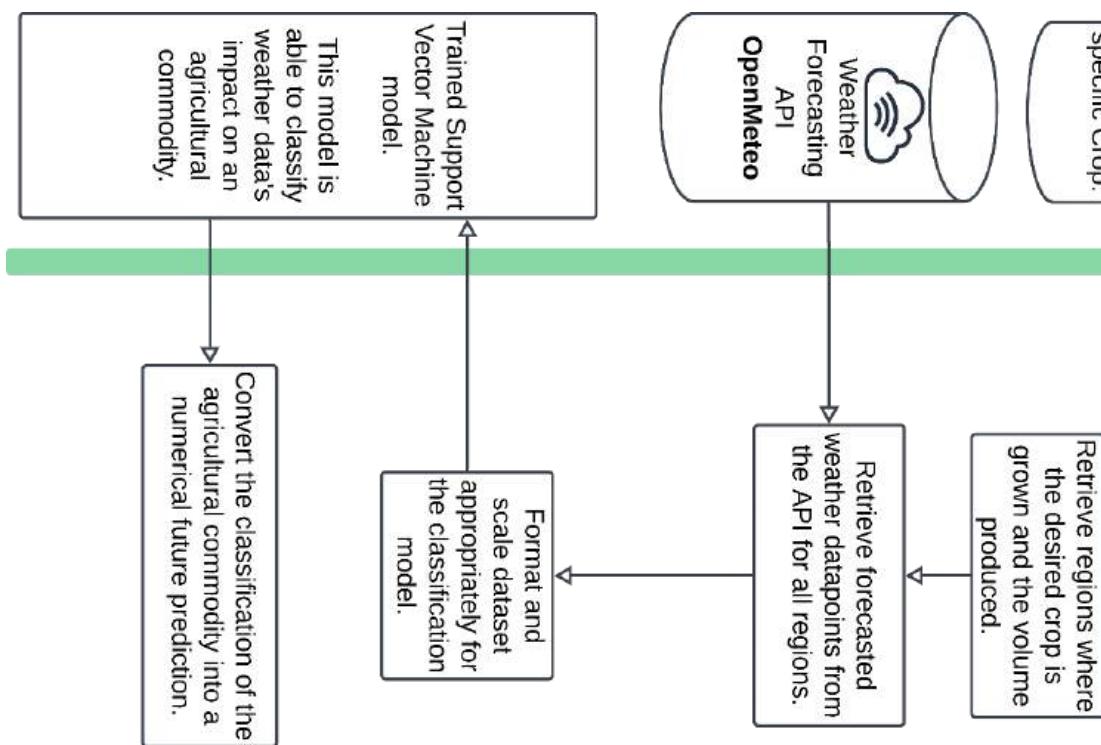


Processing Layer

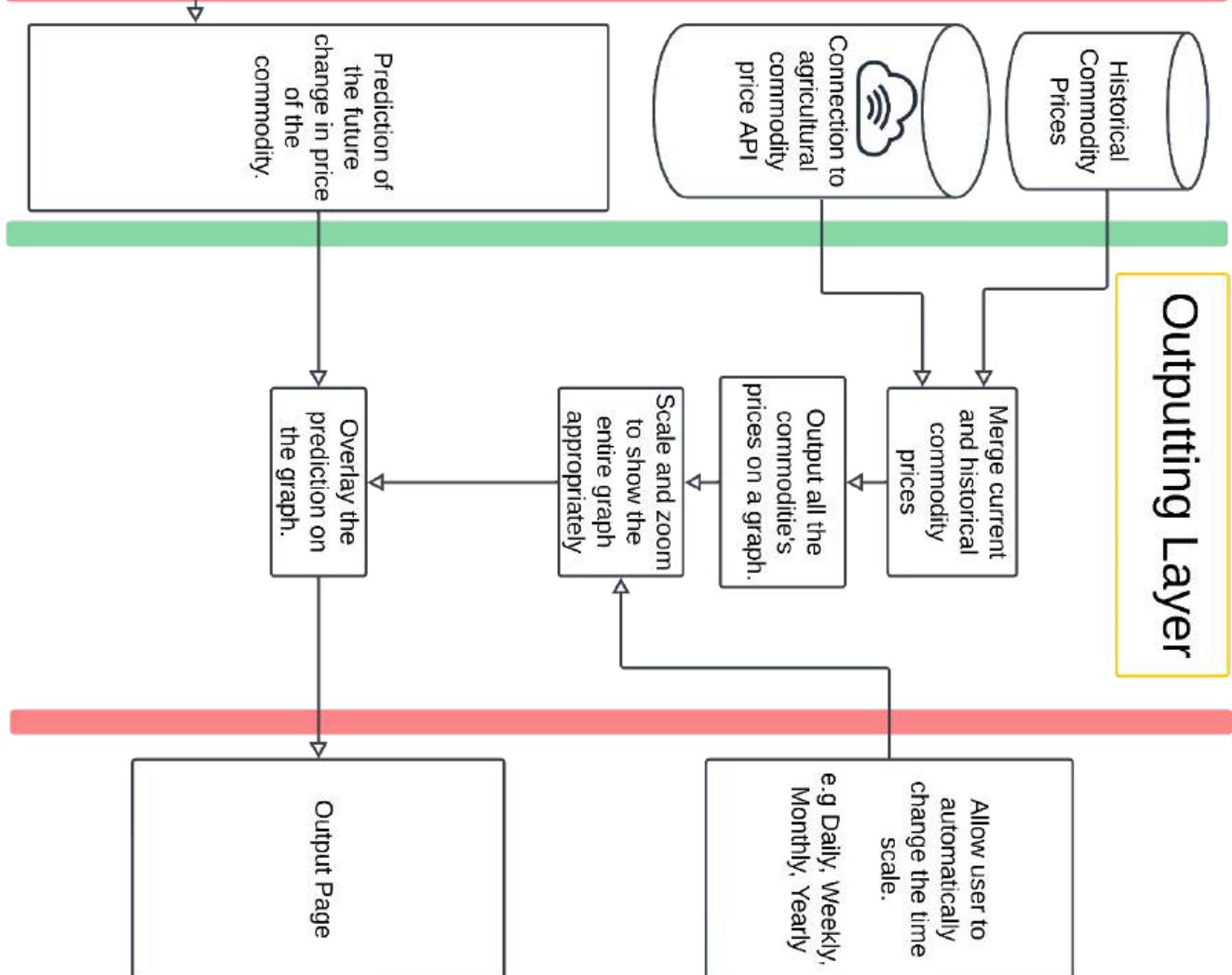
The Machine-Learning Model
 This component will be explained in more detail in the following section



Predicting Layer



Outputting Layer



Proposed Screen Designs and Usability Features

Based on the internal structure of the project we can begin to design the user interface of the pages. These pages will be the way the user interacts with the whole project so a clear user interface should be a foundational thought throughout this entire stage. Separate pages are used in each section of our program to clearly represent what the program is doing in the backend. In the table below we define the pages we need to build into each layer and why they need to be implemented.

Input	Processing	Predicting	Output
<p>This layer includes the inputting of the crop the user wants to predict while also allowing users to access other pages such as settings and a tutorial.</p> <p>This layer's main function is as an input layer. The pages we require are:</p> <ul style="list-style-type: none"> • Start Page • Crop Input Page • Advanced Settings Page • Tutorial Page <p>The Start page functions to introduce the user to the program and allow them to access other pages of the program. This page should contain a clear starting point to easily navigate the entire system.</p> <p>A crop input page would highlight the selection</p>	<p>Processing and Prediction are separate distinct layers, however the overall UI across both layers are merged. This was suggested after an interview with stakeholders where it was suggested that the UI should indicate what is happening in the background of the program. Combining this also means we can have a seamless transition between the input and output stage.</p> <p>This section's main function is as a background processing layer. The pages we require are:</p> <ul style="list-style-type: none"> • Loading Page <p>The loading page's functionality is to clearly indicate to the user what is happening in the background while also showing them the progress on these activities. These activities include retrieving data from all the external sources, correlating historical commodity prices and then predicting the future change in price. All of these</p>	<p>This is the final layer where the program outputs the solution. This output will include both historical prices for the crops commodity alongside an overlaid prediction as to the crops predicted price movement.</p> <p>The section's main function is as an output layer and to visualise the predictions. The pages we require are:</p> <ul style="list-style-type: none"> • Graph + Information Page 	<p>This page's functionality is to provide the prediction and other useful information directly to the user. Having a single output page simplifies the user's interaction. Taking</p>

<p>of crops that a user can select to analyse.</p> <p>An advanced settings page contains a range of settings you can change to edit how the model works, these settings are called hyperparameters. In this case this includes advanced settings such as the number of epochs to train the model and other settings which are specific to the implementation of the model. This page allows advanced users to tweak the model to fine-tune how it processes data and other parameters.</p> <p>A tutorial page's main function is to clearly inform how to use every function of our tool. It should be thorough and include all elements that a user will come across. This page should be accessible and easily understood to a wide range of people.</p>	<p>activities don't require user input and instead happen within the system. Therefore to retain a clear UI and to indicate to the user the current processes occurring. The system should reference what is happening in the backend so the user can clearly see how the system is processing data overall.</p> <p>A clear loading screen means that the user gets an insight into what the program is doing and therefore if errors occur they can handle them appropriately through this page. During this stage errors are the most likely to occur as there is an extreme movement and processing of data. This means that the loading screen should clearly react to errors within the system so the user can diagnose and quickly solve them. For instance this could include a lack of storage space or an unstable connection to a datasource or the API. In both these cases a clear error prompt given to the user can help fix the issue and keep the system up and running.</p>	<p>inspiration from IBM's single page design for their Environmental Suite their page outputs and displays all the information clearly on one page. Separating different sections of the page into separate widgets which perform different functions such as viewing future crop yield predictions to looking at historical weather events.</p> <p>Elements that are going to be included are the historical and the current commodities price. Overlaid on top would be a prediction as to the future price movement. Other information may also be included if relevant to the system. This may include summarised weather data and accuracy indicators.</p>
--	--	---

Similar Products Screen Design :



Stakeholders have highlighted requiring a clear user interface. Due to the wide range of stakeholders we have asked them all to try competitor's products in similar fields, to build their foundational knowledge on commodity markets and how to trade and invest in the stock market. In addition to how to interpret and understand predictions of agricultural commodities. We set up a Robinhood trading account to immerse them in a purely stock trading application. Robinhood is a financial services company that aims to make investing accessible to beginners. Robinhood is a trading platform where you can buy and sell a range of stocks / commodities. It is

available on multiple platforms such as their mobile and web applications. Robinhood is similar to the functionality of our tool which aims to provide a clear interactive graph and system. Alongside having a clear usable user interface their platform helps stakeholders understand how similar products work and also what to expect in our tool. Robinhood's main focus is to be an application to buy and sell stocks. They have limited prediction capabilities and focus entirely on developing their products. This means that stakeholders focused their time on interacting with Robinhood's interface and looking at how they approached the problem of graphing information.



Another tool we experimented with was Vesper. Vesper is a stock market prediction tool which focuses a lot more on accurate predictions of agricultural commodities. Making it extremely similar to our tool. Due to this stakeholders instead focused on looking at analysing stocks and outputting their respective predictions. One clear shortcoming of this tool is their clear focus on commercial applications so their predictions were advanced and had to be

understood. The main advantage of giving stakeholders experience of using Vesper was to thoroughly understand how similar solutions display solutions and how they use their user interface to convey this. We can see an example forecast in the image above. Vesper has a range of

After they trialed both Robinhood and Vesper on both their phone and a laptop they were interviewed to help gauge the initial ideas of how to structure the overall user interface of our product.

Dear Yash,

Hope you are doing well. I tried both software solutions and found the experience very enjoyable.

Using Robinhood I was able to quickly search through and find a variety of stocks and commodities. The main features I enjoyed were the ability to quickly go back and forth through pages and not be stuck within a loop of searching and skipping through pages. Robinhood's interface was really simple and I could jump through the pages I wanted quickly.

Vesper on the other hand was a different story. It had a similar interface, but felt almost cluttered and overly complex. However once I was able to access the predictions I wanted, it was detailed enough and helped me make decisions efficiently.

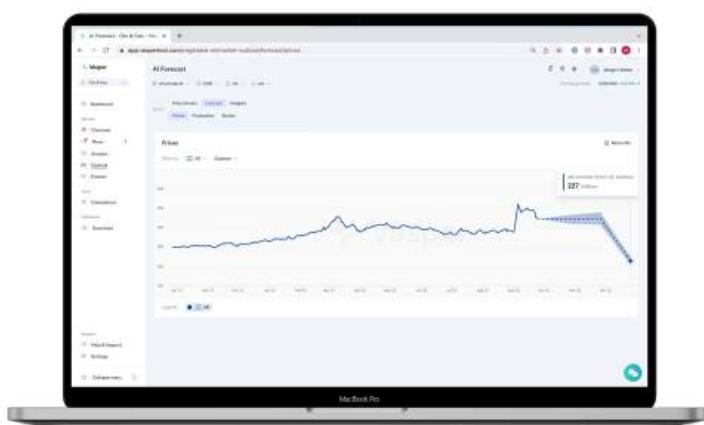
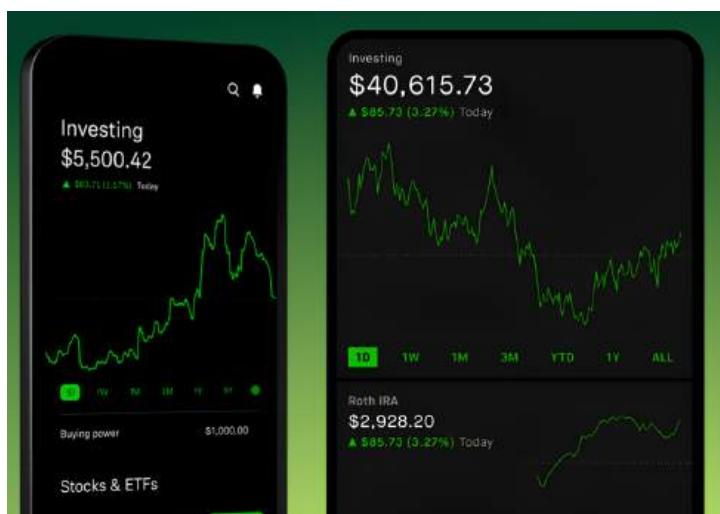
Thanks,
Mr Rao

Interviewing another stakeholder, similar points were reiterated. He highlighted

Robinhood's use of a concise interface alongside limiting the colours and amount of things on the screen. Reachability was another issue that was highlighted. Manu found Robinhood's interface touch centric and said "I can easily reach and access all points on the page. The interface was designed clearly with the users in mind". Whereas Vesper focused entirely on their prediction interface. Stakeholders found the extreme accuracy and complexity viable for users that rely on this tool commercially. The user interface and overall usability reflected the targeted end-user for this tool. The end-user's mainly composed of commercial entities that require such detailed information.

Stakeholders value simple and concise pages. This system is aimed at a large market of potential users so focusing on abstracting as much complex information from the average user improves the accessibility of the solution as there is no need to understand and use complex concepts such as machine learning, data formatting and retrieval of databases instead the system handles everything in the background.

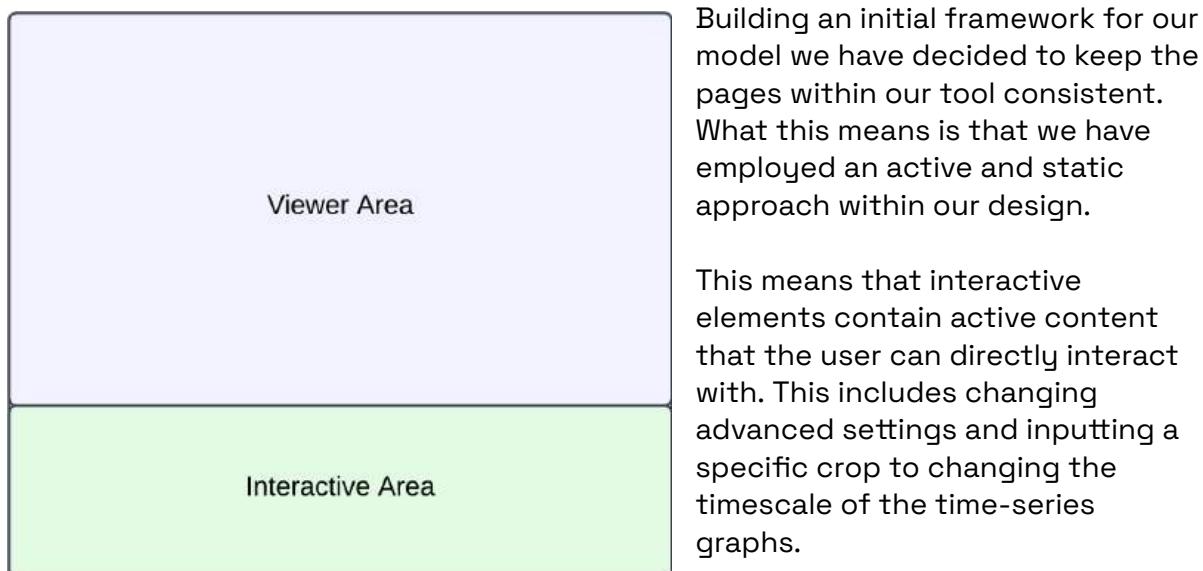
Our main intention for our tool is to incorporate both simplicity of the interface and components such as the prediction. With the complexity of rigorous data analysis and the addition of editable hyperparameters. To help build the foundations of the system we have looked into how stakeholders use tools such as Robinhood and Vesper.



Looking into how Robinhood displays data and how they format their entire system showed us how simplicity should be woven within the entire solution. Stakeholder's enjoyed Robinhood's overall usability. Particularly its monotone interface which highlighted different elements and prevented user's from feeling overwhelmed with information. In the picture above we can see how Robinhood visualises graphs and their simple application. A clear graph followed by an easily accessible time scaler. In this case we can see that all the space is utilised with key elements throughout the interface. Particularly their dedication to accommodating

touch support has meant that the tool is easily interactable and clear. Looking into how Vesper visualises predictions, stakeholders could easily gauge the change in a crop's price in the near future. Vesper's interface focused entirely on providing a prediction. This meant that stakeholder's sometimes found the interface confusing while trying to input advanced hyperparameters and other important data within the system. Ultimately Vesper's ability to give such a clear and concise prediction overlaid on top of the graph set it apart from similar products.

Our tool aims to combine key elements from both these programs fusing both a clear accessible user interface which is able to cater to people of all levels and also in the background produce a prediction based on a mixture of parameters. The ultimate output being a simple prediction which a user is able to interpret easily.



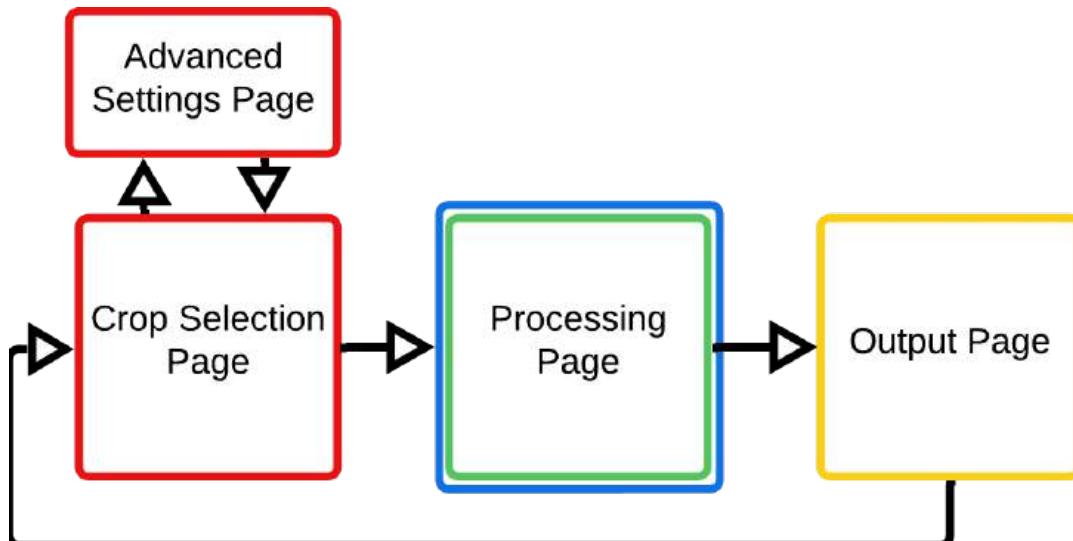
On the other hand the viewable area hosts static elements. These elements can't be interacted with and instead house all the outputted information. In this case it would include a visualised graph to other important data such as a loading screen. Elements such as prompts could also be considered instead of an entire tutorial page teaching the user how to navigate and interact with our software.

Talking with stakeholders about this planned initial framework I understood that this idea would make the application both intuitive and user-friendly as it guides the user to interact with the product in a specific way.

However one problem Sharath pointed out was that it might condense pages too much adding friction to interacting with elements. Such as in the output page where we will have to have an interactive graph which may not be entirely suitable to this direct segmentation of elements as either active or static.

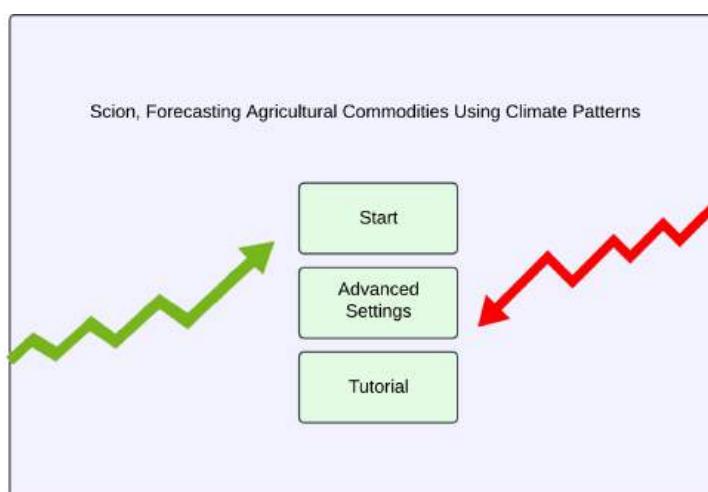
The separation of elements into active and static elements makes for a more efficient user experience. This framework helps to keep the user interface throughout the entire solution consistent and easily accessible.

Following on from our initial structure plan we have designed a page structure. This dictates the exact pages a user will interact with and how they will flow through the entire system.



Input Pages :

While looking at how simplicity should be retained within the entire system we looked at how competitors use start and tutorial pages. Robinhood's interface relies entirely on the user learning how to interact with the system directly instead of forcing them to visit and interact with unnecessary pages. This in fact means that their pages remain easier to use overall as the user learns over time. It is for this reason that we have decided to implement elements from the start and tutorial pages within the program instead of standalone pages themselves. Embedding these features keeps the amount of interactive elements limited and makes it easier to develop the solution as we can focus entirely on how the user navigates and uses the solution.



At the start we designed a start page, but after stakeholder feedback we understood that it added unnecessary friction for the user as they would need to click through multiple pages to produce a prediction. In the image we can see the inclusion of advanced settings and a tutorial page which would have cluttered the start page and hidden advanced settings and tutorials behind the initial page.

Fixing this means we can incorporate both advanced settings and tutorial segments within the main program itself. To do so we could make the input layer span only two main pages.

Therefore in the input page we are focused on designing the following.

- Crop Input Page
- Advanced Settings Page

Crop Input Page Design and Usability :

The design of this page should be simple and easily allow the user to choose a selection of crops.

Taking into account the active and static elements within the design we have chosen to have a selection of crops at the bottom of the page and the user can select one. This page represents the first page that the user interacts with so it is necessary that it is clear how to interact with the program. Stakeholders wanted to have a tutorial so we embedded that within the static areas of our software. Looking at the information box we can see that all the tutorial information has been condensed and can be easily understood. Following on from that necessary interactive elements are easily clicked from the bottom of the screen. Stakeholders liked the design and said it fully met their criteria.

Usability Features	Explanation	Justification
A clear and easy to use input page design. Specifically for our implementation of both an active and a static area of the screen.	<p>Segmenting the active and static section of our page makes the software solution increasingly user friendly as there is a consistent input method between all the pages. Users can focus on the primary task of analysing crops without distracting elements.</p> <p>Additionally this means that implementing a minimalist approach allows all the decision points to be allocated into a specific space of the page. In this case they are all clustered together in the active area of the screen. Instead of navigating through multiple buttons using multiple buttons across the page they are all consistently placed together in</p>	<p>Our design prioritizes presenting all the data in a clear and concise manner. Removing unnecessary elements such as a start page to ensure the user can quickly make their specific selection of the crop on this main inputting page. This meant we kept the main elements on the page user centric in the sense that they provide the user with quick access to selecting the crop and advanced settings without needing to filter through a start page. Instead we provide the user with a single selection so users can focus on what data they want to indeed analyse.</p> <p>In our proposed implementation we have decided to go with an active and static segmentation of our screen. Particularly within the starting user interface this keeps everything consistent by segmenting different types of elements</p>

	<p>the same part of the page. Maintaining a clear user interface as the end-user navigates and traverses through multiple pages in our system.</p>	<p>and promoting familiarity throughout the entire application. This is because instead of overloading users with elements placed in different areas of the screen we can provide a predictable experience as the user is able to understand the visual and technical minimalism within our crop input page.</p>
An integrated tutorial setup to embed how to use our page within an immediate page instead of the user relying on a separate page.	<p>The static area on the page can be dedicated to instructing the user how to access and use the solution. In this case instead of dedicating a completely separate and independent page it is woven within the original pages to be both transparent and informative as to processes occurring currently in the background and how to interact with the solution. This streamlines the entire tutorial process as the user is constantly informed.</p>	<p>Our main justification for this usability decision was to streamline our entire process. By doing so we can validate that the user is accurately using our system. In this case we can also provide custom feedback as to how to interact with all the specific elements of our program without needing the user to always access an original tutorial on the start page. Comparing a page just with documentation to an interactive changing window that suitably prompts and recommends actions is what stakeholders preferred.</p>

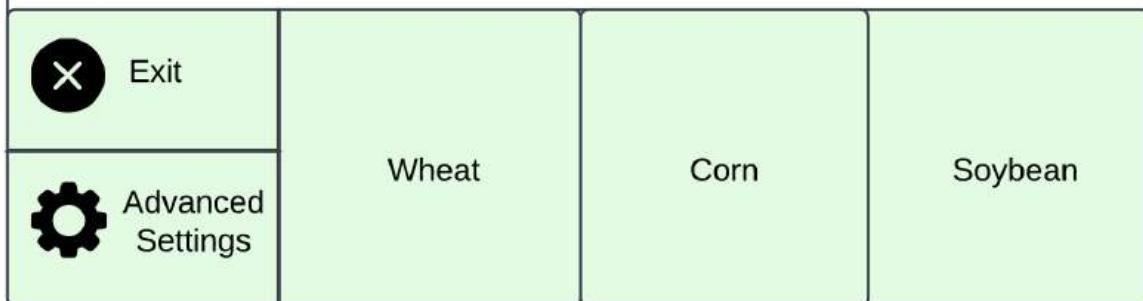
SELECT A CROP BELOW

This box provides static information. For example the tutorial prompts below.

Select a crop to analyse.

Access advanced settings to edit hyperparameters and other settings.

Click exit to exit the program.



Advanced Settings Page Design and Usability :

Advanced setting's page main end-users are advanced users who want to be able to fine tune their experience easily. This means they want to be able to quickly change multiple values specific to the machine learning model. The page below is a proposed design which I showed to Sharath and Manu who said they were interested in having and being able to access advanced controls.

Stakeholder Feedback from Manu

This design was great and has my full support. It was clearly formatted with information at the top and all the buttons clearly placed at the bottom. However it feels a bit too simple as I would be forced to use things such as sliders and use buttons to directly change to precise values. I think that being able to directly change and edit the values would be more effective.

SELECT ADVANCED SETTINGS BELOW

Hyperparameters are variables that can be configured and are external to the machine learning model. These hyperparameter values can not be estimated from the data.

Hyperparameters will be automatically configured if you don't set values to them.

C Value

Gamma Value

Other Hyperparameters

Exit	C Value	Gamma Value	Other Values
	1 - +	1 - +	

Feedback from Sharath hinted at a similar problem as advanced users would have the knowledge and experience of using advanced tools such as Vesper. The need to maximise customization is a necessity. This is because a slider and a box that can increment and decrement values may be deemed unnecessary when it is faster and more efficient to directly interact with the hyperparameters.

Taking into the feedback I received I implemented it directly into a new design by adding large text boxes that you can quickly change without needing to go through an inbuilt system which may take longer and be less precise. The tutorial page also highlighted the advantages of this layout as important elements are all segmented into their appropriate sections. Sharath elaborated saying that the usage of static and active boxes was beneficial to this tool as the user can easily discern how to change specific parameters and also the impact of editing hyperparameters.

SELECT ADVANCED SETTINGS BELOW

Hyperparameters are variables that can be configured and are external to the machine learning model. These hyperparameter values can not be estimated from the data.

Hyperparameters will be automatically configured if you don't set values to them.

C Value
Gamma Value
Other Hyperparameters

X Exit	C Value 1203.23	Gamma Value 432	Other Values : Value 1 0.01 Value 2 0
--	-------------------------------	-------------------------------	--

The reason that this page is completely separate is to prevent the user feeling overwhelmed with the amount of variables that they can edit and change. Specifically as our tool is dedicated to a wide range of users with varying levels of experience. Keeping entirely separate pages separated allows for navigation to feel seamless as it is natural to click through each page. Instead of overloading a page with information.

Usability Features	Explanation	Justification
The ability to directly manipulate hyperparameters and apply it onto the model.	This allows granular control of exactly how the Support Vector Machine (SVM) model is trained on the training data. In this case this allows for the setting of custom hyperparameter values for C and Gamma before a model is trained specifically if the user is an advanced user and wants advanced settings when using the program.	Our advanced settings page gives users more control and freedom to fine tune exactly how the model will perform. Hyperparameters are set before training occurs and allows the user to change how the model analyses training data. Giving users full control over the internal parameters of the model means that analysis can be made specific to their needs.

	<p>Additionally this also means the program is able to cater to advanced users and users of all skill levels as it broadens the potential application of our tool allowing users the choice to build their own custom model instead of the program either automatically optimising hyperparameter values or it being set to default values.</p>	<p>Often in other systems such as Vesper, how a model works behind the scenes is hidden and inaccessible to the user. Vesper's solution produces a single prediction and gives that prediction to all the users without them being able to suitably tune and customise it to their own specification. Our stakeholders specifically wanted these features as we want our tool to be flexible for all users with different levels of experience.</p>
Separating advanced settings into a separate distinct page.	<p>Separating pages for both beginners and advanced users allow this tool to cater to multiple skill levels which in this case can be represented by the diverse range of stakeholders that are interested in our tool.</p> <p>Maintaining the design consistency this distinct advanced settings page can apply directly to only appealing to advanced users and therefore allow for advanced users to have access to a range of technical features such as custom hyperparameters and additional custom settings all in one place.</p>	<p>Having a separate distinct page for advanced settings helps to separate advanced settings into a completely different page and explain elements on each page suitably. Revealing advanced features only when requested means that users are not intimidated by the selection and amount of complex settings they have access to. In our case this page is developed only for advanced users in mind to fine tune the model's hyperparameters. As our tool should be able to use default parameters and optimise these accordingly a range of beginner and novice users will not need to access advanced setting pages. This means that using a separate page that is only viewable once requested caters to all the users and stakeholders and not just focusing on a specific subset.</p>

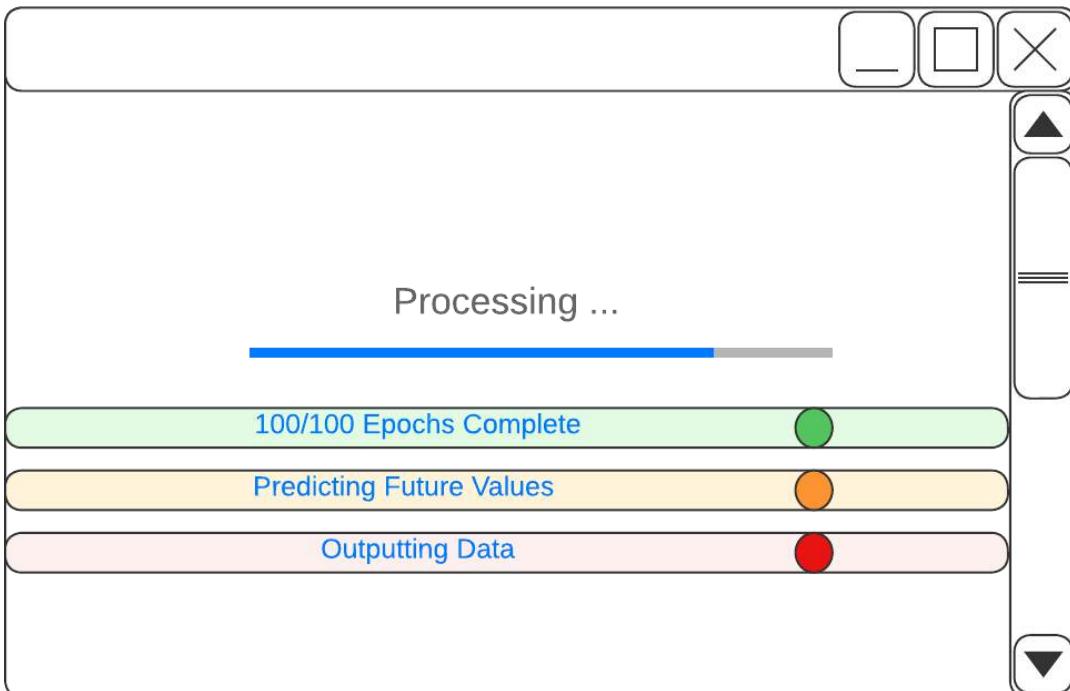
Processing / Predicting Pages :

The majority of background processes occur in this layer. Due to this there is an added requirement to actively handle errors effectively. Throughout the entire process this stage will be the most risk of errors occurring. This is because of the need to retrieve and handle datasets simultaneously. So ultimately the shared design and functionality philosophy of this page aims to both inform the user about what is occurring in the background and be able to clearly indicate errors that are occurring during this process.

Within this page we use a single page that is able to display both the background progress and indicate when an error occurs.

- Processing Page

Processing Page Design and Usability :



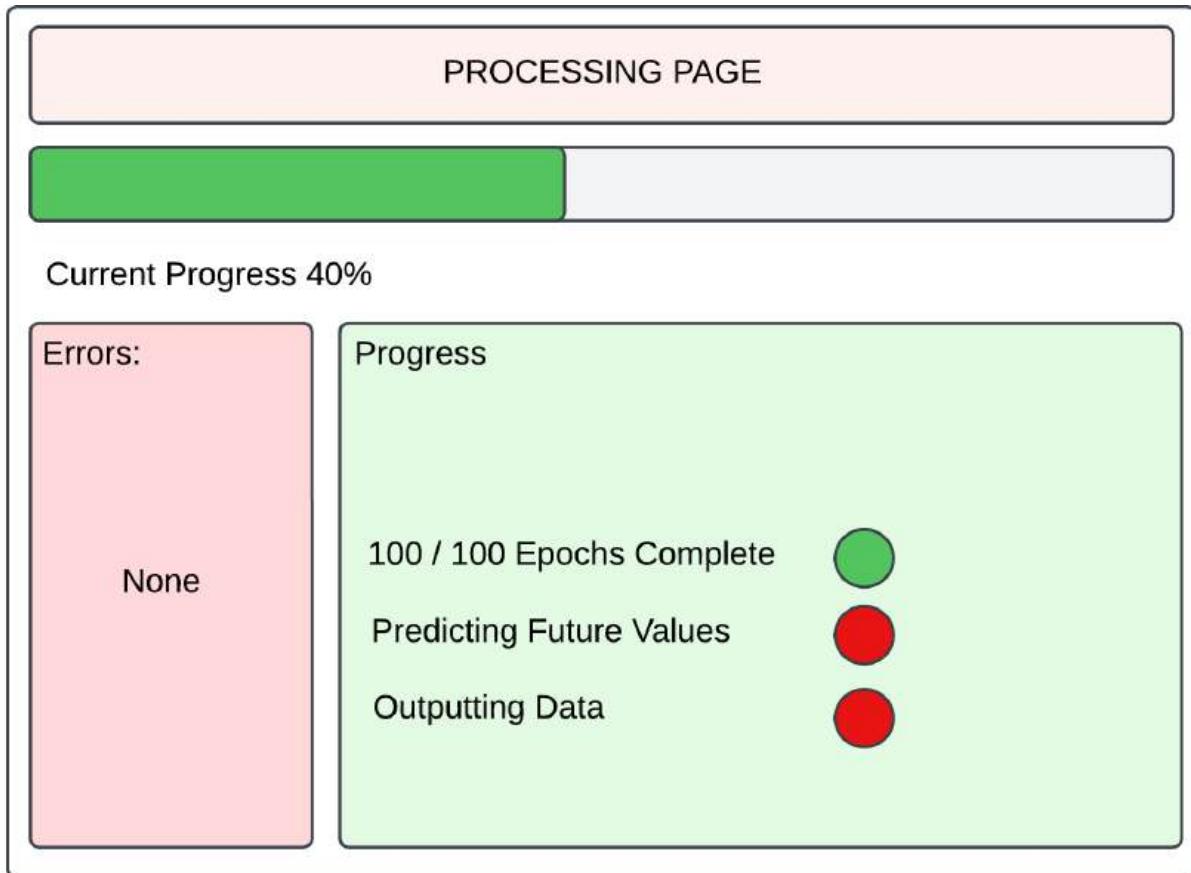
Errors:

Unable to Retrieve Historical Commodity Data from the internet.

An initial design included just a simple loading screen. In this design the user can see the current progress of the system through the progress bar at the top of the page. Alongside this progress bar is a list of the exact processes that are being executed currently. Such as epochs which is the amount of training cycle that the model will have to go through and other processes that are due to be executed. Stakeholders such as Shreeharsh appreciated knowing what is happening in

the background as they can easily react to unexpected errors that may arise. However a concern that was raised was the need to display these errors to the user. In this case our stakeholder said he thought segmenting the processing and error pages such that they are distinct pages meant it may be unclear to the user if an error occurs as it is a completely separate page that opens.

To counter this we redesigned both pages merging them to be a single page that contains both progress and a dedicated error panel so the user can quickly have issues diagnosed all on one page. Looking at the redesigned page below we can see that user's are constantly reassured and have access to all elements at once. We kept the progress bar as the main element and built around that with a detailed progress widget which shows what processes are occurring at each moment in time. Shreeharsh said it fully met his criteria as it was easily interpretable and didn't require using multiple different pages.



Usability Feature	Explanation	Justification
A progress bar that shows the current real-time progress of our entire system.	A progress bar allows the user to be informed in real-time as to the current status of the system. Meeting requirements as to the transparency of processes happening in the background. Additionally as stakeholders are able to recognize what is happening in the background we can also implement an error reporting system to output errors if they occur directly to the user.	Including a progress bar in our solution gives the user real-time data as to the progress of our solution. During extremely computationally intensive tasks such as when data is being processed or a model is being trained there is no need for the user to input any data. Due to this a clear visual cue that is embedded within the processing page keeps end-users informed while also promoting a more seamless user-experience.
A progress widget embedded under the	Transparency as highlighted by the stakeholders is a necessity as they should be able to recognise processes currently occurring in	Specifically stakeholders such as Shreeharsh wanted to understand exactly the processes occurring in the backend. A progress bar widget means

<p>progress bar which shows the exact operation that is occurring at each time.</p>	<p>the backend and how it is occurring. Additionally this allows the user to have a greater understanding of the steps taken for data to be processed into a series of approachable steps.</p> <p>This boosts the trust of the system as if in the future any potential errors occur also adding to the maintainability of the system as a whole. This is due to users and developers being able to see exactly the component that failed and why with clear and concise error messages directly generated and outputted to the user. In this case this allows the user or developer to easily debug and fix the error without needing additional support.</p> <p>This also aligns with the original stakeholder requirements as part of the user interfaces design which clearly segments linked data elements together.</p>	<p>that specific information about the processes that are currently happening in the background can be displayed to the user to keep them up-to-date with what is currently being executed. Transparency means that users can easily identify what is happening in the background. This means that specific stages during the “processing layer” will be named and outputted to the user. The main stages within the processing layer which we can identify from the entire structure of our solution is the following.</p> <ul style="list-style-type: none"> • Retrieval of data • Processing of the data to produce a machine learning model that can classify weather data. • Retrieval of future forecasted weather data. • Applying the model to classify the future weather data. • Finalising Predictions. <p>A range of these subsets can be clearly shown on the screen for the user making their entire user experience more enjoyable and informative. As instead of the complexity of our solution being hidden we can show it to the user in a clear manner.</p>
<p>An error widget embedded under the progress bar which shows the immediate visibility of an error if it were to occur.</p>	<p>Proactively and automatically managing errors means that the program is able to handle a range of inputs and quickly react to potential issues.</p> <p>This increases the resilience of the system as it is first able to catch errors before they occur using a series of validation steps across the entire program.</p> <p>Additionally it is able to manage and solve potential errors if they</p>	<p>An integrated error panel will serve a similar planned function. In our case we aim to be as transparent as possible with the user. The immediate visibility of any errors is crucial to this.</p> <p>Instead of constantly informing users what is happening the integrated errors panel aims to do the opposite. Instead only when an error needs to be handled by the user and can't be automatically handled within the system is when they will be addressed on this particular page.</p>

	<p>occur and only informing the user if necessary.</p> <p>Our system should have multiple validation sections to verify that the code is working as required and that a valid output can be produced. In some cases the system can automatically handle errors. Examples include reconfiguring an API connection such as if only partial data has been received. The system can automatically request all the data.</p> <p>Ultimately implementing processes that follow this framework means the solution is able to be in a technical system more robust and have complex error-handling systems.</p>	<p>A fluid machine learning model that can classify a range of different data points. This is because a machine learning model is able to actively learn patterns instead of directly memorising it. So unseen data can be appropriately classified, something a generic algorithm is unable to do as its decision boundaries are not fluid with the data and are instead defined by fixed rules.</p> <p>Web scraping a website with a new user interface to find a specific dataset. Websites get constantly updated with new elements so a fixed algorithm is unable to find a specific element. Instead with a system that is able to actively search and understand the HTML structure of a website we can find specific elements that are hidden within it.</p> <p>Using techniques such as imputation to fix anomalous and missing datasets. In this specific technique we make unusable datasets clean by fixing the mistakes within the values of a dataset.</p> <p>These are examples of where a program actively is able to handle errors in some cases this isn't possible such as if a dataset is completely corrupted, we don't have access to the internet to make a request to an API or any other example of an error that needs immediate attention to appropriately diagnose and fix. In this case we have no choice, but to ask the user for help. The error portal helps us to clearly show the user what is the issue, preventing the entire program from crashing. This means that the user can easily debug and fix the issue without needing to access the source code. Instead errors are clearly documented and presented to the user within the embedded error panel.</p>
--	---	--

Output Pages :

An output page main function is to provide the user with their desired prediction overlaid on top of the historical and current time series data. For simplicity this page is a single page with only the time-series commodity graph shown. The main focus solely being to provide the user with data to quickly understand the movements of an agricultural commodities price movement. To allow for users to quickly analyse multiple commodities quickly we are planning to include features such as split screen and a button to return to the starting page to select a new crop.



The main focus of this page is to enable the user to understand a prediction within their specific agricultural commodity.

- Prediction Visualisation

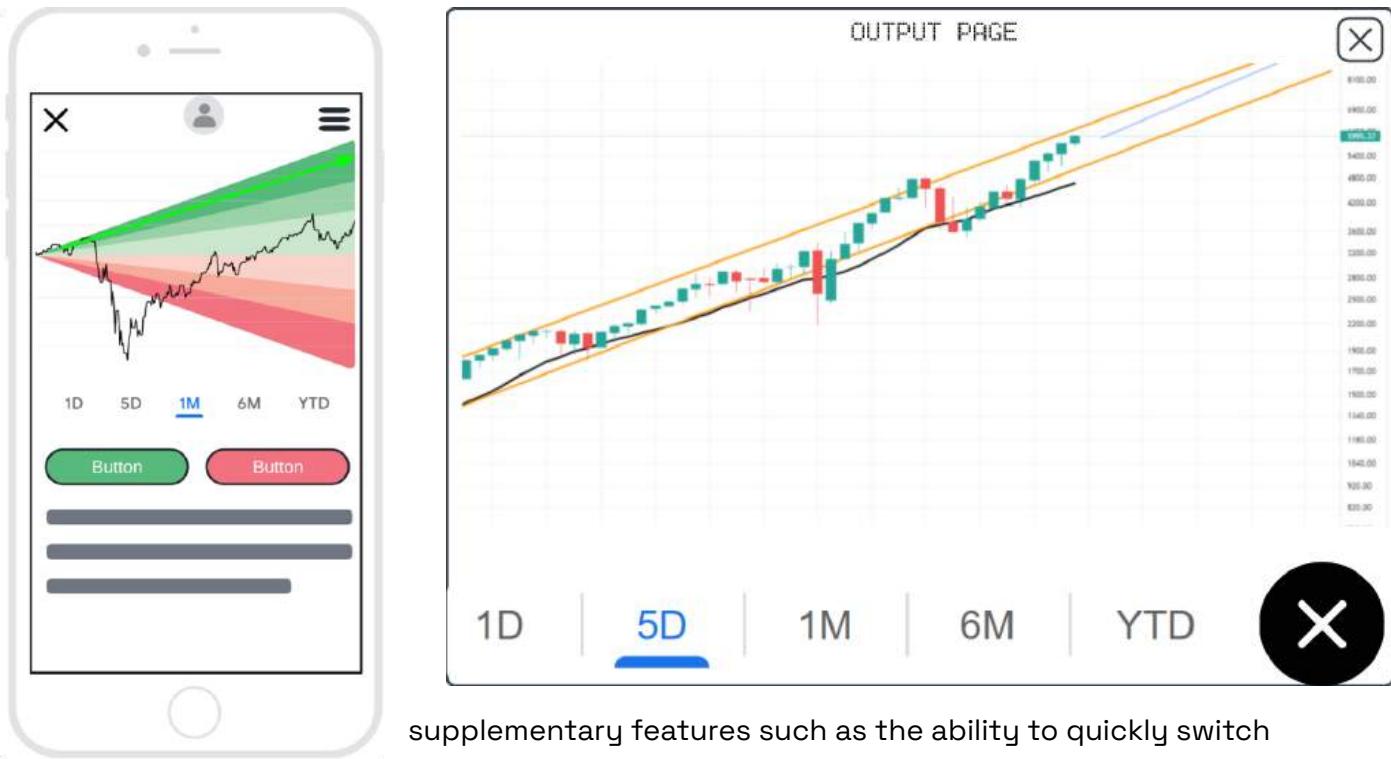
Output Page Design and Usability :

A prediction visualisation includes 2 key elements. The prediction itself and then historical and current commodity prices. Merging these two elements we get a graph. Stakeholders also mentioned wanting to be able to easily zoom and pan across the data and also features such as the ability to specify a timescale. For instance users can switch between a monthly and a daily view quickly. To do this we asked stakeholders for their initial ideas about the layout after using Robinhood. We can see below how Robinhood structures and displays their graphs.

Poornima, one of my stakeholders as a beginner, didn't understand how to use Robinhood's platform. However after the initial learning curve she saw how the elements could be easily understood. Specifically how the graphing function works,

while interacting with Robinhood's desktop trading platform Robinhood Legend I saw clearly how Robinhood uses a systematic approach when they output data. In the graph above we can see how their interface is entirely dedicated to the graph with interactive elements to change the timescale and other features nested at the top and bottom of the screen. All of this makes Robinhood's platform concise so you can quickly glance and view without needing to go through multiple steps.

Designing the visualisation of the prediction means we have to focus first on the graph and how it is plotted on the page. While alongside accounting for



supplementary features such as the ability to quickly switch between different time frames and zoom, pan throughout the entire graph. Overlaying the prediction appropriately will also be another major component of this page.

A previously designed wireframe shows an example mobile application. In this case we can see a clearly labeled graph with a prediction and the necessary time scale selector. Building on this, stakeholders requested making the online interactive element being the graph and taking forward Robinhood's almost minimalist interface.

Stakeholder Feedback from Shreeharsh :

Shreeharsh said the final output page clearly prioritised clarity and conciseness. So all users can quickly and easily understand all the presented information. Dedicating the page entirely to the graph means that user's don't have a choice but to interact with it. Preventing the program from overloading the user with information.

Concerns on a mobile application however would be the overall responsiveness and user experience as our software would only have access to low power hardware and a smaller display area. Users will always expect a fast and responsive interface and with the current design it might be less appealing than the fully functioning interface on a high power device such as a laptop. However these constraints of the specific system is fully outweighed by the familiar and extremely intuitive user experience across all the systems

Stakeholder's enjoyed this design as it merged both the interactive elements with the static graph allowing users to naturally and natively interact with the data. The graph is clearly outputted with an overlaid prediction. The exact way we output a future prediction is under development and depends on multiple factors such as the model and how it processes and outputs data. However the main format will be a clear line overlapping the graph showing the future movement of the agricultural commodity.

Usability Features	Explanation	Justification
The ability to directly interact with the graph we produced and the lack of any other elements on the output page.	<p>An interactive graph means that the user is able to engage with the predictions and historical data simultaneously instead of it being necessary to have two separate programs to display the data. Our solution combines everything into a single agricultural commodity prediction application.</p> <p>These interactive elements were inspired by applications such as Robinhood which users find extremely simple and easy to use. With intuitive navigation and interactive elements to interact with the commodity price data.</p> <p>Additionally an interactive graph allows the solution to be able to be used on a range of devices such as touch-screen mobile phones to desktop computers further introducing the accessibility of our software on multiple pieces of hardware.</p>	<p>The interactive graph serves as our final output of the entire solution. We aim to provide a detailed and comprehensive view of historical commodity prices and the future predicted commodity prices within this final step.</p> <p>To do this we display the entire output as an interactive graph that we can appropriately zoom / pan / scale to understand the intricacies within the data. This feature encourages the exploration of the data and caters to users who will want to examine in detail specific time segments.</p> <p>Stakeholder's wanted a simple interface that is easy to use. Inspired by Robinhood's user interface and their concise method to navigate both around their whole solution and within the graphed data points. We decided to have a minimalist philosophy when we implemented this stage as the only information required by the user is the final prediction. This meant that our solution focused on delivering the primary information instead of</p>

		overloading the user with a cluttered and inefficient page that is of limited actual use to the user.
Predetermined time-scale selection to quickly switch between multiple time-segments .	This added flexibility allows users to analyse trends in the short and long term quickly using time-scale selection buttons to click between different ranges. This is a key feature that was included in similar software solutions that include interactive graphs so allow for an intuitive interactive graph interface.	We are planning to implement the ability to switch between different predetermined time scales such as (daily, monthly) to provide an added amount of flexibility for the user when viewing the data as they can quickly switch between analyzing short-term and long-term trends in the data. The reason we did this is to reflect key elements within Robinhood's platform and the user interface of the majority of stock market visualisation platforms. This means it enables the user to quickly click a timescale to switch onto a specific time segment on the graph without needing to zoom / scale / pan to see that time segment. This function will directly interact with the graph to allow seamless transitions between different time segments.

Summary of the Processes at Each Stage

Detailed summary of the process including key variables and structures.

The program has been split into distinct stages as we can see from the structure of the solution. These graphs split the program into 4 distinct layers as data follows a linear path through them. Travelling across the entire structure to get to the final output which is an appropriate prediction. Each stage is independent of each other. With only key information retrieved from the previous stage. Within the input stage there are datasets and functions that can be accessed globally which are indicated when they are inputted into a layer. This means each layer retrieves data from the previous layer alongside accessing data from other external sources such as an array stored on the device. This intersection of using both local and external storage makes the program need to handle retrieving and executing functions on multiple datasets simultaneously. Doing this means we have a reliance on using separate layers and independent functions within each layer. Each layer has clear transferable data points between layers to help limit the amount of overhead data that may clutter the system if we develop the program linearly. Multiple moving

parts at once also means we can use techniques such as parallel processing to speed up the total execution speed of our software.

Input Layer :

The input layer's first user focused action is to prompt them to select a specific agricultural commodity to analyse. In this case we load the crop selection page which we defined beforehand. This screen will have the ability to do two actions: select the crop to analyse and to access the advanced settings page.

First examining the crop selection page we see that a user can simply pick a crop from the selection of all the crops and it will save it as a variable called crop. The crop variable will be a string and be a predetermined identifier for what crop is being analysed. This variable is used as an identifier within functions in all the layers to simplify what data needs to be retrieved. This is because all the datasets contain a wide range of data. For example OpenMeteo, our source for historical and forecasted weather data, stores data for locations around the world. Specifying the exact crop means we can retrieve data for the required latitude and longitude coordinates and analyse that. The main reason we use a weather API is because of the issue as there is a wide selection of potential crops there will be too much data to store weather datasets for every single crop. OpenMeteo once directly requested will reply with a pandas dataframe with all the requested values.

In our case requests from OpenMeteo will be retrieved and contain the 4 weather metrics. This includes the temperature, humidity, precipitation and cloud cover. This will be stored in a pandas dataframe which is a two-dimensional data structure which contains the data in a tabular form. This data structure can be easily converted into a .csv file which is a file format that stores tabular data.

Once we get the specific crop we can focus our efforts on compiling all the data specific to that agricultural commodity. Extra information about each crop is available from an external database which stores the coordinates of where crops are grown and the quantity that each region produces. In this case we retrieve our data from 3 locations. We first need to find the latitude and longitude coordinates of regions where the crops are grown. This data will be in the form of a two-dimensional array as it contains both the latitude and longitudinal coordinate within a single entry. In total within the specific crop's array locations there will be a total of 3 entries signifying the 3 major locations where it is grown.

Within this same crop information database there will be information as to the spread of volume between all the locations. Given as a percentage or the raw quantity that this region produces we retrieve that data. This will be stored as an array with 3 values so the program can estimate what impact a single region's climate has on the overall price. This is because if a particular region produces more produce then it by definition has a greater impact on the price than a region that produces less.

Now we have three pieces of information: the crop, regions where it is grown and the quantity each region produces. This data will be passed through the layer as the

crop, a two-dimensional dataframe of coordinates and an array which contains the spread of volume between the regions it is grown. n

Now we have the information specific to the crop and details about the regions it is grown in we can retrieve the respective historical climate data and current weather data. To do this we use the specific crop and its respective coordinates of where it is grown. Paired with the current date. Using all this information we can construct an API request for OpenMeteo. Following their documentation they need precise coordinates and the range of dates and then pass it in a suitable format to the API as a request. To create a valid date range we do the following: Find the date of the starting datapoint and we can take this as the starting date of the cached historical agricultural commodities price data. We do this as both databases (price and weather) are fundamentally paired and analysed together. This is necessary as they have the same range of data points so if we request information that is not in both we can't use it as training data to train the machine learning model. To do this we can take the starting date of the agricultural commodity price data and then the current date to construct a time range for weather data. This time range shows the range of dates that we need to retrieve weather data for. In this case OpenMeteo requires them to submit the data as a string in the format ISO8601 which is a standardised international date format. Once we send the API request to OpenMeteo they should respond with a panda's dataframe which we could save as a data frame or a csv file. In total we send 3 total requests as we have 3 major locations per crop that we need to retrieve. We will save these as 3 separate files ending in 1, 2 and 3. For instance historicalWeather1 and forecastedWeather1.

Alongside retrieving a range of weather data points we can also in parallel retrieve the respective crops commodity prices. We need to focus on up-to-date current market data and also historical market data. The commodity market data changes day-to-day so relevant data is required. To do this we will retrieve current agricultural commodity data for the specific crop. Alongside updating our cached database which stores the historical prices of agricultural commodities. Retrieving commodity price data can be done in two potential ways. Relying entirely on an API such as AlphaVantage which provides real-time and historical commodity data. Or using a module such as beautifulsoup4 in python which is able to capture the underlying HTML of a page and extract data. Beautifulsoup4 is an external python module which allows users to simplify web scraping for data. The exact implementation will be discussed in our design section. However after one of these processes we should have access to commodity price data. Doing so would mean we can retrieve the current commodity market data and save it onto our device and by doing so update the existing cached historical commodity market data.

Our solution has a dependency on databases. This is because foundationally our tool is a data analysis tool which means we need to focus on two core aspects. Handling databases and processing databases. In this layer the input layer we are dedicating our resources to handling databases. Doing so we need to take into account the needs of future layers which involve processing these said databases. The two main databases we pass onto the next layer which involves entirely processing data are the price and weather databases / datasets. Conformity between layers is important therefore as each layer should conform to the

requirements of future layers. What this means is that the database format we produce should be easily accessible and easily implemented into future areas of the program. To do so we have decided to use both pandas dataframes and comma separated value (.CSV) files to store databases. This is because they are both indexed appropriately and can store large amounts of data. This is necessary as we are taking into account decades of both commodity price data and weather data so using a sustainable file format is necessary. Another reason for using these formats is the interchangeability between both as they're both in tabular form they can be easily read as a .CSV file or transformed into a panda's dataframe. This means our code retains our maintainability and can be easily updated to account for new data for instance.

Looking at the decomposed input layer structure we see the need to merge data into their respective datasets. This process happens as we need to simplify all the distinct outputs into a single file between layers. What this means is that we are collating all weather / price datasets and producing a single weather and a single commodity price dataset.

Discussing merging weather data from multiple sources and timeframes into a single weather dataset. We need to try to capture the environments that the crop is grown in fully and represent it numerically in a single dataset. The values across this dataset should help us estimate the ultimate change in price of the agricultural commodity as we can use the model to infer how the weather affects the resulting commodities price. By creating this dataset we can use forecasted weather to predict the future price of the agricultural commodity as the original historical weather dataset represents the historical crops growing environment precisely. This process is required as we are retrieving weather data from multiple sources and have only a single commodity price that we can correlate with. To merge all the weather data into a single file we use the 3 separate weather data frames which are labeled 1, 2, and 3 to represent the region they are grown. Alongside the spread of volumes array which shows the quantity of produce that these major locations grow. First we have to standardise each two-dimensional weather data table. Standardising involves multiplying all the data points in the table by the percentage amount of produce that the region produces. For instance if a region where a crop is grown accounts for 1% of the worldwide output then it would have a respective 1% impact of the agricultural commodities price. What this means is that if a region grows more of a crop it will therefore have a greater impact on the price of an agricultural commodity. Once we have the standardised weather for all the values we can merge all three datasets. This means that we can approximate the worldwide weather's impact on a specific crop using a single database.

These final merged dataset might however contain missing data or anomalous results. First we need to format the data into an appropriate file format. As discussed before this means we may store the weather and price datasets as a .CSV or a Pandas DataFrame. This file format between the two should be the same as they will be retrieved and processed together when training the model on historical price and weather data. Once we format the data into an appropriate file format we can pass both separate datasets into a data validation step. This function's main purpose is to detect errors in the values within each dataset. What this means is

that missing values for instance are handled and changed either to a zero or can be generated using a process called Imputation. Imputation is the process of replacing data with substituted values. This involves looking at other information within that datapoint and making an educated guess at what that data point includes. This exact process of imputation will be discussed within the more technical, algorithms side of the design section. Imputation will mean we can replace missing pieces of data with an estimation so we can train the model with a complete dataset.

Following on from this we need to detect and handle anomalies. Anomalous results are results that deviate from the expected pattern . For instance an extremely large temperature that is outside the normal range. If there is a large presence of anomalies it may limit the accuracy of the model as the training data is incorrect to some degree. Handling anomalous results means we have to cleanse the data appropriately. In this cleansing the data means we will have to as before look for anomalous values and then use the same process of imputation within a function to generate a suitable value that it can hold. The process of imputation would be a separate function within itself as it needs to be able to look at the data without bias and generate a value that is both within the standard range and also fits the other data points well. For example if we are looking at data in a cold region such as the Alps within Switzerland we can expect the temperature all year round to remain cold. Therefore if we impute data for the temperature value we can expect it beforehand to remain low. All of this means we have to take multiple factors into account before generating and imputing a new value into our dataset.

Following on from the beginning, after the initial crop selection user interface the end-user will have the option to access advanced settings and a tutorial. This directly links in with the proposed screen designs which we designed previously. To access the advanced settings page the user will click a button nested within the interactive area of the screen redirecting the user to an advanced settings page. Within this page it will contain a range of hyperparameters the user can change specific to the machine learning model and other editable technical settings. In this case parameters refer to data learned by the model itself directly from the training so the user will not be able to manually adjust them. In comparison hyperparameters are adjusted before the training process begins and directly affects how the model learns from the training data. If the user is a beginner the model will take default parameters and automatically optimise itself by changing the hyperparameters. In this case we could implement a function that does an exhaustive search over a hyperparameter grid. What this means is that multiple different hyperparameter combinations are arranged on a finite grid and the algorithm does an exhaustive search over all of them. Training the model over all the different hyperparameter combinations and using the most effective one. This is a technical aspect of training the model so will also be discussed in more depth during the implementation side.

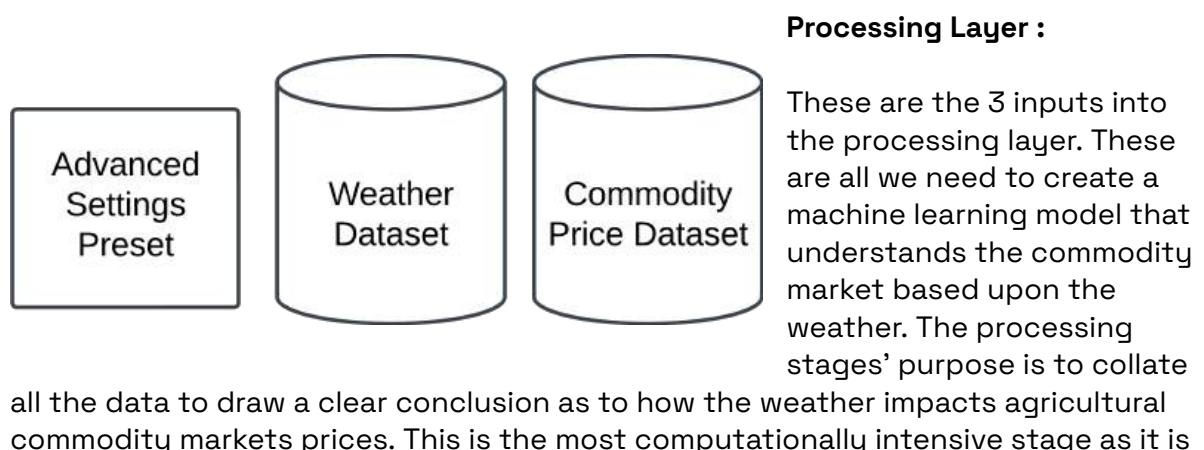
Once the user inputs the precise hyperparameter values within the advanced settings page these will be saved as boolean values if they take true or false or floating point numbers if they are able to take fractional values or integers if the hyperparameters take real numbers that are whole numbers. The specific hyperparameter definitions have not been included and are specific to the machine learning model we plan to implement; this will be looked into in the algorithms and implementation respective sections. If the user leaves a hyperparameter value blank

or unchanged we will automatically use a default value. This is due to our user base also accommodating beginners so we don't require users to fully understand changing specific hyperparameters. Default values will be predetermined values which give a normal expected result, but the program isn't fully optimised for these hyperparameters. The hyperparameter variables will then need to be appropriately stored in a file before we go to the following processing page. This is because we employ a layer structure within our code to make it modular. Each module or layer can't be run without all of the required elements present and inputted within that module. We take this file with all the stored hyperparameters and pass it onto the next layer.

One element we haven't mentioned which we have amended from our previous design is the tutorial. Instead of a standalone tutorial page we have decided to embed tutorial prompts throughout the entire solution to tell users how to interact with the respective solution. As we have shown in our proposed screen designs we have integrated a concise tutorial throughout all the separate pages. These tutorials will be descriptive and tell the user exactly how to interact with the program. In this case instead of creating a large manual at the start of the program we have successfully integrated it within all elements of the solution making it accessible to people of all experience levels. This also simplifies the overall navigation of the program as the user has important information that they need directly in front of them instead of needing to either keep the tutorial open in a separate page or navigate to the start of the programming if an error occurs.

The final output of the input layer therefore is the following: A weather dataset, a commodity price dataset and an advanced settings file. Both datasets should contain complete values and should be matching in size. Both datasets should also not include anomalous or missing values and should be in a correct format ready for processing in the next stage. The advanced settings file on the other hand should contain a two-dimensional array with all the hyperparameters and other settings. This file should be easily traversed to load the hyperparameters and other settings into the system and apply them.

In summary the input layer acts as the initial input of all the data into the program we can clearly see how all the data is retrieved, suitably preprocessed and then outputted. The outputs in this case being the inputs into the next processing layer.



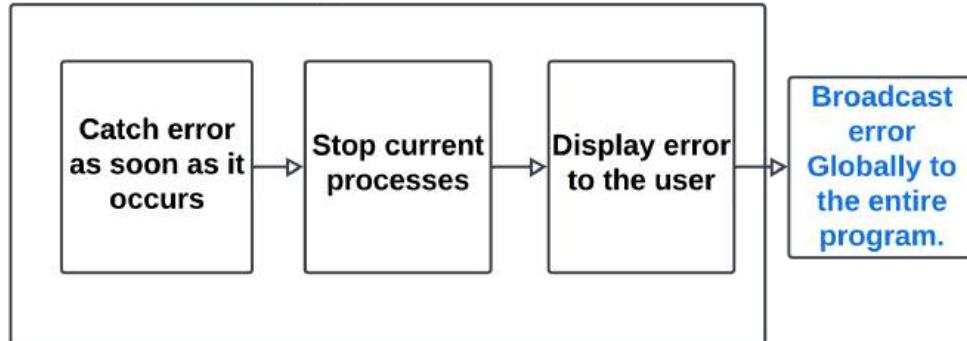
in this stage the model is trained. In this processing layer the main output is a trained machine learning model. We have decided to implement a support vector machine which is a supervised machine learning algorithm that is suited for classification. The exact details as to how it works will be discussed in the algorithm's section as the entire structure of the model is extremely technical, however we will discuss the overall processes that occur below.

The weather and crops commodity price datasets are used in conjunction with each other to draw conclusions between how the weather impacts the price of an agricultural commodity. The final output of this layer being a fully trained model which is able to classify the impact of forecasted weather on the future price of a commodity.

Firstly we need to initialise the first two datasets. This involves formatting the data appropriately and making sure it can be traversed effectively by the model. During this stage we could also implement validation techniques such as verifying that the complete dataset is intact and present. In this case we will validate all elements are present and the file is not corrupted.

As mentioned beforehand this stage is the most prone to an error occurring as it involves the most movement and processing of data in the background. Due to this we need to implement an error handling function throughout the entire program. This would connect directly to the processing page and display clearly to the user if an error were to occur. This error catches the error preventing it from corrupting the current action and closing the entire program. Pauses the current processes to prevent further errors from occurring. Then displays the error directly to the user and in all cases would broadcast the error globally. Broadcasting the error to be able to be detected throughout the entire program means that it can be embedded directly into the processing page without the need for separate error functions for different types of errors. Instead all errors can be handled by the exact same function. An error function that prevents the entire system from crashing when something incorrect occurs makes the system more maintainable because if a component doesn't work correctly in the backend such as an API returning incorrect information it can be handled directly by the user.

global errorFunction()



As we are planning to implement a machine learning algorithm within our program we need to scale our data. Scaling involves mathematically standardising data to having

a mean of 0 and a standard deviation of 1. The exact method we use to scale data will be shown within the algorithms component. Instead here we discuss the need to scale data. Scaling data means that all data points can be represented appropriately as they don't take extremely large values . Instead they are represented as values that are not as spread out which is what the standard deviation value defines. If we were to train the model on unscaled data it would have both poorer accuracy and a slower training time as the computer would be overloaded calculating relationships between extremely large numbers. In a sense scaling helps to summarise the data while also keeping the relationships the same.

Before inputting data into the mode we might validate again to verify that it fully meets the model's requirements and both datasets are able to be processed by the model appropriately.

As our model is a Support Vector Machine machine it is suited for classification tasks instead of prediction tasks. In this case this means that an SVM is able to label data in multiple dimensions instead of making an exact price prediction based on all of the time series data. Using an SVM to classify unlabelled data will mean we will input weather data into the model and it will output a label as to the impact on the price. For example if the weather is extremely cold and the crop requires high temperatures then we can assume that the crops price should drop as we can estimate a loss of yield. To tackle this problem we have looked into using a different approach that involves labelling the data ourselves as the change in the commodities price and then training the model using the two datasets. The training stage therefore would involve taking averages of each datapoint to construct a datapoint that takes a time segment into account. For example if we are looking at daily price and weather data. We could create a new segment which looks at the data in weekly chunks. Each chunk of weather data will show the average weather data over that period and each chunk of price data will show the average price data over that period and also a label of the change in price during that period. As SVM (Support Vector Machines) can classify data in multiple dimensions and with multiple labels we can classify data points with different labels. This means that we don't have to only look at if the weather has a positive or negative impact on the commodities price. Instead we can be more precise and look at percentage changes. Each chunk or bucket will be labeled accordingly as their change from the previous price for instance positive, negative, neutral or actual percentage changes.

Once everything is labeled and ready for training we can input this into our model. Taking inventory we know we have weather data for each time segment which includes all four variables which is cloud cover, temperature, humidity and precipitation. The four variables are called features. On the other hand we use the labelled price data and we call these variables labels. We can use both these features and a single label to correlate both the data and make the model able to differentiate between each data point. This model will also be trained using the specific hyperparameters that the original advanced settings file contains.

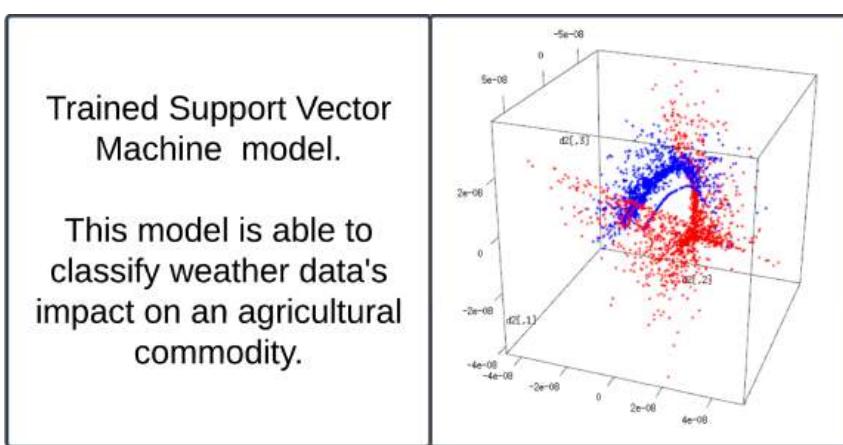
As we are training the model algorithmically this will be discussed in the algorithms section as this includes a machine-learning model within the structure diagram. The final desired output of this processing layer is a trained model. This model should be

able to correctly classify weather and label its forecasted impact on the price of the agricultural commodity.

Looking at the user facing proposed screen designs we can see that there are two main components that are included that will have to be implemented simultaneously to the purely computational part. This is the error function and progress indicators.

As discussed before we have defined an `errorFunction` which is able to handle all types of errors. Both preventing the entire system from crashing and also informing the user what just happened. A key part of this planned `errorFunction` is its ability to broadcast its messages throughout the entire program. This means that we can easily implement an error page within the processing page. This error pages function is to continuously detect errors using concepts such as threading to run two pieces of software simultaneously in python. Once an error is detected it can be sent to the user via that page preventing the entire page from crashing.

A progress indicator would also employ a similar threading technique. This means that the user can see currently both the percentage of operations that have been completed in total and also what is actually happening in the background. To do this we would use the same threading techniques to run two pieces of code simultaneously. Threading allows you to create two separate instances that run code at the same time this means we can train the model at the same time we output the progress directly to the user. Making a progress bar involves retrieving the total number of steps that a program is going to take and then finding out the current steps that have been completed. Using both these pieces of information we can construct a progress bar which shows how far along the program is. A progress bar will keep the user constantly aware that the program is running and how far along it is. Other progress indicators include highlighting what exact process is being run currently and displaying it to the user. Both of these progress indicator processes require threading, to gather up-to-date relevant information for the user.

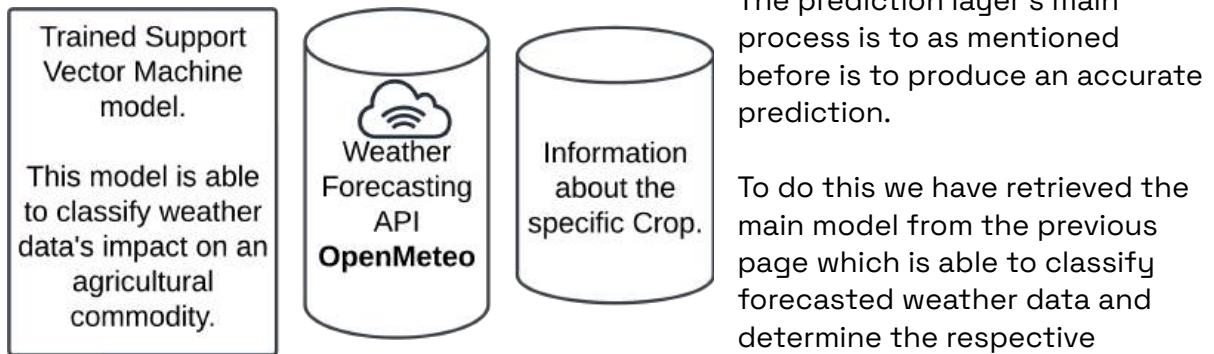


Ultimately we will have processed all the data and end up with a fully trained model. Alongside this our processing page will indicate that all the processes that need to be executed are complete and then the user will be redirected to the next layer. The outputs of the

processing layer is a fully trained support vector machine. In the image we can see a collection of points that are scattered through the vector space. A trained support vector machine should be able to easily distinguish and classify each point as being either red or blue just using their vector. So in essence all a support vector machine does is draw a line between all the data points and then use that to categorise data

with labels. The line in our instance as we are working in multiple dimensions is called a hyperplane. So effectively the only operation in this layer that we were processing was to find the multidimensional hyperplane in the weather data that can most effectively categorise it as one of the labels. As we plan to have multiple labels there will be multiple different hyperplanes splitting up the data. The final trained model we will export into the next layer may not need to be a file instead we can directly pass it from one function into another.

Predicting Layer :



commodity. Two other datasets we have access to is OpenMeteo's weather forecasting API which allows us to request weather forecasts into the near future this means we will be able to access future weather values for locations all around the world. The second dataset we need is specific information about the crop. This specifically would be an array with the coordinates of where the crop is grown alongside an array with the spread of quantity that each location grows.

As we did beforehand when retrieving and merging all the historical weather data into a single dataset we will need to do the same here. We need to individually send an API request to OpenMeteo requesting a weather forecast for that location as far into the future as possible. Currently OpenMeteo's tool allows weather forecasting multiple metrics up to 16 days into the future. Once we are able to successfully build and send the request, the program should be able to handle the replied weather dataframe. This dataframe contains future weather data, specifically the values of all the metrics over 16 days into the future. We will need to request 3 dataframes in total as there are 3 total major locations per crop.

Once we do this we can focus on merging all three datasets. As we approached before all we need to do is multiply each dataset by the percentage that it produces. Then we sum all the elements in the database to create a single merged database. Something we didn't mention beforehand is the usage of the same index for arrays that contain the location data and the spread of volume data. This means that we can quickly switch between a specific location and the respective quantity of produce that location produces.

We can only input time segments into our trained support vector machines so we will need to turn the merged dataset into a collection or a single time segment. This time segment will need to be the same size as our original data as we will be making a prediction using this information. Scaling will also have to be done as all the

training data had to be scaled so to reflect this process we will do it again to this piece of data.



We can now produce a prediction. To do so we load the model into memory and then pass through a single time segment datapoint. The output of this prediction would be a label as to the future price change of the agricultural commodity.

We could also experiment with our forecasted data as instead of creating a single time segment and therefore a single future price prediction we can split the data into multiple separate time segments and make a range of predictions.

As we can see in the image we can see that each individual weather data segment can be analysed to produce a separate prediction. This will give end-users an idea of how the price will move over the next 16 days.

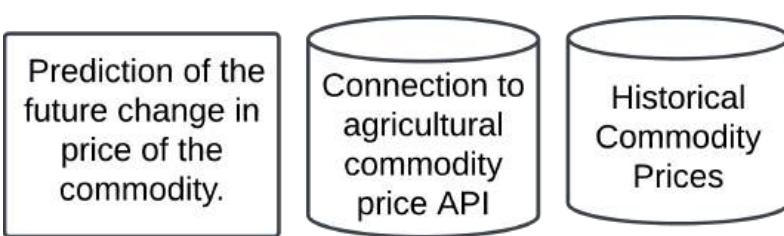
This process will involve converting multiple separate forecasted weather data points into numerical future predictions. To do this we will need to have a range of forecasted weather data points which OpenMeteo suitably provides. We then split the spread of weather data points into separate individual time segments for example, OpenMeteo provides 16 days of data. We can analyse data in segments of 4 days to gauge how the price will move in each 4 day period. Once we have each 4 day segment which contains the average weather metrics during those four days we can input it into the SVM model which will classify each segment and produce four different predictions. Each separate prediction will contain a predicted change in price during those four days. Using this we can create an array with the suspected price movement. Such as an array with corresponding daily time segments and their predicted change in price such as an increase of 0.5% or a drop of 1%. This would mean over a longer time period such as 16 days we can estimate how the price will move overall in finer detail than a single prediction.

One problem that we may encounter is that fundamentally predictions closer to the current date will be more accurate than data further into advance so this piece of software will produce better predictions when it is closer to the desired time so over a long term such as on the 16th day the software may have prediction limitations if we compare its relative accuracy to a price change within a week.

After this we can directly pass a tabular two-dimensional array directly to the next layer. This table will have either multiple time segments or a single time segment. As we are using OpenMeteo's maximum 16 days prediction time we can predict a maximum of 16 days in advance. So in comparison to other tools we can successfully try to implement a prediction tool that is longer than a week such as in IBM's

Environmental Suite and Vesper's prediction tools. This array will contain predictions of each time segment, the starting date and the length of each time segment. Using all this information and an index where we loop through all the values the function in the next stage should be able to interpret all the data and produce a suitable plot of all the commodity price data.

Outputting Layer :



During the output layer the main goal is to suitably output a prediction of the crops future agricultural price. From the preceding layer we already have the predicted movement in the stock's price so all we need

to do is apply this to a graph that contains crops commodity prices. The graph format should be clearly interactive and able to zoom and pan. Stakeholders also suggested having the ability to switch between different time scales without needing to zoom and pan into a specific segment.

All these features were implemented correctly within the proposed screen design so we need to summarise the features as to how to translate between the backend structure and the interactive frontend structure. As the design of the software uses limited static objects there is a need for our backend structure to be responsive to changes. For instance when the user pans to a specific time segment the graph should be correctly visualised alongside the prediction elements. The majority of the output page is taken up by the graph so that will be the main individual object that the user will be interacting with.



We will need to use either a single commodity price dataset or merge historical and current commodity prices to get a complete database of the crops commodity prices. With this information we can plot a time-series graph of the commodity prices over multiple years. This will mean that the user will not have to access other sources for price data instead it is all shown correctly on the user's screen. Plotting data correctly means that the entire graph is scaled and correctly placed so the

user has a suitable view of the entire graph. Looking at the image we can see the proposed screen design of the output page.

At the bottom of the graph we can see two predetermined interactions. An interactive time scale as well as an exit button. The interactive time scale allows the user to automatically pan and scale the graph to fit the desired time segment

appropriately on the screen. For instance the 5 day view currently selected shows the changes in commodity prices over the last 5 days. This disregards any information before this 5 day period so to access it you would have to select a larger time range like over 6 months to a year. Zooming and panning the graph will involve mathematically transforming it in the x, y axes. Zooming will involve either stretching it in the x, y axes to see more or less of the graph. This means we will also have to change the prices on the y axis of the graph and how it is scaled as the graph is different. Alongside this the x axis will also have to be scaled appropriately as you're looking at a different date range. Using predetermined interactions will have to also take into account what data points are plotted on the graph as you will have to manually select the most recent datapoint and predict from there. Panning the graph is easier as you do not have to rescale the data . Instead you are moving the graph linearly in either the x, y axes. Panning involves shifting the graph either left/right or up/down to view all the data. In this case the x, y axes will also have to react by showing older or newer dates within the x axis and the y axis will have to show larger or smaller price data points. In both these cases we need to highlight the fluidity of the system as it should be able to quickly handle and react to changes within the page.

This graph will need to accurately represent data as this is the only output of our entire program and will be the key embedded element that will display the estimated prediction for their agricultural commodity.



We have to then on top of this graph overlay a prediction. This prediction's main function is to show how the crops commodity price is forecasted to move in the near future based upon only future forecasted weather movements. The crops commodities price as a time series graph will be overlaid with a prediction. In essence what this means is that the user will want to have a general idea as to what will happen to the crop's price. To do this effectively we will take the outputted prediction from the previous processing layer and further process it. As mentioned beforehand in the processing layer the prediction could be implemented as two different variants. Firstly we could have a single prediction looking at the proposed screen design that we designed above we can see that a single line dictates the prediction. This prediction is just overlaid on top of market data and contains the suspected trend. This would mean if the



prediction is a single value we can estimate that overall the price of the agricultural commodity over the next 16 days will have a n % change. With n representing the percentage change in the commodities price. This may prove too simple so we could tackle it another way looking at how over multiple separate time segments the price will change. Looking at the screenshot above we can see an example of this type of prediction. Each labeled box represents a time segment and shows how each time segment changes with time.

Plotting this on the screen design we can see that predicting multiple time segments at a time and then merging them may prove more effective at discerning the day-to-day trends of the agricultural commodity. Predictions on the annotated screen design are between the 2 light pink boxes. Looking at this image we can see that the single prediction is unnecessarily vague and can't represent exactly how the commodity changes in price. This is referring to the single blue line that stems away from the commodity market data. In comparison the precise time segmented predictions prove more accurate and can more accurately show how the price changes over time. This is referring to the red / green lines following the historical commodity prices.

To implement the predictions within the graph we need to do the following: have predetermined time segments and corresponding numerical predictions for both. As we mentioned before with both these pieces of information we can output predictions with varying levels of detail. For more precise detail we can take smaller time segments and overlay predictions based on that. On the other hand we could also take larger time segments or even a single time segment and make a suitable prediction.

As described in the previous layer we will have access to an array with predictions of each time segment, the start date and the length of each time segment. Using all this information as part of the prediction dataset. We can plot predictions of the graph in more depth. To do so all we need to know is the starting date in the ISO8601 date standard and set that as our initial x coordinate. This x coordinate will be the current date today and will store the most up-to-date current price of the agricultural commodity.

Using the array of predictions which stores numerical percentage predictions across each time segment we can plot how the price is affected one by one by multiplying the previous price by the predicted percentage change to produce the new price. This means that we can take the most recent value and apply this n% change to produce the following price of the agricultural commodity after the end of that time segment. We can then plot this directly on the graph as we can work out the x coordinate using the length of each time segment and the index of this time segment compared to the first segment. For example if we know time segments are of length 4 days and we are predicting the price of the second time segment we know that it is 8 days after the starting date. So we can do date based addition to get the x coordinate of the predicted price.

To get the y coordinate we would just multiply the old price by the percentage change we predict which produces the price at the end of the time segment also

known as the new y coordinate. To retrieve the old price at the start of the time segment we would just find the time segment's starting date and what commodity price is held within the database. To retrieve the percentage change we would just match the exact time segment we are looking at with the value that is held in that index within the predictions array.

Doing this multiple times across each connected time segment we can construct a predicted future graph of points. Each coordinate represents a predicted price in that time segment. Once we have this data we can plot the respective line between the old price and the projected price across each time segment. This would involve using $y = mx + c$ the equation of a straight line alongside the region that the time segment determines to produce a bounded straight line. The bounds describe the region that the line should be located as it should be a finite line and not exist before or after this time segment. We can then repeat this technique using a for loop across all the following time segments to produce the allocated graph.

If the user is to zoom / pan / scale the graph as we have all the predefined commodity price datasets and predicted changes in price. There is no specific requirement to retrieve or process any new data. Therefore all we have to do is adjust the x,y axes appropriately to the correct scale as described beforehand and replot the data. This means that our tool remains responsive and can easily react to changes. Using an interactive graph also means that the user doesn't have to consult multiple different sources before it produces a result instead all the data is suitably presented.

Specific Algorithms we plan to Implement

Described the solution fully using appropriate and accurate algorithms justifying how these algorithms form a complete solution to the problem.

Algorithms are used throughout the process with the main bulk of algorithmic logic being within building the model itself. Other algorithms and algorithmic thinking will be implemented both before creating training data for the model and also during the output stage.

During the four-stage process Input, Processing, Predicting and Output we will need to break down what algorithms will be used. In the table below we can see what specific algorithms are used within each layer and the explanation and justification why they are implemented specifically.

Algorithms implemented within the input layer :

Algorithm	Explanation	Justification
Merging all three	This algorithm involves	A single representation of

Algorithm	Explanation	Justification
weather datasets into a single file	suitably merging all the datasets to represent the weather in all three separate locations as a single comprehensive dataset. This is done by multiplying the percentage of the crop that region produces relative to global supply and then adds each element within each database.	the crop's weather across all major regions means there is a single file that needs to be processed by the model which contains all the data so the model is unbiased to all the separate locations and can make a well rounded prediction.
Suitably amending the format of data	Formatting the data means that each file is usable and can be traversed correctly. For instance the file format of data that is retrieved from all the APIs and external sources may be different. This step means that they can all be read and compared without needing further conversion.	Data formatting ensures the ultimate compatibility with the model so we can successfully verify that the dataset is both readable and intact before we pass it onto the machine learning model. In this case any corrections will amend the dataset directly preventing it from needing further processing in the prediction layer.
Algorithms that detects errors within a dataset.	Anomalous or missing results will need to be fixed before we pass data on directly to the model to make sure the model is trained on data that is of a high quality.	To do this we need our program to be able to detect anomalous or missing values within a dataset and appropriately flag it.
Imputation algorithm that is able to fill in missing elements	Imputation is the process of replacing data with substituted values that are within a normal range. This could involve replacing anomalous or missing values with an expected element.	This means the program should be able to make an educated guess at what that data point includes as the machine learning model is unable to be trained on incomplete datasets.

Algorithms implemented within the processing layer :

Algorithm	Explanation	Justification
Standardizing all the training data by using a scaling function	This algorithm standardizes data by formatting it to have a mean of 0 and a standard deviation of 1. The mean is the average value of the entire dataset and the standard deviation is a measure of how spread out the values within the dataset are.	This ensures all the data points are represented appropriately for when it is used to train the machine learning model. This standardisation process means the model has a quick training speed and high accuracy when training compared to when we train it with unformatted datasets.
The machine learning model. A Support Vector Machine (SVM)	<p>This algorithm's main function is to classify how much of an impact the forecasted weather has on the future price of an agricultural commodity. This step comprises multiple algorithms that process data and create a model that is able to accurately classify data. The main steps that occur within the SVM are the following:</p> <ul style="list-style-type: none"> • Define SVMs hyperparameters and parameters • Generate a kernel function to split the data • Create a hyperplane to classify all the data <p>All three of these sections will have to be tackled individually to be able to produce a trained model that is able to classify inputted weather data.</p>	A Support Vector Machine is well-suited directly for classification tasks as we are working with weather variables with multiple metrics so being able to handle multiple dimensions is necessary and then correlating this with a single label is complicated. This means that we will have to implement complex algorithms to discern the data and are unable to use simpler methods such as a K Nearest Neighbour algorithm which we discuss more about below.
Grid search cross validation is a technique used to find and optimise the hyperparameters for	Grid search cross validation works by creating all the combinations of the hyperparameters for a model and then training and testing	The output of this grid search cv process is a collection of hyperparameters which are the most optimal for

Algorithm	Explanation	Justification
a model.	the model on every single possible combination of hyperparameters. The cross validation technique described a technique to evaluate the performance of a model on unseen data.	the support vector machine model so the user's model is always extremely optimised to produce the most accurate results and therefore predictions.

Algorithms implemented within the predicting layer :

Algorithm	Explanation	Justification
Data retrieval algorithm	<p>This algorithm should be able to retrieve forecasted weather data for all regions from the API. There will also be a need to retrieve current agricultural commodity data from the web using techniques such as web scraping which would be described under data retrieval and processing.</p> <p>A data retrieval algorithm should be able to first gather all the data either using an API and making a request or using techniques such as web scraping which would involve catching an instance of a website and analysing the HTML to extract data. In both cases the algorithm should also be able to handle the retrieved data.</p>	<p>This would mean that the model always has access to the most up-to-date weather and commodity price data. This is important because the model will use all the data that is available to it to make a prediction about the future change in price of the crops commodity.</p> <p>An algorithm that is able to do this successfully will mean retrieved data can be accessed and further processed easily.</p> <p>All of this will mean we can guarantee the model is trained on high quality data so it can therefore produce high quality insights.</p>
Dataset merging	Combining weather data for all three separate major locations for where the crop is grown into a single dataset keeps the type of data that the model receives to be consistent as the same process is happening	This simplifies the prediction process as we ensure there is both consistency of the type of data we input into our model and all the weather patterns spread across

Algorithm	Explanation	Justification
	on the data before values are directly predicted from it.	the world are represented correctly.
Standardising data using a scaling function	We standardised the training data so it is also necessary to standardise the weather data that we are going to input into our SVM model. This scaling function will also standardise the data to have a mean of 0 and a standard deviation of 1 as mentioned beforehand.	This will mean we can ensure that all the data points are correctly represented and in the correct format for our model. Improving both the accuracy of our predictions and keeping the processes we do to all data points consistent.
Using threading to show a processing bar at the same time data is being processed.	Threading is a technique to allow multiple separate programs to run simultaneously as separate threads. Each thread can be run independently of the others.	Utilising threading within our program will mean we can successfully show progress indicators such as what is happening in the background and show errors that may have occurred at the same time the model is being trained and data is being processed.
Applying the support vector machine to predict data	<p>The main output of the previous processing layer would be the support vector machine which is able to classify forecasted weather data and determine the exact predicted change in the price of an agricultural commodity.</p> <p>We will predict the change in price of the agricultural commodity as multiple units of time across the entire forecast. This will mean we will split data and produce a prediction of the future movement of the agricultural commodity within specific time segments. Each time segment will then have an allocated prediction stored in</p>	<p>The output of the classification of the support vector machine will be a single label that determines the predicted future change in price of an agricultural commodity this would be used to determine how the price moves in the future</p> <p>Using multiple predictions over the entire forecasted weather dataset allows us to see exactly how the commodities price changes over time. So the user is able to access predictions in higher detail.</p>

Algorithm	Explanation	Justification
	an array and passed onto the next layer.	

Algorithms implemented within the outputting layer :

Algorithm	Explanation	Justification
Time-series graphing of data. This would mean we appropriately plot datasets and data points on a graph.	This algorithm would go through the entire dataset and plot each as either a single data point. Or groups of data as a dataset. This would mean on the graph we can clearly see both the historical and current prices of the crops commodity alongside an overlaid prediction as to the future movement of the price.	This is implemented to allow the user to interact directly with the outputs of our program. The graph should be interactive and able to zoom / pan / scale across the entire chart. This allows the user to explore the data in more detail and use a time scale selector to switch between different time scales.
Overlaying the future predictions of the price and how it moves directly on the graph.	As mentioned in the previous layer this will incorporate the produced predictions and apply this onto our graph. This involves separating each time segment and producing a line that extends between the old commodities price and the predicted new commodities price across the entire structure. Doing this across all the time segments we can produce an accurate prediction. This will need to stay in the same location if we were to move or interact with the final output graph.	This prediction will show exactly how the crop's commodities price is forecasted to move in the near future directly based upon only the future forecasted weather. This allows users to see predictions in different levels of detail as time segments may be larger in length to try to capture how the actual commodities price will change in the near future.

While first developing the project we were looking at using a range of machine learning algorithms to implement and how we would implement each. This is because using “artificial intelligence” is a vague term that describes multiple different types of algorithms. Researching a range of algorithms we looked at implementing a K Nearest Neighbour algorithm to classify weather data points.

KNN works by taking historical data (Climate and Commodity data) and working on them together. It works by using a concept called clustering. In 2 dimensions if we have two groups of data points that exhibit similar characteristics they may be able to be classified by looking at other data points near that datapoint. For instance if we have an unlabelled datapoint and it is located near a lot of the same labelled datapoint we can label it appropriately. K Nearest neighbour expands on this idea, using the idea that similar data will form clusters together.

These Clusters are referred to as ‘neighbours’. We use a value of k which is an integer which refers to the radius that we count neighbours from. This means if data points are in between clusters we can draw clear conclusions between their outcomes.

Small values of k makes a model extremely sensitive to noise as if a datapoint is in between very close clusters there will be significant overfitting to data points. As the algorithm is only analysing a few neighbours. Whereas larger values of k will make a model less sensitive to noise, as it will smooth out smaller differences in between clusters leading to an overall less accurate result. The specific k-value will have to be chosen in respect to the dataset we have and how it is spread across multiple dimensions.

Different crops have different sensitivities to the climate as you will clearly see some crops are weather resistant with a limited yield impact by the climate such as corn whereas cocoa beans have a high sensitivity to the climate so will see greater fluctuations in yield due to the climate. Therefore the corresponding cluster spread throughout the data will be smaller as small changes in the respective climate equate to large changes in yield. In comparison, weather resistant crops will have large clear spreads of clusters making it easier to categorise data as it is clear that only large changes in the climate will impact the crops yield.

KNNs also work in multiple dimensions finding labelled points near a datapoint and using that to classify it even if graphically we can't represent it.

Overfitting may also occur as we are trying to force a datapoint to conform to what is around assuming that there are clear distinguishing factors between the weather and the crop's commodities change in price.

Researching KNNs further I stumbled upon a SVM which is a support vector machine. SVMs work better with high dimensional data as instead of using nearby data points to classify something we use a hyperplane to fully separate data points into different classes. This in our scenario seems more accurate and less prone to issues such as overfitting. Overfitting is the process when the model learns the training data too well so it performs well on training data, but isn't able to correctly classify unseen data correctly. Ultimately we chose to go through with implementing a SVM as we are working with data in multiple dimensions and SVMs are suited directly at catching non-linear relationships between data which should directly translate to a model that is able to classify data points at a higher accuracy.

Suitability and Technical Details behind Algorithms

Described the solution fully using appropriate and accurate algorithms justifying how these algorithms form a complete solution to the problem.

A list of all the algorithms we describe in detail within this section.

Algorithms we plan to implement in our program
Merging multiple weather datasets into a single dataset.
Detecting anomalous / missing values within datasets and using imputation to replace the errors within the dataset with a suspected datapoint.
Standardizing data using a scale function and also formatting the data correctly.
Training a Support Vector Machine to be able to successfully classify data.
Grid search cross validation is used to optimise the hyperparameters of the model.
Using threading to run multiple programs simultaneously as separate threads.
Data retrieval using Application Programming Interfaces and techniques such as web scraping. Alongside saving retrieved data appropriately.
Using a trained support vector machine to classify weather data points
Plotting time-series data on an interactive graph.

Merging Weather Datasets - How does this algorithm work? :

The main technical premise behind this algorithm is to combine all three separate weather datasets to produce a single dataset that can be used to recreate the approximate average climate affecting the produce.

We have decided to use pandas data frames within our solution as it stores data in a tabular form and can allow us to easily analyse and manipulate data without having to develop separate functions from scratch which may be less efficient. Also reading openMeteo's documentation panda's data frames are the default data type used when the API responds to a weather data request.

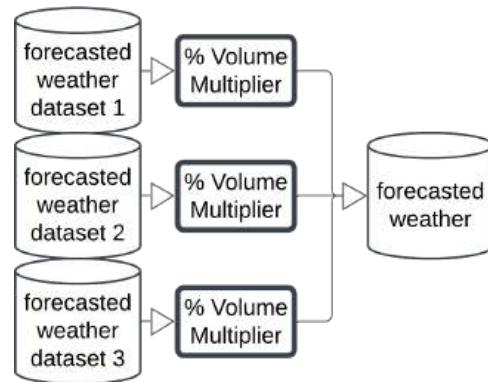
As we now have a standard data type between all three weather datasets this means we can consistently convert data using a single function without needing to handle inconsistencies within the data. This also means there is not a need to align the datasets timestamps as they will all have a common timeframe as our api

requested the same weather request for all three datasets. This is different when we compare both commodity price and weather datasets as we know they are from different sources so there is an added step of making sure we have a shared timeline between all the data at hand.

1. The input of the algorithm is either the three weather datasets themselves or the variable names of the three datasets. Alongside the datasets we have the percentage spread of volume across each region. This number signifies the amount of produce each major location produces. Multiplying each dataset by its relevant indexed volume percentage we get the weighted weather datasets.
2. We can now merge all the elements across all three datasets to get a single database. As all the datasets were assigned their appropriate weighting before we merged all the datasets, all we need to do is add each table together and we produce a single database.
3. This single database however might contain errors if we use a function to add all the values. This is because the date is often a string value and if we try to add multiple strings they will just concatenate together. To fix this we will replace the date with the date values from one of the datasets.
4. We can then delete the remaining dataframes as we do not require them within our program anymore.

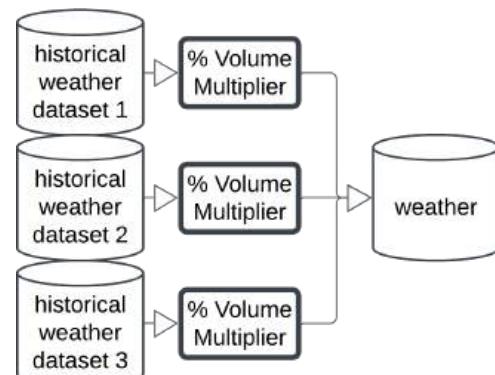
Merging Weather Datasets - How will this algorithm fit into our solution? :

This fits into our solution in two places looking at the structure we need to merge weather datasets before training our model to create a single weather dataset, while also being used when we are merging future forecasted weather data into a single database.



As mentioned before this means for comprehensive weather analysis the model will only have to classify a single weather dataset with a single commodities price dataset. This is because the SVM is only able to be trained on two datasets at a time so there is a need to accommodate this. This means that our overall system can save time by combining all the weather data into a single data instead of having to train the model three separate times on three different pieces of weather data.

Keeping our training of the model both more quick while also maximising the efficiency of our system as we are not wasting computational power.



Merging Weather Datasets - Justification of implementing this algorithm :

A machine learning model is only able to handle a single complete training dataset at once time. As crops are grown in multiple locations there is a need to represent the weather of crops grown in multiple countries and regions. However an issue arises due to there being multiple locations internationally where crops are grown, but only a single standardised commodity price for the crop. This means that it is necessary to represent all the weather variables as a single climate dataset which can be appropriately correlated with the crop's commodities price.

Merging all the crop's weather also reduces the total amount of data that is needed to be processed as only a single commodity price dataset and climate dataset need to be trained by the model. Optimising the efficiency of the entire system as it is able to train the model quickly. Additionally there is only a marginal loss of accuracy as the produced merged weather dataset accounts for the production output of each region to produce a single weather dataset that holistically represents the original 3 separate regions' weather trends.

Pseudocode:

```
volume_percentages = [percentage1, percentage2, percentage3]

#data1, data2, and data3 are already defined weather dataframes

data1_weighted = data1 * percentage1
data2_weighted = data2 * percentage2
data3_weighted = data3 * percentage3

merged_data = MERGE(data1_weighted, data2_weighted, data3_weighted)

merged_data.date = data1.date

DELETE data1, data2, data3, data1_weighted, data2_weighted, data3_weighted

OUTPUT merged_data
```

Detect and Impute anomalous or missing values within our dataset

- How does this algorithm work? :

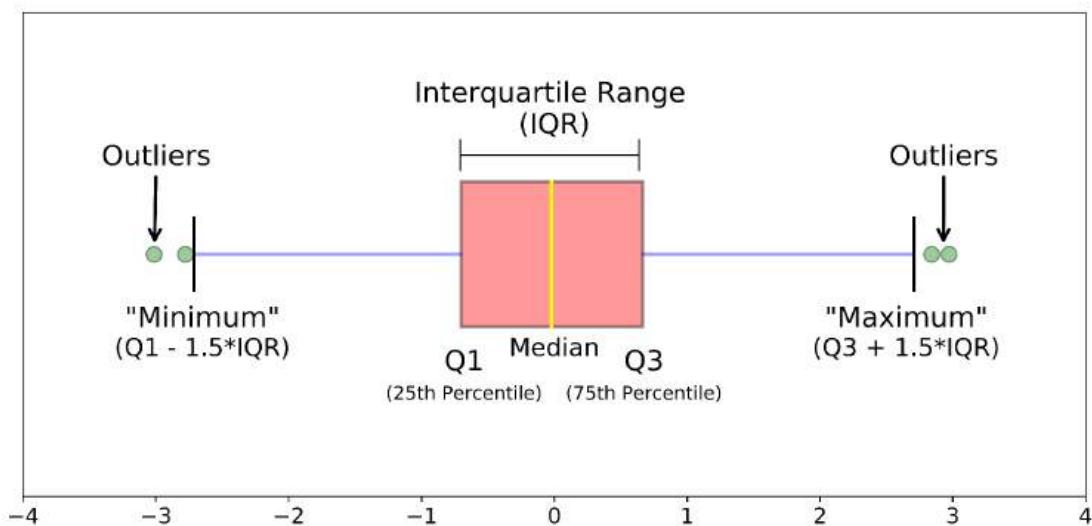
As we are working with only numerical data we can use statistical methods to help flag anomalous data. Missing values on the other hand can easily be recognised.

Missing values include empty data values such as using None or “ ” as well as placeholders for data that isn't present such as using “N/A” or “?”. These values will be specific to the individual dataset and will need to be coded appropriately to take this into account. Missing values can be quickly identified as missing.

To detect anomalies we can do the following :

1. Analyse and find the Interquartile range of all the data. To do this you would sort the dataset in ascending order.

- Find the 25th percentile value. This is the value that 25% of the data falls below it.
- Find the 75th percentile value. This is the value that 75% of the data falls below it.
- To find the Interquartile range (IQR) we can do $75\text{th Percentile} - 25\text{th Percentile} = \text{Interquartile Range}$. The interquartile range measures the statistical spread of data. For instance if data is extremely spread out the interquartile range would be large compared to data that has a small interquartile range which suggests it is not as spread out.
- After we know the approximate statistical spread of the data we can create a lower and upper bound for data. In this case if a value is larger or smaller than the respective upper and lower bound we can identify an anomaly or an outlier. We can define the lower and upper bounds using the following formulae. If a value is identified as an anomaly we can flag it appropriately to be directly fixed.



$$\begin{aligned} Q1 - (k \times IQR) &= \text{Lower bound (Minimum)} \\ Q3 + (k \times IQR) &= \text{Upper bound (Maximum)} \end{aligned}$$

Where,

$Q1$ represents the first quartile or the 25th percentile.

$Q3$ represents the third quartile or the 75th percentile.

k represents a constant factor that determines how high or low the bounds will be. A larger value of k means the respective upper and lower bounds require values that are more extreme to be considered an anomaly.

IQR represents the predetermined Interquartile range.

If a value is present either above the upper bound or below the lower bound we can flag it as an anomaly. If a value is missing on the other hand we can flag it appropriately as a missing value.

Imputation on the other hand occurs when an anomalous or missing value is flagged using the method we described above. Imputation involves replacing anomalies and missing values with appropriate substitutions. These appropriate substitutions are data points that are within the appropriate range for the entire dataset and the exact datapoint.

1. First we need to retrieve the location of the data point we want to impute.
2. Using the location of the data point we can find the column that contains this anomalous or missing data point.
3. As we are using mean imputation all we need to do is take an average of the observed values within the same column. If our dataset is extremely large or if the dataset has large fluctuations within the data points we can focus on finding an average of the observed values to only nearby elements. For instance only taking an average of the 10 elements near the specific datapoint.
4. Once we have the new value we can directly update and save the data frame as our value should replace the erroneous datapoint correctly. In this case we are using mean imputation to present a result.

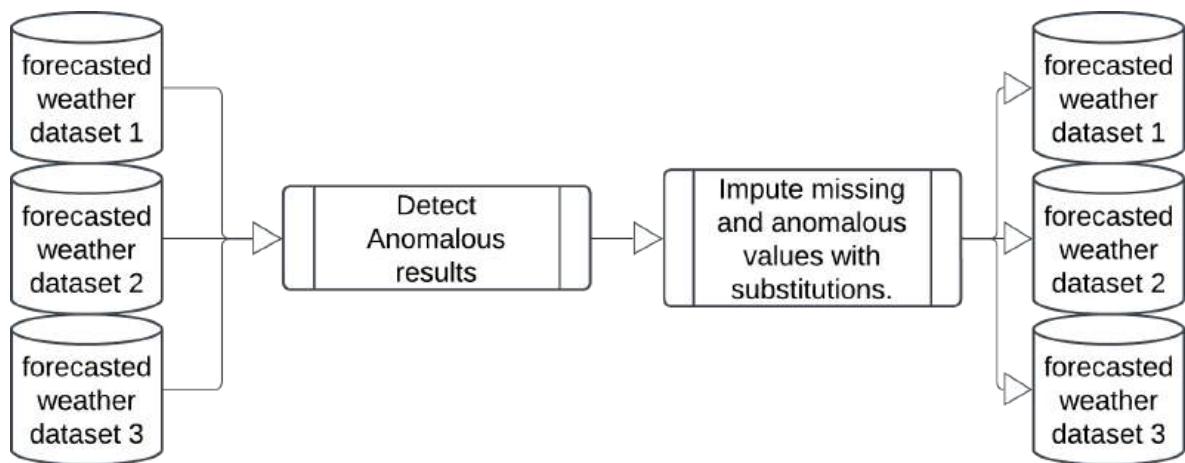
Other methods of imputation include using a linear regression model to produce a value. In this case we can directly use the other values within the datapoint that are present and then fill in the missing value using a linear regression algorithm. A linear regression algorithm works by finding a linear relationship between all the data points. This would involve drawing a straight line that fits the data in 3 dimensions. Using this line with our present values we can replace the missing value with a predicted one that takes into account the data included within the datapoint instead of an average value.

Both methods of imputation such as using mean imputation or a linear imputation involve trying to generate data that fits a datapoint so the model has a complete dataset. As we expect the databases to be mainly intact and complete with values we won't require imputation across a large proportion of the dataset instead it is used as redundancy to prevent missing or anomalous values from getting through the entire dataset. This is required as a support vector machine model can't be trained with an incomplete dataset as it is unable to handle missing values. Also retaining anomalous values within our data set may make our support vector machine biased as using extreme values within our data points may confuse the model making it perform worse overall. Ultimately a complete and robust dataset is an advantage both for the model and its subsequent predictions.

Detect and Impute anomalous or missing values within our dataset

- How will this algorithm fit into our solution? :

This detection of anomalous or missing values algorithm will be used throughout the entire program. This happens at two major stages, during the input layer as data is being retrieved and during the prediction layer as data is being fed into the model to produce a prediction. During data retrieval this algorithm is applied before the merging of the datasets to ensure that any anomalies or missing values are extracted before the data is handed to the forecasting model. What this means is that data is checked for any sort of potential erroneous values before it is fed directly into training the model. During data prediction before any datasets are given to the model such as forecasted weather data it will need to be checked for any missing or anomalous values that will negatively impact the prediction capabilities of the final model. Handling data before it enters the model means that it is thoroughly cleaned which ensures that valid predictions can still be produced even with imperfect input data.



Whenever the detection function flags a value as either missing or anomalous another function will be called to handle this instance. In this case during either data retrieval or when datasets are being inputted directly into the model the imputation element may be called.

In this case imputation will mean either a datapoint within a weather dataset will be substituted accordingly using data across the entire dataset. Or a datapoint within a commodity price dataset will be substituted accordingly. Imputing a value within the commodity price dataset will be different as it is time-series data so it will need to be a price between the previous price and the proceeding commodity price.

All of these processes will mean the forecasting model is trained and uses clean and valid data. Ultimately handling any potential issues that are caused by missing or anomalous values before any data is used.

Detect and Impute anomalous or missing values within our dataset - Justification of implementing this algorithm :

When training a machine learning model the dataset has to be complete; this is due to the mathematical model not being able to make predictions on data points

that have incomplete outputs. In this case only data points that have all the necessary values are used to train the model. Additionally datasets with anomalous values such as extremely large or small values that are outliers outside the normal range introduce bias while the model is training. This is because they distort the original pattern causing the original model to become less accurate.

This is more important due to the fact we are using external data sources so it is necessary to validate that they are complete and accurate. In this case we can use a subroutine to detect anomalous or missing values and a second subroutine to replace it with an appropriate substitution so overall the dataset is more representative of the original environment in the case of climate data or commodity prices. Ultimately this means that the model overall when trained on all the data points performs more accurately.

Pseudocode:

```
datasets = [dataset1, dataset2, dataset3]

FOR EACH dataset IN datasets:

    Q1 = calculate_percentile(dataset, 25)
    Q3 = calculate_percentile(dataset, 75)
    IQR = Q3 - Q1

    lower_bound = Q1 - (5 * IQR)
    upper_bound = Q3 + (5 * IQR)

    FOR EACH value IN dataset:
        IF value IS MISSING OR value < lower_bound OR value > upper_bound:
            flag_value(value)

            column_mean = calculate_mean(get_column(value))
            replace_value(value, column_mean)
        ENDIF
    ENDFOR
ENDFOR
```

Scaling data - How does this algorithm work? :

Scaling data is a function used to standardise a dataset. This standardisation transforms the data so that it has a mean of 0 and a standard deviation of 1. The mean is an average of the values within each column. If a dataset has multiple columns then each transformed dataset's columns will have a mean of 0 and a standard deviation of 1. The standard deviation is a statistical measure to how spread out the dataset is.

Making the mean of the data 0:

1. Find the mean of each column also called a feature.
2. Subtract every element within the column with the respective mean of that column. This means the data is centered at 0.
3. Verify that this process has correctly happened by checking the mean of each column is equal to zero. This means that the data is centered appropriately. Repeat this same process if the data is not centered.

Making the standard deviation of data 1:

1. This is done after the data is centered so each feature / column has a mean of zero.
2. The formula below is used to find the standard deviation of each feature / column.
3. Once we know the standard deviation of each feature / column we can scale all the values appropriately by dividing the value by the standard deviation of each datapoint to produce the standardised data.
4. We can verify all the data is correctly scaled by making sure the mean of each column is equal to 0 and the standard deviation of each column is 1.

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

(σ) or *Sigma* is used to represent the standard deviation of each datapoint.

N is the size of the entire population in this case the amount of values in each column.

x_i is the individual value.

(μ) or *Mew*. is used to represent the mean of each feature

Summarising this entire formulae into a single line of code we need to scale the data using the following function.

$$Z = ((x - \text{mean}) / \text{std}(x))$$

Where Z represents the scaled value.

X represents the original datapoint.

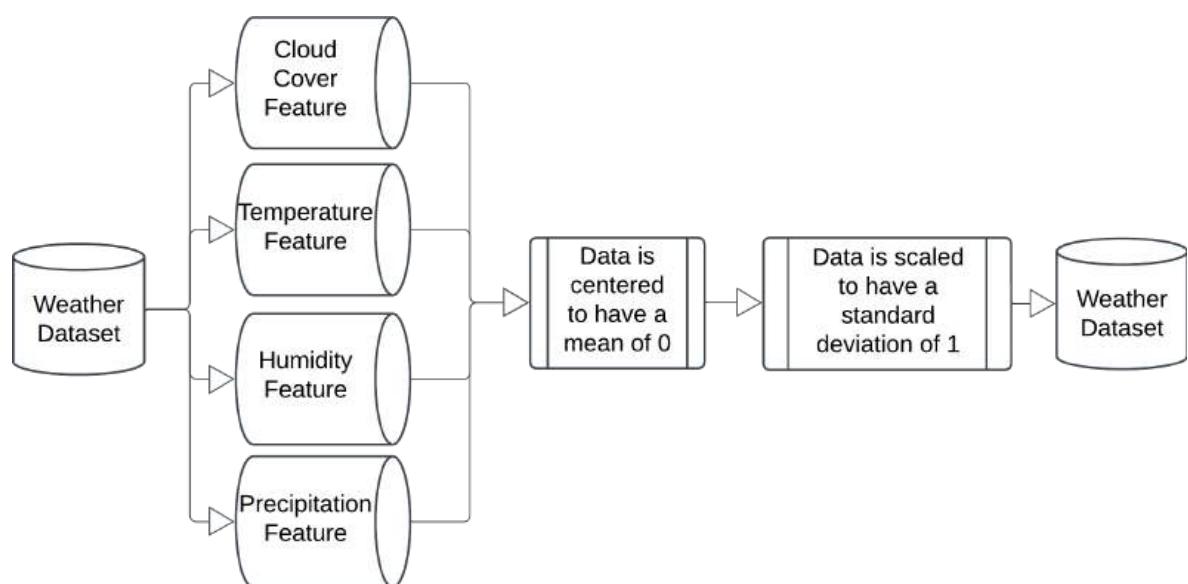
Mean is the mean of the feature / column.

And std(x) is the standard deviation of the original data point compared to the feature.

Scaling data - How will this algorithm fit into our solution? :

All the data before it interacts with the final model will have to be scaled to help reduce the training speed of the model and the overall accuracy . This means that any data that will come into contact with or will pass through the support vector machine will need to be scaled as our training data will have been scaled the same way so therefore any input data to make a prediction on will also need to be suitably scaled.

By incorporating a scaling algorithm we can correctly identify and work with the requirements of the support vector machine to achieve better performance and produce accurate predictions of how agricultural commodity prices will move in respect to changing weather events.



Detect and Impute anomalous or missing values within our dataset - Justification of implementing this algorithm :

Scaling is required by the machine learning model. This is to reduce the overall computational power required and also improve the accuracy of the machine learning model. As we have multiple weather variables in the single weather dataset including precipitation, temperature, humidity and cloud cover. Each of the variables has a specific scale of values that may introduce bias as large values within the weather variables might skew the model into producing inaccurate predictions. As the large value causes the model to prefer it in comparison to a variable with a smaller range of numbers.

In comparison by implementing scaling of the entire climate dataset before it is trained all the data values have a mean of 0 and a standard deviation of 1 making them all represent the same initial conditions, but are scaled so they all have equal weights and impact when training the final model. This limits biases and makes the final model more accurate mitigating the impact of outlier values.

Pseudocode:

```
INPUT data

# Center the data
FOR EACH column in data
    sum = 0
    FOR EACH value in column
        sum = sum + value
    ENDFOR

    mean = sum / LENGTH(column)

    FOR EACH value in column
        value = value - mean
    ENDFOR
ENDFOR

# Scale the data
FOR EACH column in data
    std_dev = standardDeviation(column)

    FOR EACH value in column
        value = value / std_dev
    ENDFOR
ENDFOR

OUTPUT data
```

Support Vector Machines - How does this algorithm work? :

The main concept behind support vector machines SVMs are as a supervised machine model that is used for classification tasks. In our case supervision refers to the process of providing a model with labeled data during the training stage and then allowing the model to understand how the data relates to the correct classification. Classification is used to assign unseen data points to specific categories or classes. The model wants to distinguish between the main features between the data and be able to successfully classify it.

In our case we want to implement a Support Vector Machine algorithm to classify the weather's impact on the price of an agricultural commodity.

Looking at our previously planned structure implementation of the model we can see how we plan to implement a support vector machine. This was highlighted in the structure stage of the design section and located within the grey machine

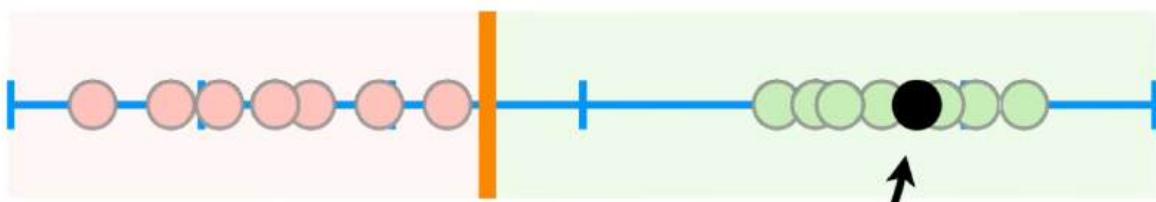
learning model box, this contained functions that had to be implemented individually and had very specific functions that we describe in more detail below.



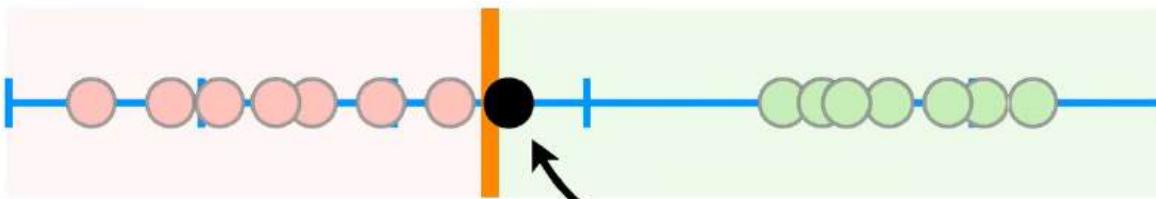
Before we look at our implementation of a support vector machine let's first describe how our support vector machine works internally.

As we mentioned SVMs are used to classify data. Looking at an example dataset above we can see a range of values that describe multiple observation values in one dimension. The colour of each observation indicates the label of the data. In our case the features represent the values that the data holds and the labels represent the actual classifications of the data.

Looking at our dataset we can segment it appropriately using a single plane. This plane in one dimension can be used to segment the data. When we get an unseen datapoint we can use our chosen threshold to classify the data.

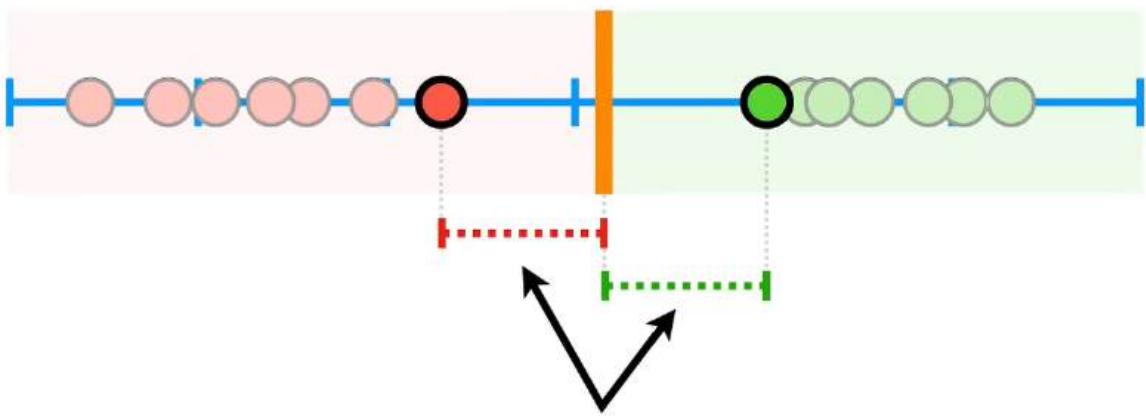


In this case the threshold has been used to segment the red and green classes so when an unseen observation was located within the green segment it is easily classified as a green datapoint.



However, this method has an issue because if the observation was located here we wouldn't be able to classify it appropriately. This is because the threshold is incorrect looking at the data as the observation is clearly closer to all the red labels and further away from the cluster of green labels, but because of the location of the classifier it is classified as a green datapoint.

Instead, to produce a more suitable threshold we can take into account the data points on the edges of our solution to produce a more suitable margin across the data. Taking into account the midpoint of the data on the edges of the cluster we can produce a more accurate threshold or plane to separate all the data. The shortest distance between the observations and the threshold or plane is called the margin. Looking at the image below we can see the margin that is labelled. We want a margin that can both accurately segment data and avoid misclassifications. We discuss how we optimize the values of hyperparameters to produce well defined margins using a Grid Search Cross Validation algorithm below.



In our case using a single threshold / plane to classify datasets will not always work as we are assuming data will be distributed in a clearly segmentable pattern.



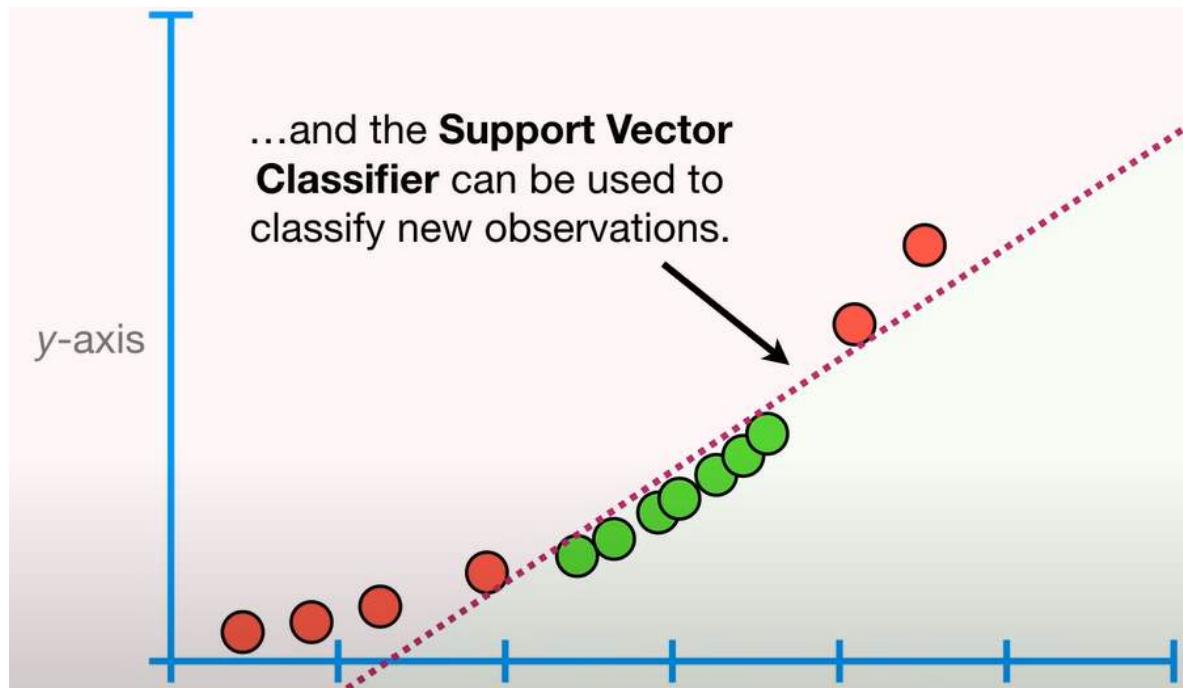
Looking at a dataset below we can see an example of this. Data is spread across the entire dataset in a non-linear fashion with a cluster of green labelled dots in the centre and red dots located to the side. In this case it may not be completely clear as to how to separate data using a single plane as misclassifications will be inevitable. This is because a single plane can only separate data in one dimension in one way. This means that one side will have to correspond to a red label and the other side will have to correspond to a green label. This clearly will not work with our dataset as it is not segmentable in that manner. In this case we can use support vector machines to classify the data. Looking at our original structure we will do the following.

First we can't use a single plane at the moment to classify the data so we use what is called a kernel function to mathematically transform the data.

As our current dataset above is only one-dimensional we can use what we call a kernel function to transform each datapoint into a higher-dimensional space. In this case this will mean we can use this specific function to add another axis to our dataset transforming it into a new dimension. There are a range of types of kernel functions we can use to in effect separate the data however to explain the concept in this case we have squared the x-coordinate of the data. This transforms the data into what we can see below.

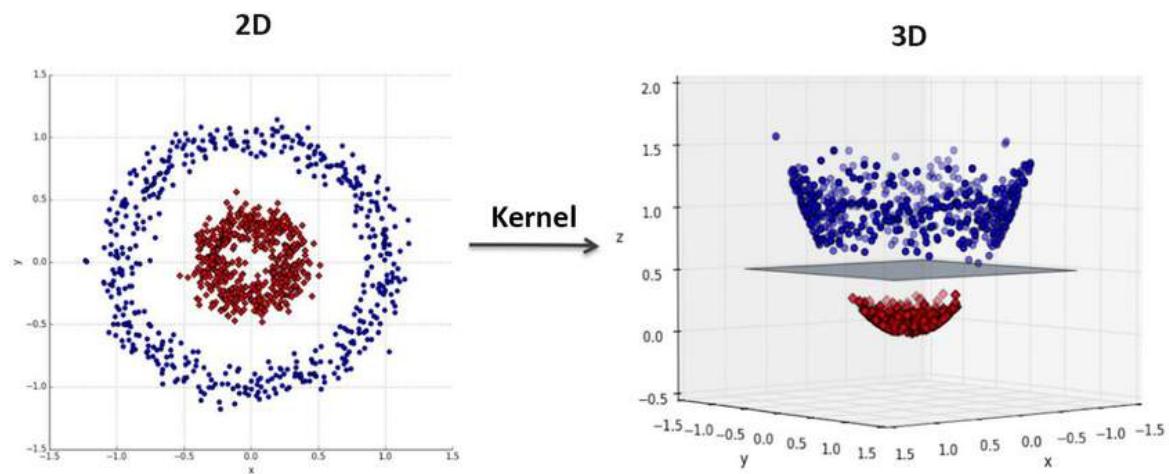
We can then use what we call a Support Vector Classifier to create a plane to classify the respective data. In this case we can now easily separate the red and green points using a single line as they can be represented as distinct segments instead of being intertwined with each other in the previous step. There are multiple types of kernel functions of different complexities which we can implement which we discuss below.

The Support Vector Machine is the specific combination of how the support vector classifier leverages the power of the kernel function to solve a range of classification tasks.



The Kernel Function :

Looking at the image below there is a range of different kernel functions the main role of these kernels is to capture patterns of varying degrees of complexity. They all use different methods to get to the end-result so make them suited for a multitude of different applications depending on how a dataset is distributed. However, they all have the same purpose in capturing the patterns within data and extracting them to clearly show how data is distributed in an added dimension.



In the image above we can see a similar example, but in two-dimensions in this problem we are involved with non-linear data and are unable to produce a

hyperplane to split it. A hyperplane is the technical term we use to describe a plane that segments data. Mapping this data into an added dimension means we can easily find a linearly separable boundary between them allowing us to computationally segment and classify data.

Another reason we use a kernel function is their applicability in a range of situations such as in more than three dimensions. This means that we can classify data in more dimensions using a kernel function. Multivariate classifications rely upon the same fundamentals of using a kernel function to segment the data in an added dimension and a support vector classifier to classify the data as a whole. This whole process comes under the umbrella of a support vector machine which we will implement. Now we will look into specific kernel functions and how they work.

1. Linear Kernel :

$$K(x_1, x_2) = x_1^T x_2$$

Linear kernels are the simplest kernel function and works by creating a single decision boundary by linearly separating data into two dimensions. This is accurate for data that is linearly separable, but won't be able to capture the patterns within non-linear datasets.

2. Polynomial Kernel:

$$K(x_1, x_2) = (x_1^T x_2 + r)^d$$

A polynomial kernel works by introducing polynomial terms to create a more complex decision boundary. These can create flexible curves in multiple dimensions, however due to its adaptability will mean that overfitting may be an issue as it is able to in essence memorise the training data instead of trying to understand the underlying patterns.

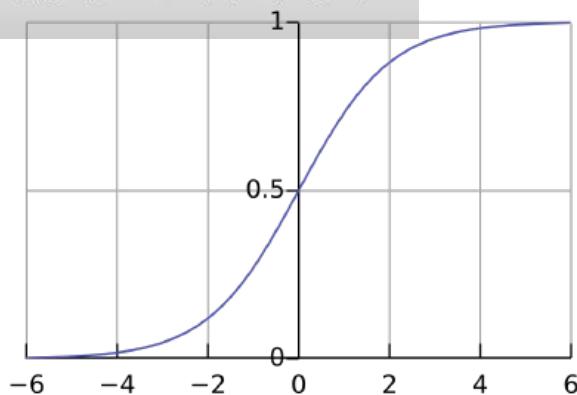
3. Radial Basis Function (rbf) Kernel :

$$K(x_1, x_2) = \exp(-\gamma \cdot ||x_1 - x_2||^2)$$

A radial basis function is the most popular and widely used kernel. It works by creating circular boundaries around data points to group them and separate them in multiple dimensions.

4. Sigmoid Kernel :

$$K(x_1, x_2) = \tanh(\gamma \cdot x_1^T x_2 + r)$$

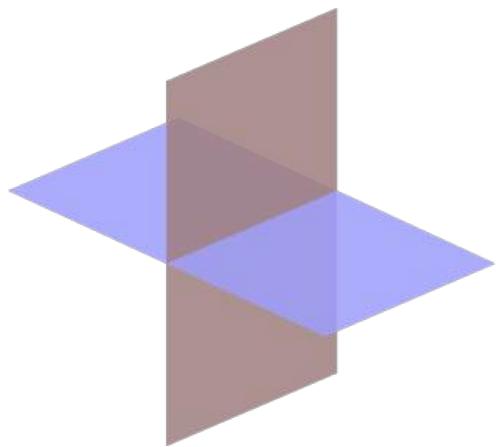


A sigmoid kernel can introduce non-linear decision boundaries. A sigmoid function looks like the image below and tries to separate points by separating points based on where they fall on the curve. For instance extremely large values within the sigmoid function produce the value 1 whereas extremely low values within the sigmoid function produce the value 0. Using multiple sigmoid functions we can classify multiple features and labels at once separating data using complex boundaries.

Support Vector Classifier:

A support vector classifier on the other hand uses this transformed data and finds the linear relationship between the data. It tries to find a linear hyperplane to

separate the data as the kernel function has already suitably separated the data into clear segments that can easily be classified.

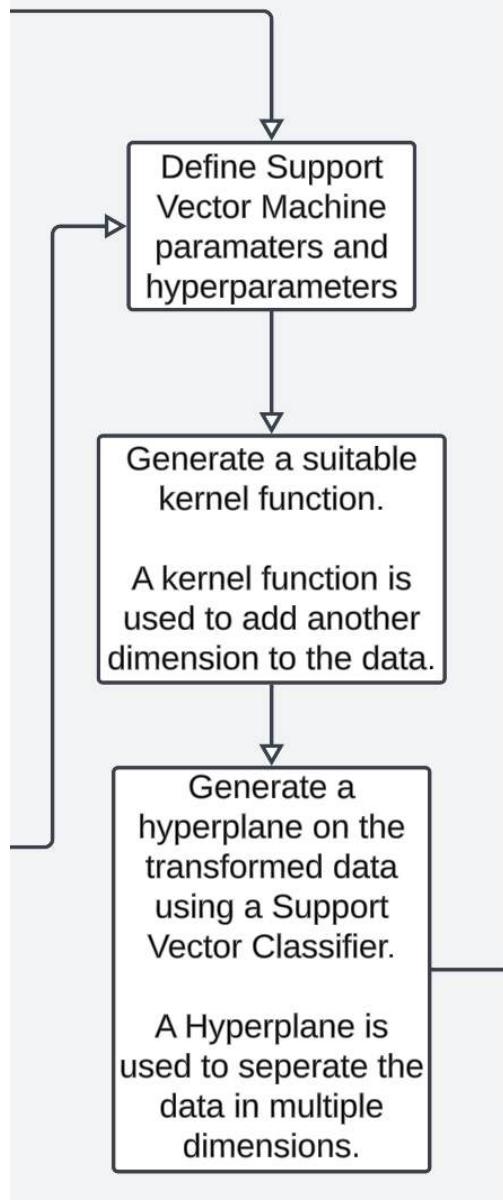


A support vector classifier just decides on the most optimal decision boundary to separate this already transformed data. In this case this can be a three-dimensional hyperplane to a hyperplane in any dimension as the SVC only operates within the kernel function's transformed space.

Support Vector Machines - How will this algorithm fit into our solution? :

The Machine-Learning Model

This component will be explained in more detail in the following section



In our case we want to implement a Support Vector Machine algorithm to classify the weather's impact on the price of an agricultural commodity.

This is our intended proposed structure of our entire support vector machine model. Here we include a clear definition as to what happens within the entire algorithm based on the fundamentals we described above.

1. Hyperparameters are set before the training process and influences how effectively the model will learn. Parameters refer to internal values within the model and are changed during the training process.

2. A suitable kernel function is generated to segment all the data. As described above in our use case as we have multiple weather features and multiple variables we will be operating our kernel function in multiple dimensions.

3. We pass the linearly segmented data to a support vector classifier to create a hyperplane that is a decision boundary to separate data points into different classes. In this case we are classifying whether the weather and whether it has a positive or negative impact and to the specific extent on the price of the selected agricultural commodity.

Once that entire process is complete we produce a single model that is able to classify weather data and its exact impact on the price of an agricultural commodity.

Support Vector Machines - Justification of implementing this algorithm :

Support Vector Machines (SVMs) allow the processing of multiple weather variables at once. This is because it represents each datapoint as a vector quantity and then uses a kernel function to segment the data which can then be classified and a prediction outputted. The kernel trick allows complex

relationships in multiple dimensions of data to be captured. As opposed to linear models which capture linear relationships between data which inherently limits the models ability to understand the interlinked variables and prices.

Additionally our dataset is summarised as a series of data points instead of over a time series so can easily be represented as individual vector coordinates and an attached label as to the impact it had on the agricultural commodities price. The final output of the training of the model is a Support Vector Machine which is able to accurately predict the future price movements of agricultural commodities.

Pseudocode:

```
gamma = 1
c = 1
kernel = rbf

SVM = new svm

DEFINE svm(hyperparameters, data):
    kernelFunction = generate.kernel_function(data)

    data = kernelFunction(data)

    hyperplane = generate.hyperplane(data)

    model = (kernelFunction, hyperplane)

RETURN model
```

Grid Search Cross Validation - How does this algorithm work? :

Grid search cross-validation is a technique used in machine learning to find the most optimal hyperparameters for a support vector machine model.

This involves finding the best combinations of hyperparameters for a model by testing and evaluating a range of combinations of hyperparameters. In the image we can see the accuracy of a model across multiple different combinations of gamma and C value. These are hyperparameters set before training the support vector machine model. The validation accuracy refers to how well the model actually performs on the test data.

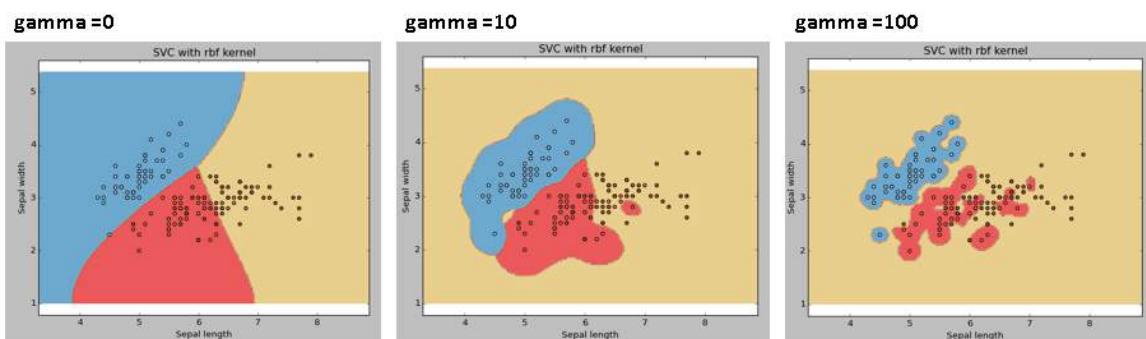
In this case the gamma hyperparameter in the context of a SVM controls how spread out the hyperplane will be and how it will conform and contour to all the data points.

So a high gamma value looking at the image below will cause the SVM to create a hyperplane that conforms to individual data points extremely well. This means

that a high gamma value will mean the hyperplane has an overall low spread across the entire dataset as it is specifically contouring to cover all the data points.

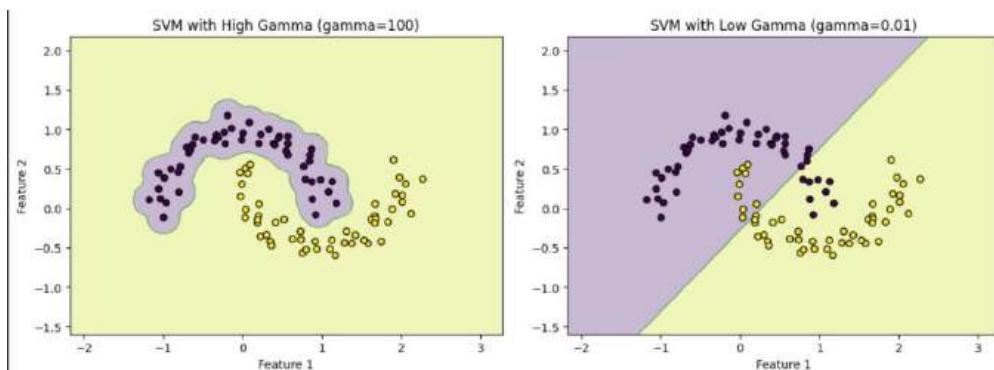
On the other hand a low gamma value, looking at the image below will cause the SVM to more broadly spread across the entire data so it isn't as specific at covering the points. This means that it covers a wider range of points across the entire graph.

Looking at how different gamma values affect a two dimensional hyperplane across the data. We can identify that the gamma value directly controls how effectively the hyperplane which as described before contours and spreads across the data. Different datasets might have a different spread of data so we will need to optimise these values specifically.



In this image we can see a selection of different gamma values and their impact on the SVM creating a hyperplane to segment the data. We can clearly see that having a low gamma such as gamma = 0 creates a wide hyperplane that is spread across the entire graph.

Whereas looking at gamma = 10 and gamma = 100 the produced hyperplane used to segment the data contours specifically to the data points. This means that it inevitably covers less area around it and instead focuses on individual data points.



Another hyperparameter which we may optimise is the C value. The C hyperparameter value is also known as the regularization parameter. A correct C value means there is a balanced trade-off between maximising the margin between the data and minimizing the amount of misclassification.

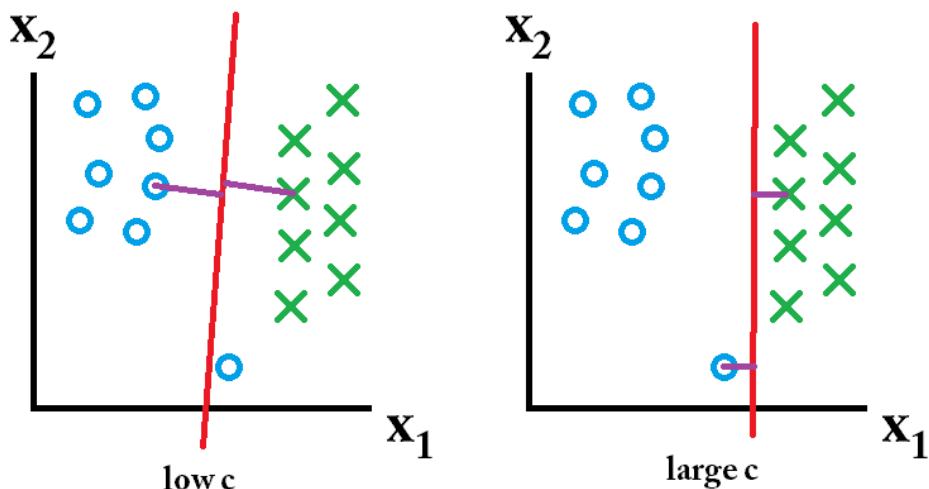
Maximizing the margin means there is as wide of a gap between the hyperplane and data points of different classes. This will mean data is clearly segmented. A wide margin will also allow the model to by this large generalization classify unseen data more accurately. However a large margin as it is generalizing the data into large chunks it is unable to make out very specific examples. For instance if data is located close to each other using wide margins may cause misclassifications as data is unable to be clearly segmented appropriately.

Minimizing misclassification on the other hand, refers to reducing the amount of incorrectly classified data that the SVM produces.

Therefore our aim is to find an optimum C value that both maximises the margin between data points and minimizes misclassifications.

So how does the C value directly influence the Support Vector Machine :

C values are used to create margins or hyperplanes between data.



Using a small C value makes the model prioritise creating a wide margin between the data. This allows some misclassifications within the training data however aims to clearly segment data. As said beforehand this will mean the model is better at generalization as it has a wide margin and is unable to capture specifics within the decision boundaries. In the image above we can see what happens when we use a low c value. The wide margins means that most data is correctly classified as the model tries to clearly segment it.

Using a large C value on the other hand makes the model prioritises creating a narrow margin between the data. This prioritises correctly classifying all the training data as we are trying to fit the margin exactly throughout the data. In the image we can see an example when the C value is set to a large c value. In this case we have created a divider that has a small margin so the specifics of the dataset are retained. An issue this may cause however that has been mentioned before is overfitting.

When using both these hyperparameters to train our SVM it may be tempting to use extremely large gamma and C values, as it is clear that the data will be clearly partitioned using precise hyperplanes. This may be evident in our training data as the SVM can avoid misclassifications and fit data accurately.

This pitfall of SVMs and other machine learning algorithms is called overfitting. As discussed beforehand overfitting is the process when a model performs extremely well on training data however when it is evaluated on unseen testing data performs poorly. Ultimately this is because the model due to the large C and gamma values has memorized the patterns within the training data instead of understanding how the training data is spread across. This extreme sensitivity seems counterproductive as if a model is able to capture every single pattern it should perform better, however in the real world this is not the case as it is inevitable that it misclassifies data as it can't replicate the real world randomness of data and extract the more obvious patterns within it.

To handle this we need a way to find a balance between the overall precision and sensitivity of a model. The core idea therefore is to continuously test and evaluate a model based upon unseen data as well. This is where cross-validation is implemented. Cross-validation is similar to making a model do a practice “mock” test before it is given a real examination.



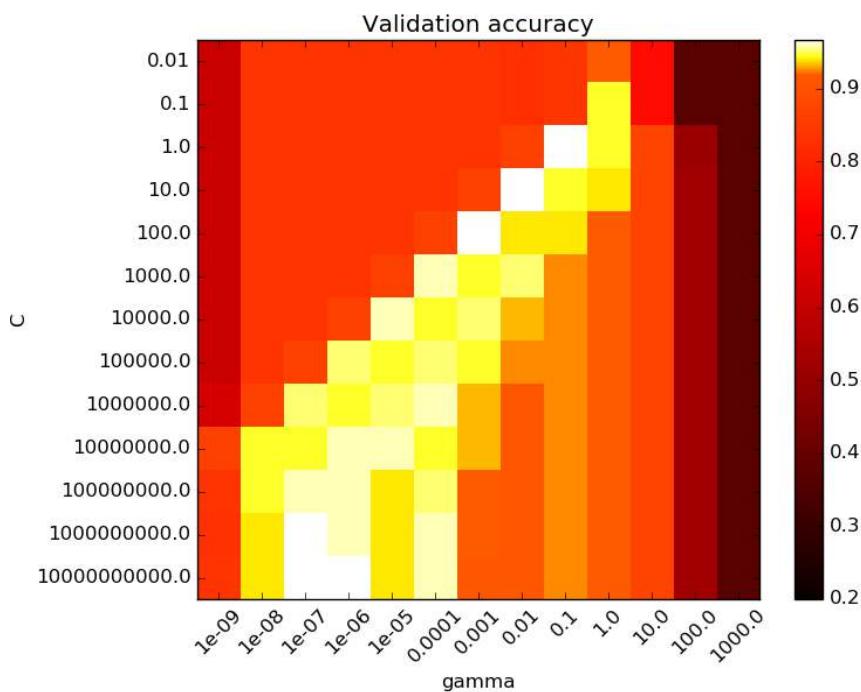
What is Cross-Validation ?:

1. First we split all the datasets into multiple separate subsets. Each individual subset is called a fold. Each fold is the same size and contains a different piece of data. If you combine all the folds together you will have the entire dataset.
2. When training the model instead of using the entire dataset for both testing and training we can now use a combination of folds and then test the dataset on an unused fold. As we have multiple separate subsets of the dataset as folds. We can use multiple different combinations of folds to iteratively train the model. This means that all data points are used for both

training and testing data at some point within the program. This will mean that the model will be trained and tested on multiple different combinations of folds and tested on the unused fold. This process happens iteratively so the same model will be trained and tested on fold 1 and then onto the subsequent fold.

3. Using this method we can evaluate the overall performance of the model as we know it is unbiased as all data points at some point will have either been used for training or testing the data. We will evaluate the model at each fold measuring the accuracy on the testing set of that individual fold. For each trained model's metrics we can then average all the accuracy data from each fold to get a more reliable figure as to how the model performs based on different data variations. This would be in comparison to using a single test-train-split or testing and training with the same dataset. This is because we can elegantly reflect how the model performs on multiple different variations of the data and not a single accuracy indicator.

Ultimately cross-validation works by reducing overfitting of the data as we are training and testing using all the possible combinations of data. This means that if the model performs well on training data, but not well on the testing data we can use cross validation to realise that the model is overfitting and not reliably understanding the data. This method has an extremely robust method to evaluate all the data as we are using a combination of models to give accuracy results instead of relying on a single accuracy result produced by a single model. However the main reason we are implementing cross-validation is to find the best hyperparameter. This is because we can try multiple different hyperparameter values that are prone to overfitting such as an extremely large gamma and C values and produce a significantly less unbiased evaluation as to the model's success in classifying new data points. This is because if we evaluate an SVM that uses extremely large gamma and C values when we train data it will inevitably overfit. This overfitting on paper may produce a high accuracy if we analyse the model's ability to classify either the same training data or a few unseen testing data points. Using a combination of all the data points to evaluate the model means that we can see how the model actually performs in the real world as we can maximise the amount of data that is tested on the model.



Our main goal is aligned with this, trying to use cross-validation to optimise the hyperparameters of a support-vector-machine. In doing so we can find the best combination of hyperparameters that produce the highest accuracy in classifying unseen data reliably.

How a Grid Search Cross Validation algorithm can be implemented :

1. Define a list of hyperparameters and a list of values that they could be. In this case this could include providing an array of potential gamma and C values. This list would define a finite list of possible gamma and C values. For example in the image above we can see the following C values 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000, 1000000000 and 10000000000. We can also see the following gamma values.



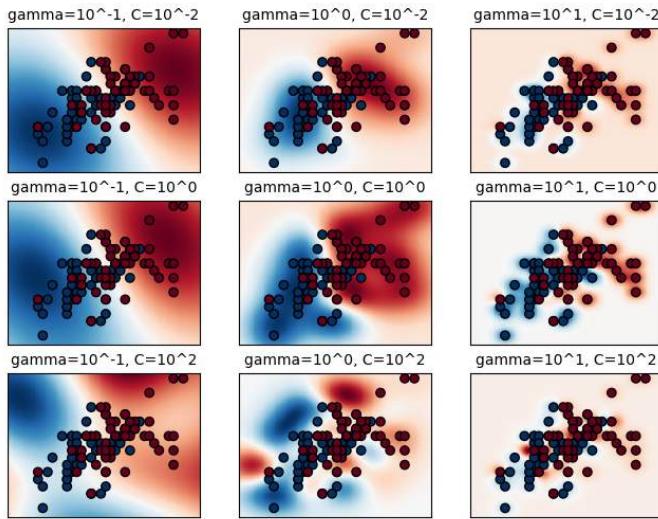
Both these hyperparameters will be stored as an array within the program.

2. We can then gather all the hyperparameters we want to evaluate using our algorithm and produce a grid. This grid contains all the possible combinations of our hyperparameters. Looking at the grid above we can see every exhaustive combination of the hyperparameter arranged in a grid structure. This structure can be in multiple dimensions such as if we are trying to compare multiple hyperparameters at a time. This includes trying to evaluate more than 2 hyperparameters as well as if we wanted to try to evaluate different kernel functions which are used in the SVM to map data to higher dimensions within the dataset. Multiple different kernels can be evaluated using this grid. The final grid therefore may exist in multiple

different dimensions ,but will contain all the possible combinations of the hyperparameters and other settings we are looking to evaluate and then optimise.

3. We then split the entire dataset we are using to train the SVM into separate dataset subsets called folds. This splitting of data into separated folds means we can use cross-validation to get a reliable and robust estimate as to the model's performance across all the data and to clearly spot when overfitting occurs. Cross-validation is the main element used to reliably evaluate each model.
4. As we already have a SVM training function we can input all the hyperparameters directly into the SVM function and produce a model. As we are using cross-validation when we train the model we will cross-validation to train multiple models on different combinations of our fold dataset leaving separate folds for training and testing. This will mean we can verify that all data points have been used as both training and testing. This process will happen iteratively training and testing on all the combinations of the dataset. Once this is complete we will take an average of the accuracy of the model at each fold. This gives us the overall unbiased accuracy of the model found by using cross-validation so the SVM is unable to memorise the patterns within a dataset and instead cross-validation will highlight an unbiased accuracy measurement as to how well it performed.
5. We would repeat this function where we evaluate the model's hyperparameters using cross validation across every single element in the produced grid. This allows us to see how changing the value of the hyperparameters impacts the overall accuracy of the model as a whole. Plotting the Grid-Search Cross Validation algorithm and the relative accuracy across the entire range of hyperparameters we can optimise these values appropriately. Looking at the image we can see areas that are white with optimal hyperparameters and areas that are dark red which indicates hyperparameters that don't allow the model to converge to a hyperplane that accurately splits data effectively.

Grid Search Cross Validation - How will this algorithm fit into our solution? :



The above image shows an implementation of the grid search cross algorithm across multiple different gamma and C hyperparameter values. In this case we can see how different hyperparameters produce different hyperplanes to segment each separate feature effectively.

The final output of this algorithm would be an array of hyperparameters that we can use to optimise the training of our support vector machine model. When doing so we can use the entire weather and price dataset as we are not worried about overfitting. This is because we have effectively found a solution to it by using the most optimal specific hyperparameter values.

This means that our model is able to effectively both learn patterns and generalize data effectively. Which is the optimal solution we are looking to find that yields the best performance of the model on the end-users device.

However, issues with implementing this algorithm specifically into our program is the suspected computational power required. This is because a grid-search algorithm is inherently exhaustive so we cover and need to process a wide range of values. This will be especially important when we want to analyse multiple hyperparameter values on limited low-power hardware. Potential other techniques include a randomized search to search for a limited amount of time for the optimal hyperparameters and choose the one that gives you the most optimum result.

Ultimately what this will mean for our tool is the ability to consistently produce accurate results while retrieving different forms of data as we can fully automate the process of tuning our hyperparameters and advanced settings. This ultimately enhances the overall user experience as users are no longer required to use default values and the program can remain fluid by being able to handle and optimise the model as a whole when used within different settings.

Grid Search Cross Validation - Justification of implementing this algorithm :

After interviewing Mr Sharath as part of maintaining an accurate system, grid search cross validation is a key piece in doing so. While the model is being trained we can set custom hyperparameters which link into how the model is trained upon the data. Using grid search cross validation to test all the defined hyperparameter combinations of gamma and C we can ensure that the produced SVM has the highest accuracy to produce valid predictions. This automates the process of tuning the model as instead of a static value the program is dynamic and is able to automatically select the optimal hyperparameter combination to train the entire dataset on.

Cross-validation also means that the hyperparameters are tested on a diverse range of unseen weather and price data scenarios so are able to effectively evaluate the accuracy of it. Stakeholders such as Mr Sharath said that this successfully met the original requirements and meant that the program is able to be both robust and adaptable to new scenarios and weather events. Calibration of the model also means that the model's accuracy is always maximised and able to make highly accurate predictions consistently.

Pseudocode:

```
SET hyperparameter_grid TO all combinations of hyperparameters  
SPLIT data INTO folds  
  
FOR EACH combination IN hyperparameter_grid  
    INITIALIZE classifier WITH combination  
  
    FOR EACH fold IN folds  
        TRAIN classifier ON all folds EXCEPT current fold  
        EVALUATE classifier ON current fold  
    ENDFOR  
  
    CALCULATE average performance across all folds  
ENDFOR  
  
SELECT combination WITH best average performance
```

Using threading to produce a loading screen

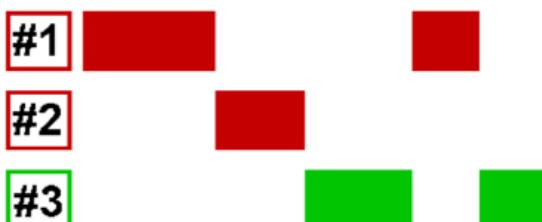
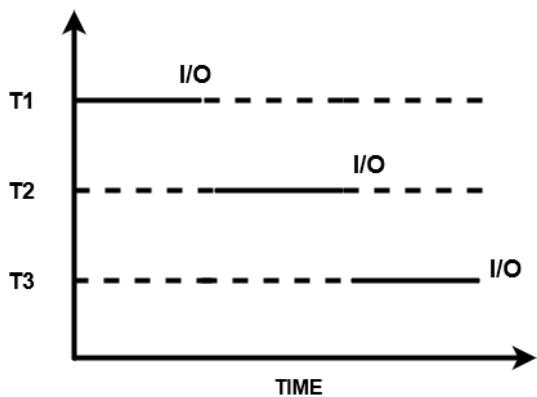
- How does this algorithm work? :

Threading is a process where we create multiple instances of a program to run applications simultaneously as multiple separate threads. This means that we can run multiple separate functions concurrently. Each thread runs independently and doesn't interact with other threads that are running. In this case we can consider each thread as a separate instance within the program as it is able to interact with the original program such as if it wants to call a range of specific functions or access the data stored within variables.

As each separate thread runs as a completely separate and independent instance we can use multiple threads to do tasks in parallel. Instead of traditionally sequentially running code in a chronological order. This means that we can run two functions that do separate things at the same time allowing us to do things such as produce loading screens and run programs in parallel.

Advantages of using this is that it may be efficient to run tasks using threading when they are both computationally intensive and can be easily split up into separate distinct tasks. This would mean that we are able to easily utilise the multiple cores within either a CPU or a GPU. These multiple cores can execute

separate tasks and therefore threads in parallel making them suited for this framework. Modern day computers are suited to threading as they often contain more than one core that can execute tasks independently.



Within python after some research a potential issue for successfully implementing threading natively would be their Global Interpreter Lock (GIL). This is a process that limits the performance of threads in certain situations. This is because the GIL only allows one thread to be executed in python at a time even if it is running on hardware that is capable of running on multiple cores. This may mean even if on the system we can see threads being run they are indeed virtual performance improvements and show instead the

possibilities of how we can run multiple processes at a time. This is an exception for tasks that physically require access to external resources such as a network request. This occurs to keep the synchronisation of each thread.

There is a need to keep multiple threads synchronised as data might be used interchangeably between threads. For instance if we have one thread that is able to execute quicker and execute a function before that data is required we might run into conflicts between threads as there needs to be appropriate mechanisms to coordinate access between them. Creating and managing threads brings up different issues and causes our system ultimately to become extremely complicated and convoluted. This is called a race condition and occurs when multiple threads are trying to access and modify the same resource at the same time this may cause unintended and unpredictable behaviour.

The Global Interpreter Lock therefore tries to limit the entire system by controlling access and having threads run in a predictable fashion.

Using threading to produce a loading screen

- How will this algorithm fit into our solution? :

Our main application of threading is to produce a loading screen. To prevent our application of native threading within our system we are only implementing separate threads that have no impact on each other. This is to conform to python's standards of synchronous sequential threads run while obeying python's global interpreter lock (GIL). Multithreading involves a program that uses multiple threads to concurrently display loading screen data and execute the main processing layer in the background.

Fitting the algorithm into our solution requires using separate threads to update the loading screen to display the current progress of training the model and any errors that have occurred while also ensuring that in the background data is being processed and correctly matches what is happening on the loading screen.

The reason why we have chosen to implement threading is to give the user appropriate user feedback about the background processes that are occurring while our solution is actively running. This means that the user can easily debug our program directly from the errors section of the page or using the visual feedback from what processes are occurring to show and explain exactly the reason why processes are being executed. Ultimately this will link with our goal of improving the user interface and therefore user experience.

Using threading to produce a loading screen - Justification of implementing this algorithm :

Threading allows the program to give the user continuous real time feedback as to what is occurring in the background. Stakeholders such as Mr Manu said that this would be beneficial as during long running tasks, the program is transparent as to processes that are occurring. Additionally using independent threads for the user interface and backend means that as there are no shared data processes between them it confirms python's Global Interpreter Lock limitations.

Due to the necessity of the user interface, threading can also be implemented to highlight errors when they occur directly to the user on the loading screen in real time allowing users to debug them immediately while also providing a clear user interface while the program is processing data.

Pseudocode:

```
CREATE thread1
| DISPLAY loading screen
| UPDATE loading screen with progress

CREATE thread2
| EXECUTE main program logic
```

Data Retrieval of both commodity prices and weather data

- How does this algorithm work? :

Our algorithm will need to access two pieces of data. Weather data and the crops commodity prices.

The weather data will need to include the four main weather metrics that our solution requires. The temperature, humidity, precipitation and cloud cover. All of the weather data will need to cover both historical weather data points and forecasted weather datasets. To do this we have located a datasource that contains historical weather data. MPI-ESM is a comprehensive earth system model that is developed by the Max Planck Institute Earth System Model. This model contains multiple climate research components and contains approximate weather data for all four of our required metrics. The specific model we are using is MPI_ESM1_2_XR (51 km) which covers the entire earth in 51 km squared segments. This dataset is a high resolution model for the earth's weather system and contains data with high detail and accuracy which is crucial for our tool. An issue we faced is the size of this tool as the complete dataset is over terabytes in size. Splitting this historical weather dataset into only the variables we require still means that the file is over 250GB in size which is over our projects requirements for storage size and this would make it inaccessible to non-advanced users that may want to run our program on a low-power device that is unable to store that much data. This meant that we had to implement an Application Programming Interface to retrieve weather data from. In this case this was openMeteo which provided open source access to this model. There were rate restrictions that limited calls per device to 10 000 calls a day however most beginner users are unlikely to hit usage limits.

OpenMeteo also provided forecasted weather datasets which forecast the future weather of any location on earth up to 16 days in advance with reasonable accuracy. Using both of these solutions OpenMeteo provides we have access to a constant stream of weather data.

Commodity prices on the other hand could also be retrieved using techniques such as using an API such as AlphaVantage to get up to date data. However, after

researching the documentation, applying an API that has low rate limits and a limited amount of listed agricultural commodities may limit the amount of user interactions with our tool and the appropriate quality of each interaction. This meant that we looked at different methods to get our data.

Most commodity market APIs had either extremely expensive prices per API request or had extremely low free API calls per day. All of this meant we had to retrieve the data from another completely separate source.

Live market data is readily available in real-time online on multiple websites for free so I planned to use techniques to extract data directly from the website instead of relying on unstable APIs which were inherently limiting in their scope. This means we can algorithmically access a website and pull only the specific information we need. Using beautifulsoup4 I plan to access the underlying HTML code behind a website and make a request to retrieve this. Once I do this I can parse my HTML data to either retrieve values themselves or the location of where values can be accessed within elements on the page. Once I have this I can suitably extract and format the data into a format that is useful for our program. Automating our retrieval of data internally instead of relying on an external API to retrieve our datasets. This process is called web scraping and involves scraping and extracting data from a website.

Once we have retrieved the respective commodities price data and the weather data we can pass it onto the formatting stage. This stage involves converting all our datasets into a consistent format that is able to be processed by our support vector machine model effectively. The data might contain anomalous or missing values that need to be handled. This process can be given to the appropriate algorithm embedded within the cleansing function which will detect the anomalous or missing value and impute it with a suitable substituted value. This exact process is mentioned within our algorithms section. In our case we are using the World Bank Group's Pink Sheets which is a monthly updated .XLS (Excel) file that contains multiple prices for commodities around the world.

Once all of that is complete we can store the cleansed data in an appropriate structure such as a pandas dataframe or a .CSV file which can then be easily passed into the model as data for training / testing / predicting.

Data Retrieval of both commodity prices and weather data

- How will this algorithm fit into our solution? :

Using up to date and relevant data is a core component of our solution. This is because the main use-case of our program is as an up-to-date commodity prediction tool. This means that we are entirely responsible for gathering all the necessary data points from various sources , compiling it and producing a valid solution.

As our software is able to handle data retrieval and the added complexity that comes with that such as using APIs and web scraping tools this algorithm makes our solution adaptable as we are able to not require constant user input of data

sources and instead can automate actions to be completed in the background by our software.

Our solutions overall design prioritises both efficiency and accuracy so effective data retrieval is integral to our system as we are building at scale a data analysis tool that is able to successfully predict the price of agricultural commodities using climate patterns.

Using suitable data retrieval techniques is also cost effective as our program can easily access commodity prices and weather data from external sources that are both up-to-date, relevant and in processible file formats.

Data Retrieval of both commodity prices and weather data - Justification of implementing this algorithm :

Retrieving up-to-date weather data and real-time commodity prices is a requirement as we are trying to make accurate predictions. In comparison to just using historical data we are able to capture more patterns and predict with higher accuracy by focusing on accommodating up-to-date datasets.

Additionally as we are using algorithmic methods to process data there is a requirement for data to be of a high quality and resolution. Datasets can be retrieved from external sources which are free and open source using APIs such as OpenMeteo and techniques such as web scraping to gather data. Automation of these data retrieval processes reduces the need for the end-user to manually upload or gather data. Instead we have a future proof design that is able to consistently retrieve and process data.

Pseudocode :

```
# Weather data retrieval

weather_data1 = GET_WEATHER_DATA(coordinates1, start_date, end_date)
weather_data2 = GET_WEATHER_DATA(coordinates2, start_date, end_date)
weather_data3 = GET_WEATHER_DATA(coordinates3, start_date, end_date)

# Commodity price data retrieval (web scraping)

current_price = SCRAPE_COMMODITY_PRICE(website_url)

UPDATE_CACHED_PRICE_DATA(current_price)
```

Using a Support Vector Machine to classify data

- How does this algorithm work? :

The output of the processing layer is a fully trained support vector machine model. This model's main purpose is to efficiently classify unseen weather data points and produce a prediction as to the predicted future change in price of the agricultural commodity.

As our model is fully trained and the relevant hyperplane has already been produced, to classify our datapoint with a label that defines what category our data falls into all we need to do is pass it through the model.

The model will use the same kernel function and produce a transformed location. This transformed location can be compared with the hyperplane which represents the decision boundary and depending on where the unseen weather data point is located we can use its relative position to the hyperplane to assign it a class. This class is the category that it falls into for instance whether the price of the agricultural commodity will change and to what specific extent.

We can do this same process quicker by using an algorithm called the kernel trick. The kernel trick involves allowing the SVM to directly produce a segmentable mapping without calculating the coordinates of the data points in multiple dimensions which will require a lot of unnecessary computation. Instead we can use a kernel function to calculate the similarity between data points if it were in the higher dimension. This means that we can approximate the decision boundary to categorise data not in multiple dimensions. The Radial Basis kernel function operates in infinitely many dimensions so it isn't possible to appropriately visualise it. This means that we can avoid transforming data and instead process the data appropriately as if it was already transformed.

The kernel trick algorithm allows for both performance and flexibility improvements. As the SVM is able to classify a point quicker as it is not using a direct translation into a higher dimension it is instead transforming a higher dimension into the current dimension the dataset is in. Flexibility is also involved as it means that extremely complex functions such as a RBF can be used appropriately to segment the data.

Ultimately using SVMs to classify data means that our entire program is not static and is instead able to fluidly react to how data exists in the real world.

Using a Support Vector Machine to classify data

- How will this algorithm fit into our solution? :

This algorithm fits securely within the prediction side of our solution as we are retrieving forecasted weather data and need to produce predictions as to the weather's impact solely on an agricultural commodity. The SVM crucially captures the complexity of the multiple weather patterns and works with non-linear relationships to produce a single classification that is used to predict the ultimate impact on a commodities price.

Implementing specific examples such as the kernel trick within our program means that the solution is efficient as our solution is able to successfully produce an insightful output effectively all while running on low power hardware that is accessible to a range of users.

The forecasted data will be retrieved from the OpenMeteo API and will be suitably formatted to be able to be processed by the machine learning model. The final

trained SVM model will be implemented within the prediction stage to produce a suitable prediction. This means that this step is an intermediary step before the final output of the solution is produced.

This specific algorithm is the a main component of our entire solution so it is crucial that it is able to accurately handle and produce a range of predictions , particularly when proposed designs show it producing multiple predictions across multiple separate time segments. This means that the function will be called and used during every request within our program retaining the importance of this specific component as it is a product of all the previous inputting and processing activities. After this stage the output from all the classifications are delivered to the user to clearly visualise on an interactive graph.

Using a Support Vector Machine to classify data - Justification of implementing this algorithm :

Classification of forecasted weather data using the support vector machine (SVM) locally on the users own device allows for data to be kept both secure and processed quickly. Additionally, using a support vector machine allows for the training of models and classification of data points on the device as it doesn't require extremely powerful resources to capture correlations in data. Instead data in multiple dimensions can be effectively processed to produce a support vector machine that can quickly classify a wide array of data points which in this case is the forecasted weather data.

Pseudocode :

```
svm_model = LOAD_MODEL("svm.model")

forecasted_weather_data = GET_FORECASTED_WEATHER_DATA(coordinates)

predictions = []

FOR EACH timestep IN forecasted_weather_data:
    prediction = svm_model.PREDICT(timestep)
    APPEND prediction TO predictions

DISPLAY predictions
```

Plotting data on an interactive graph - How does this algorithm work? :

Plotting data involves three main functions. Zooming / Panning / Scaling the data. Also there is a need to be able to select a custom time scale for data to switch between looking at a yearly scale and one in decades. Having a clear interface allows the user to be able to interact easily with our graph and access all the data points quickly. In this case within our graph we will have a time-series graph of all the price data points and an overlaid prediction as to indicate the future price movements of the agricultural commodity.

Four main aspects of this algorithm are the following. Data input, scaling of data, visualising data, overlaying predictions and implementing interactive elements.

Data input involves being able to access and retrieve time-series data. In this case we will need to have a constant source of time-series data that covers all the commodity prices. As discussed before we will use other algorithms embedded within their respective function to access this relevant data. As our data will be in a relevant tabular file format we can extract both axes data points from this dataset.

We will need to use specific commodity price measurements across specific days to populate data within our graph. To appropriately plot this onto a graph we require the correct scaling of data. This involves automatically adjusting the x-axis and the y-axis which represents the date and price of each datapoint respectively. This means within the commodity price dataset we will need to also extract the date values and the commodity price values to represent the x and y components of our final coordinate. Once we do this we need to scale the graph appropriately to include all the values or show specific segments of the graph in detail. To do this we would adjust the class width of each x, y axis value to produce a correct graph. This might involve finding the minimum and maximum x, y axis value to represent the maximum values we will need to display. This means that we can limit the amount of data we can represent both utilising the screen space we have efficiently and preventing data points from being cut off the entire graph. We can then set appropriate class widths across both axes and scale both axes to correctly distribute all the points appropriately on the screen.

A scaling algorithm will do the following.

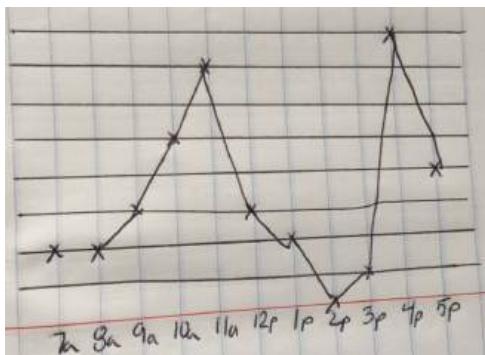
1. Calculate appropriate intervals or also called the class widths to spread the labels on each axes accurately.
2. Adjust each axis independently to fit within all the data points within the screen appropriately.



Visualizing this data means we will have to present it in a suitable format. When looking at commodity prices they are normally shown not as a list or table of values, but instead as a connected line graph that shows clearly how the price changes across time. We can plot data points as a line graph by implementing the following algorithm.

1. Plot each datapoint correctly taking into account the x, y values are correct.
2. Find the first datapoint that is plotted on the graph and save the relevant coordinate.
3. Select the $(n + 1)$ th datapoint that is plotted on the graph and save the relevant coordinate. (n) refers to a positive integer for example the first datapoint that is plotted on the graph which can be referred to as $n=1$.

- Now we have two separate points on the graph. We can use the relevant $y = mx + c$ straight line formula to connect these two points to show how the price approximately changes as time goes on. As a $y = mx + c$ line is infinitely long we will need to define limits or regions to only plot the required components of our line. This will mean that we can set the appropriate x-axis limits that will be the size of our class width. This means that the line will not span infinitely as long and instead is clearly bounded within a segment. Looking at the following image we can see what we aim to produce.
- Increment n to the subsequent value. This is such that n is equal to the subsequent integer. This will mean that when we call $n + 1$ we will be looking at the next point on our graph.
- We can now repeat this action to correctly cover and connect every single datapoint within our entire dataset. Producing separate lines to connect each datapoint and then connecting all the components of our line individually until all our data and desired lines have been plotted.



Looking at this image we can see how using multiple straight lines we can represent separate data points as a line graph.

We can now focus our efforts into plotting the overlaid prediction on top of the historical and current price data. This will enable end-users of our solution to see a clear visualization as to how the commodity price is expected to move

in the future. Looking at our planned output page we can begin to plan how

exactly we are going to implement an algorithm that does the following. Plot the future prediction of commodity prices directly on top of the historical price data.



To do this we will use outputs from the previous layer. This involves using the produced time segment prediction data that was outputted from the prediction layers operations. This means that there are predefined

time segments that have been forecasted and correctly predicted. These time segments are ranges of dates that have been analysed as a whole and then an appropriate time segment prediction is produced. This will mean we have an array with suitable predictions as to the commodities price across each specific time

segment. Using the same method as above we plan to plot each individual time segment's data points and then plot it directly on the graph. Looking at the image above we can see our planned implementation as to the final product of our solution.

1. This involves retrieving the length of each time segment, an array with predictions as to the values of forecasted prices within each time segment and access to the graphing algorithm.
2. Using all of this we can create an array with all the x coordinates of the prediction data as we can retrieve the length of each time segments and the amount of time segments we have stored in our array to estimate where the time segment will start as it will directly fit in after the historical prices of the commodity data.
3. We can then as described beforehand plot each datapoint starting at the first one. In this case when $n = 1$.
4. Increment n and find the second datapoint. Now we have two separate points on the graph. We can use the relevant $y = mx + c$ formula and connect these two points to show how the price approximately changes as time goes on. As a $y = mx + c$ line is infinitely long we will need to define limits or regions to only plot the required components of our line. This will mean that we can set the appropriate x-axis limits that will be the size of our class width. This means that the line will not span infinitely as long and instead is clearly bounded within a segment.
5. Repeat this until all the prediction data points have been plotted.

Interactive elements are also an important component of our outputted solution. This is because stakeholder's want the ability to zoom / pan / scale across the graph easily. Zooming enlarges and widens the overall view of specific segments of our graph so the user is able to customize the view they look at data. Panning involves allowing the left and right movement of our graph so users are able to access both future predictions and the historical prices of commodities. Custom time scales are an implementation of both zooming and panning as its main use is to allow users to access predefined segments of data without having to interact with the zooming and panning user interface. This feature's main aim is to allow users to pick specific custom ranges to view data within.

The easy traversal of our graph is key so implementing features to allow users to zoom through our data and pan across multiple components is crucial.

Plotting data on an interactive graph - How will this algorithm fit into our solution? :

The graphing algorithm will clearly fit into the output layer as we need to visualise both predictions and historical prices of the agricultural commodity. This involves preparing and presenting all the available data to the end-user in a suitable format. This enhances the user experience as it allows an end-user to be able to explore the data without looking at a predetermined forecast that isn't customizable. This algorithm prepares and presents all the required data to the end user in a clear and interactive way. This means that the end-user can easily interact with our solution to analyse the data using their methods easily. Our solution is central to the

solution as it is the only output that is produced. This algorithm is a key part of the output layer as it compiles all the predictions that are generated by our original model and presents them in a clear and concise manner. Supporting the financial decision-making of our users and allowing them to quickly make decisions based on the support of thousands of price and weather data points.

Plotting data on an interactive graph - Justification of implementing this algorithm:

A graph is the final output of our solution; this means that it is a necessity as it will allow the user to easily access and understand the historical agricultural prices overlaid with the future predictions of the agricultural commodity they are analysing. The main feature of the program is to produce a range of predictions that allow the end-user to analyse the agricultural commodity without needing to access multiple sources. Instead our solution collates and processes all the data automatically and presents it all on a single page that is accessible to the user.

A single interactive graph is an essential component of our program as it means that the user can accurately gauge the performance of the crop's commodity based on the historical data and future forecasted price movements. Additionally there are features that allow the user to quickly access data such as time scale switching and zooming and panning across the entire graph as requested by stakeholders to allow for added intractability.

Pseudocode :

```
FOR each datapoint in dataset
    Plot datapoint with x and y coordinates

n = 1

WHILE n < number of datapoints
    point1 = coordinates of nth datapoint
    point2 = coordinates of (n+1)th datapoint

    Calculate slope (m) using point1 and point2
    Calculate y-intercept (c) using point1 and m

    x_start = x coordinate of point1
    x_end = x coordinate of point2

    FOR x from x_start to x_end
        y = m * x + c
        Plot line segment at (x, y)

    n = n + 1
```

Key Variables, Structures and Classes

Identified, explained and justified the key variables / data structure / classes

Key Variables :

Variables are named memory locations within our solution that hold a value. Within our program we will implement multiple variables to share data between functions and to allow for the ultimate processing of data. There are two specific types of variables we can implement. These are global variables and local variables. We explain the differences below.

Global variables have a global scope. A global scope means that variable is accessible throughout the entire program. This means that you can call a global variable even inside a function.

Local variables on the other hand have a local scope. A local scope means that the variable is only accessible locally within the exact function or block of code. This means that you can't access it outside of this function or block of code.

crop variable:

The main variable we use throughout the entire is the crop. This crop variable is assigned the crop the user picks at the crop selection page. This variable is a global variable as it will be called and used within functions to define what data they need to retrieve. This is because the weather and commodity prices are different if you analyse a different crop. This is because a separate crop will be grown in a different location so a specific crop will require a specific weather dataset and different crops are listed as different agricultural commodities on commodity markets. The datatype of this variable will be a string that will define what crop the user selected. Such as 'cocoa' or 'wheat'.

latitude and longitude variables:

The latitude and longitude refer to the coordinates of the location where the crop is grown. Within our functions we will need to refer to a latitude and longitude coordinate component to define the location in the world where a crop is grown. We will also need to use the latitude and longitude variables to define the location we want to retrieve weather data from using the API. These will be used as a local variable within a function and will store a float datatype as coordinates have a fractional component.

startDate variable:

As mentioned before both the weather dataset and the price dataset need to be of the same size. The price dataset will be inherently limited as the commodity market would have only been established in the 1960s compared to weather data which is accessible decades and even centuries before this. This means that our commodity data constrains our dataset. The constraint means that we inherently limit the weather data. This is because we need both datasets to be the same size to suitably

process it so we can use this information to set the starting date value of the startDate variable. This refers to the date that the commodity and weather data starts at. The date is in the form ISO8601 as it is a widely accepted international standard so it is used to keep all the variables within the program in the same format. ISO8601 is in the form of a string so the startDate variable will be stored as a string. This variable can have a local scope as it can be calculated within a function and directly passed into another function instead of having to be accessible throughout the entire program.

endDate variable:

The endDate will be a string data type and also in the form of ISO8601 to keep consistency within the date system of our program. The endDate is similarly constrained. In this case it is constrained by either the availability of commodity prices or the current date. This is because if the most recent commodity prices is a day out-of-date for instance that means that to train our model we only currently have prices until that restrictive date. This will mean that to produce a complete dataset for our model we will have to be inherently restrictive in setting the respective end date. If instead the commodity prices are up-to-date and relevant with the current prices our end date can be the current date that the user is using our solution. In either case the current date highlights the day that

websiteURL:

As we have decided to use web scraping or an API to access up-to-date commodity price data we need a datasource to access it from. In this case we can consider using a website and web scraping the required data directly from it. The websiteURL will be the data source of the commodity prices and will store the URL of the specific web page where we can access this data. The websiteURL variable will be a string as it needs to store a collection of characters to form the URL. This will be a variable with a global scope as it will need to be used within a function to retrieve a dataset from that specific website.

commodityPricesDataframe:

This is the dataframe that contains all the commodity prices including both historical and current commodity prices. We will load the cached database into this dataframe which will be stored in RAM for quick use instead of having to retrieve it directly from secondary storage such as the user's hard drive. The commodity prices are stored in a dataframe as it is a data type that stores tabular data. The original dataset might be in the form of a CSV or another two-dimensional data type which makes a panda's dataframe data type suited for this specific application.

weatherDataframe:

This is the dataframe that contains all the weather data including both historical and current weather data. We will load the cached database into this dataframe which will be stored in RAM for quick use instead of having to retrieve it directly from secondary storage such as the user's hard drive. The weather dataset is stored in a dataframe as it is a data type that stores tabular data. The original dataset might be in the form of a CSV or another two-dimensional data type which makes a panda's dataframe data type suited for this specific application.

cHyperparameter and gammaHyperparameter:

These are the hyperparameter variables which store the hyperparameters which the user has access to edit in the advanced settings page. The hyperparameters will be a float data type to allow for fractional numbers. These hyperparameters can also be set to default initial values if the user doesn't customise it.

finalCHyperparameter and finalGammaHyperparameter:

These hyperparameters are of the data type float as well to allow for fractional numbers. These are the final hyperparameters that will be set after the user uses the grid search cross validation algorithm to pick the most optimal hyperparameters for that dataset. This means after the grid search cross validation algorithm is complete the program can easily pass the most optimal hyperparameter values to the user.

Kernel:

The kernel is the function used to transform the data. There is a range of kernels that the user can select from a linear, polynomial, sigmoid and the radial-basis-function kernel. This means that there are multiple methods that the program can use to get to the end-result of the classified data. To accommodate multiple kernels within our program we can use the variable kernel of datatype string to store the kernel we are using to separate the data.

defaultHyperparameters:

This variable will store the default values of the unoptimised hyperparameters if the user doesn't select any specific ones through the advanced settings page. The variables data type will be of a one-dimensional array with a list of values that represents the default hyperparameter values.

hyperparameterGrid:

The hyperparameter grid contains all the possible values for the hyperparameters. Its datatype will be a two-dimensional array which will be represented in python as a nested array. This means that we can store all the possible hyperparameter values which we can use to generate a grid with all the possible combinations. We didn't store an array which contained all the possible hyperparameter values as it would have been a waste of storage space instead of actively generating them when needed.

weatherDataframe1, weatherDataframe2 and weatherDataframe3:

These are the dataframes for the historical and current weather data. They are stored as dataframes for easy retrieval and manipulation of datasets. They are of pandas dataframe data type which stores tabular data.

X:

In machine learning the “X” represents the features of the model. These are all the combined weather metrics. In this case all three weather dataframe (1, 2, and 3) are combined into a single weather data frame that is used to train the support vector machine model. This is because a support vector machine only accepts a single dataset for training at once. The data frame within X will contain all four weather

metrics precipitation, temperature, humidity and cloud cover. Across all three of the major locations where the crop is grown.

scaledX:

As a support vector machine model is more accurate when all the data is standardised. In our case centering and scaling the data so they have a mean of 0 and a standard deviation of 1. We need to further process our dataset to be appropriately standardised. This involves retrieving the previous variable X and then using a standarsing function to scale and centre it to have the aforementioned properties.

Y:

These are the labels to the dataset. This is because the labels are stored in a separate dataset. The original weather dataset X will have multiple data points. Each datapoint will be categorised with a label that specifically represents the change in price of the agricultural commodity. Labels include exact percentage change of the commodities price during that time or a label that indicates the change in price for instance strong positive, positive, neutral, negative and strong negative.

forecastDataframe1, forecastDataframe2 and forecastDataframe2:

These are the dataframes for the forecasted weather data. They are stored as dataframes for easy retrieval and manipulation of datasets. They are of pandas dataframe data type which stores tabular data.

Model:

The trained model itself will be stored in a model named model. The data type is unclear as it stores a model instead of a physical piece of information. Instead the model consists of fully trained kernel function and the hyperplane which is used to categorise data into distinct categories appropriately. This model can then be passed into the prediction layer to be used for predicting.

timeSegment:

As we are making multiple predictions across a time range of our weather forecast and not a single prediction we can use time segments to show the change in price across the entire graph. The variable timeSegment will be the length in days of each time segment that we will forecast independently. The time segment data type will be an integer as it is a number of days.

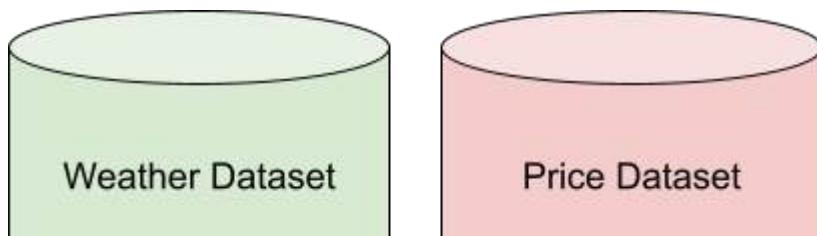
Predictions:

The predictions will be stored in an array that contains all the predictions. This is across all the time segments that have predictions against them. The predictions will store the time segment prediction as an array with multiple percentage changes indicating the prediction within each time segment.

Structures :

A structure is a user-defined data type that is used to group together related data. We can use structures within our solution to group and organize data into a single entity or structure. A structure can be used to hold data. As our tool is

foundationally a data



analysis tool we require the use of structures within our solution. We require the following structures within our program: a weather dataset and a price dataset. A structure allows us to organize the complexity of a collection of data into a single dataset that we can easily implement within our program. A structure will often contain a related collection of data in our case we will implement a structure to store historical commodity prices of multiple crops and a structure to store historical and forecasted weather data that contains multiple weather data metrics.

We store our datasets on the device the user is using to make the data processing more efficient. Such as instead of making multiple API requests to individually find the weather of a datapoint we can retrieve an entire dataset with a single API request and store it as a dataframe on our device. This means that we can easily access and format the entire file and all the values within it instead of worrying about the individual manipulation of each datapoint as we can simplify data operations making our entire process quicker and more efficient.

We have also implemented caching, which involves storing frequently accessed data such as commodity prices on the device as a comma-separated values (.CSV) dataset so that instead of having to call an API to get multiple data points we can access it quickly on the device. Another feature of this structure is the ability to easily append data to it as you can add current values to the cached commodity price dataset to make sure it is up-to-date. As it is in a tabular form, appending elements to update a dataset can be done easily using this structure.

In this case we use both a pandas dataframe structure and a .CSV structure to store our datasets. These databases can be easily traversed as they are in a tabular format which means that we can store time-series data accurately within the database efficiently. We need to traverse the database efficiently as we will need to retrieve data from specific dates which means the already indexed format makes this operation simpler as instead of manually searching through data we can locate the correct index quickly and access our specific data points within the entire database. Comma-separated values (.CSV) and a pandas dataframe can be used interchangeably as they describe the same two-dimensional and tabular data structure. These can be thought of simply as organized containers with compartments that allow the easy storage and access of data at any point in time.

Using structures also links to our need for the integrity of data. As there is a need to use only valid and relevant data we can have an object-oriented approach when we interact with datasets. We can use encapsulation which is a core concept behind object-oriented programming to only be able to interact with our dataset using predefined classes. These predefined classes contain methods which are functions that define the behaviour of the entire class. The class can only perform actions that have been defined as a method. This is described in more detail in the following section.

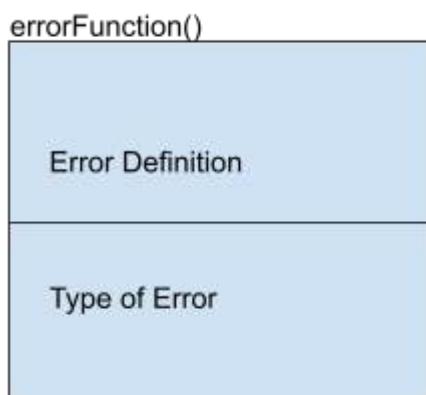
Classes :

Classes within python are blueprints for creating objects. It defines a set of attributes and methods that define the object. An object is an instance of a class.

Within python we can use classes to perform specific actions and produce outputs. In our program specifically we use classes to manipulate data within the class and produce suitable output data that can be passed onto the next stage. The advantage of using a class instead of implementing a specific function within our program is the ability to reproduce that function by calling that class and inputting custom data through it, this links to the modular nature of our program.

errorFunction(errorDefinition, type):

This class is used to alert the user due to an error and show it directly on the screen. This error function is called when the program is unable to progress due to an error within the program. The error function will show what type of error has occurred as defined by the error definition and then what will happen. In this case there are two types. A soft-reset which will reset the program back to the starting page and a hard-reset which will reset the entire program and initialize it from the beginning.



verifyCoordinateValidity(latitude, longitude):

This class takes in the coordinates of where the crop is grown which is split into latitude and longitude coordinates and checks its validity before it is passed to the next stage of the program. This involves making sure the latitude is between (-90 and 90) and the longitude is between (-180 and 180). If the coordinates are valid the class returns valid values whereas if the coordinates are invalid the class returns invalid values.

(-90 and 90) and the longitude is between (-180 and 180). If the coordinates are valid the class returns valid values whereas if the coordinates are invalid the class returns invalid values.



openMeteoHistoricalClimate(latitude, longitude, startDate, endDate):

We need to be able to access historical weather data so we need to implement a class that is able to retrieve historical weather data from the open source weather API openMeteo. This involves passing the latitude, longitude and a start date and an end date through the API as a request. After you run this program with all the specified variables openMeteo will respond to the request with a panda's dataframe with all the information. If invalid values are entered either openMeteo will not respond or an error will be flagged to the user to prevent this we only pass valid

data into the class as we can see above where we verify the

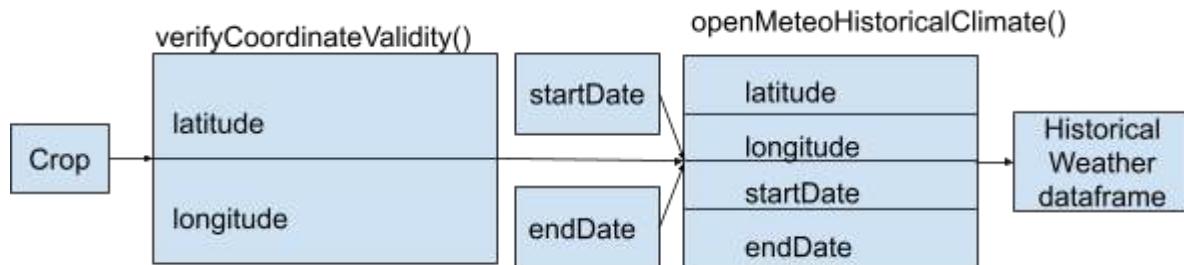
openMeteoHistoricalClimate()



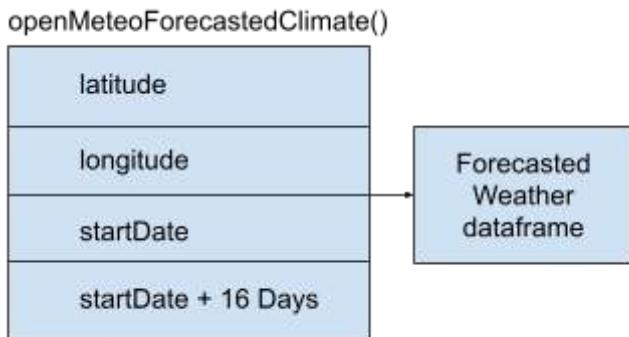
coordinates before implementing them.

retrieveHistoricalClimate(crop):

This class will be able to use all the functions above to create a request and format the data appropriately. This means that it will use the crop the user selected. Retrieve all the data points that are necessary which are the latitude, longitude, startDate and the endDate. Using all of this we can create a request by first verifying the coordinates are valid using the verifyCoordinateValidity class and then pass on the respective coordinates and dates into the openMeteoHistoricalClimate() function. We will do this process three times for each major crop location and also if the coordinates are invalid we will display an error.



openMeteoForecastedClimate(latitude, longitude, startDate):



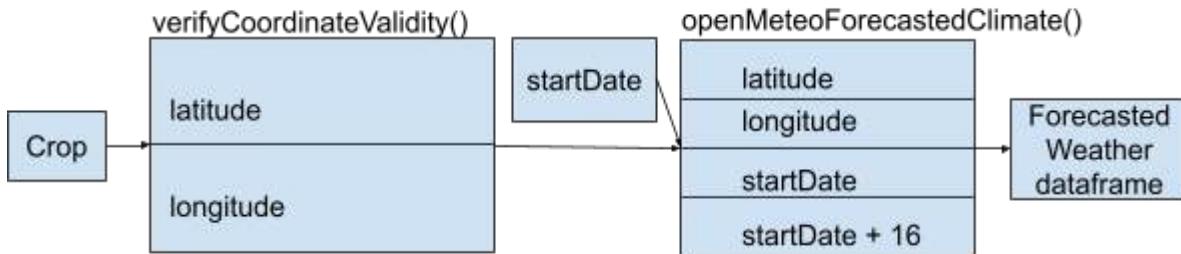
We need to be able to access future forecasted weather data so we need to implement a class that is able to retrieve future weather data from the open source weather API openMeteo. This involves passing the latitude, longitude and a start date as a request. There is no endDate in this request as the program will automatically select the longest date that you can forecast weather to. In the current

openMeteo documentation this is 16 days after the starting date. After you run this class with all the specified variables openMeteo will respond to the request with a panda's dataframe with all the information. If invalid values are entered either openMeteo will not respond or an error will be flagged to the user to prevent this we only pass valid data into the class as we can see above where we verify the coordinates before implementing them.

retrieveForecastedClimate(crop):

We use a single class to retrieve the forecasted climate for a crop. We pass the crop we want future weather data for through the class and it retrieves the latitude and longitude coordinates. These coordinates are then verified using verifyCoordinateValidity and at which point we pass the coordinates to the API retrieval function openMeteoForecastedClimate. This class takes in the coordinate

and the starting date. The final output of the entire class is a retrieved forecasted weather dataframe.



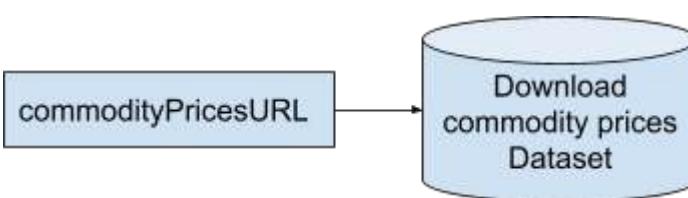
getDatasetLink(URL):

We are dependent on an external source to retrieve current commodity data from. Due to this we need a class that is able to scrape a web page and accurately locate the file where commodity data is stored. This is because within a HTML webpage there will be a lot of unnecessary elements that don't contain the required current commodity prices. To tackle this we will input the URL of a specific web page and the `getDatasetLink` class will respond with the link for the current commodity prices dataset.



downloadDataset(priceDatasetURL):

This class will be used to retrieve the commodity prices from the online dataset. As the data source for current commodity prices will not contain other unnecessary elements, such as the original HTML of the web page which might have contained user interface elements and other pieces of information. This means that we can directly download the entire dataset using the dataset URL.



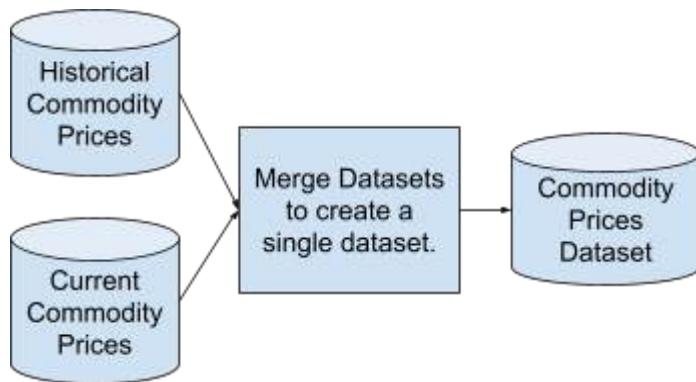
retrieveCommodityPrices():

Combining the retrieval of the location of the commodity prices dataset with the downloading of the dataset. We can use a single class to execute both instructions at once. This means that everytime we run the program we can use up-to-date and relevant data as we are directly web



scraping it off the internet.

update_HistoricalCommodity_Prices():

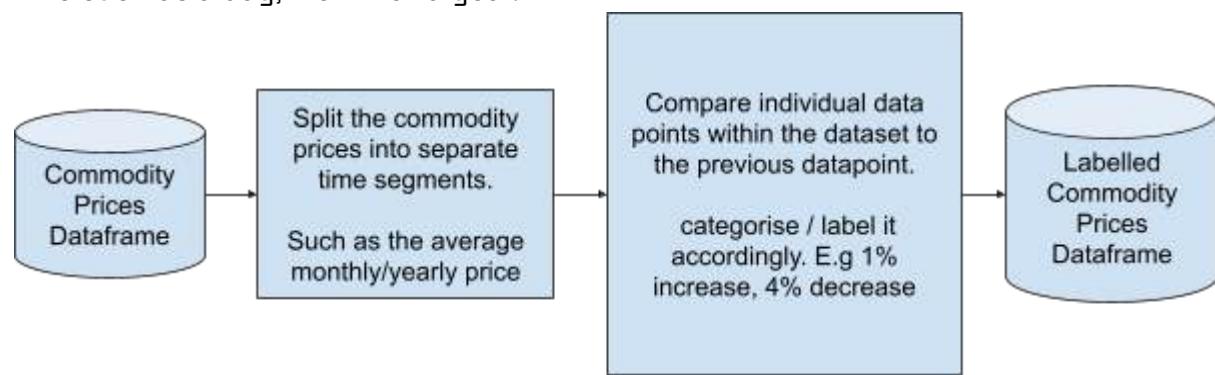


As we are planning to implement a separate historical commodities prices database and then retrieve the current commodity prices by web scraping it off the internet there is an added complexity in combining both datasets into a single dataset. To do this our class will retrieve both historical and current commodity price datasets and then merge them

both into a single commodity prices dataset so the cached database of historical commodity prices are always kept up-to-date and relevant.

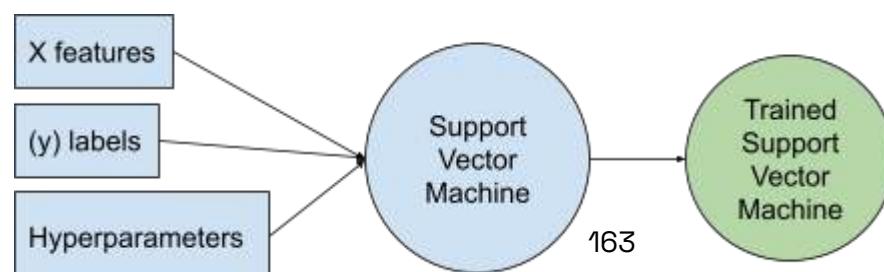
classifyCommodityPrices_dataframe(dataset):

Classification involves labelling our entire dataset to show the change in price of each datapoint across the dataset. This means that there is a percentage change value that categorises each datapoint in comparison to the previous price. This is done so we can use the matching weather data to try to locate patterns across the weather and the commodity price data using our support vector machine model. We split the commodity prices into separate time segments which are then averaged to look at not a single point of time, but how the commodity prices change by average over that time segment. A time segment defines a specific amount of time such as a day, month and year.



train_SupportVectorMachine(X, y, hyperparameters):

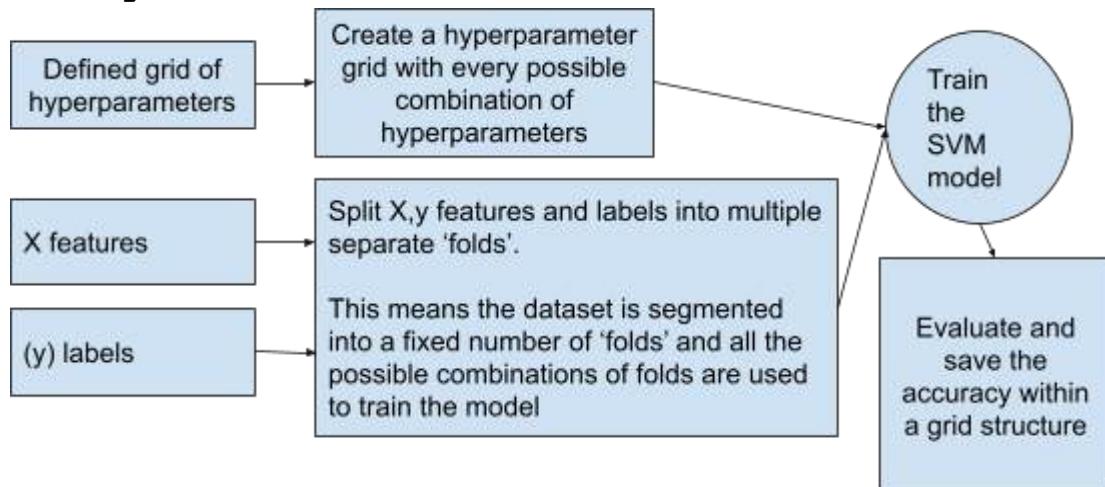
We train the support vector machine using these three key variables within the class. X represents the features of the dataset in this case this is the scaled weather data. (y) represents the categorised change in price of the agricultural commodity. Finally the hyperparameters are an array of C, gamma and kernel values. With all of this information we can correlate the datasets to try to define both a kernel function and a hyperplane to split the data.



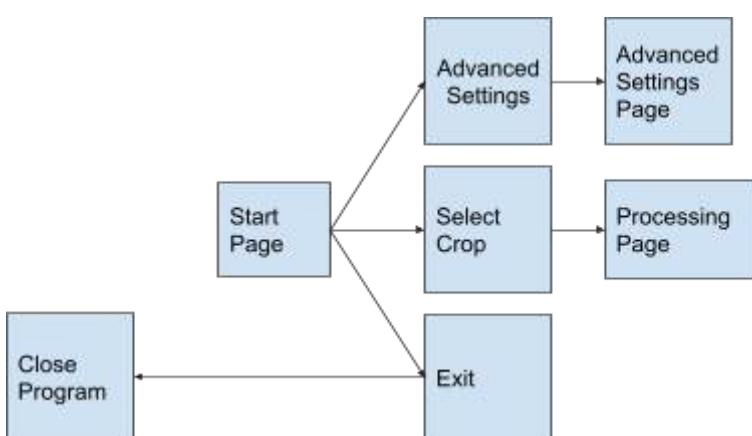
gridSearch_crossValidation(possibleHyperparametersArray):

Grid search cross validation involves trying to find the most optimal hyperparameters for a model. We input all the possible C, gamma and kernel hyperparameter values into our array. This is then used to create a grid of all the possible combinations of hyperparameters. Our original dataset will be split into separate folds so we can test and train the model on all of the data points instead of using a subset for testing and a subset for training. This also means that using cross validation by splitting data into separate folds that we can generate a more accurate accuracy value as the model is not able to memorise the patterns of the data. This would be a problem if we tested and trained on the entire dataset which would mean we can hit problems such as overfitting which will occur as the model is able to memorise all the data and perform extremely well in classifying the original dataset.

We can then combine the hyperparameter grid with the model to have an exhaustive search to find the most optimal hyperparameters. To evaluate the model we will pair this process with cross-validation to effectively define the relative accuracy of the specific model. Once all the models have been trained with every combination of hyperparameters we can just pick the hyperparameters that have the highest accuracy.

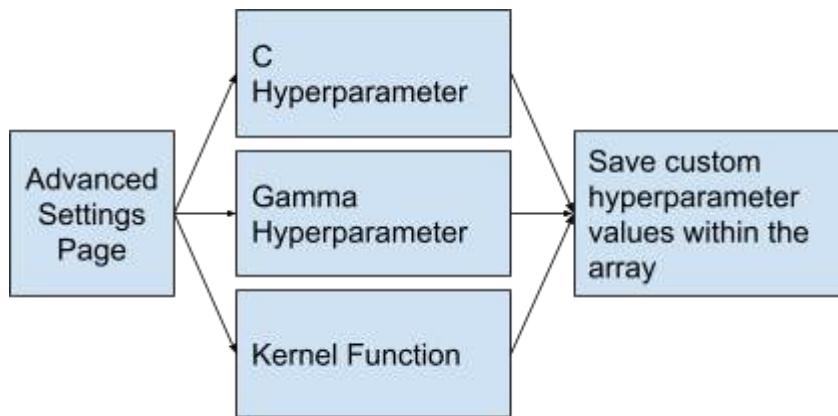


tkinterStartPage():



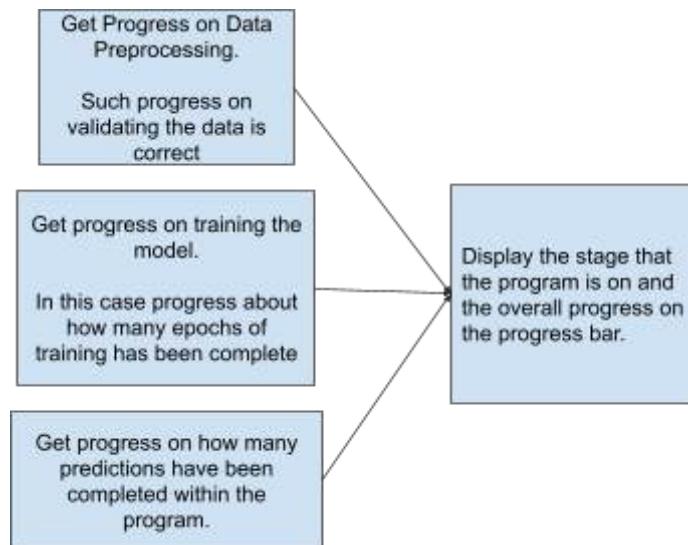
This class will contain all the user interface elements to access within the start page. When the user selects a specific crop it will be saved within the variable 'crop'. The main interface elements within the user interface page should be the ability to select a crop, advanced settings and exit. This class should allow the user to access all three of these functions on a singular page.

tkinterAdvancedSettingsPage():



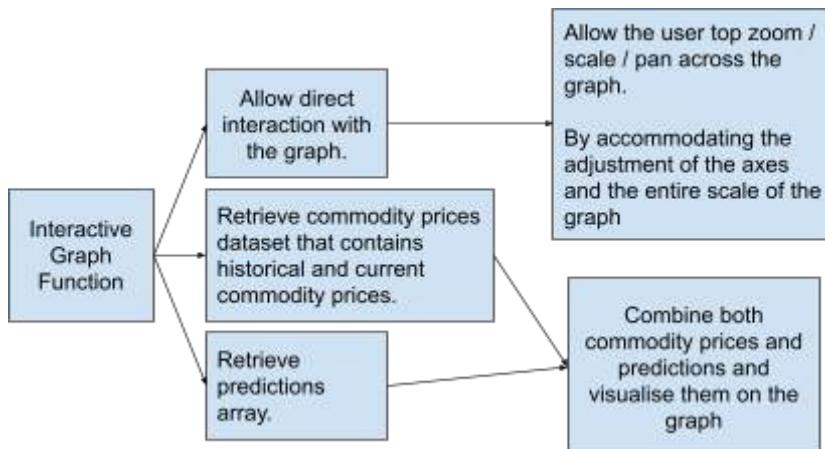
optimise the model as the user wants the model to be trained on their own custom hyperparameters.

tkinterLoadingPage():



This class will contain the user interface elements that are accessible to be edited on the advanced settings page. This will include the kernel function, c and gamma value. Once they are edited they're custom values will be saved to train the model with. In this case if the user inputs custom hyperparameters grid-search cross validation won't occur to find and

tkinterGraphPage():



The graph page will need to output the produced predictions overlaid on the historical and current agricultural commodities price data. To do so we will implement a single class to handle all these processes. The final product should be an interactive graph that the user is able to zoom / pan / scale around to view all the data at once.

Validation within each Stage of the Solution

Identified, explained and justified necessary validation within our solution.

Our model requires validation to ensure data is always accurate, consistent and meets the requirements of processes we are using. This is important as we can use validation techniques to accurately identify anomalies, missing values and spot errors before it goes through our program.

Our main aim in implementing validation is to improve the overall accuracy of the support vector machine which is created to classify forecasted weather data to make predictions as to the change in price of an agricultural commodity.

Validation Technique	Explanation	Justification
Before input coordinate data into the weather retrieval API we use a class to verify that the coordinate components are valid.	This is to make sure the weather API is only inputted with valid coordinates. If the coordinate is invalid it will be skipped and the next location will be processed.	This is necessary as we are working with multiple locations where the crop is grown so validating the coordinate is correct means that we do not retrieve incorrect data which could cascade into inaccurate predictions.
Cleaning and formatting data before it is used to train the model.	This process is done to catch inaccuracies within the data we retrieve externally as we need to verify it is valid. This means that we flag missing and anomalous values and ensure the support vector machine can process it.	The support vector machine is extremely sensitive to the quality of the data. It is not able to classify missing or anomalous data effectively so flagging these values and substituting them with appropriate ones leads to accurate training of the model.
Error handling implemented across the program	Our program is handling multiple new pieces of information at a time such as retrieved weather and commodity price data. Having clear error handling means that	If the program is able to anticipate potential errors such as data not being retrieved then they can catch them before it compromises the stability of our entire program as unexpected issues

Validation Technique	Explanation	Justification
	instead of the system crashing the user can know what has gone wrong and can take steps to solve it.	may occur if this is not the case such as the program constantly crashing.
Hyperparameter validation involves making sure custom hyperparameters inputted by the user are correct.	As the user has the option of inputting custom values for the hyperparameters validation could ensure that the values are valid and can be used.	This means that the model is trained correctly and within the users requirements that were defined as part of the customs hyperparameters.
Cross-validation is used to make sure that we can evaluate the support vector machine's accuracy effectively without bias.	This biased nature may be caused by overfitting data. Cross-validation ensures that the model is accurate by using all the data for both training and testing. This is in comparison to testing the accuracy of a dataset with the same model for both training and testing.	This cross-validation is important to ensure that the model is able to effectively generalise data. Generalisation means that the model is able to understand the underlying meaning of data instead of memorising the training data. With cross-validation we can measure how the model performs effectively to unseen data and therefore its real-world performance.

Identifying Test Data for Development and Alpha Testing

Identify and Justify test data that will be used during development. This includes test data that will be used to test the functionality of our solution as it is developed.

Test data for development will be split into the four stages within our process and the expected output for all of them. Alpha testing specifically is a process that involves testing the program while it is being developed so this means that we will take into account the internal structure of the program.

These four stages can be differentiated as four layers that data will pass through in the following order. Input, Processing, Prediction and Output.

As part of thinking ahead for development we will test the following functionality within each layer making sure to verify and validate each layer's key steps. The following tables indicate what should be tested within each layer when developing the program. These are the key processes that we aim to happen across each layer as data cascades through all four layers. Each layer also includes tests for edge cases; these are scenarios or inputs that are outliers and are extreme scenarios that the system should still be able to handle.

Input Layer:

Identification	Explanation	Justification
Successfully allow the user to select a crop from a range of other crops	The program should be able to allow the user to select and save a specific crop they want to analyse.	This is a crucial step as the user should be able to select a crop which will be referenced across the entire program.
Retrieve historical and forecasted weather data for the selected crop	Retrieval of historical and forecasted weather from external sources will need to be handled.	Weather data is used as the features component to train the model.
Retrieve historical and current commodity prices for the selected crop	Retrieval of historical and current prices from external sources such through web scraping tools will need to be implemented and work autonomously.	Commodity price data is used as the labelled component to train the model.
Allow the user to access and change advanced settings such as hyperparameters.	The program should allow users to adjust advanced settings such as the hyperparameters which affect how the machine learning model is trained.	User's want to be able to customize the model to meet their needs. This customization of hyperparameters should be applied and move to the next layer.

Input Layer Edge Cases:

Identification	Explanation	Justification
Inputting extremely large valid values for hyperparameters within the advanced settings	The program should be able to handle a range of inputs including ones outside the normal range. The advanced settings page allows inputting custom values for	Advanced settings should be able to handle the input of hyperparameter values that are outliers and also ones within the normal range without the system

page.	both gamma and C hyperparameter values.	incorrectly producing errors.
Corrupted commodity price datasets and climate datasets.	Datasets may be corrupted which means that values may be missing / anomalous or data may need to fully be retransmitted if the dataset is incomplete.	Utilising external sources for data means that our solution will need to be able to handle both missing and anomalous values within datasets.
Retrieving data from external APIs with an unstable internet connection.	Validation measures should be in place to request retransmission of packets in case they haven't been appropriately sent.	The solution should be able to automatically debug itself and maintain a complete dataset before the machine learning model is generated.

Processing Layer:

Identification	Explanation	Justification
Merging weather data from multiple sources into a single dataset	Each crop is grown in multiple locations so there are multiple weather locations to retrieve data from.	Merging all three weather datasets ensures that the model has complete data that represents the environment that the crop is grown in to make accurate predictions.
Detecting anomalous and missing values and amending them.	We need to validate that the data effectively checks for errors within the dataset and substitutes them effectively.	This will mean that the model is trained on high quality data and this does not negatively impact the model's predictions.
Standardising weather data by scaling and centering it.	Standardising data involves transforming it to have a mean of 0 and a standard deviation of 1.	This means that all the data is on the same scale and doesn't negatively influence the model. Also improving the training speed and accuracy.

Processing Layer Edge Cases:

Identification	Explanation	Justification
Train Support Vector Machine with extremely large gamma and c hyperparameter	Allowing the user to set custom hyperparameter values means that the system should be able to deal with requests that	Large gamma and c hyperparameter values means that the model may have to overfit the training data to produce an output

values.	require the model to do extreme processing.	with the initial required case. In this case the system would indicate the real time progress through the progress bar.
Large number of sample data points to train the Support Vector Machine model on.	If the training data set has a large number of individual data points the system may either sample a proportion of these data points or distill them into a smaller dataset.	This means that the model is able to handle all dataset inputs at the training stage without sacrificing either time or necessary initial processing power.

Prediction Layer:

Identification	Description	Justification
Validating that a valid hyperparameter grid has been produced that shows all the possible combinations of hyperparameters.	We need to do this so we can verify that during grid search cross validation an effective combination of hyperparameters are produced.	This would mean if it works effectively that we can effectively optimise the model by having an exhaustive search across the entire grid to find the optimal hyperparameters.
Predictions should be correctly produced and indexed to the time segment they are predicting.	This means that we should be able to process the array of predictions and generate the change in price over that same time period.	Our entire solution relies on both the effective production and visualisation of predictions so this is a crucial step in connecting both of these steps.

Prediction Layer Edge Cases:

Identification	Explanation	Justification
Extreme weather events within the forecasted weather data	Weather outside the normal range is most likely still within the valid range of specific weather variables therefore they should still be taken into account when making predictions.	Extreme weather events, if representative of the environment, may have knock-on effects on the agricultural commodity market therefore it is necessary to still retain them within the original dataset.
Missing or	Missing values will need to	Datasets that contain

anomalous values within the forecasted weather dataset.	be substituted before a prediction can be made for that datapoint. Anomalous values outside the valid range for weather variables will also have to be substituted so the dataset overall is kept accurate.	missing or anomalous values may cause the model predicting the future price of the crops commodity to be increasingly biased causing it to make inaccurate predictions even though the original dataset may have contained a large majority of representative values.
---	---	---

Output Layer:

Identification	Description	Justification
Graph the predictions and the historical commodity prices correctly.	All the separate graphs should all be visualised in harmony across the entire solution.	This means that users can easily follow and make trends between all the data points.
The graph should be interactive.	Interactive means that the user themselves can interact and view data personally without being set into predetermined fixed solutions.	Interaction is a clear component as it means our system is fluid and feels seamless to use.

Output Layer Edge Cases:

Identification	Explanation	Justification
Missing historical data for agricultural commodity prices.	Datasets with missing values should be substituted by taking the average of the time segment and substituting the missing value with the averaged one.	Historical datasets may contain missing values if sourced from external sources such as an API.
Extreme zooming and panning into specific time segments	The solution should be able to handle large changes in the x,y axis scaling and view. This means that the final solution should easily allow fluid transitions between multiple different viewpoints.	As the entire graph is interactive it should allow for quick and responsive transitions between multiple different viewpoints. Additionally this is necessary due to the inclusion of touchscreen devices which the solution will need to

		handle.
Quickly switching across predetermined time segments such as viewing data across daily values to yearly values.	The solution should be able to quickly switch across multiple time segments while retaining a fast response time.	Graphs should render quickly so the user is able to efficiently extract data and a valid output from our solution without needing to wait for the solution to process values.

Justifying Test Data for Black Box Testing

The previous section indicates the testing processes that will happen during the development stage alongside this we will implement a black box testing approach to the entire program.

Black box testing is a method of software testing that examines the functionality of an application without looking at the internal structure of the application. This means that we would be looking at the final output of each layer and not the steps internally that the solution will take to come to this output.

Focusing on ensuring the model produces expected outputs we will split the expected outputs for the entire program from a tool that forecasts agricultural commodities using climate patterns into expected outputs within each layer. This means that each layer has a component that it produces in order to correctly embed itself within the entire program. The final acceptance testing will involve stakeholders using and trying the entire completed solution however within key points of the program each layer will be tested in a similar format.

Input Layer:

The expected output of the input layer is as a starting page of the program. This means that during this layer, elements should allow for crop selection and access to advanced settings as they are purely initialising settings. The main outputs of this stage should be a clear user interface that the user is able to navigate through. It should be both responsive and clear and concise. Accessibility is another component as all types of stakeholders should be able to easily interact with our product.

Processing Layer:

This layer involves a lot of static information production. This is because in this stage all the data is being processed. There are three main static components that should be verified in this layer. These are the progress bar, the progress table and an error table. The progress bar should successfully show the real-time view of all the data being processed at a time. While the progress and error tables should constantly refresh to indicate changes within what is currently being processed and any errors that might occur. There is not a specific component to test. Instead we would be testing the successful execution of the entire processing layer as this stage is purely static.

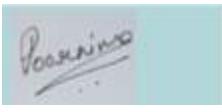
Prediction Layer:

The prediction layer uses the support vector machine model produced in the previous layer and manages its appropriate execution. This layer is also part of the processing layer as it integrates directly into the same user interface. Therefore there is not a specific component to test. Instead we would be testing the successful execution of the entire prediction layer as this stage is purely static

Output Layer:

The output layer outputs the final prediction to the user. Specific components do not have to be tested; instead, immediately after prediction finished we will need to verify that a detailed prediction is overlaid appropriately on the graph alongside the interactive graph that should be able to be interacted with appropriately. This layer compared to the previous processing and prediction layers are fully interactive as to give end-users the ability to visually interact with the data instead of seeing a static prediction.

Stakeholder Design Feedback and Sign Off

Stakeholder	Feedback
Mrs Ganeshgudi	  After discussing the steps we have taken to refine the solution Mrs Ganeshgudi agreed with the final proposed solution. This was after implementing an active and static approach to our user interface which she agreed with as it meant the whole process was simplified and made to be concise. All of this meant that it appealed to her as it met her intended specification of a solution that is both easy to use and informative.

Mr Shreeharsh	 Mr Shreeharsh Rao	 16 / 03 / 2025
		<p>Mr Shreeharsh specifically liked the implementation of time segment predictions to show the way the future price will move as compared to a single static prediction that we had in the beginning. This particularly was important to him as his goals of using this tool for research into how the crop market is impacted by the weather. Ultimately we agreed about the intended solution as it allowed both novice and advanced users to use this tool in greater detail.</p>
Mr Manu		 16 / 03 / 2025
		<p>Mr Manu was initially concerned with the limited accuracy of this model. This was mainly because of our implementation of only using a limited number of four main weather metrics. However after implementing a larger time range of data from 1960-2024, we agreed to have a trade off between the debatable quality of the data compared to the sheer quantity of data. All of this meant that we agreed on the proposed solution. A main point was that we wanted to keep our tool accessible to all users so limiting the amount of weather metrics we use was a choice we had to make after consulting our diverse collection of stakeholders.</p>
Mr Nagendran		 15 / 03 / 2025
		<p>Mr Nagendran is an advanced user. After his also similar concerns with the accuracy of the model we looked into the specific optimisation of our hyperparameters. What this meant is that we used a grid search cross validation to make sure that if the user doesn't change the hyperparameters via the advanced settings page our tool will automatically optimise all the hyperparameters. This was a breakthrough for Mr Nagendran as it meant he could rely on this tool even as time went on as he understood how the patterns in a dataset might change over time and that different hyperparameters might be able to perform better in certain scenarios. Ultimately we agreed on the proposed solution as it was a clear harmony between advanced settings and the simplicity of our solution. By implementing optimization autonomously within our program we were able to capture accuracy for both advanced and novice users.</p>

Development of the Coded Solution

Building an Initial Prototype

Our first step of developing this solution was to build a simple proof of concept of this program. The reason we did this is to show stakeholders an example implementation of our program and give a clear starting point in which we can iterate through our code and build on top of the foundational ideas we have already defined.

We first implemented an index with all the crops so that we can access a standardised array across the entire program. This is also why it has a global scope. The reason we are implementing this function is to be able to have an array of definitions of the possible crops the user can access and use. This also allows for maintainability in the program as we can easily amend crops by adding and removing them from the main array.

```
# This is a global crop index with all the possible crops we have available.

global cropIndex
cropIndex = ['cocoa', 'soybean', 'corn', 'wheat', 'sugar']
```

This is the error function that we have implemented within the solution. This is a proof of concept as we need to build a function to be able to catch and handle errors without it impacting other functions within the program. Catching an error involves detecting it and executing this error function. Handling the error involves both informing the user as to the exact error that has just occurred and the next steps either the user or the program will take to fix it and get back to normal operations.

```
# This is an error alert function and will be what is called when an error is caught within the program.

#This function allows the program to make decisions when errors occur.

def errorAlert(error,type):
```

```

print('error encountered')
print(error)

#The type of the error defines the next steps to be taken after
the error has been detected.

#In this case there are two types: a soft and a hard reset.

#A soft reset will return back to the start page, whereas a hard
reset will initialize the entire program.

if type == 'soft reset':
    print('soft reset - returning to start page')
else:
    type == 'hard reset'
    print('hard reset - resetting entire program')

```

17 errorAlert('no internet','hard reset')

✓ 0.0s

```

error encountered
no internet
hard reset - resetting entire program

```

Testing this program involved calling the function with a predetermined error in the correct format. We can see when we call the error function above with the exact error code and suitable solution the program is able to display the error and then execute a specific task to amend the program. This adds to the program's resilience as it is able to handle a range of errors. The error function will be implemented alongside validation segments as all inputs, specifically data needs to be validated to verify that it meets the predetermined requirements.

Weather data of a single crop is grown in multiple locations therefore we need to accommodate all the locations within a program. To do this we use an array to store the appropriate coordinates of the top three major locations of where a crop is grown. This is expanding on the original cropIndex array and defines the exact regions each crop is grown. These regions are used to define the environmental weather conditions that crops are grown in.

For example in 2022 Ivory Coast, Ghana and Indonesia accounted for the majority of worldwide cocoa bean production so these three were the locations chosen for the top three producers of cocoa. The cocoaCoordinates array defines the main regions within each country that the crop is grown. This allows easy access for the program to access these locations within an indexed array and easy maintainability as you

can easily update the coordinates to represent the current locations of global production.

Cocoa bean production

2022, millions of tonnes

 Ivory Coast	2.23
 Ghana	1.10
 Indonesia	0.68

country accounts for.

The volume spread is an array for a specific crop that shows the amount of volume of crops produced that each of the three locations produce relative to each other.

For instance for worldwide cocoa bean production the following countries produce the most during 2022. Ivory Coast, Ghana and Indonesia. The table below shows the relative percentage that each

Country	Percentage
Ivory Coast	56 %
Ghana	27 %
Indonesia	17 %

The component below defines the coordinates of the region where the crop is grown and the volume each region produces. These two arrays are defined for each individual crop. In this case this means that there are 5 available crops that can be analysed. Cocoa, Soybean, Corn, Wheat and Sugar.

```
# crop + Coordinates indicate the top three major producers of that
crop around the world.errorAlert
# These three major producers are represented as three distinct
coordinates with a latitude and a longitude component
#
#The volume spread indicates what percentage of the produce that each
location produces.

cocoaCoordinates = [[6.053110308791517, -5.632365625597256],
                    [5.723258880120529, -2.0289499687551404],
                    [-1.4526661026550038, -80.30948687187554]]
cocoaVolumeSpread = [0.660, 0.187, 0.153]
```

```

soybeanCoordinates = [[-19.64619384124217, -54.26514130936858],
                      [42.29404827977364, -92.81193554839746],
                      [-35.06747636366788, -58.071092994544266]]
soybeanVolumeSpread = [0.487, 0.360, 0.153]

sugarCoordinates = [[-22.591058675268453, -48.380706513546365],
                     [27.512949904632368, 80.63046034857008],
                     [16.17804967961032, 103.56698522101857]]
sugarVolumeSpread = [0.471, 0.404, 0.125]

wheatCoordinates = [[33.91053065183956, 113.67123697952496],
                     [47.904101854372406, 2.055730082984269],
                     [27.166528613792373, 80.59817535483508]]
wheatVolumeSpread = [0.357, 0.353, 0.290]

cornCoordinates = [[42.685090443052125, -93.32174099559683],
                   [40.954928136698115, 117.03116790283333],
                   [-11.258015767551417, -54.79968906997016]]
cornVolumeSpread = [0.474, 0.377, 0.149]

```

As once the program is running there is no need to change the values of the coordinates; we can iterate the program to maintain their values and not allow the array to be mutable by implementing an immutable tuple. This means that the sequence of elements and data values itself within the tuple are immutable and cannot be changed while the program is running. An example of an array and a tuple are shown below.

```

array = [1, 2, 3, 4]
#Prints the entire array.
print(array)

#Arrays are mutable and allow individual data elements to be
#changed while the program is running
array[3] = 0
print(array)

/python/output :
#This shows that an array's data elements can be amended while the
#program is executing instructions.
[1, 2, 3, 4]
[1, 2, 3, 0]

```

Whereas on the other hand we can use immutable tuples and try to see what happens if we try to change a value.

```

tuple = (1, 2, 3, 4)
#Prints the entire tuple.
print(tuple)

#Tuples are immutable and don't allow individual data elements to be
changed while the program is running.
tuple[3] = 0
print(tuple)

/python/output :
#This shows that an array's data elements can be amended while the
program is executing instructions.
(1, 2, 3, 4)

TypeError: 'tuple' object does not support item assignment

```

Our entire program relies on data elements that are defined above. Therefore we have decided to improve on the current implementation by saving them as a tuple. This means that they can easily be kept constant within the program as tuples are immutable. So its state cannot be edited while the program is running. This adds to the robustness of the program as essential values are stored securely and can't be edited while the program is running.

```

cocoaCoordinates = ((6.053110308791517, -5.632365625597256),
                    (5.723258880120529, -2.0289499687551404),
                    (-1.4526661026550038, -80.30948687187554))
cocoaVolumeSpread = (0.660, 0.187, 0.153)

soybeanCoordinates = ((-19.64619384124217, -54.26514130936858),
                      (42.29404827977364, -92.81193554839746),
                      (-35.06747636366788, -58.071092994544266))
soybeanVolumeSpread = (0.487, 0.360, 0.153)

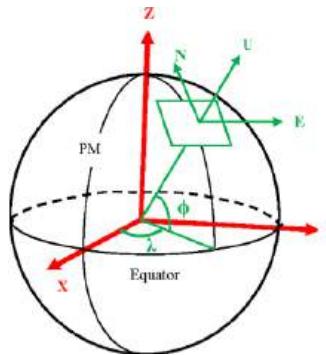
sugarCoordinates = ((-22.591058675268453, -48.380706513546365),
                     (27.512949904632368, 80.63046034857008),
                     (16.17804967961032, 103.56698522101857))
sugarVolumeSpread = (0.471, 0.404, 0.125)

wheatCoordinates = ((33.91053065183956, 113.67123697952496),
                     (47.904101854372406, 2.055730082984269),
                     (27.166528613792373, 80.59817535483508))
wheatVolumeSpread = (0.357, 0.353, 0.290)

cornCoordinates = ((42.685090443052125, -93.32174099559683),
                   (40.954928136698115, 117.03116790283333),
                   (-11.258015767551417, -54.79968906997016))
cornVolumeSpread = (0.474, 0.377, 0.149)

```

Initially the coordinates of the respective crops passed directly into the external API for weather data retrieval processes. However if the coordinates for any reason were corrupted or incorrectly passed into the model there aren't any validation steps to prevent passing incorrect coordinate values into the API. Therefore it is essential to iterate the previous program and add a component that validates the coordinates.



We need a way to validate whether a coordinate is valid or not. This is because it is in a specific format for the openMeteo request which only accepts latitude and longitude components in the Geographical WGS84 coordinate system. This is a standard that we need to conform to as the external weather data which we are retrieving requires this. Failing to validate that the data is in this specific format will mean that the coordinate may be misinterpreted as a different coordinate system which may

be in a different geographic region irrelevant to the original data point. Validation in this adds both to the maintainability of this solution in the future and the accuracy of our results. As the user can guarantee that at multiple stages within the solution there are error detection and validation.

```
def verifyCoordinateValidity(latitude,longitude):#Returns validity of
the coordinates
    try:
        if -90 < latitude < 90:
            validLatitude = True
        else:
            validLatitude = False #Invalid latitude. The value is not
between -90 and 90
    except:
        validLatitude = False #Invalid input. The value is invalid as
it is not a numerical value.

    try:
        if -180 < longitude < 180:
            validLongitude = True
        else:
            validLongitude = False #Invalid longitude. The value is
not between -180 and 180
    except:
        validLongitude = False #Invalid input. The value is invalid
as it is not a numerical value.
```

```

    if validLatitude == True and validLongitude == True:#The entire
program will return either a valid / invalid statement that
corresponds to the coordinate components.

        return 'valid values'

    else:

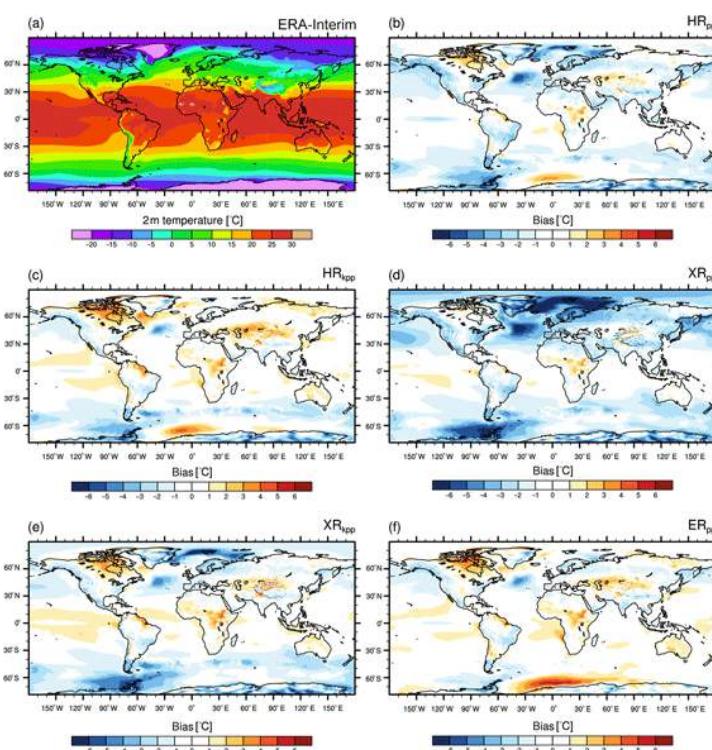
        return 'invalid values'

#We have in built validation within this piece of code by using a
try: except: statement

#if we pass a string value for instance through the first if
statement that compares numbers it will produce an error.

#By using an except case we can set the latitude and longitude to be
false.

```



Initially we decided to try and download the entire climate dataset and store it on the user's device. As the Max Planck Institute Earth System Model is open source it is freely available to download for offline use. This means that we can potentially store historical climate data on the user's device locally. This may mean that retrieval tasks occur quicker. In the image we can see the areas that the model covers and the average temperature of these regions.

However the overall size of the model which was 200GB meant

that it was not suitable to be downloaded locally on the users device. Instead we opted for a system that relies on external datasets that can be requested and transmitted to the user's device. The implementation of this historical weather dataset retrieval process is detailed below.

Retrieve historical weather data from openMeteo:

We use the component below to retrieve historical weather data from openMeteo. This module requires validated latitude and longitude variables to be passed onto it and it responds with a dataset of daily historical weather data for the exact region. A problem I first experienced was the program crashing when it was unable to connect to the API. This was because there were no validation techniques implemented to maintain the connection in the case it was ever either disconnected or the program was entirely offline. I used the retry-cache module to constantly try to reconnect to the data source. This meant it was straightforward for the solution to request and retrieve data without any additional errors as the program was able to handle intermittent network downtime.

Alongside this I have chosen to implement a specific type of database when storing datasets called a Pandas Dataframe. We have implemented a pandas dataframe because of its ability to be used alongside and interchangeably with a .CSV(Comma-Separated Values) file type. Pandas also natively support data manipulation which allows us to efficiently and quickly execute operations and large datasets.

After processing and outputting all the parameters a single historical daily weather dataframe called daily_dataframe.

```
def openMeteo_historical_climate(latitude, longitude):  
  
    #This is a validation technique used to retry the connection  
    #constantly if the connection were to disconnect for any reason.  
    cache_session = requests_cache.CachedSession('.cache',  
    expire_after = 3600)  
    retry_session = retry(cache_session, retries = 5, backoff_factor  
    = 0.2)  
    openmeteo = openmeteo_requests.Client(session = retry_session)  
  
    url = "https://climate-api.open-meteo.com/v1/climate"#This is the  
    url we use to directly access openMeteo's weather API.  
  
    #Parameters define data we input into the historical weather API.  
    params = {  
        #The location (coordinates) are expressed with the latitude and  
        #the longitude coordinates.  
        "latitude": latitude,  
        "longitude": longitude,  
  
        #The start and end-date have been set to predetermined values.
```

```

    "start_date": "1960-01-01",
    "end_date": "2023-11-01",
    "models": "MRI_AGCM3_2_S",
    "daily": ["temperature_2m_mean", "cloud_cover_mean",
"relative_humidity_2m_mean", "precipitation_sum"]
}

responses = openmeteo.weather_api(url, params=params) #We pass the
parameters that we defined straight into and through the API.

response = responses[0] # Process first location. Add a for-loop
for multiple locations or weather models which we can do using
multiple parameter tables.

#Our implementation processes locations one at a time however all
the data can theoretically be retrieved at once.

#This is so we can clearly distinguish different pieces of data
which would be difficult if it was all stored within a single
dataframe.

#These are identifiers which highlight the location and timezone
of where the crop is grown.

#These identifiers can be used to debug the program to find out
what data is included within the dataframe.

debugCoordinates = (f"Coordinates {response.Latitude()} °N
{response.Longitude()} °E")

debugElevation = (f"Elevation {response.Elevation()} m asl")
debugTimezone = (f"Timezone {response.Timezone()}"
{response.TimezoneAbbreviation()}")
debugTimezoneDifference = (f"Timezone difference to GMT+0
{response.UtcOffsetSeconds()} s")

# Process data in a daily format. By doing so we can extract the
four metrics we are looking for.

#These metrics are temperature, cloud_cover, humidity and
precipitation

daily = response.Daily()
daily_temperature_2m_mean = daily.Variables(0).ValuesAsNumpy()
daily_cloud_cover_mean = daily.Variables(1).ValuesAsNumpy()
daily_relative_humidity_2m_mean =
daily.Variables(2).ValuesAsNumpy()
daily_precipitation_sum = daily.Variables(3).ValuesAsNumpy()

```

```

#We need to generate a sequence of dates. To do this we repeat an
interval between each date.

daily_data = {"date": pd.date_range(
    start = pd.to_datetime(daily.Time(), unit = "s", utc = True),
    end = pd.to_datetime(daily.TimeEnd(), unit = "s", utc = True),
    freq = pd.Timedelta(seconds = daily.Interval())),
    inclusive = "left"
) }

#These populate the data frame with the necessary dates with the
four metrics we just retrieved from the API.

daily_data["temperature_2m_mean"] = daily_temperature_2m_mean
daily_data["cloud_cover_mean"] = daily_cloud_cover_mean
daily_data["relative_humidity_2m_mean"] =
daily_relative_humidity_2m_mean
daily_data["precipitation_sum"] = daily_precipitation_sum

daily_dataframe = pd.DataFrame(data = daily_data)

#We return the entire historical weather dataset as a single
panda's dataframe.

return daily_dataframe

```

Retrieve all the historical climate data for all three crop locations:

We also need an efficient way to retrieve all the locations for all three locations where the crop is grown. All these three separate distinct datasets will also have to be merged into a single one.

After this we need to successfully process the data so it's in the correct format for training the machine learning model. This means that we multiply each weather variable by the respective percentage that region contributes to the worldwide supply. We then combine all the adjusted data into a single dataframe by adding their numerical values. An error I encountered was concatenating the data column as even though it is made up of numbers when concatenating the same date it is treated as a string.

I expected the dates to be unaffected by the addition of the variables however what I noticed in the database after was that they produced a concatenated string. This would negatively impact the solution as the dates would not be readable in the correct ISO8601 format which is specified as “DD-MM-YYYY” stored as a string data type.

Expected Output	Actual Produced Output
01-01-1999 + 01-01-1999= 01-01-1999	"01-01-1999" + "01-01-1999"= "01-01-199901-01-1999"

To fix this I captured the initial dates of a dataframe before I processed it and stored it as a buffer. Then I overwrite the incorrect dates with the corrected date data column. This meant that there was no need to amend the concatenation function, instead by overwriting the incorrected concatenated dates I retained the original date in the respective ISO8601 format.

At the end of the program we produce a single combined historical weather data frame that contains all the data for the specified crop. This component allows the creation of a single dataframe that contains all the historical weather data for a crop. Allowing for the quick retrieval of data using a single compact function. Simplifying the process from a series of data retrieval and processing components to a single component that you pass the specific crop into and it outputs the historical climate data.

```
def retrieve_historical_climate(crop):#This entire function can
retrieve all the historical weather data for a crop and accordingly
merge all its datasets.

    global historicalWeather1, historicalWeather2, historicalWeather3
#We make these data frames have a global scope so they can be
accessed across the entire program.

    #We use a for loop to loop through all three crop locations and
extract the location and the respective coordinates.

    #This process occurs three times as we need to get the 1st, 2nd
and 3rd historical weather dataset. This is because there are three
major locations where each crop is grown.

    for i in range (3):
        latitude, longitude = globals()[crop + 'Coordinates'][i]
#Using globals() instead of using multiple if statements for each
crop we can write efficient code. This means that we can directly
call the crop's coordinates.

        coordinateValidity = verifyCoordinateValidity(latitude,
longitude)#We use validation to verify that the coordinates are
correct and valid.

        if coordinateValidity == 'valid values':#If the specific
coordinates are valid we can generate each historical weather data
```

```

frame one at a time.

        globals()['historicalWeather' + str(i + 1)] =
openMeteo_historical_climate(latitude, longitude) #Generates 3
different data frames, one for each country

        #If data is incorrect we catch it before the error is allowed
to propagate and call the error function.

        elif coordinateValidity == 'invalid values':
            errorAlert('Crop information database contains invalid
longitude and latitude coordinates.', 'hard reset')
        else:
            errorAlert('Coordinates could not be validated. No data
has been requested from the API OpenMeteo.', 'hard reset')

        historicalWeather1.to_csv('historicalWeather1.csv', sep=',',
index=False, encoding='utf-8') #Saves the dataset as a .csv file.
This allows for physical debugging as we can see the .csv file in
human readable format as a csv is tabular.

        historicalWeather2.to_csv('historicalWeather2.csv', sep=',',
index=False, encoding='utf-8') #.csv files and data frames are
interchangeable so it is efficient to save all weather datasets as
CSVs.

        historicalWeather3.to_csv('historicalWeather3.csv', sep=',',
index=False, encoding='utf-8')

        #Converts .csv back into a pandas dataframe. This uses the utf-8
encoding.

        historicalWeather1 = pd.read_csv("historicalWeather1.csv",
header=0)
        historicalWeather2 = pd.read_csv("historicalWeather2.csv",
header=0)
        historicalWeather3 = pd.read_csv("historicalWeather3.csv",
header=0)

        #This is where we manually multiply all the datasets with the
respective percentage volume which that location produces.

        #We multiply every column within the dataset by the respective
volume spread that location produces.

        #
        #This is possible due to the consistent indexing and variable
naming scheme within our program as the index of the coordinate is
equal to the index of the percentage spread.

```

```

#
historicalWeather1["temperature_2m_mean"] =
(historicalWeather1["temperature_2m_mean"] * globals()[crop +
'VolumeSpread']) [0])
historicalWeather1["cloud_cover_mean"] =
(historicalWeather1["cloud_cover_mean"] * globals()[crop +
'VolumeSpread']) [0])
historicalWeather1["relative_humidity_2m_mean"] =
(historicalWeather1["relative_humidity_2m_mean"] * globals()[crop +
'VolumeSpread']) [0])
historicalWeather1["precipitation_sum"] =
(historicalWeather1["precipitation_sum"] * globals()[crop +
'VolumeSpread']) [0])

historicalWeather2["temperature_2m_mean"] =
(historicalWeather2["temperature_2m_mean"] * globals()[crop +
'VolumeSpread']) [1])
historicalWeather2["cloud_cover_mean"] =
(historicalWeather2["cloud_cover_mean"] * globals()[crop +
'VolumeSpread']) [1])
historicalWeather2["relative_humidity_2m_mean"] =
(historicalWeather2["relative_humidity_2m_mean"] * globals()[crop +
'VolumeSpread']) [1])
historicalWeather2["precipitation_sum"] =
(historicalWeather2["precipitation_sum"] * globals()[crop +
'VolumeSpread']) [1])

historicalWeather3["temperature_2m_mean"] =
(historicalWeather3["temperature_2m_mean"] * globals()[crop +
'VolumeSpread']) [2])
historicalWeather3["cloud_cover_mean"] =
(historicalWeather3["cloud_cover_mean"] * globals()[crop +
'VolumeSpread']) [2])
historicalWeather3["relative_humidity_2m_mean"] =
(historicalWeather3["relative_humidity_2m_mean"] * globals()[crop +
'VolumeSpread']) [2])
historicalWeather3["precipitation_sum"] =
(historicalWeather3["precipitation_sum"] * globals()[crop +
'VolumeSpread']) [2])

#As all the historical weather data frames have been processed we
can just add them using an additional function which adds the

```

```

respective numbers.

combined_dataframe = historicalWeather1 + historicalWeather2 +
historicalWeather3

#The date which is a string has also been concatenated so we need
to fix that.

combined_dataframe["date"] = historicalWeather1["date"]

return combined_dataframe

```

An issue that arose after testing by stakeholders to retrieve weather data for a crop such as 'cocoa' was the total length of time it took to retrieve and process data. Taking between 30 - 60 seconds in total to process.

Looking at the weather data we are using in the image below we can see that it contains over 23000 data points. This is because we are retrieving historical weather data from 1960 to 2023 as daily weather data points. This means that the total number of weather data points that we are both retrieving from the internet, processing and saving within our database requires powerful hardware. As specified within our hardware requirements we want our solution to be able to be used by both beginner and advanced users on low-power commercially available hardware such as a laptop.

	date	temperature_2m_mean	cloud_cover_mean	relative_humidity_2m_mean	precipitation_sum
1	2023-10-20 00:00:00+00:00	25.72043	00.00270	90.00111	1.7441014
23306	2023-10-21 00:00:00+00:00	25.674377	51.81887	90.0659	5.8103914
23307	2023-10-22 00:00:00+00:00	25.922302	67.77478	91.07071	8.531095
23308	2023-10-23 00:00:00+00:00	25.820225	100.0	93.07551	7.9951544
23309	2023-10-24 00:00:00+00:00	26.068153	99.68661	92.0803	9.140444
23310	2023-10-25 00:00:00+00:00	26.166082	100.0	92.08509	15.044407
23311	2023-10-26 00:00:00+00:00	26.364008	92.59842	91.08989	11.961441
23312	2023-10-27 00:00:00+00:00	26.311934	67.55433	90.09468	13.540952
23313	2023-10-28 00:00:00+00:00	26.359861	61.510242	90.09947	6.827524
23314	2023-10-29 00:00:00+00:00	26.457788	50.466152	89.10428	6.7582054
23315	2023-10-30 00:00:00+00:00	26.255714	78.42206	91.10906	13.297853
23316	2023-10-31 00:00:00+00:00	26.30364	85.37797	91.11386	16.681513
23317	2023-11-01 00:00:00+00:00	25.701567	88.33388	92.11866	16.59772
23318					

To understand how to conform to our original proposed requirements I talked to stakeholders about potential solutions to our problem. Talking with Sharath he said that to keep accuracy and reduce processing time it is best to look at data at a daily scale. The reason the process was taking too long was because of the return function which was unable to output so many data points at once effectively perhaps because it was not optimised for panda's dataframes. The data

manipulation on the other hand which involved multiplying each column occurred quickly and efficiently so it seemed to be the quantity of data we were handling.

Sharath also suggested this might be an issue for the machine learning model as we cannot train a support vector machine on 23000 data points as the hardware requirements we have do not tolerate that.

To tackle this issue we aimed to reduce the number of datapoints that we return at the end of the function. To implement this we need to convert all the columns into yearly averages and then return it. Using panda's dataframe data manipulation techniques we achieved this by replacing `return combined_dataframe` with the following piece of code.

```
#Let's format the data into yearly averages as we have too many
daily weather data points.

combined_dataframe["date"] =
pd.to_datetime(combined_dataframe["date"])# Convert "date" to
datetime type

combined_dataframe["year"] = combined_dataframe["date"].dt.year# Extract the year

combined_dataframe_yearly_averages =
combined_dataframe.groupby('year').mean()# Calculate the yearly
averages of each column using a data frame manipulation function.

combined_dataframe_yearly_averages =
combined_dataframe_yearly_averages.reset_index() #Resets the index of
combined_dataframe_yearly_averages

#Define the title labels of our new dataframe.

combined_dataframe_yearly_averages =
combined_dataframe_yearly_averages[['year','temperature_2m_mean',
'cloud_cover_mean', 'relative_humidity_2m_mean',
'precipitation_sum']]

#We save this combined dataframe to a csv so we are able to call
it within all components of the program as it is now a global file.

combined_dataframe_yearly_averages.to_csv('combined_historicalWeather
.csv', sep=',', index=False, encoding='utf-8')

#We can return the data frame as a single dataframe that contains
all the combined historical weather values that have been processed
```

```

    to be in yearly time segments.

    return combined_dataframe_yearly_averages

```

Using this solved the original problem as we are able to get yearly averages for the forecasted data speeding up the total execution time and also reducing the total amount of data points we will have to train the support vector machine on all while maintaining the overall accuracy of our data.

In the image below we can see the database that is produced. It has the necessary four values, but with yearly averages instead. We have a reduced total of 64 data points we can use to train the model.

year	temperature_2m_m...	cloud_cover_mean	relative_humidity_2m_m...	precipitation_sum
1960	24.618158388819673	71.76048248687705	84.55257823443443	4.004928399228385
1961	24.4458610305726	65.1706210357863	83.77055745830685	3.9590846602858742
1962	24.552533959750686	68.03420652825206	83.33126928292329	3.5658462978525485
1963	24.719594511668497	71.61908733085752	84.81131542123562	4.633582897680685
1964	24.213332706540985	65.72292918235519	82.0920609126858	3.1921383330867625
1965	24.414164607569866	69.02200849958822	84.25905026173973	3.9732294314197945
1966	24.468981584043835	66.17629705758357	82.38468944010137	3.674241147933022
1967	24.3356309920137	67.64556301164932	81.53701430051233	3.2665962383151097
1968	24.421837332275956	69.56315590230192	84.17628471978415	3.7270545247812024

An error I encountered while unit testing this component was the format of the data not being correctly recognised when the program parsed the database.

```

UserWarning: Could not infer format, so each element will be parsed individually,
falling back to 'dateutil'. To ensure parsing is consistent and as-expected, please
specify a format.

combined_dataframe['date'] = pd.to_datetime(combined_dataframe['date']) # Ensure 'date' is datetime type

```

It occurred due to this line within the `retrieve_historical_climate(crop)` function above. That uses pandas to convert the date from a string format into the correct datetime object which is standardised and allows the date to be easily within the entire solution without needing to do additional conversions to different formats. Instead everything is standardised within the ISO8601 format.

```
pd.to_datetime(combined_dataframe["date"])
```

Even though the datetime component worked correctly standardising the dates into ISO8601 it did so after analysing each date individually and then converting them using a python library called dateutil. To fix this I manually instructed the program to assume there is a mixed number of formats and it did it automatically instead of switching to the dateutil library as a failsafe or redundancy. The corrected datetime implementation is below.

```
combined_dataframe['date'] =  
pd.to_datetime(combined_dataframe['date'], format='mixed')
```

Retrieving forecasted weather data:

Retrieving a current forecast of weather from an API requires a similar sequence of instructions. However instead of getting historical data we are retrieving the future forecasted weather. We use the same verified coordinate inputs as the component that retrieves historical climate data. Additionally we take in the initial date that we want data from. This is to allow for added maintainability in future versions as our program can easily be edited to take predictions for a future date for instance.

We use the same API call mechanism to both connect to our weather data provider OpenMeteo and retry-cache to keep data transmission consistent. This added consistency is due to retry-cache being able to constantly retry requests and make sure data is received correctly.

All retrieved weather within our solution is from OpenMeteo which provides global weather data from historical datasets to extremely accurate future predictions.

The output of this entire component is a single forecasted climate dataframe to a maximum of 16 days in advance containing data that covers daily data points with measurements every hour of the day.

```
def openMeteo_forecasted_climate(latitude, longitude, startDate):  
    #We only input the start date as our program will always use the  
    longest possible forecast length automatically.  
    #The longest possible forecast length is 16 days after the  
    current date.  
  
    # Setup the Open-Meteo API client with cache and retry on error.  
    cache_session = requests_cache.CachedSession('.cache',  
    expire_after = 3600)  
    retry_session = retry(cache_session, retries = 5, backoff_factor  
    = 0.2)  
    openmeteo = openmeteo_requests.Client(session = retry_session)
```

```

#openMeteo has a maximum forecast 16 days after the start date.
This is 15 days as we are including today's date as it is a 0 indexed
counting system.

date_placeholder = datetime.strptime(startDate, '%Y-%m-%d')#
Parse the ISO8601 startDate to extract what the day is.
endDate = date_placeholder + timedelta(days = 15) #Add 15 days to
the current date
endDate = endDate.strftime('%Y-%m-%d')# Format the result back to
ISO 8601 and set that as the endDate of the weather forecast.

#all required weather variables to retrieve are listed here
url = "https://api.open-meteo.com/v1/forecast"
params = {
#The coordinate components
"latitude": latitude,
"longitude": longitude,

#We want data in the highest quality possible so we will retrieve
data in hourly steps.
#Hourly steps means that there will be more data points per day
however we only have a maximum of 16 days so our system will be able
to handle it.

#Name the four metrics we want predictions for.

"hourly": ["temperature_2m", "relative_humidity_2m",
"precipitation", "cloud_cover"],

#define the start and the end data.
"start_date": startDate,
"end_date": endDate
}

#Pass the defined parameters into the forecasting openMeteo
API.and save the responses.
responses = openmeteo.weather_api(url, params=params)

# Process first location. Add a for-loop for multiple locations
or weather models.

#We process all the responses sequentially so it is clear as to
what location is related to what response.

response = responses[0]

```

```

    #We save the identifiers for the program in case errors occur and
    we need to debug exactly what has happened and where forecasted
    weather data has been retrieved for.

    debugCoordinates = (f"Coordinates {response.Latitude()} °N
{response.Longitude()} °E")

    debugElevation = (f"Elevation {response.Elevation()} m asl")

    debugTimezone = (f"Timezone {response.Timezone()}"
{response.TimezoneAbbreviation()}")

    debugTimezoneDifference = (f"Timezone difference to GMT+0
{response.UtcOffsetSeconds()} s")

    # Process hourly data. The order of variables needs to be the
    same as requested.

    hourly = response.Hourly()
    hourly_temperature_2m = hourly.Variables(0).ValuesAsNumpy()
    hourly_relative_humidity_2m = hourly.Variables(1).ValuesAsNumpy()
    hourly_precipitation = hourly.Variables(2).ValuesAsNumpy()
    hourly_cloud_cover = hourly.Variables(3).ValuesAsNumpy()

    #Create an hourly time column for the data.
    hourly_data = {"date": pd.date_range(
        start = pd.to_datetime(hourly.Time(), unit = "s", utc =
True),
        end = pd.to_datetime(hourly.TimeEnd(), unit = "s", utc =
True),
        freq = pd.Timedelta(seconds = hourly.Interval()),
        inclusive = "left"
    )}

    #Save each component that has been retrieved from the forecasted
    weather API into a single hourly data dataset.

    hourly_data["temperature_2m"] = hourly_temperature_2m
    hourly_data["relative_humidity_2m"] = hourly_relative_humidity_2m
    hourly_data["precipitation"] = hourly_precipitation
    hourly_data["cloud_cover"] = hourly_cloud_cover

    hourly_data = pd.DataFrame(data = hourly_data) # Convert dataset
    into a panda's dataframe

    return hourly_data #Return the hourly dataframe.

```

Retrieving the forecasted climate:

We can retrieve the forecasted climate of all three weather locations all at once by using a single function below. This builds on top of the function above as it adds a layer of processing we need to embed within our program and produces a single forecasted climate dataframe that we can directly pass into our model to make predictions.

It follows the same processes as retrieving the historical climate data from openMeteo where each region for where the crop is grown is individually retrieved and stored in a dataframe. Then all three data frames that represent the forecasted weather across 3 different regions are combined to produce a single dataframe that represents each location's weather by the volume it contributed to the global production of that specific crop.

Ultimately the purpose of producing a single weather forecast dataset is that it means that the model which when we input data into only produces a single prediction that conveys the information about the worldwide supply of that specific crop. Instead of having multiple separate predictions across multiple separate regions our goal as set out by our stakeholder requirements was to aggregate multiple regions and produce a single answer that can be both interpreted by the user and successfully outputted on a graph.

```
def openMeteo_forecasted_climate(latitude, longitude, startDate):
    #We only input the start date as our program will always use the
    longest possible forecast length automatically.
    #The longest possible forecast length is 16 days after the
    current date.

    # Setup the Open-Meteo API client with cache and retry on error.
    cache_session = requests_cache.CachedSession('.cache',
    expire_after = 3600)
    retry_session = retry(cache_session, retries = 5, backoff_factor
    = 0.2)
    openmeteo = openmeteo_requests.Client(session = retry_session)

    #openMeteo has a maximum forecast 16 days after the start date.
    This is 15 days as we are including today's date as it is a 0 indexed
    counting system.

    date_placeholder = datetime.strptime(startDate, '%Y-%m-%d')#
Parse the ISO8601 startDate to extract what the day is.
    endDate = date_placeholder + timedelta(days = 15) #Add 15 days to
```

```

the current date
endDate = endDate.strftime('%Y-%m-%d') # Format the result back to
ISO 8601 and set that as the endDate of the weather forecast.

#all required weather variables to retrieve are listed here
url = "https://api.open-meteo.com/v1/forecast"
params = {
    #The coordinate components
    "latitude": latitude,
    "longitude": longitude,

    #We want data in the highest quality possible so we will retrieve
    data in hourly steps.
    #Hourly steps means that there will be more data points per day
    however we only have a maximum of 16 days so our system will be able
    to handle it.
    #Name the four metrics we want predictions for.

    "hourly": ["temperature_2m", "relative_humidity_2m",
    "precipitation", "cloud_cover"],

    #Define the start and the end data.
    "start_date": startDate,
    "end_date": endDate
}

#Pass the defined parameters into the forecasting openMeteo
API.and save the responses.
responses = openmeteo.weather_api(url, params=params)

# Process first location. Add a for-loop for multiple locations
or weather models.
#We process all the responses sequentially so it is clear as to
what location is related to what response.
response = responses[0]

#We save the identifiers for the program in case errors occur and
we need to debug exactly what has happened and where forecasted
weather data has been retrieved for.
debugCoordinates = (f"Coordinates {response.Latitude()} °N
{response.Longitude()} °E")
debugElevation = (f"Elevation {response.Elevation()} m asl")

```

```

        debugTimezone = (f"Timezone {response.Timezone()}"
{response.TimezoneAbbreviation()}")
        debugTimezoneDifference = (f"Timezone difference to GMT+0
{response.UtcOffsetSeconds()} s")

        # Process hourly data. The order of variables needs to be the
        same as requested.
        hourly = response.Hourly()
        hourly_temperature_2m = hourly.Variables(0).ValuesAsNumpy()
        hourly_relative_humidity_2m = hourly.Variables(1).ValuesAsNumpy()
        hourly_precipitation = hourly.Variables(2).ValuesAsNumpy()
        hourly_cloud_cover = hourly.Variables(3).ValuesAsNumpy()

        #Create an hourly time column for the data.
        hourly_data = {"date": pd.date_range(
            start = pd.to_datetime(hourly.Time(), unit = "s", utc =
True),
            end = pd.to_datetime(hourly.TimeEnd(), unit = "s", utc =
True),
            freq = pd.Timedelta(seconds = hourly.Interval())),
            inclusive = "left"
        )}

        #Save each component that has been retrieved from the forecasted
        weather API into a single hourly data dataset.
        hourly_data["temperature_2m"] = hourly_temperature_2m
        hourly_data["relative_humidity_2m"] = hourly_relative_humidity_2m
        hourly_data["precipitation"] = hourly_precipitation
        hourly_data["cloud_cover"] = hourly_cloud_cover

        hourly_data = pd.DataFrame(data = hourly_data) # Convert dataset
        into a panda's dataframe

        return hourly_data #Return the hourly dataframe.

def retrieve_forecasted_climate(crop): #This entire function can
    retrieve all the forecasted weather data for a crop and accordingly
    merge all its datasets.
    global forecastedWeather1, forecastedWeather2, forecastedWeather3
    #We make these data frames have a global scope so they can be
    accessed across the entire program.

```

```

    currentDate = datetime.today()#Retrieve the current data in the
correct ISO8601 format for the API.

    currentDate = currentDate.strftime('%Y-%m-%d')

    #We use a for loop to loop through all three crop locations and
extract the location and the respective coordinates.

    #This process occurs three times as we need to get the 1st, 2nd
and 3rd forecasted weather dataset. This is because there are three
major locations where each crop is grown.

    for i in range (3):
        latitude, longitude = globals()[crop +
'Coordinates'][i]#Using globals() instead of using multiple if
statements for each crop we can write efficient code. This means that
we can directly call the crop's coordinates.

        coordinateValidity = verifyCoordinateValidity(latitude,
longitude)#We use validation to verify that the coordinates are
correct and valid.

        #If the specific coordinates are valid we can generate each
forecasted weather dataframe one at a time.

        if coordinateValidity == 'valid values':
            globals()['forecastedWeather' + str(i + 1)] =
openMeteo_forecasted_climate(latitude, longitude, currentDate)
#Generates 3 different data frames, one for each country
        elif coordinateValidity == 'invalid values':
            errorAlert('Crop information database contains invalid
longitude and latitude coordinates.', 'hard reset')
        else:
            errorAlert('Coordinates could not be validated. No data
has been requested from the API OpenMeteo.', 'hard reset')

        forecastedWeather1.to_csv('forecastedWeather1.csv', sep=',',
index=False, encoding='utf-8') #Saves the dataset as a .csv file.
From numpy to pandas dataframe.

        forecastedWeather2.to_csv('forecastedWeather2.csv', sep=',',
index=False, encoding='utf-8')

        forecastedWeather3.to_csv('forecastedWeather3.csv', sep=',',
index=False, encoding='utf-8')

        #Saves the dataset as a .csv file. This allows for physical
debugging as we can see the .csv file in human readable format as a
csv is tabular.

```

```

#.csv files and data frames are interchangeable so it is
efficient to save all weather datasets as CSVs. This also means that
it is globally accessible across the entire program.

forecastedWeather1 = pd.read_csv('forecastedWeather1.csv',
header=0)
forecastedWeather2 = pd.read_csv('forecastedWeather2.csv',
header=0)
forecastedWeather3 = pd.read_csv('forecastedWeather3.csv',
header=0)

#This is where we manually multiply all the datasets with the
respective percentage volume which that location produces.

#We multiply every column within the dataset by the respective
volume spread that location produces.

#
#This is possible due to the consistent indexing and variable
naming scheme within our program as the index of the coordinate is
equal to the index of the percentage spread.

#
forecastedWeather1["temperature_2m"] =
(forecastedWeather1["temperature_2m"] * globals()[crop +
'VolumeSpread'][0])
forecastedWeather1["cloud_cover"] =
(forecastedWeather1["cloud_cover"] * globals()[crop +
'VolumeSpread'][0])
forecastedWeather1["relative_humidity_2m"] =
(forecastedWeather1["relative_humidity_2m"] * globals()[crop +
'VolumeSpread'][0])
forecastedWeather1["precipitation"] =
(forecastedWeather1["precipitation"] * globals()[crop +
'VolumeSpread'][0])

forecastedWeather2["temperature_2m"] =
(forecastedWeather2["temperature_2m"] * globals()[crop +
'VolumeSpread'][1])
forecastedWeather2["cloud_cover"] =
(forecastedWeather2["cloud_cover"] * globals()[crop +
'VolumeSpread'][1])
forecastedWeather2["relative_humidity_2m"] =
(forecastedWeather2["relative_humidity_2m"] * globals()[crop +
'VolumeSpread'][1])

```

```

    forecastedWeather2["precipitation"] =
(forecastedWeather2["precipitation"] * globals()["crop" +
'VolumeSpread'][1])

    forecastedWeather3["temperature_2m"] =
(forecastedWeather3["temperature_2m"] * globals()["crop" +
'VolumeSpread'][2])
    forecastedWeather3["cloud_cover"] =
(forecastedWeather3["cloud_cover"] * globals()["crop" +
'VolumeSpread'][2])
    forecastedWeather3["relative_humidity_2m"] =
(forecastedWeather3["relative_humidity_2m"] * globals()["crop" +
'VolumeSpread'][2])
    forecastedWeather3["precipitation"] =
(forecastedWeather3["precipitation"] * globals()["crop" +
'VolumeSpread'][2])

#As all the historical weather data frames have been processed we
can just add them using an additional function which adds the
respective numbers.
combined_dataframe = forecastedWeather1 + forecastedWeather2 +
forecastedWeather3

combined_dataframe.drop('date', axis=1, inplace=True) #Instead of
time segments the forecasted weather dataframe we produce only has a
single row.

#This single row contains the average value for each column over
the entire forecasted weather dataset.

#As given by openMeteo's documentation future forecasts may
contain missing data.(empty values)
#Instead of physically removing data points we can just extract
complete data points and equate that to the original dataset.
#This effectively deletes empty data points as they aren't
included within the new dataframe.

combined_dataframe =
combined_dataframe.loc[(combined_dataframe['temperature_2m'] != None) &
&
(combined_dataframe['cloud_cover'] != None) &
&
(combined_dataframe['relative_humidity_2m'] != None) &

```

```

(combined_dataframe['precipitation'] != None) ]

combined_dataframe = pd.DataFrame(combined_dataframe.mean())
#Finds the average forecasted weather for the next 16 days

combined_dataframe.to_csv('combined_forecastedWeather.csv',
sep=',', index=False, encoding='utf-8')#Save it as a csv so we can
use this data frame for debugging and access it from the rest of the
program.

return combined_dataframe.transpose()

```

"Pink Sheet" Data New!

[Commodity prices](#)

December 2024 (PDF)

[Monthly prices](#)

December 2024 (XLS)

[Annual prices](#)

December 2024 (XLS)

Retrieving the current and historical agricultural commodities price:

Our solution needs access to both historical and the current agricultural prices. To do this we have planned to use a cached database alongside using a web scraping tool to access the current commodity prices.

Implementing this would mean we can streamline the entire process by relying on the cached database if for any reason we don't have access to current data. This is because we can retrieve all the current and historical weather data directly from the internet through published commodity market prices through the World Bank Group Pink Sheets. These prices are constantly up-to-date so would clearly be relevant for our

program.

Source: <https://www.worldbank.org/en/research/commodity-markets>

The code below shows the process we have taken to web scrape all the data. Initially we set the target URL for the dataset as worldBank_url. The webpage's HTML is captured by requests and passed onto BeautifulSoup. This module parses all the HTML and searches for the tagged Annual prices data. We have also included validation at this stage as data is being retrieved from external sources if any errors occur it can be caught before the program crashes or stalls. Such as if there is no internet available to download the data or a broken link with no access to the file. The URL of where the file is hosted is downloaded as an excel file directly onto the user's device. The contents of the database are then loaded into a pandas dataframe and returned as an output of the component. This output contains a

single dataframe file that contains commodity price data for a range of commodities including the specific agricultural commodities we are analysing.

```
global worldBank_url
worldBank_url =
"https://www.worldbank.org/en/research/commodity-markets#1"

def get_dataset_link(mainWebsite_url): # Returns the path of
commodity price dataset from the World Bank Group Commodity Markets
page.
    #url: The URL of the World Bank Commodity Markets page.

    #We use validation here with the implementation of a try: except:
statement
    #This means that we can catch the error before it crashes the
entire program.

    #Reasons that we can't access the website may be a firewall or a
lack of an internet connection so we need to communicate these to our
users.

    #an alertDialog within the except clause prints the error why the
user can't access the web page and executes an action to fix it.

    try:

        page = requests.get(mainWebsite_url)
        page.raise_for_status() # Verifies the webpage is reachable.
        html = BeautifulSoup(page.text, 'html.parser') # Extracts the
html

        # Find the specific link containing "Annual prices"
        xlsx_link = html.find("a", href=True, string="Annual prices")

        if xlsx_link:
            return xlsx_link["href"]
        else :
            alertDialog('Could not access commodity market data', 'soft
reset')

    except requests.exceptions.RequestException as error: # Prevents
the code from crashing if the user can't access the webpage and
prints the reason.
```

```

        errorAlert(f"Error accessing web page>>> {error}", "hard
reset")

def download_dataset(fileURL):
    try:#Validation is also used here as we are downloading the
entire file chunk by chunk as you can see.
        #If for any reason while downloading it fails the error is
caught by the except clause and prints the error.
        response = requests.get(fileURL, stream=True)
        response.raise_for_status()

        with open('commodityPrices.xlsx', 'wb') as f:#Downloads the
entire file
            for chunk in response.iter_content(chunk_size=8192):
                f.write(chunk)

        return os.path.abspath('commodityPrices.xlsx')#Returns the
absolute location where the data is stored.

    except requests.exceptions.RequestException as error: #Error
handling
        errorAlert(f"Error downloading file>>> {error}", "hard reset")

def retrieve_commodity_prices(): #Retrieves current and historical
agricultural prices.
    commodityDatasetLink = get_dataset_link(worldBank_url)
    commodityPricesPATH = download_dataset(commodityDatasetLink)

    #Our entire program uses data frames to process data due to
openMeteo's dependency on them and the embedded data manipulation
that can be easily used on them.
    commodityPricesDataframe = pd.read_excel(commodityPricesPATH,
sheet_name = 'Annual Prices (Nominal)')#We read the retrieved dataset
as a pandas dataframe.

    return commodityPricesDataframe

```

Talking with stakeholders we found out that the returned dataframe is in the wrong format. This is because the .xlsx file we retrieved from the World Bank Commodity Pink Sheets had unnecessary empty spaces and a title. All of these unnecessary elements will have to be removed to have a correctly formatted commodity Prices Dataframe which we can effectively pull data from. While unit testing this

components outputted agricultural commodity dataframe the empty rows and addition of titles meant that the title of the fields of the database was incorrectly labelled. This meant that the produced dataframe wasn't able to be easily traversed without the additional formatting which I detail below.



World Bank Commodity Price Data (The Pink Sheet)								
annual prices, 1960 to present; nominal US dollars								
current series are available in nominal and real dollar								
Updated on September 04, 2024								
Crude oil; avege	Crude oil; Brin...	Crude oil; Durb...	Crude oil; WTI	Coal, Austral...	Coal, South Afric...	Natural gas, US	Natural gas, Euro...	Liquefied natural gas, Imp...
(\$/bbl)	(\$/bbl)	(\$/bbl)	(\$/bbl)	(\$/tonne)	(\$/tonne)	(\$/mmBtu)	(\$/mmBtu)	(\$/mmBtu)
CRUDE_PETRO	CRUDE_BRENT	CRUDE_DUBAI	CRUDE_WTI	COAL_AUS	COAL_SAFICA	NGAS_US	NGAS_EUR	NGAS_IMP
1960	1.63	1.63	1.63	—	—	0.14	0.40	—
1961	1.57	1.57	1.57	—	—	0.15	0.40	—
1962	1.52	1.52	1.52	—	—	0.16	0.40	—
1963	1.50	1.50	1.50	—	—	0.16	0.38	—
1964	1.49	1.49	1.49	—	—	0.15	0.38	—
1965	1.42	1.42	1.42	—	—	0.16	0.41	—
1966	1.38	1.38	1.38	—	—	0.16	0.42	—
1967	1.33	1.33	1.33	—	—	0.16	0.46	—
1968	1.32	1.32	1.32	—	—	0.16	0.45	—

To do this we will use the panda's dataframe functions to help to extract only the necessary data we need. We can replace the `return commodityPricesDataframe` in the above code with dataframe manipulation scripts. This involves using the drop function to clear the rows at the top which are empty.

Additionally we use iloc also called integer-location based indexing within Pandas to select rows and columns by their integer position within the dataframe. This allowed me to extract only the specific commodity fields at the position 0, 11, 24, 28, 35 and 45. These represent the date, cocoa, soybean, corn, wheat and sugar respectively.

```

for i in range (5):#We drop 5 rows of empty spaces to make the titles
be the clear column headers of the dataframe.

commodityPricesDataframe =
commodityPricesDataframe.drop(index=(i))

commodityPricesDataframe = commodityPricesDataframe.iloc[:,[0,11,24,28,35,45]]#We extract only the relevant commodities which we
are using to make predictions to be the entire dataset

commodityPricesDataframe =
commodityPricesDataframe.drop(commodityPricesDataframe.index[[1, 2]],
axis=0)This line also removes unnecessary elements.

return commodityPricesDataframe#Returns the correctly formatted
database

```

Testing this by calling the function as `retrieve_commodity_prices()` creates this dataframe. This is correct as it contains all the necessary elements in the correct format and all arranged appropriately. This means that we can effectively traverse and query this database to return specific fields such as the change in the price of sugar across 1960 to 2024.

NaN	Cocoa	Soybeans	Maize	Wheat, US HRW	Sugar, world
1960	0.589017	91.833333	44.5	57.993333	0.066208
1961	0.478558	109.333333	44.956667	58.605833	0.05945
1962	0.457967	100.54	48.653333	64.3325	0.061683
1963	0.552342	110.093333	53.770833	64.486667	0.183233
1964	0.505742	110.505833	54.631667	67.526667	0.126325
1965	0.365308	116.868333	55.6725	59.461667	0.044217
1966	0.517633	125.6625	58.296667	62.954167	0.039808
1967	0.598117	114.725	54.1025	65.71	0.042058
1968	0.720883	110.800833	47.963333	62.771667	0.041933
1969	0.897717	106.746667	52.286667	58.394167	0.070592

Classifying commodity price data into separate categories:

Classifying commodity prices is necessary as there is a need to have distinct labels that define a category. As a prototype version we implemented a simple negative and positive label that defined the change in price of the commodity over that year as either positive or negative. As we are using a support vector machine to classify data into separate categories we need distinct labels to categorise all the data into. The code below categorises the respective commodity price data.

It is necessary to classify the price of the commodity as we are training the model on labelled historical data. These categories represent the labels that will be given to the machine learning model to train itself on. The machine learning model will then be able to itself label future forecasted weather data as to what impact the weather will have on the respective price of the agricultural commodity.

```
def
classify_commodityPrice_Dataframe(crop,commodityPricesDataframe):#Labels to the selected crop's change in price as either positive or negative.

#This for loop matches the crop with its location within the
```

```

cropIndex array.

#The index within the crop in the cropIndex array is the same as
the index within the commodity prices dataframe.

for i in range(len(cropIndex)):
    if cropIndex[i] == crop:
        cropRow_DataframeReference = i
        break

commodityPricesDataframe = commodityPricesDataframe.iloc[:, [0,i
+ 1]]#Extract only the crop column you need and the dates.

pd.set_option('display.max_rows', None)#Display the maximum
amount of rows and columns so all available data is able to be
accessed,
pd.set_option('display.max_columns', None)

commodityPricesDataframe =
commodityPricesDataframe.iloc[1:].reset_index(drop=True)
commodityPricesDataframe.iloc[:, 1] =
pd.to_numeric(commodityPricesDataframe.iloc[:, 1])

#Calculate the change of the price numerically as the difference
from the previous year's commodities price.
commodityPricesDataframe['Change'] =
commodityPricesDataframe.iloc[:, 1].diff()
commodityPricesDataframe.loc[0, 'Change'] = 1

#Categorise the change into either positive or negative by using
two seperate bins which are positive or negative infinity.
#Therefore positive numbers are positive and negative numbers are
negative.
commodityPricesDataframe['Change'] =
pd.cut(commodityPricesDataframe['Change'], bins=[-float('inf'), 0,
float('inf')], labels=['Negative', 'Positive'])

return commodityPricesDataframe#Returns the processed dataframe.

```

We can test the effectiveness of this by calling the function using this code. For instance if we want to categorise the change in the yearly price of wheat. We can use:

```
classify_commodityPrice_Dataframe('wheat', retrieve_commodity_prices())
```

The output produced is the labelled commodity prices. The change in this case assessing whether the price of the agricultural commodity was either Positive / Negative or Neutral. In future iterations of our prototypes we will improve the accuracy of our model by adding a greater range of labels on the initial training data. For instance instead of the model only being able to categorise the general trend of the agricultural commodity if at the labelling stage it was given percentages of the yearly change then the final trained model can attach percentage predictions onto future forecasted data.

World Bank Commodity Price Data (The Pink Sheet)	Unnamed: 35	Change
	1960	57.993333
	1961	58.605833
	1962	64.3325
	1963	64.486667
	1964	67.526667

Training the support vector machine:

Once we have all the labelled commodity price data which we refer to as labels and the climate data which we refer to as features we can pass these along with any hyperparameters into the model to train.

At this stage the model will apply either the default hyperparameters or custom hyperparameters set by the user. Advanced hyperparameters will be a part of a separate advanced settings page within the GUI (Graphical User Interface) of our solution.

Before training data is passed onto the model it is preprocessed by initially dropping the year field for the commodities price and the average price. Therefore each data element is only represented by the label in the change in the commodities price alongside the climate variables during that year.

Validation occurs when we verify that the number of samples in our features and labels dataframes match. This is because there needs to be the same index of values of both for the model to be trained correctly. If there is a mismatch between the two tables then the data cannot be correlated. Talking with stakeholders they set

out in their requirements that they want the machine learning model to be trained with data that is both complete and represents entirely all the data available. What this means is that we cannot implement a solution that trains on the data partially or potentially incorrect data. Therefore if there were to be an error such as a mismatch between the training datasets then we flag an error through the error function alerting the user to take more specific action such as reestablishing a connection with data sources or repairing corruption of the program or database files.

Ultimately the output of this component is a trained machine learning model which is able to make predictions using future forecasted climate events. Such as if extreme wind or temperature will have a negative impact on soybean production over the next few weeks.

```
#Trains the SVM and returns the trained model.

#We input the following into the support vector machine to train it.
# The historical weather, commodity price datasets and
hyperparameters.
#Hyperparameters are adjustable settings that affect how effectively
a performs.

def trainingSupportVectorMachine(historicalWeather, historicalPrice,
hyperparameters):

    X = historicalWeather # The weather is the feature and contains
the four climate metrics
    y = historicalPrice # The labelled prices are the labels and is
what we are trying to train our SVM to accurately predict.

    if hyperparameters == 'no hyperparameters': # If there are no
given hyperparameters, set it to default ones.
        C_value, gamma_value, kernel_value = 10000, 0.01, 'rbf'
    else:
        C_value, gamma_value, kernel_value = hyperparameters['C'],
hyperparameters['gamma'], hyperparameters['kernel']#This allows
advanced users to have access to custom hyperparameters when they
train their model.

    X.drop('year', axis=1, inplace=True)
    y = y.iloc[:, 2:]
    y=y.values.ravel() #Turns dataframe into a 1 dimensional array as
```

```

we only have a single column we can convert it into a single row.

    if len(X) != len(y):#This involves validation as the model checks
    if there is missing data. This will happen if one dataset is larger
    than another dataset so therefore there are missing data points.
        errorAlert('missing data. The length of historical weather
and price data do not match.', 'hard reset')

    # Scale the data so it has a mean of 0 and a standard deviation
    of 1. This gives us previously described performance increases.
    X_scaled = scale(X)

    # Train the SVM on ALL data
    clf_svm = SVC(C=C_value, gamma=gamma_value, kernel=kernel_value)
    clf_svm.fit(X_scaled, y)

    return clf_svm#Return the trained SVM model.

```

We can test if we can create a support vector machine that is trained on cocoa commodities producing a single support vector machine model. The program to test generating a model retrieves the historical climate data for cocoa alongside labelled prices and passes directly into training the SVM (Support Vector Machine) using default hyperparameters.

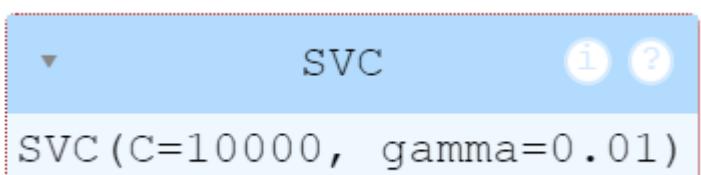
```

weather = retrieve_historical_climate('cocoa')
labelledPrices =
classify_commodityPrice_Dataframe('cocoa', retrieve_commodity_prices())
)

trainingSupportVectorMachine(weather, labelledPrices, 'no
hyperparameters')

```

The output of the entire program is a fully trained support vector machine that is able to accurately classify commodity data. As we can see the final support vector machine has the default hyperparameters as we did not specify specific ones. This means our unit test was successful and the previously created components all can work together to generate a model.



Automatically Find the Optimal Hyperparameters:

We want the program if they are a beginner user to automatically optimise itself. Stakeholder's like Shreeharsh wanted these features as it meant that the model will always be performing most efficiently without needing manual setting by the user. Shreeharsh said the automation part was the most important as it meant that even beginner users like himself were able to make accurate predictions. In comparison to using static default hyperparameters the model would continually update the hyperparameter values based on the current data without needing to request users to set the values themselves.

Responding to his specific request and requirements we built a grid search cross validation algorithm to automatically find the optimal hyperparameters. It is important to know that if an advanced user sets custom hyperparameters those will be prioritised and hyperparameter optimisation will not take place. The program uses grid search cross-validation which involves a grid search over a hyperparameter grid which contains all the different combinations of hyperparameters alongside cross validation testing each individual combination. In the end the hyperparameter values that perform the most accurately on the test data are outputted as the optimal hyperparameters.

```
#Outputs the optimal hyperparameters as a dictionary and creates a processing txt file.

def findOptimal_hyperparameters(crop):
    param_grid = [
        {'C': [1, 10, 100, 1000, 10000], #regularisation parameter C.
         'gamma': ['scale', 1, 0.1, 0.01, 0.001],#Different gamma values
         'kernel': ['rbf', 'poly']}]]#We can choose different kernel functions ,but RBF works well for most.

    #We retrieve all the historical price and weather dataframes.
    X = retrieve_historical_climate(crop)
    commodityPricesDataframe = retrieve_commodity_prices()
    y = classify_commodityPrice_Dataframe(crop
,commodityPricesDataframe)

    #We format each appropriately.
    X.drop('year', axis=1, inplace=True) #We remove the year of the weather dataset as it is unnecessary to train the model.

    #We remove everything except the change in price labels.
    y = y.iloc[:, 2:]
    #As it has a single column we use .ravel() to turn it into a
```

```

one-dimensional array.

y = y.values.ravel()

#We use validation within this part of the program to make sure
we have two complete datasets.

if len(X) != len(y):
    errorAlert('missing data. The length of historical weather
and price data do not match.', 'hard reset')

X_scaled = scale(X)#Scaling and centering the features (X).

#This is the grid search cross validation algorithm.

from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
import numpy as np
param_grid = {'C':[1, 10, 100, 1000, 10000],
              'gamma': ['scale', 1, 0.1, 0.01, 0.001],
              'kernel': ['rbf', 'poly']}

#The program automatically does grid search cross validation as
described.

optimal_params = GridSearchCV(SVC(), param_grid, cv=10,
scoring='accuracy', verbose=2)
optimal_params.fit(X_scaled, y)

return optimal_params.best_params_ # We return the optimal
hyperparameters.

```

This grid search cross validation worked correctly as we can see here when Sharath, our stakeholder, tested optimising a dataset. This was done through a module instead of an algorithm we built from scratch as this is a prototype and we will implement a grid search cross validation algorithm natively within our program.

Fitting 10 folds for each of 50 candidates, totalling 500 fits	The outputs of
.C=10000, gamma=1, kernel=poly; total time= 3.1s	the program showed
.C=10000, gamma=1, kernel=poly; total time= 2.6s	the current processes
.C=10000, gamma=1, kernel=poly; total time= 1.2s	that are taking place. It
.C=10000, gamma=1, kernel=poly; total time= 7.2s	highlighted the number
.C=10000, gamma=1, kernel=poly; total time= 2.6s	of total fits which in
.C=10000, gamma=1, kernel=poly; total time= 13.5s	this case represented
.C=10000, gamma=1, kernel=poly; total time= 12.7s	the 500 different
.C=10000. gamma=1. kernel=polv: total time= 3.9s	combinations we are

testing within the hyperparameter grid. The program showed the currently tested hyperparameters and the time it took to do so.

The final output of the grid search cross validation algorithm was a single line that indicated the most optimal hyperparameters. The datatype of this was a dictionary so the values could easily be passed back into the program.

```
{'C': 10000, 'gamma': 0.01, 'kernel': 'rbf'}
```

The entire process took 76.5 seconds in total which meant that it would have taken longer than the allocated time constraint of 60 seconds for the entire process. As the entire process will take more than a few seconds Mrs Ganeshgudi suggested we add a progress indicator to see clearly what percentage of the program has been completed and how much is left to go and reduce the number of hyperparameters we are testing to shorten the total time needed for the entire algorithm.

● 72 `findOptimal_hyperparameters('cocoa')`
73

✓ 1m 16.5s

Thinking ahead this meant we had to somehow capture the progress of the program and have the ability to display it to the user. Python natively didn't have a method to capture the output of a program within the text box, so we need to find a way to somehow log the progress output of the grid search algorithm and then produce a suitable progress indicator. Replacing `return optimalHyperparameters` with the code below we can do this. In this case we can use a technique called subprocesses to produce a completely separate process or thread that is independent from the rest of the solution. This meant we could output the progress in real-time on a text file. We can work out the total progress of the entire solution just by working out how many lines were written on the final file.

```
# Run GridSearchCV in a subprocess and capture output. Virtual
environment!
command = ["python", "-c",
           "from sklearn.model_selection import GridSearchCV; "
           "from sklearn.svm import SVC; "
           "import numpy as np; "# Import numpy if needed for
x_scaled and y
           "param_grid = {'C':[1, 10, 100, 1000, 10000], "
           "             'gamma':[0.001, 0.01, 0.1, 1, 10, 100]}",
           "grid = GridSearchCV(SVC(), param_grid, cv=5);"
           "grid.fit(x_scaled, y);"
           "optimalHyperparameters = grid.best_params_"
           "print(optimalHyperparameters)"]
```

```

        "'gamma': ['scale', 1, 0.1, 0.01, 0.001], "
        "'kernel': ['rbf', 'poly']); "
        "optimal_params = GridSearchCV(SVC(), param_grid,
cv=10, scoring='accuracy', verbose=2); "
        # Assuming X_scaled and y are numpy arrays converted
        # into a list. As a subprocess is a virtual environment.
        f"X_scaled = np.array({X_scaled.tolist()}); " # Convert to list for string representation
        f"y = np.array({y.tolist()}); "
        "optimal_params.fit(X_scaled, y);"
        "print(optimal_params.best_params_)"
    ]

#Run the command above and then dump all the outputs line by line
into a text file.
with open('processingOutput.txt', 'w') as outfile:
    subprocess.run(command, stdout=outfile,
stderr=subprocess.STDOUT, text=True)

with open('processingOutput.txt', 'r') as file:
    for line in file:
        pass
    optimalHyperparameters = line
    optimalHyperparameters = optimalHyperparameters.strip()

#Extract the optimal hyperparameters from the last line of
the outputted text file and return it to the user.
#We use ast.literal_eval to interpret the string in the text
file into a dictionary.
optimalHyperparameters =
ast.literal_eval(optimalHyperparameters)

return optimalHyperparameters

```

Testing this we can now capture the progress of our grid search cross validation algorithm by capturing the amount of lines that have been written within the processingOutput.txt file. In this case we still retain the same optimal hyperparameter value output as before, but now are able to capture the processing stages progress. In case we process another crop's prediction consecutively the

processingOutput.txt file will be fully overwritten to show the new processing stages progress.

```
  processingOutput.txt
493 [CV] END .....C=10000, gamma=0.001, kernel=poly; total time= 0.0s
494 [CV] END .....C=10000, gamma=0.001, kernel=poly; total time= 0.0s
495 [CV] END .....C=10000, gamma=0.001, kernel=poly; total time= 0.0s
496 [CV] END .....C=10000, gamma=0.001, kernel=poly; total time= 0.0s
497 [CV] END .....C=10000, gamma=0.001, kernel=poly; total time= 0.0s
498 [CV] END .....C=10000, gamma=0.001, kernel=poly; total time= 0.0s
499 [CV] END .....C=10000, gamma=0.001, kernel=poly; total time= 0.0s
500 [CV] END .....C=10000, gamma=0.001, kernel=poly; total time= 0.0s
501 [CV] END .....C=10000, gamma=0.001, kernel=poly; total time= 0.0s
502 {'C': 10000, 'gamma': 0.01, 'kernel': 'rbf'}
```

✓ 1m 18.5s

```
{'C': 10000, 'gamma': 0.01, 'kernel': 'rbf'}
```

As required our program captures the final line when all the data has been processed and stores it within a dictionary data structure and returns it as an output of the operations.

Modules we used within this prototype:

We used modules within this program to simplify doing complex tasks. For example retrieving specific elements from a HTML page is difficult, but when we used BeautifulSoup we were able to both find and extract a spreadsheet from a website without having to understand the underlying process of porting a website into python. The following modules all do different specific tasks. These modules and libraries allow us to both simplify and improve efficiency of the solution when we develop it and allow us to add more complex logical components within our program without needing to understand or develop low-level functionality within the solution.

```
#We install pandas to store the dataframes. Dataframes are used to
store our weather and commodity price datasets.
import pandas as pd

#Sklearn is an open-source machine learning library in python and is
what we use within our program to implement an SVM.
from sklearn.svm import SVC

#We import a scale module from sklearn to make the weather(features)
dataset have a mean of 0 and a standard deviation of 1.
from sklearn.preprocessing import scale
```

```

#Grid-search cross validation module is also used to do the grid
search cross validation if our program doesn't do it manually.
from sklearn.model_selection import GridSearchCV

#CSV is imported as we are also working with .CSV files and pandas
dataframes interchangeably.
import csv
#Datetime is used to handle dates. They are all in the form ISO8601
so can be easily accessed and understood.
from datetime import datetime, timedelta

#These are modules required by openMeteo as we are accessing an
external database through an API.
import openmeteo_requests
#These two modules are used to retry connecting to an API if the user
disconnects from the server.
import requests_cache
from retry_requests import retry

#Web scraping modules are included here beautifulsoup4 is used to
retrieve the HTML code and requests to access elements within a
website.
from bs4 import BeautifulSoup
import requests

#Once we have the link to an external dataset we will use os to
download it.
import os

#When we implemented a loading screen we used subprocess to run
multiple threads of code at once.
import subprocess

#Abstract syntax tree used for the loading screen.
import ast

```

Prototype Prediction:

All of these separate modules can all be combined to produce a single program that allows the user to do the following.

1. Input and retrieve the data.

2. Process the data to produce a single model.
3. Predict the future commodities price using the model and forecasted weather data.
4. Output the predictions to the users.

Combining all the modules into one involved using the following code to test the success of our entire program. This involves calling and combining all the components we built previously and making them run sequentially. This serves the frameworks of our final solution as we will need to take similar steps to produce a prediction while also implementing additional functionality such as a GUI running concurrently while data is processed in the background.

```
def predict(crop,model):#This is a simplified prediction layer.

    #We retrieve and standardise the forecasted weather
    futureWeather_dataFrame = retrieve_forecasted_climate(crop)
    futureWeather_dataFrame_scaled = scale(futureWeather_dataFrame)

    #We input the forecasted weather data into the model producing a
    #single prediction which we return.
    prediction = model.predict(futureWeather_dataFrame_scaled)

    return prediction


crop = input('CROP >>>')#Input the crop you want to analyse.

#We retrieve all the historical weather and price data.
historicalWeather = retrieve_historical_climate(crop)
labelledPrices = retrieve_commodity_prices()

#classify_commodityPrice_Dataframe(crop,retrieve_commodity_prices())

#We find the most optimal hyperparameters using a grid search cross
#validation algorithm.
hyperparameters = findOptimal_hyperparameters('cocoa')

#We input all these values, weather, price and optimal
#hyperparameters into the model for it to be trained.
trainedModel = trainingSupportVectorMachine(historicalWeather,
labelledPrices, 'no hyperparameters')
```

```
#We apply this trained model with forecasted weather to produce a prediction.
predict(crop, trainedModel)
```

However while testing our program, validation techniques caught an error before it was fed into training the model. This stopped the execution of our entire program and told us that there is missing data. This is because the weather and price data are different lengths.

```
error encountered
missing data. The length of historical weather and price data do not match.
hard reset - resetting entire program
```

Investigating closer we can see there is an inconsistent number of samples between both weather and price data and they can't be suitably formatted to train the support vector machine model. This also highlighted the need for internal error catching mechanisms which allowed for both easy debugging and maintainability of the program as it is able to automate resolving errors. On the contrary without this system any errors would cascade through the program and produce additional errors therefore consistent validation stages allow the program to be robust.

ValueError: Found input variables with inconsistent numbers of samples: [64, 260]

To fix this we need to look into the format of both the weather and the commodity price data that is fed into the model. The weather data was in the correct format, but the commodity data on the other hand was not. Looking at the commodity prices it was not labelled with the yearly change in price and did not contain the specific crop we are analysing. The image below shows the current dataset when we output it using `print(labelledPrices)`.

	World Bank Commodity Price Data (The Pink Sheet)	Unnamed: 11	Unnamed: 24
5		NaN	Cocoa
8		1960	91.833333
9		1961	109.333333
10		1962	100.54
11		1963	110.093333
12		1964	110.505833
13		1965	116.868333
14		1966	125.6625
15		1967	114.725

To fix this we need to both classify the commodity prices and pick the specific crop we are analysing. To do this we can replace the value from just retrieving all the commodity prices using `labelledPrices = retrieve_commodity_prices()`.

To retrieving actually labelled data and only for the specific crop using

```
labelledPrices =  
classify_commodityPrice_Dataframe(crop,retrieve_commodity_prices())
```

This produces the correct output data which we can see below. This is in the correct format to train the model with as it contains the date, price of the agricultural commodity and the respective percentage change.

	World Bank Commodity Price Data (The Pink Sheet)	Unnamed: 11	Change
0		1960	0.589017 Positive
1		1961	0.478558 Negative
2		1962	0.457967 Negative
3		1963	0.552342 Positive
4		1964	0.505742 Negative
5		1965	0.365308 Negative

Testing the weather data in this situation we can see below it is in the correct format as it retrieves the merged historical weather dataset which contains all four weather metrics. We output the historical weather data using `print(historicalWeather)`.

year	temperature_2m_mean	cloud_cover_mean	relative_humidity_2m_mean	precipitation_sum
1960	24.618158	71.760482	84.552578	4.004928
1961	24.445861	65.170621	83.770557	3.959085
1962	24.552534	68.034207	83.331269	3.565846
1963	24.719595	71.619087	84.811315	4.633583
1964	24.213333	65.722929	82.092061	3.192138
1965	24.414165	69.022008	84.259050	3.973229

Ultimately the final output of this program is whether the forecasted price of the agricultural commodity will either go up or down in the next 16 days. Stakeholder's tested this entire program by using a command line interface that lets you input the crop's name and the program processes everything in the background to finally produce an array which indicates the respective prediction of that agricultural commodity.

```
CROP >>> (Press 'Enter' to confirm or 'Escape' to cancel)
```

```
wheat
```

```
CROP >>> (Press 'Enter' to confirm or 'Escape' to cancel)
```

This is the produced final output of our initial prototype. Stakeholders documented interacting with our system as part of the initial prototype which is shown in the testing video. This involved testing the program with default hyperparameters and didn't involve optimising them.

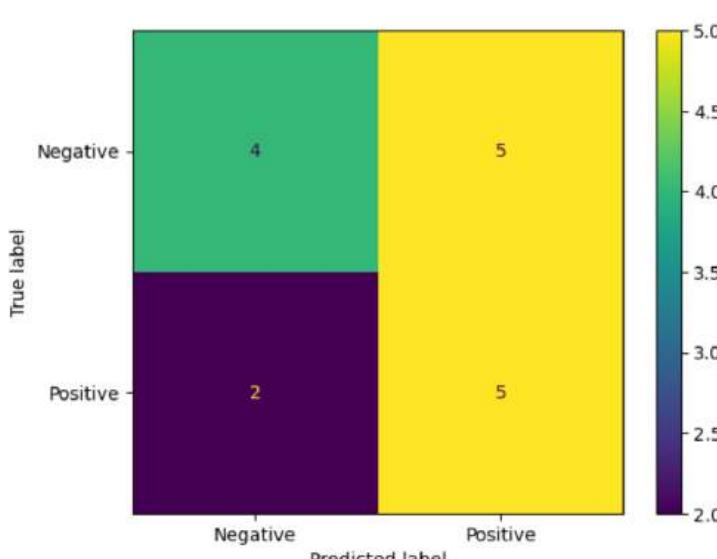
Producing a prediction as well with a model that selects the optimal hyperparameters does this same process, but takes longer at up to a single minute of processing time. These optimised hyperparameters will mean that the produced prediction has been produced by the most accurately fine-tuned model and not just with default hyperparameter values.

```
array(['Positive'], dtype=object)
```

Stakeholders gave feedback on this section. Talking with Sharath who understood the basic

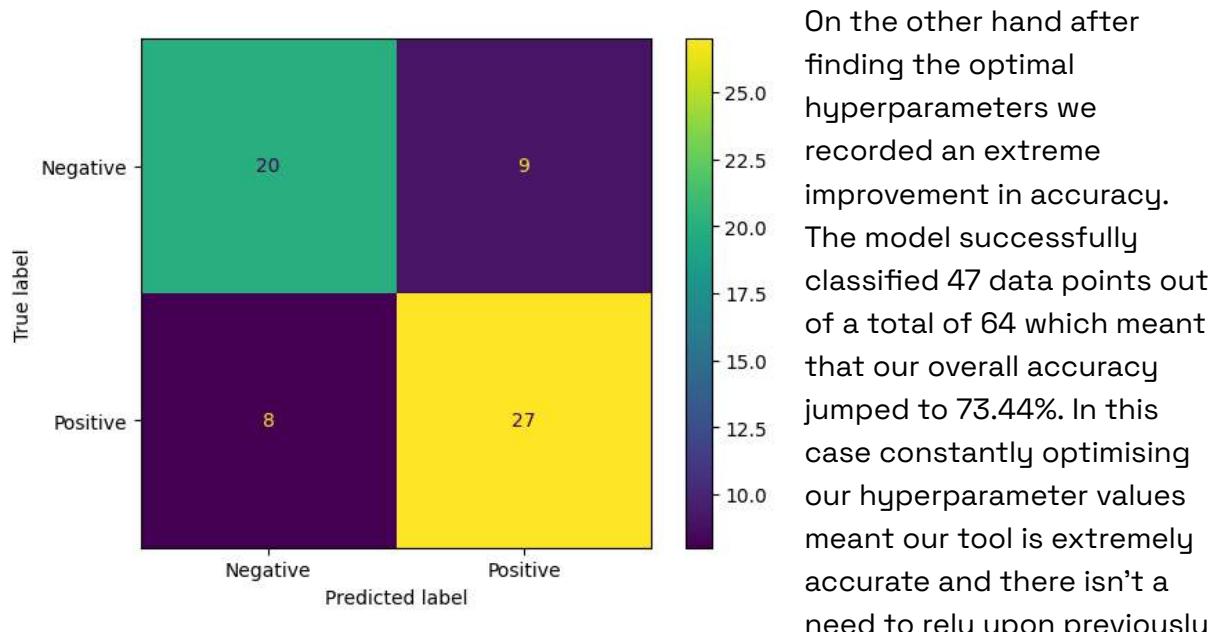
internal structure of our solution found the program worked extremely well to produce a solution quickly when using default hyperparameters, but optimising inevitably took more time. However specifically he found the produced prediction to be extremely vague as it did not specify the exact price, but instead said the price is going to either change positively or negatively in the next 16 days.

Suggested improvements from stakeholders were to provide a graphical user interface (GUI) and produce more accurate predictions that meet the requirements we stated above.



Adding to the testing stage of the model we decided to evaluate the performance of a model with random hyperparameters versus one with optimum hyperparameters. Looking at the confusion matrix below we can see the predicted labels the model assigned to data points compared to their true labels. The confusion matrix

allows us to look in depth at how our model assigns labels to data points and measure the overall accuracy of the support vector machine. In this case our model correctly classified 9 data points out of the total 16 data points giving us an overall accuracy of 56.25% which is extremely low. We used a test-train split within this data to separate data into either being used for testing or training so the model is trained with unseen data and can't memorise the training data.



Talking with Mrs Ganeshgudi our stakeholder who is a beginner trader and unlikely going to use advanced settings such as setting custom hyperparameters she wanted to always use the most optimised model. This was because the total time it took to optimise the hyperparameters was well worth the significant jump in accuracy. Sharath on the other hand still wanted to be able to customise the model so advanced user's are able to still manually input the hyperparameters while beginner users are reassured that the program itself will automatically optimise itself to the dataset.

start date
2023-10-08 00:00:00+00:00
startDate + 15 days
2023-10-23 00:00:00+00:00

To make predictions in more detail we will need to split the forecasted weather into more specific time segments such as splitting it into four separate 4-day segments.

Looking at the forecasted weather data we can see that if the forecast starts at midnight on the 2000-01-01 for instance it will end 16 days after this date which is

2000-01-16. This is 16 days as the current date is counted as well.

Looking at the .CSV forecasted weather file we can confirm this looking at the date values in an example forecast. The first step we will do is recreate the same algorithm we used to turn daily averages into yearly averages for historical weather data. Instead now we are turning daily values into 4 day time segments. In total the final output should be four data points which show the average weather metrics across that specific four day period.

We are working within the `def retrieve_forecasted_climate(crop)` function which currently only retrieves an array of the 4 average values of the metrics.

```
combined_dataframe['date'] =  
pd.to_datetime(combined_dataframe['date']) # Ensure 'date' is  
datetime type  
  
#Defines the split of the data into four time segments. Each of  
which has a length of 4 days.  
combined_dataframe['time_segment'] =  
pd.cut(combined_dataframe['date'], bins=4, labels=['Days 1-4', 'Days  
5-8', 'Days 9-12', 'Days 13-16'])  
  
# Calculate mean values for each grouped time segment. This means  
that we have four different data points instead of a single  
datapoint.  
combined_dataframe = combined_dataframe.groupby('time_segment',  
observed=False).mean()  
combined_dataframe.to_csv('combined_forecastedWeather.csv',  
sep=',', index=False, encoding='utf-8')#Save it as a csv so we can  
use this data frame for debugging and access it from the rest of the  
program.  
  
return combined_dataframe
```

Testing this we can see that instead of returning a single datapoint such as below.

temperature_2m	relative_humidity_2m	precipitation	cloud_cover
26.918595	76.698573	0.026222	85.353125

Each component represents the entire forecast's average value for each metric.

Stakeholder's wanted more detail so the accommodation of time segments allowed this as we can now make predictions in greater detail as we have a total of four separate data points across the forecast instead of a single one.

time_segment	temperature_2m	relative_humidity_2m	precipitation	cloud_cover
Days 1-4	26.918595	76.698573	0.026222	85.353125
Days 5-8	27.108367	77.134729	0.021212	63.959469
Days 9-12	26.674001	71.255531	0.000000	39.547000
Days 13-16	26.366191	73.142542	0.010128	54.538885

The above image shows the output of the program when we retrieve the forecast for environments where cocoa beans are grown across the following four day time segments. We called this function using `retrieve_forecasted_climate('cocoa')`.

Validating Data + Imputation :

Validation is a key part of the entire program that's why we need to build steps in place to detect anomalous and missing values within our dataset and use a function to impute them with the necessary values. Within the weather dataset instead of extracting only the data points with values within them and using that we can instead use a function to traverse the entire dataset and find and replace missing and anomalous values. Imputation is the process of replacing these erroneous elements within a dataset with substituted correct values.

We will build a function that both detects errors and replaces the value with a substituted value using imputation. Our algorithm will be the IQR algorithm which was described in the design section to detect anomalous results by calculating the IQR range of the data and creating valid bounds where if the data is too high or low it will be flagged and replaced with a suitable substitution. Missing values will also be flagged and replaced with a suitable substitution.

Imputation is the process of creating the substituted values. We agreed with stakeholders that our imputation algorithm will involve getting the average of nearby data points and substituting it with that.

```
def impute_anomalies(dataframe):  
  
    #This is the k value and defines how large the bounds will be.  
    #A larger k value will mean the bound will cover less of the  
    #entire dataset.  
    #This means that the function will be less sensitive to  
    #anomalies.  
    k = 2
```

```

for column in dataframe.columns:

    #This checks if the column only contains numerical numbers as
we can't impute string values.
    if pd.api.types.is_numeric_dtype(dataframe[column]):

        #Find the upper and the lower quartiles.
        Q1 = dataframe[column].quantile(0.25)
        Q3 = dataframe[column].quantile(0.75)

        IQR = Q3 - Q1#Find the interquartile range

        #Define the upper and lower bounds which data have to be
within.
        lower_bound = Q1 - k * IQR
        upper_bound = Q3 + k * IQR

        #.loc is used to access and modify specific elements
within the specific column.

        #This means there is no need to create an inefficient for
loop to loop through all the values and columns.

        #The condition selects only elements that fall outside of
the bounds we defined.

        #These specific elements are then replaced with the mean
of the column.

        dataframe.loc[(dataframe[column] < lower_bound) |
(dataframe[column] > upper_bound), column] = dataframe[column].mean()

    return dataframe

```

The above function was tested by making manual anomalies within the data and then passing that entire dataframe through our function that detects and imputes

anomalies to see what happens.

temperature_2m_m...	cloud_cover_m...	relative_humidity_2m_m...	precipitation_s...
25.883562	35.62847	81.69292	0.7705929
26.586628	34345	7	0.0
26.117956	6	866	0.6265436
26.049284	0	634	0.5540783
25.630611	74.33678	85.47261	1.5161823

The model correctly classified all the anomalies and imputed them with the respective correct normal values within the dataset.

temperature_2m_m...	cloud_cover_m...	relative_humidity_2m_m...	precipitation_s...
25.883562	35.62847	81.69292	0.7705929
26.586628	34.550148	77.67668	0.0
26.117956	68.14569	81.60866	0.6265436
26.049284	100.0	82.540634	0.5540783
25.630611	74.33678	85.47261	1.5161823

There is also a need to build a function that detects missing values and imputes a suitable substituted value. We can use the same structure as before. Finding missing values involves looping through only columns that contain numerical values and filling in a specific element with the mean of the entire column and returning the updated dataframe. To do this I used a specific function within pandas that fills missing values such as empty elements with the specific columns mean. Empty elements in this case are elements that hold no data and not a value representing 0 units of the climate variable.

```
def impute_missing(dataframe):  
  
    for column in dataframe.columns:  
        #This checks if the column only contains numerical numbers  
        #as we can't impute string values.  
        if pd.api.types.is_numeric_dtype(dataframe[column]):  
  
            #.loc is used to access and modify specific elements  
            #within the specific column.
```

```

    #This means there is no need to create an inefficient for
loop to loop through all the values and columns.

    #These specific elements are then replaced with the mean
of the column.

    dataframe[column] =
dataframe[column].fillna(dataframe[column].mean())

return dataframe

```

We tested this function that imputes suitable substituted values into a missing element by passing a dataframe with a completely empty datapoint. This means that the element stored contains no values and is empty.

temperature_...	relative_humidity_...	precipitati...	cloud_co...
25.0845	92.0	0.0	72.0
24.9345	92.0	0.0	86.0
temperature_...	relative_humidity_...	precipitati...	cloud_co...
25.234499	91.0	0.0	44.0
25.0845	92.0	0.0	72.0
24.9345	92.0	0.0	86.0

After passing the entire dataset through the dataframe we can see that the produced result is a dataframe that is complete. A complete data frame means that there are no missing elements as we can see in the same dataset, but after processing where suitable values have been imputed or replaced.

Prediction Layer:

We need a function that passes these four data points into a prediction function that takes in the forecasted weather as well as a trained support vector machine model and produces an appropriate prediction over each time segment. Instead of our initial prototype which produced a single prediction, the addition of distinct time segments over a set time period allows us to model the change in the agricultural commodities price. This is as an array and can be directly passed onto the next component which can visualise the predictions on an interactive graph. Separate independent components are used together to slowly build up the skeletons of the solution and allow the development to be streamlined.

```

#This is the prediction layer. We take the entire model and the crop
and produce a prediction based on forecasted weather data.

def predict(crop,model):

    #We retrieve and standardise the forecasted weather
    futureWeather_dataFrame = retrieve_forecasted_climate(crop)

    #Delete the first date column as it is unnecessary and not a
    numerical value.
    futureWeather_dataFrame =
futureWeather_dataFrame.drop(futureWeather_dataFrame.columns[0],
axis=1)

    #Standardise and scale the data.
    futureWeather_dataFrame_scaled = scale(futureWeather_dataFrame)

    #We input the forecasted weather data into the model producing
    the predictions which we return as an array.
    prediction = model.predict(futureWeather_dataFrame_scaled)

    return prediction

```

To test this we will create a model and pass this through the prediction component to produce four predictions across the entire array. This means that we can validate if data can successfully pass through multiple components at a time to finally build into a range of predictions as outputs of the entire solution. Testing the end-to-end functionality of our program as data seamlessly flows through multiple components to finally at the end produce a solution. This end-to-end functionality only works if all components are correctly integrated together which testing allows us to validate.

```

crop = 'cocoa'

#We retrieve all the historical weather and price data.
historicalWeather = retrieve_historical_climate(crop)
labelledPrices =
classify_commodityPrice_Dataframe(crop,retrieve_commodity_prices())

#We find the most optimal hyperparameters using a grid search cross
validation algorithm.

#We input all these values, weather, price and optimal

```

```

hyperparameters into the model for it to be trained.

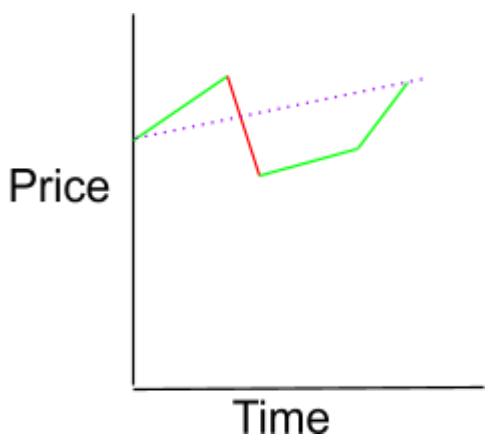
trainedModel = trainingSupportVectorMachine(historicalWeather,
labelledPrices, 'no hyperparameters')

#We apply this trained model with forecasted weather to produce a
prediction.

predict(crop, trainedModel)

```

The final output is a more detailed prediction than what stakeholder's wanted as instead of being a single value it is four separate predictions across the time segment. Looking at the image using this prediction function we are able to make



more detailed predictions as we can accommodate the price's movement across each four-day period. In comparison our old method relied on having a single prediction along the entire graph which is vague as you can't make our short term fluctuations and movements.

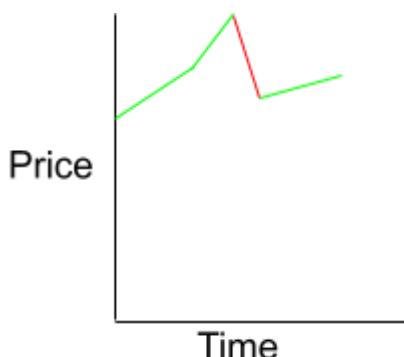
Interviewing Mr Manu, one of our stakeholders who particularly wanted the tool to have a high accuracy after testing it, he was impressed

with the amount of information it can project as the system can look through 16 days of data and then make decisions as to what will happen in each four day period across the entire forecast accurately. However he said that the detail each prediction was given was low and it was difficult to see the variation.

```
array(['Positive', 'Positive', 'Negative', 'Negative'], dtype=object)
```

The above forecast is what Mr Manu produced for cocoa commodity prices in the next four day period. He said that just having two simple possible variables that are either positive or negative gives little detail as to the actual change in price. To do this he wanted a system that produced accurate percentage values which he can estimate exactly what the future price will be.

To do this we will need to iterate our commodity price classifier to label the precise percentage of the change in price instead of just a simple positive or negative label.



This image shows our clear goal with classification as we want the change in price to be clearly quantifiable and not as a vague tool that is unable to

show the actual change in price of the agricultural commodity.

Updating Commodity Prices Classifier to categorise data points in more detail:

We update the commodity prices classifier to show the change in price of the commodity in percentage points. Updating this component enhances the accuracy of our model as we are able to make predictions with finer accuracy instead of broader positive and negative labels. This means that insights are more detailed and more numerical as you can effectively track the change in the raw price of the agricultural commodity.

The percentage change is given as a floating point number which is extremely accurate compared to giving it as a binary label that specifies if the price changed positively or negatively.

```
def
classify_commodityPrice_Dataframe(crop,commodityPricesDataframe):#Labels to the selected crop's change in price as either positive or negative.

    #This for loop matches the crop with its location within the cropIndex array.

    #The index within the crop in the cropIndex array is the same as the index within the commodity prices dataframe.

    for i in range(len(cropIndex)):
        if cropIndex[i] == crop:
            cropRow_DataframeReference = i
            break

    commodityPricesDataframe = commodityPricesDataframe.iloc[:, [0,i+1]]#Extract only the crop column you need and the dates.

    pd.set_option('display.max_rows', None)#Display the maximum amount of rows and columns so all available data is able to be accessed,
    pd.set_option('display.max_columns', None)

    commodityPricesDataframe =
commodityPricesDataframe.iloc[1:].reset_index(drop=True)
```

```

commodityPricesDataframe.iloc[:, 1] =
pd.to_numeric(commodityPricesDataframe.iloc[:, 1])

# Calculate percentage change
commodityPricesDataframe['Percentage Change'] =
commodityPricesDataframe.iloc[:, 1].pct_change() * 100
commodityPricesDataframe.fillna(0, inplace=True) # Fill NaN with
0 for the first row

# Round percentage change to an integer
commodityPricesDataframe['Percentage Change'] =
commodityPricesDataframe['Percentage Change'].round(0)

return commodityPricesDataframe#Returns the processed dataframe.

```

While developing this component I encountered an error while using a specific function within the Panda's module. The pandas module allows for easy data manipulation for structured data, In this case the weather databases stored as Pandas dataframes.

FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated and will change in a future version. Call result.infer_objects(copy=False) instead. To opt-in to the future behavior, set

```

`pd.set_option('future.no_silent_downcasting', True)`
commodityPricesDataframe['Percentage Change'] =
commodityPricesDataframe.iloc[:, 1].pct_change() * 100

```

The error indicates that in future versions of Panda's the module is unable to automatically convert the entire data type of the column when performing the fillna operation. Fillna captures missing data elements within the column and fills them with an element. In this case as we are initialising a new column it fills it with 0s.

As the entire column is all the same datatype containing a list of 0 integer elements. There is no need to amend the program in this case as the data type is constant. However to handle the error we have used the warnings module to filter and catch the error before it is directly shown to the user. This allows the program to catch the unnecessary error before it is shown to the user.

```
import warnings
```

```
warnings.filterwarnings('ignore', category=FutureWarning,
message="Downcasting object dtype arrays on .fillna, .ffill,
.bfill is deprecated")
```

This means that only errors that will impact the output of the program are caught. Additionally the warnings module are able to capture and handle warnings automatically so additional automated debugging tools can be developed adding to the maintainability of the solution.

I tested the classification of historical cocoa commodity prices by calling the following function.

```
classify_commodityPrice_Dataframe('cocoa', retrieve_commodity_prices())
```

The resulting data frame that we produce has a range of percentage changes. However an error of this process is that we do not have precise measurements as all the percentages are spread so far away from each other. The extreme variation as we can see below means that the program is unable to specifically categorise a data set precisely. This problem is further emphasised by the fact that we have only 64 commodity price values within our dataset.

	World Bank Commodity Price Data (The Pink Sheet)	Unnamed: 11	Percentage Change
0		1960	0.589017
1		1961	0.478558
2		1962	-4.0
3		1963	0.552342
4		1964	-8.0
5		1965	-28.0
6		1966	42.0
7		1967	16.0
8		1968	21.0

To tackle this problem we have decided to go with another approach by selectively sorting each percentage change into a finite number of bins. This means that instead of a percentage taking a range of integers between 0 and 100 we can instead be in multiples of 10 for instance. This reduces the number of categories that are possible so that we can more efficiently categorise the data.

We did this by adding this line to the program before the dataframe is returned.

```
commodityPricesDataframe['Percentage Change'] =
(commodityPricesDataframe['Percentage Change'] / 10).round(0) * 10
# Round percentage change to the nearest 10
```

	World Bank Commodity Price Data (The Pink Sheet)	Unnamed: 11	Percentage Change
0		1960	0.589017
1		1961	0.478558
2		1962	0.457967
3		1963	0.552342
4		1964	0.505742
5		1965	0.365308
6		1966	0.517633

The issue was this method was that major fluctuations within the commodities price was uncommon so by rounding all the data we significantly decreased the ability for the model to spot smaller variation and made the model only focus on large variations.

To retain the model's accuracy we kept the original format, this meant that the percentage changes are only rounded to be integer values. Integer value percentages were displayed as a positive / negative number between 1 - 100 however we are unable to use these values when multiplying data. This is because the change in percentage is defined as multiplying a number by the approximate change in volume within it. E.g a 1% increase would correctly be 1.01 as the number has gained a 1/100 component. In the same way that a 5% decrease wouldn't be -5, but correctly notated as 0.95.

If the percentage changes are correctly categorised with the correct notation we will be able to quickly extract the value and multiply the value without needing further processing to turn -5 into 0.95.

```
# Convert percentage change to correct multiplier format
commodityPricesDataframe['Percentage Change'] = 1 +
(commodityPricesDataframe['Percentage Change'] / 100)
```

Testing this we can see the produced dataset is in the correct format.

	World Bank Commodity Price Data (The Pink Sheet)	Unnamed: 11	Percentage Change
0		1960	1.0
1		1961	0.8
2		1962	1.0
3		1963	1.2
4		1964	0.9
5		1965	0.7
6		1966	1.4
7		1967	1.2

We encountered this error when we tried to train the classified commodity data with historical weather data within a support vector machine.

ValueError: Unknown label type: continuous. Maybe you are trying to fit a classifier , which expects discrete classes on a regression target with continuous values.

This meant that the model was receiving continuous data instead of discrete values which confused it. Continuous values can take any value within a given range such as height and decimal numbers. Discrete data on the other hand can only take specific, distinct values such as integers. There is a finite number of discrete data compared to an infinite amount of continuous data.

To fix this we need to make the support vector machine understand the percentage change values as discrete data and not numerically continuous data. Changing the datatype from a float to a string can achieve this as then the model will understand each datapoint's percentage change as a specific category instead of a numerical number.

Testing this change on predicting the price of soybean we retrieved the following predictions for the change in price of the soybean commodity over the next 16 days with each segment representing a four day period.

```
array(['1.4', '0.9', '1.1', '1.1'], dtype=object)
```

Stakeholders commented on the extremely large values of the predictions as a 40% increase within the next four-days is too large and by simple statistical analysis basically impossible. To get in line with normal predictions we realised the problem was due to the training data being the change in price year over year instead of our smaller time segments which are 4 day periods. To fix this we will need to divide each prediction by 365 which is the number of days within a year and multiply by 4. This number will give us the average change in price of the agricultural commodity over four days.

```
return prediction * 4/365
```

Running the same test again on soybean commodities with the returned prediction multiplied by 4/365 we encounter an error.

```
---> 16 return prediction * 4/365
```

TypeError: unsupported operand type(s) for /: 'str' and 'int'

This error was due to us previously turning all the array values into strings instead of

integers as we wanted them to represent categories. Therefore we need to turn the array's contents into suitable floating point numbers instead of strings and do the following calculations to find out how the commodities prices change over each time segment.

```
# Convert to array of floating-point numbers
prediction = prediction.astype(np.float64)

yearlyChange = np.array(prediction)
# The daily change is multiplied by four as our time segment is of a
four day length.
fourDayChange = 1 + (((yearly_changes - 1) / 365) * 4)

prediction = fourDayChange

return prediction
```

Therefore the final array that we produce when we make a prediction for soybean is below. This correctly indicates the change in price of the soybean commodity over the next 16 days in four separate time segments each with a length of four days each.

```
array([1.00438356, 0.99890411, 1.00109589, 1.00109589])
```

Additionally as part of our stakeholder requirements we tested our program on multiple pieces of hardware this included different operating systems and different hardware. As we are planning to support both laptop and mobile devices there is a need to cater to different types of hardware. Specifically RISC processors which the majority of smartphones and tablets run on. To do so I tested my solution on the MacOS operating system on an RISC processor. The original program was developed on a Windows device running python on a x86 based CISC processor architecture.

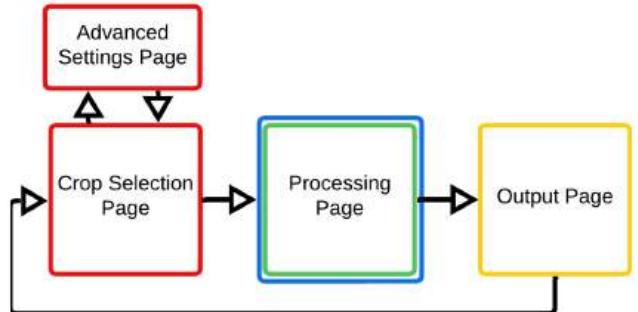
Initially I got an error flagged by the Input Method Kit (IMK) within the MacOS operating system. Upon further debugging it is a framework designed to handle text input and was automatically initialised when the solution was run. This was because the graphical user interface written with the Tkinter framework allows inputting text and therefore required macOS's native text input system.

```
Python[2776:159101] +[IMKClient subclass]: chose IMKClient_Modern
Python[2776:159101] +[IMKInputSession subclass]: chose IMKInputSession_Modern
```

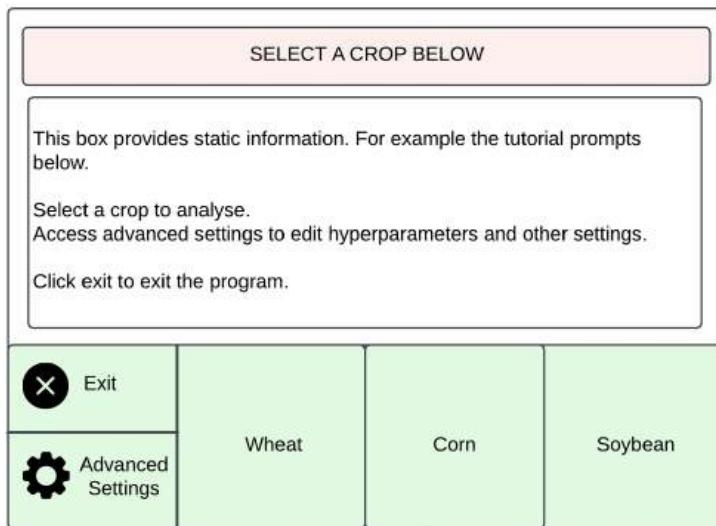
Solving this meant using a logging module to catch the error and suppress it as it allows for the system to run effectively without excessive unnecessary errors being produced. This also validates the support for multiple devices and CPU architectures as our prototype system is able to execute commands

```
logging.getLogger('IMK').setLevel(logging.WARNING)
```

Iterative Design alongside the User Interface



Stakeholders want an easy to use piece of software with a clear graphical user interface that meets the proposed requirements and the proposed screen designs.



Initially looking at the start page we can see exactly how the user should traverse through the entire program. Therefore the starting crop selection page should be built first.

We first used tkinter and tried to recreate this page as closely as possible. This meant keeping the active and static design and keeping all the buttons at the bottom. This was the first iteration of the start page below.

The page was fully functional and was made using object oriented programming techniques as we separated everything into independent functions. These independent functions allowed us to approach the interface sequentially while building each page and how to traverse through the entire solution.

The first prototype for the start-up page was extremely simple and minimalist. It applied the static and dynamic interface framework to clearly indicate how to use the program.

```

import tkinter as tk
from tkinter import ttk

# Creates the main window.
root = tk.Tk()
root.attributes("-fullscreen", True) # Make the window full screen
root.attributes("-alpha", 0.9) # Set transparency (0.0 = fully transparent, 1.0 = fully opaque)

# Make the application responsive to changes in dimensions
def make_responsive(widget):
    for i in range(5): # Allow rows and columns to expand
        widget.grid_rowconfigure(i, weight=1)
        widget.grid_columnconfigure(i, weight=1)

# Instruction Box at the top
instruction_box = tk.Label(root, text="Select A Crop Below",
                           bg="#FFCCCB", fg="black",
                           font=("Arial", 24),
                           height=3)
instruction_box.grid(row=0, column=0, columnspan=5,
                     sticky="nsew", padx=10, pady=10)

# Clear Information Page (Label for now)
info_page = tk.Label(root, text="Information will be displayed here.",
                      bg="white", fg="black", font=("Arial", 18),
                      height=10)
info_page.grid(row=1, column=0, columnspan=5, sticky="nsew",
               padx=10, pady=10)

# Advanced Settings Button with Cog Icon
advanced_button = ttk.Button(root, text="⚙️ Advanced Settings",
                             style="TButton")
advanced_button.grid(row=2, column=0, columnspan=5,
                     sticky="nsew", padx=10, pady=10)

# Buttons for Cocoa, Wheat, Corn, Soybean, and Sugar
button_labels = ("Cocoa", "Wheat", "Corn", "Soybean", "Sugar")

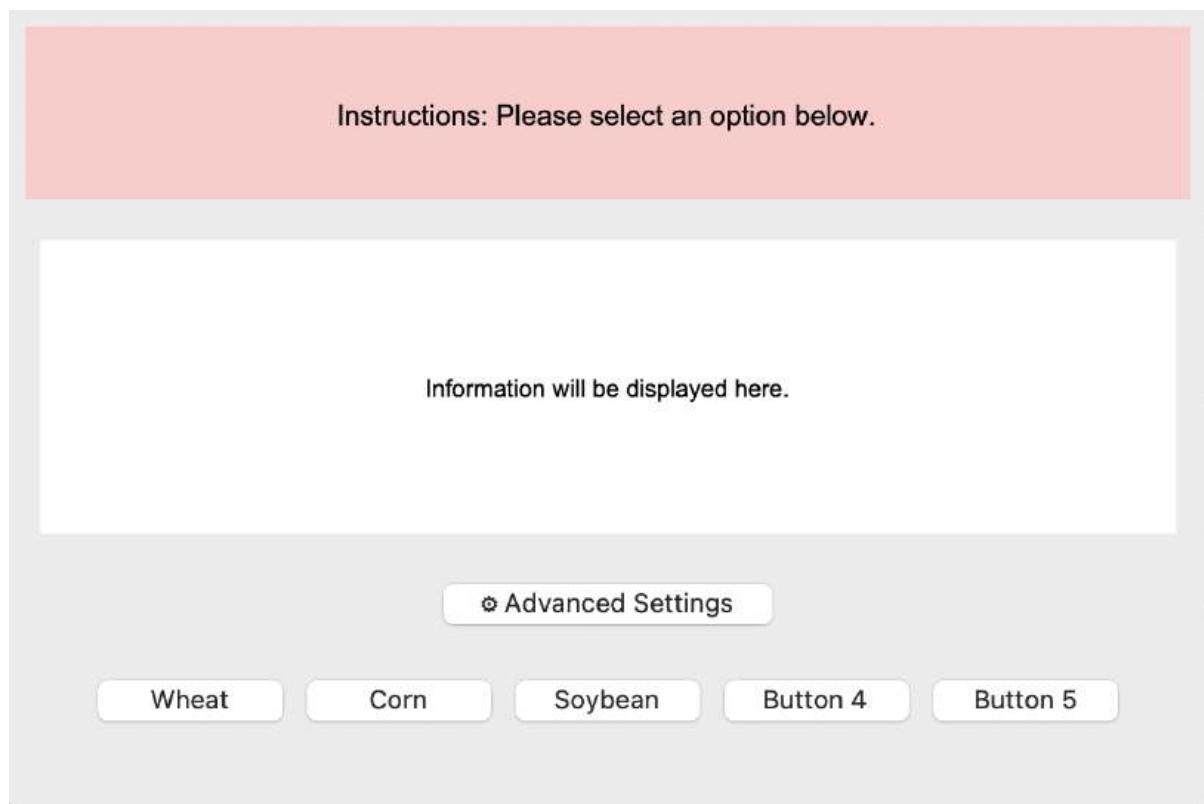
# Create and place buttons
for i, label in enumerate(button_labels):
    button = ttk.Button(root, text=label, style="Large.TButton")
    button.grid(row=3, column=i, sticky="nsew", padx=5, pady=5)

# Configure grid to make the application responsive
make_responsive(root)

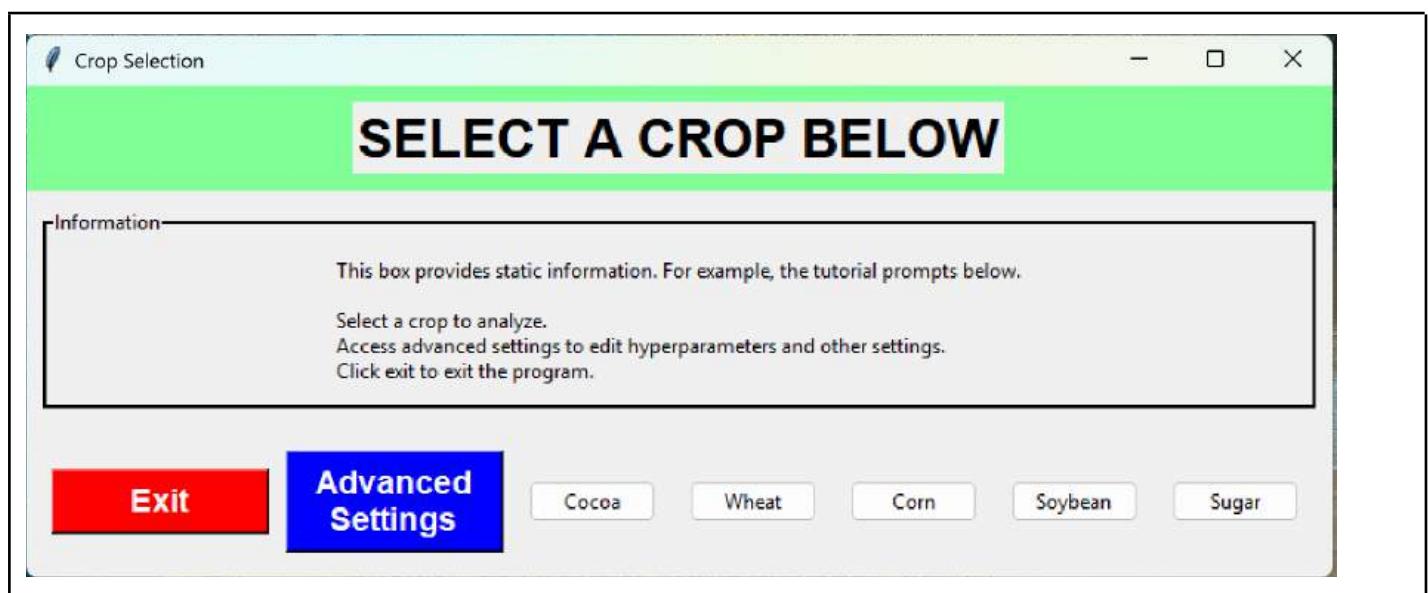
# Custom style for larger buttons
style = ttk.Style()
style.configure("Large.TButton", font=("Arial", 24), padding=20)

```

```
# Run the entire application  
root.mainloop()
```



Stakeholders found the placement of the buttons representative of the minimalist requirements. After interviewing multiple stakeholders, main points to improve included adding coloured elements to clearly differentiate between different elements. We implemented this in the second iteration of the start-up page.



```

import tkinter as tk
from tkinter import ttk

def clear_page():
    #Clears the entire page of widgets. So the entire page is empty.
    for widget in root.winfo_children():
        widget.destroy()

def crop_selection_page():
    clear_page()

    # Header
    header_frame = tk.Frame(root, bg="#80ff97")
    header_frame.pack(fill="x")
    tk.Label(header_frame, text="SELECT A CROP BELOW", font=("Arial",
24, "bold")).pack(pady=10)

    # Information Box
    info_frame = tk.LabelFrame(root, text="Information", bd=2,
relief="solid")
    info_frame.pack(fill="x", padx=10, pady=10)
    info_text = "This box provides static information. For example,
the tutorial prompts below.\n\n"
    info_text += "Select a crop to analyze.\n"
    info_text += "Access advanced settings to edit hyperparameters
and other settings.\n"
    info_text += "Click exit to exit the program."
    tk.Label(info_frame, text=info_text, justify="left",
wraplength=500).pack(padx=10, pady=10)

    # Buttons Frame
    button_frame = tk.Frame(root)
    button_frame.pack(fill="x", padx=10, pady=10)

    # Exit Button
    exit_button = tk.Button(button_frame, text="Exit",
command=root.destroy, fg="white", bg="red",
font=("Arial", 14, "bold"), width=10)
    exit_button.pack(side="left", padx=5, pady=5)

    # Advanced Settings Button

```

```

    advanced_button = tk.Button(button_frame,
text="Advanced\nSettings", command=lambda :advanced_settings(),
fg="white", bg="blue",
font=("Arial", 14, "bold"), width=10)
    advanced_button.pack(side="left", padx=5, pady=5)

# Crop Buttons
for crop in ("Cocoa", "Wheat", "Corn", "Soybean", "Sugar"):
    ttk.Button(button_frame, text=crop, command=lambda c=crop:
processing_page(c)).pack(side="left", padx=10, pady=5)

def processing_page(crop):
    clear_page()
    tk.Label(root, text=crop).pack(pady=20)

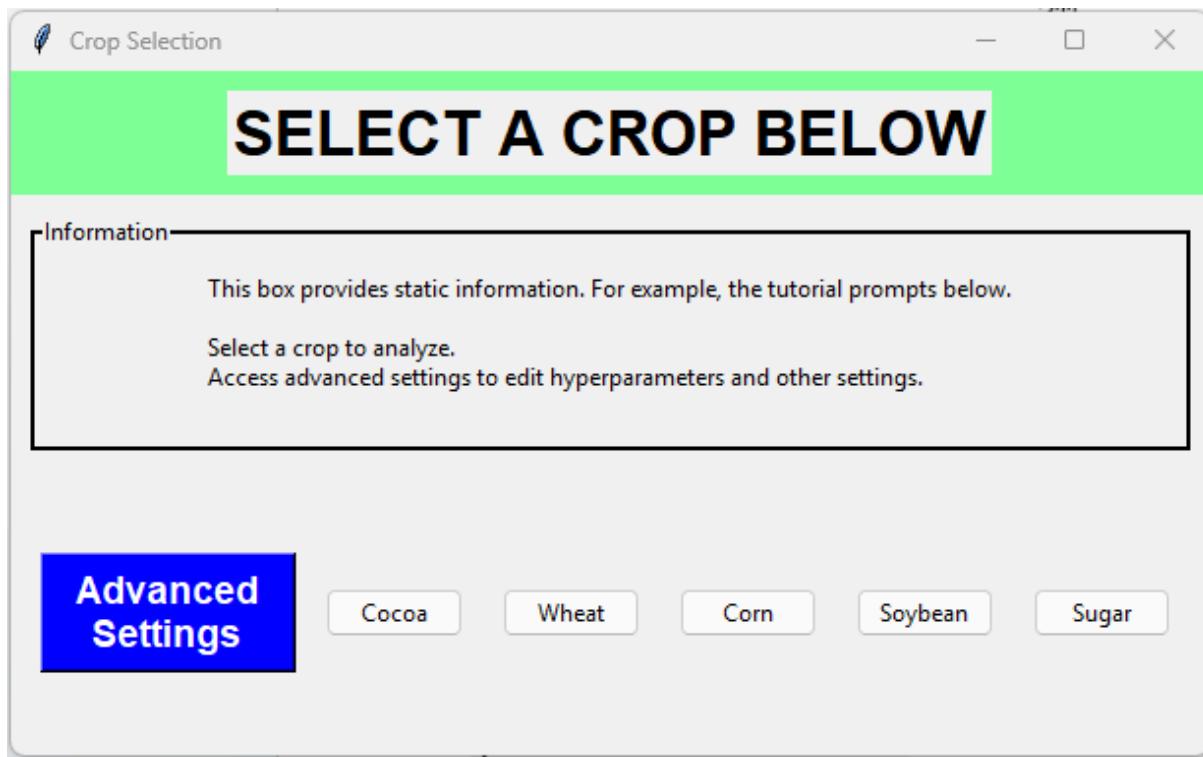
def advanced_settings():
    #This will be the advanced settings page where users are able
    to customise the hyperparameters.
    clear_page()
    tk.Label(root, text='Advanced Settings Page').pack(pady=20)

if __name__ == "__main__":
    root = tk.Tk()
    root.title("Crop Selection")
    crop_selection_page()
    root.mainloop()

```

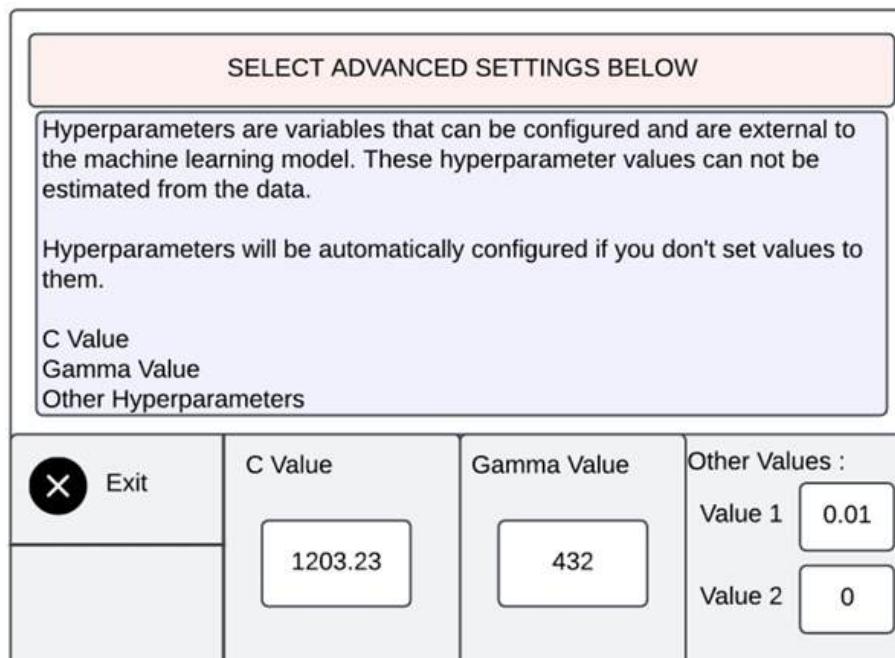
Stakeholder's liked this page saying it was clear and simple. This is the first page that the user interacts with so a clear and consistent interface is crucial. Interviewing my stakeholder shreeharsh he said he found the exit button unnecessary as when opened in a minimised window there isn't a need to have this button as the user can easily access the top of the page and close the program.

Implementing this within my program I removed the exit buttons as the operating system provided one directly.



Stakeholders said that this page was easy to navigate around and it was clear how to use the entire page.

Advanced Settings Page.

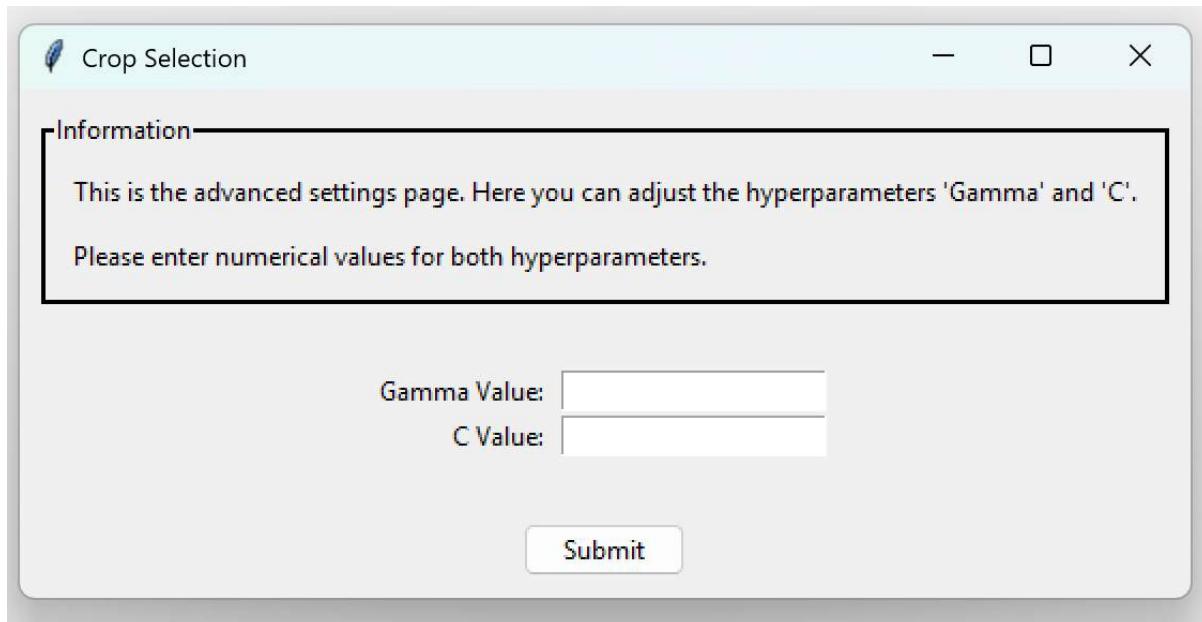


Reflecting our original design we aimed to have clearly on the screen the editable C and Gamma values while also keeping the interface static. Stakeholders suggested adding a submit button so we can verify that the values have been sent to the

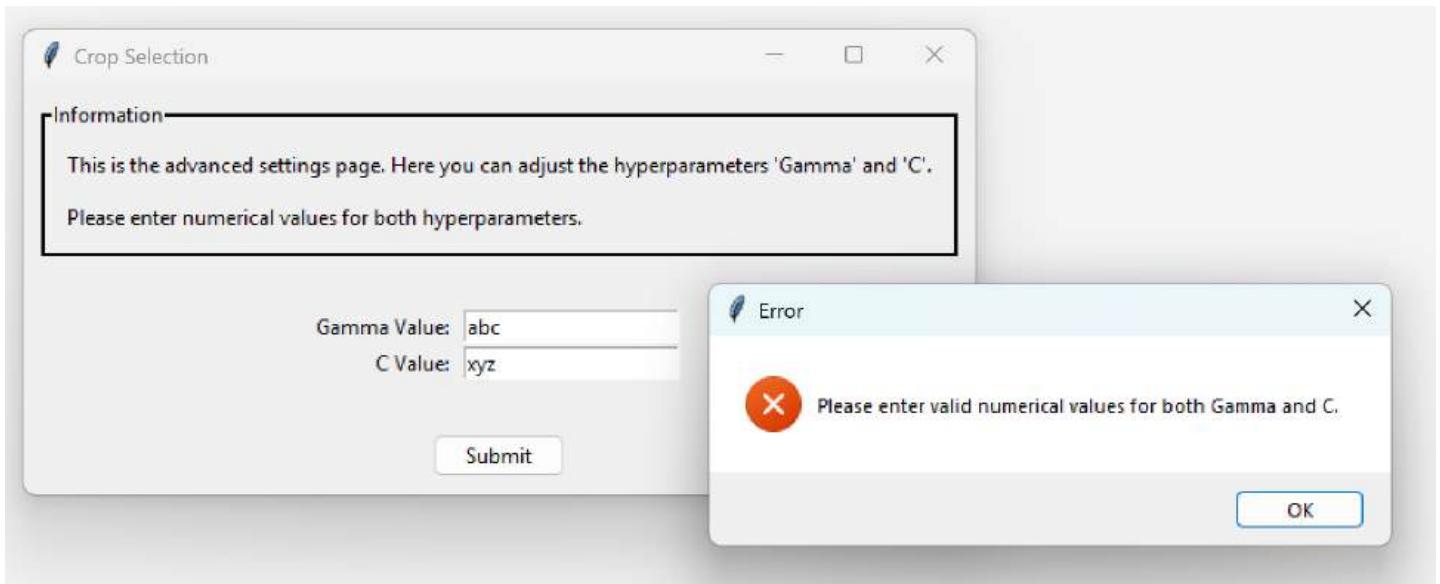
backend system further enhancing the overall user experience. To add this functionality we need to be working within the `advanced_settings()` Page function.

We built the user interface trying to model the overall static and active design that we discussed earlier segmenting and dividing the page clearly.

This was the page we decided on as it was fundamentally clear how to both interact with it and input the necessary values.



Validation was an important part of this solution so we wanted to guide even advanced users to directly interact with our product. To do so we built an error message box to display if the user were to input values that are not numerical such as a string or if they were to not enter a gamma value for the program.



Validation was implemented using a try: and except: sequence as we specified the gamma and c values should be of a float datatype the program would try to receive those values as floating point integers. However if the user instead were to enter a string instead then an error would occur within the try: statement and then the except clause would run where the user is prompted to with a message box to enter the correct gamma and c values. Below shows how we have used validation to make sure the user enters valid numerical values in both the gamma and c boxes.

```
def submit_values():
    try:
        gamma = float(gamma_entry.get())
        c = float(c_entry.get())

        customHyperparameters(c, gamma)

    except ValueError:#Here we use validation to make sure the
user inputs numerical values for both gamma and c.
        messagebox.showerror("Error", "Please enter valid
numerical values for both Gamma and C.")
```

The code below shows how we have implemented everything within the entire advanced settings page. The main function of this page is to allow advanced users to input custom hyperparameters and save it so it can be retrieved and used when the model is being trained.

```
def advanced_settings():
    clear_page()
```

```

# Information Box
info_frame = tk.LabelFrame(root, text="Information", bd=2,
relief="solid")
info_frame.pack(fill="x", padx=10, pady=10)
info_text = "This is the advanced settings page. Here you can
adjust the hyperparameters 'Gamma' and 'C'.\n\n"
info_text += "Please enter numerical values for both
hyperparameters."
tk.Label(info_frame, text=info_text, justify="left",
wraplength=500).pack(padx=10, pady=10)

# Input Frame
input_frame = tk.Frame(root)
input_frame.pack(pady=20)

# Gamma Value
tk.Label(input_frame, text="Gamma Value:").grid(row=0, column=0,
sticky="e")
gamma_entry = tk.Entry(input_frame)
gamma_entry.grid(row=0, column=1, padx=5)

# C Value
tk.Label(input_frame, text="C Value:").grid(row=1, column=0,
sticky="e")
c_entry = tk.Entry(input_frame)
c_entry.grid(row=1, column=1, padx=5)

def submit_values():
    try:
        gamma = float(gamma_entry.get())
        c = float(c_entry.get())

        customHyperparameters(c, gamma)

    except ValueError:#Here we use validation to make sure the
user inputs integer values for both gamma and c.
        messagebox.showerror("Error", "Please enter valid
numerical values for both Gamma and C.")

# Submit Button
submit_button = ttk.Button(root, text="Submit",

```

```
command=submit_values)
submit_button.pack(pady=10)
```

As we are within a function we need a way to broadcast the custom c and gamma values to the entire program. To do this we have also implemented this function which will be called when the user enters custom gamma and c values.

```
custom_cValue = None
custom_gammaValue = None

def customHyperparameters(c, gamma):
    global custom_cValue
    global custom_gammaValue
    custom_cValue = c
    custom_gammaValue = gamma
```

An error that occurred was the ability to call the customHyperparameters and set the hyperparameters however if we tried to access the custom c and gamma value variables outside of the advanced settings function we would get a `None` result even though previously we had set the value. To solve this we implemented a global scope for all the variables as it meant that it could be accessed and changed from anywhere in the program and if that value was called in another segment of the program its value would still be retained. In our previous case we came across an error because we were calling and setting the function within a local scope and as soon as we left the function it was out of scope so the value that was stored within it was lost and it was equal to `None`.

We also need to acknowledge that user's don't want to retain the custom c and gamma values after a prediction has been complete and it redirects them back to the starting page. Talking with Sharath an advanced user he said that initialising the parameters would mean that the user is able to fluidly analyse multiple crops at a time as there is no need to change the hyperparameters back to default values instead it is automatically done. This also means that users can be fully reliant on the program that automatically optimises hyperparameters as that is the default setting that caters to all users and produces the highest accuracy.

```
def initialiseParameters():
    custom_cValue = None
    custom_gammaValue = None
```

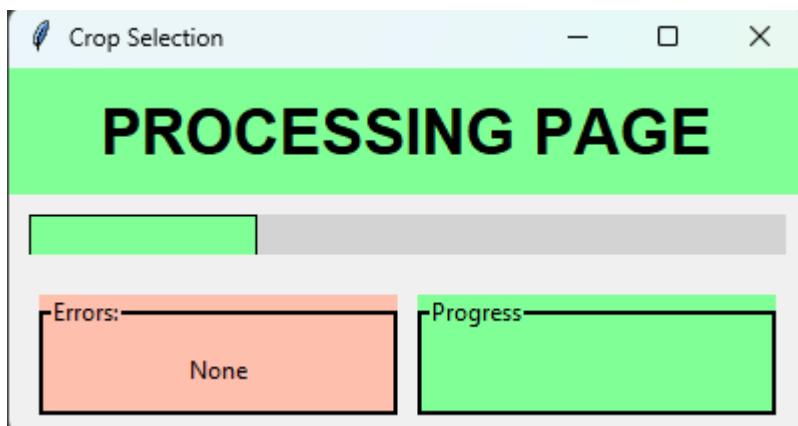


Processing Page:

This is the main page where everything happens in the backend and only progress indicators are shown to the user. These progress indicators include a progress bar at the top of the program and an error dialog box to the side and a progress dialog box to the other side.

Looking at the proposed design we can plan how we are going to implement this directly into our program.

This page will contain both the processing and the predicting layers as they both require the program in the background to process all of the data and then output a single solution.



This is the processing page we have created. To do this we have used frames and separate objects to represent the active components that change as the program is run. To do this we have created three separate functions that are able to interact with the processing page. Two separate functions to update the text boxes within both the errors and the progress blocks. What this means is that we can update the text within these boxes to inform the user what is currently happening and if there are any alerts that need to be delivered to the user. The progress bar is made using a

rectangle canvas which we resize accordingly when the progress bar has to change. This means that using all three methods of interaction with the user we can always keep them informed.

```
def processing_page(crop):
    # This is the processing page where all the data is being
    processed in the background.

    # The user will get updates as to the progress and if any errors
    occur during the entire process.

    clear_page()

    # Header of the processing page...
    header_frame = tk.Frame(root, bg="#80ff97")
    header_frame.pack(fill="x")
    tk.Label(header_frame, text="PROCESSING PAGE", font=("Arial", 24,
    "bold"), bg="#80ff97").pack(pady=10)

    # Progress Bar that shows the progress of the entire solution.
    progress_frame = tk.Frame(root)
    progress_frame.pack(fill="x", padx=10, pady=10)
    progress_canvas = tk.Canvas(progress_frame, height=20,
    bg="lightgray", highlightthickness=0)
    progress_canvas.pack(fill="x")
    progress_bar = progress_canvas.create_rectangle(0, 0, 0, 20,
    fill="#80ff97")#The rectangle represents the progress bar. We can
    change the rectangle's width to represent the change in progress over
    time.

    # makes the labelled error and progress frames that are side by
    side to each other.
    error_progress_frame = tk.Frame(root)
    error_progress_frame.pack(fill="x", padx=10, pady=10)
    error_frame = tk.LabelFrame(error_progress_frame, text="Errors:",
    bd=2, relief="solid", bg="#ffbfad")
    error_frame.pack(side="left", fill="both", expand=True, padx=5)
    progress_info_frame = tk.LabelFrame(error_progress_frame,
    text="Progress", bd=2, relief="solid", bg="#80ff97")
    progress_info_frame.pack(side="right", fill="both", expand=True,
    padx=5)

    # Error Block
    error_label = tk.Label(error_frame, text="None", bg="#ffbfad")
```

```

error_label.pack(padx=10, pady=10)

# Progress Block
progress_label = tk.Label(progress_info_frame, text="",
bg="#80ff97") # Empty label initially
progress_label.pack(padx=10, pady=10)

# We can use this function to update the text within the progress
block
def update_progress_text(text):
    progress_label.config(text=text)

#We can use this function to update the text within the error
block.
def update_error_text(text):
    error_label.config(text=text)

# Function to update the progress on the progress bar this makes
the bar larger.
def update_progress(percent):
    progress_canvas.coords(progress_bar, 0, 0,
progress_canvas.winfo_width() * (percent / 100), 20)

```

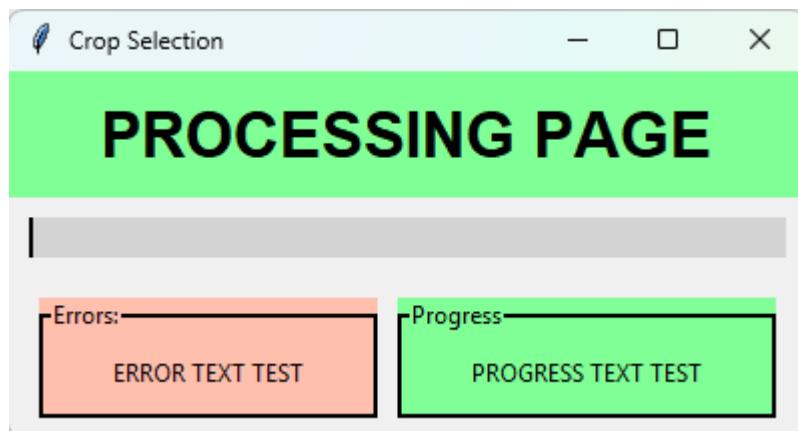
To test this page we tried to actively pass data into all three functions within this page.

We used the following script to test the program to work out if it was able to successfully handle inputs.

```

update_error_text('ERROR TEXT TEST')
update_progress_text('PROGRESS TEXT TEST')
update_progress(50)

```

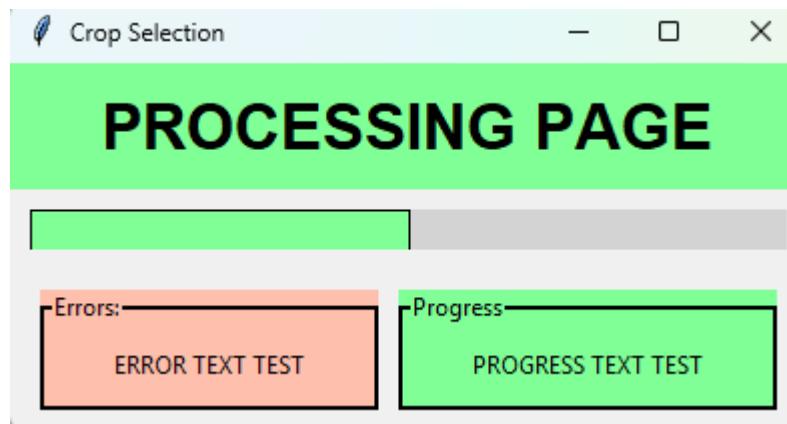


The text in both blocks rendered correctly however the progress bar did not render correctly. There were no errors produced by the program so the syntax is all correct, instead we must

be looking at a logical error as the program itself seems to be running fine. As we are rendering something in python within a tkinter module for performance benefits tkinter takes a different approach to rendering all the elements. This means that as python runs extremely quickly we may have tried to update the progress bar before the actual progress bar has been defined within the window. This may be due to Tkinter's reliance on the GPU for rendering they're elements which may occur in parallel instead of sequentially. Adjusting the update progress bar statement to wait

a single second we can fix this issue

```
update_progress(50) →  
root.after(100,  
update_progress, 50).
```



1. Retrieve the specific crop
2. Retrieve historical weather data for that crop
3. Retrieve labelled commodity prices for that crop
4. Check if the user has set custom hyperparameters. If not, find the optimal hyperparameters.
5. Train the model with the hyperparameters
6. Output the model and retrieve forecasted weather data for the crop.
7. Produce the final predictions as an array and pass it onto the next stage.

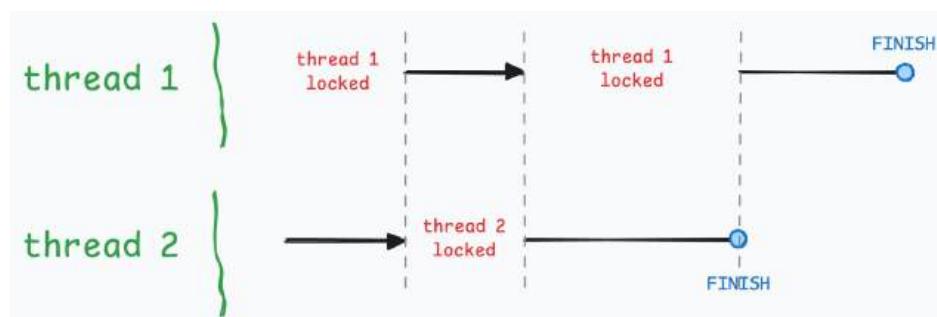
We need to do all of this while showing the progress to the user as a progress bar and indicating errors that are caused by this process.

We added to the `processingTasks` function by implementing all the modules we had before to make a prediction on the selected crop. This meant as described in design we had to implement validation techniques. This is because this is the stage that is most prone to errors due to so many moving variables and tasks being processed at once. We used a `try:` and `except:` clauses across the entire program to catch the error and report it both on the user's screen appropriately in the error dialog box, but give them a more in depth error description within the python terminal they have access to. Talking with stakeholders about the way we display errors we understood that we need to convey a message to both advanced and novice users who may encounter a range of errors. This meant that we could fit both their requirements as advanced users will be able to access and understand the internal structure of our program whereas a novice user will only understand how to

run it so accommodating an error page clearly in front of the user fulfils both their requirements.

The code below shows the processing tasks that occur within the processing phase. This is within a separate function as tkinter doesn't render objects sequentially instead it tries to compile the program all at once. What this meant is that we kept on running into an issue that when the user selected a crop tkinter tried to compile and run all the code at once. This meant that we were trying to process all the data before it was clearly rendered on the screen. We fixed this issue by both separating the program into a separate function and using

`progress_canvas.update_idletasks()` what both of these techniques did is that they forced tkinter into updating the previous progress bar before it could continue allowing all the elements to render appropriately before we started processing any data on our device.



We also encountered syncronisation issues possibly due to the Global interpreter lock which within python prevented us from running the tkinter thread fully in parallel with the processing system. What this meant is that sometimes tkinter's

graphical user interface backend was out of sync with the processes that were happening on device so they were unable to communicate. This lack of communication meant that sometimes elements would not render or we would encounter crashing of the system. This is clearly a tkinter issue when used with high power systems. We used strategically placed `time.sleep(1)` to force the program to wait for a single second for all the systems to get fully in sync with each other. While testing the minimum time that tkinter can handle before being out of sync on a laptop running the minimum hardware requirements we measured the total time to be 100ms. However we implemented a `time.sleep(1)` of 1s or 1000 ms as faster computers in the future are most likely going to get out of sync faster than slower hardware due to being able to execute more instructions per second so a likely larger gap if the two systems are out of sync.

The code below shows how we implemented the entire processing layer's actual component where processing occurs. We take in a crop and then produce the predicted future changes in price as an array.

```
def processingTasks():
```

```

try:#Validation has occurred as we are making sure that
historical weather and price data has been retrieved successfully/
historicalWeather = retrieve_historical_climate(crop)
commodityPrices = retrieve_commodity_prices()

#We update all the progress indicators to show this.
update_progress_text('Retrieved Weather + Price Data')
update_progress(10)

progress_canvas.update_idletasks()#This prevents tkinter
from running anything past this point before the previous tasks have
been completed / rendered.

time.sleep(1)#This allows tkinter to render all the
content as if we are producing data before tkinter is able to render
it, the screen remains empty.

except:
    update_error_text('ERROR - Retrieving Data see the python
terminal')#We catch the errors and display it to the user both in
tkinter and in the python command line interface.

    errorAlert('Unfortunately we were unable to retrieve
either historical weather or commodity prices from the internet.
Check your internet connection or firewall','soft reset')

try:#We validate whether the commodity prices have been
successfully labelled.
labelledPrices = classify_commodityPrice_Dataframe(crop,
commodityPrices)#Label the commodity prices

#Update the progress to show this
update_progress(20)
update_progress_text('Labelled Crop Price Data')
time.sleep(1)

progress_canvas.update_idletasks()
except:
    update_error_text('ERROR Labelling commodity prices')#We
catch the errors and display it to the user both in tkinter and in
the python command line interface.

```

```

        errorAlert('Unfortunately the commodity prices were not
able to be either accessed or labelled.', 'soft reset')

update_progress(30)
update_progress_text('Training the model...')

time.sleep(1)

#If the user has not set custom C and Gamma hyperparameter
values then we can use the grid-search cross validation to find these
optimal values and deliver it to the user.

if custom_cValue == None and custom_gammaValue == None:
    custom_hyperparameterDictionary =
findOptimal_hyperparameters(crop) #Implement a progress bar
    trainedModel =
trainingSupportVectorMachine(historicalWeather, labelledPrices,
custom_hyperparameterDictionary)
else:
    #If the user has set custom C and Gamma hyperparameter
values then we will invoke this else statement. Within this we will
use these values and pass it within a dictionary.

    custom_hyperparameterDictionary = {
        'C' : custom_cValue,
        'gamma' : custom_gammaValue,
        'kernel': 'rbf'
    }

#We can then pass these values in the dictionary into
training the model.

trainedModel =
trainingSupportVectorMachine(historicalWeather, labelledPrices,
custom_hyperparameterDictionary)

update_progress_text('Predicting the future...')

update_progress(80)
time.sleep(1)

#We want the predictions to have a global scope so we can
access it across the entire program.

global predictions
predictions = [] #We set the predictions to make sure it is
empty before we add the new predictions to the dataset.

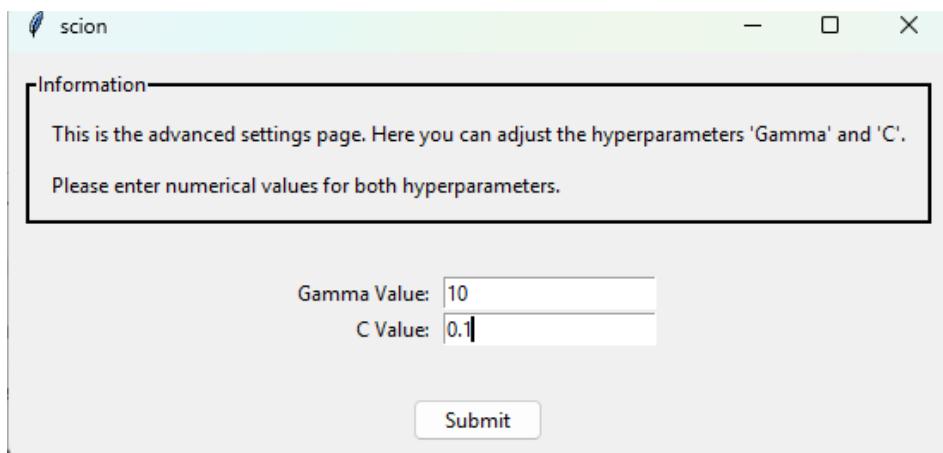
```

```

predictions = predict(crop, trainedModel)
update_progress_text('Complete')
update_progress(100)
time.sleep(1)

root.after(100, processingTasks())#We run the processingTasks
function 100ms or 0.1s after the entire program to let tkinter render
all their content correctly.

```



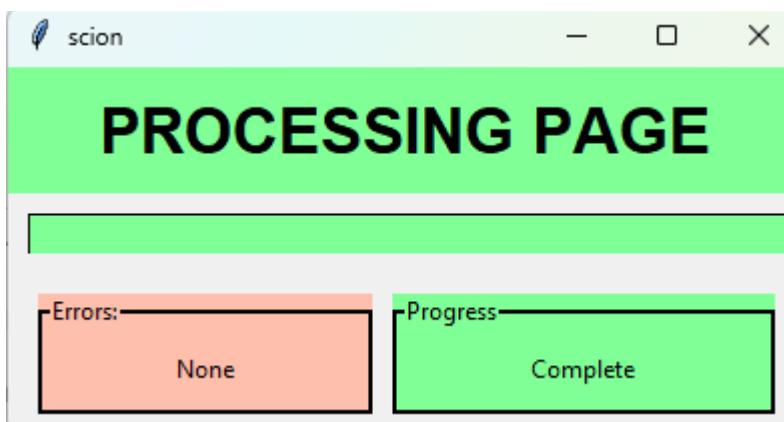
Testing this program we need to do so using two methods. One using custom hyperparameters that we set using the advanced settings page and the other using the systems solution to optimise the hyperparameter values automatically.

First to test it we viewed the advanced settings and set custom hyperparameters into the model we can see this page below. These parameters were then submitted into the model and were used when we were training it. The final output after the program indicated it was complete was the predicted change in price over the next four-day segments.

[1. 1.00438356 1. 1.00109589]

On the other hand when we let the program optimise the hyperparameters itself all we had to do is click the crop we wanted to forecast, in this case 'Wheat' on the

crop selection start page. Then the processing occurs in the background as we saw above and the final output of the program in the python command line interface is



```
[0.99780822 1.00219178 1.00438356 0.99671233]
```

Which also indicated the change in price of the agricultural commodity over the next 16 days through a four-day period.

Outputting Layer:

To implement it to be usable by a stakeholder we need to develop an interactive graph that outputs all the values. In this case we need to do the

```
def output_page(crop):#This is the output page where we output the
graph and the predictions onto a suitable interactive page.

    clear_page()
    import datetime

    commodity_prices_df = classify_commodityPrice_Dataframe(crop,
retrieve_commodity_prices())

    global predictions
    predictions = predictions.astype(np.float64)#Produces a numerical
array of all the predictions from a string to a float data type.

    # Creates the plot
    fig, ax = plt.subplots()
    ax.plot(commodity_prices_df.iloc[:, 0],
commodity_prices_df.iloc[:, 1])  # Plotting of the year and the
price...
    ax.set_xlabel("Year")
    ax.set_ylabel("Price")

    ax.set_title(f"{crop.capitalize()} Prices Over Time") #Title of
the graph (crop name is here) Prices Over Time

    # Retrieve all the dates we need...
    today = datetime.date.today()
    current_year = today.year

    # Get the last data points from the dataset.
    last_year = int(commodity_prices_df.iloc[-1, 0])
    last_price = commodity_prices_df.iloc[-1, 1]

    # Plot today's data (assuming the price is the same as the last
year's price) We are assuming this dataset is the most up-to-date
```

```

dataset we have access to.

    ax.plot(current_year, last_price, 'ro', label="Today's Price")

    # Draw a line connecting the last point and today's point
    ax.plot([last_year, current_year], [last_price, last_price],
'r--', label="Connecting Line") # Draw a line between the points

    last_date = datetime.date.today() # We set the start to be
today's date.

    last_price = commodity_prices_df.iloc[-1, 1] # Retrieve the last
price

    for i in range(min(4, len(predictions))): # Plot up to 4
predictions

        next_date = last_date + datetime.timedelta(days=4) #Add four
days to get to the next point.

        next_price = last_price * predictions[i] # Calculate the
next price using the predictions array.

        # Determine line color based on price change green is
positive red is negative change.

        if next_price > last_price:
            line_color = 'g-'
        else:
            line_color = 'r-'

        # Convert dates to years for plotting
        last_year = last_date.year + (last_date.timetuple().tm_yday /
365.25) # Accounts for leap years as 1/4 = 0.25
        next_year = next_date.year + (next_date.timetuple().tm_yday /
365.25) #We need to do this as our entire dataset has a yearly axis
and we can't plot months and days.

        ax.plot([last_year, next_year], [last_price, next_price],
line_color) #Plot the connecting line with the correct line colour.

        last_date = next_date
        last_price = next_price

def zoom_to_predicted_prices():#Allow time scale viewing to see

```

specific segments of the graph.

```
start_date = datetime.date.today() #We set this to today's
current date beforehand so we can use this same variable.

dates = []
prices = []

last_price = commodity_prices_df.iloc[-1, 1] # Get the last
price from the dataframe using the iloc dataframe function.

current_date = start_date # Initialize current_date with
start_date

for i in range(len(predictions)): # Iterate through the
predictions
    dates.append(current_date) # Add the current date to the
dates list
    prices.append(last_price * predictions[i]) # Calculate
and add the predicted price

    current_date = current_date + datetime.timedelta(days=4)
# Increment the current_date by 4 days

years = [d.year + (d.timetuple().tm_yday / 365.25) for d in
dates] # Convert dates to years for plotting as our x axis is in
years.

# Set x,y zoom
ax.set_xlim(min(years) - 0.15, max(years) + 0.15) # Zoom to
see x axis years
ax.set_ylim(min(prices) * 0.99, max(prices) * 1.01) # Zoom
to see y axis predicted prices

canvas.draw() # draw the canvas again to reflect the zoom.

zoom_button = ttk.Button(root, text="Zoom In to see Current
Changes", command=zoom_to_predicted_prices)#Button to zoom in
zoom_button.pack(pady=10)

ax.legend()
```

```

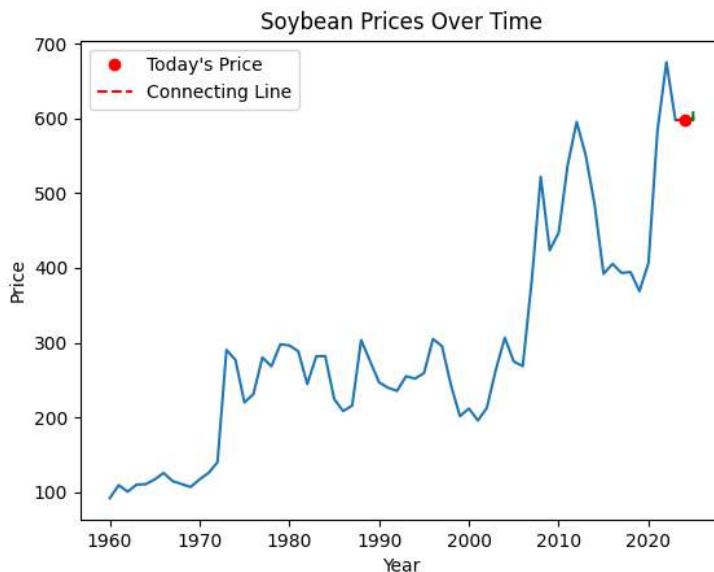
# Embeds the plot in the tkinter window
canvas = FigureCanvasTkAgg(fig, master=root)
canvas.draw()
canvas.get_tk_widget().pack()

# Adds the matplotlib toolbar
toolbar = NavigationToolbar2Tk(canvas, root, pack_toolbar=False)
toolbar.update()
toolbar.pack(side=tk.BOTTOM, fill=tk.X)

# Creates and packs the back button
back_button = ttk.Button(
    root,
    text="Back to Crop Selection",
    command=lambda: [crop_selection_page(), root.destroy()],
)
back_button.pack(pady=50)

```

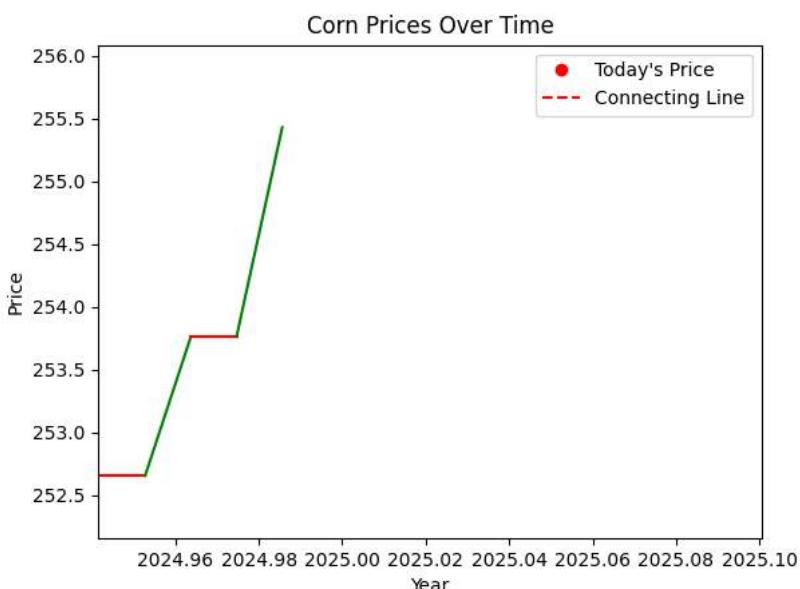
This is the final stage of the solution and allows the user to easily interact with the final solution. The predictions were produced on average 30 seconds after requested which meant that there wasn't a need to implement a time constraint as described before. A time constraint would be necessary if the program was taking more than 30 seconds to process data, however on a computer that is equal to the minimum requirements that we tested on it was effective to allow the computer to process and output the prediction in full without external structuring.



We made a range of stakeholders test the solution after this final stage to get they're opinions. Talking with Manu, a farmer that was interested in a range of commodities he was amazed at the in depth of the final solution.

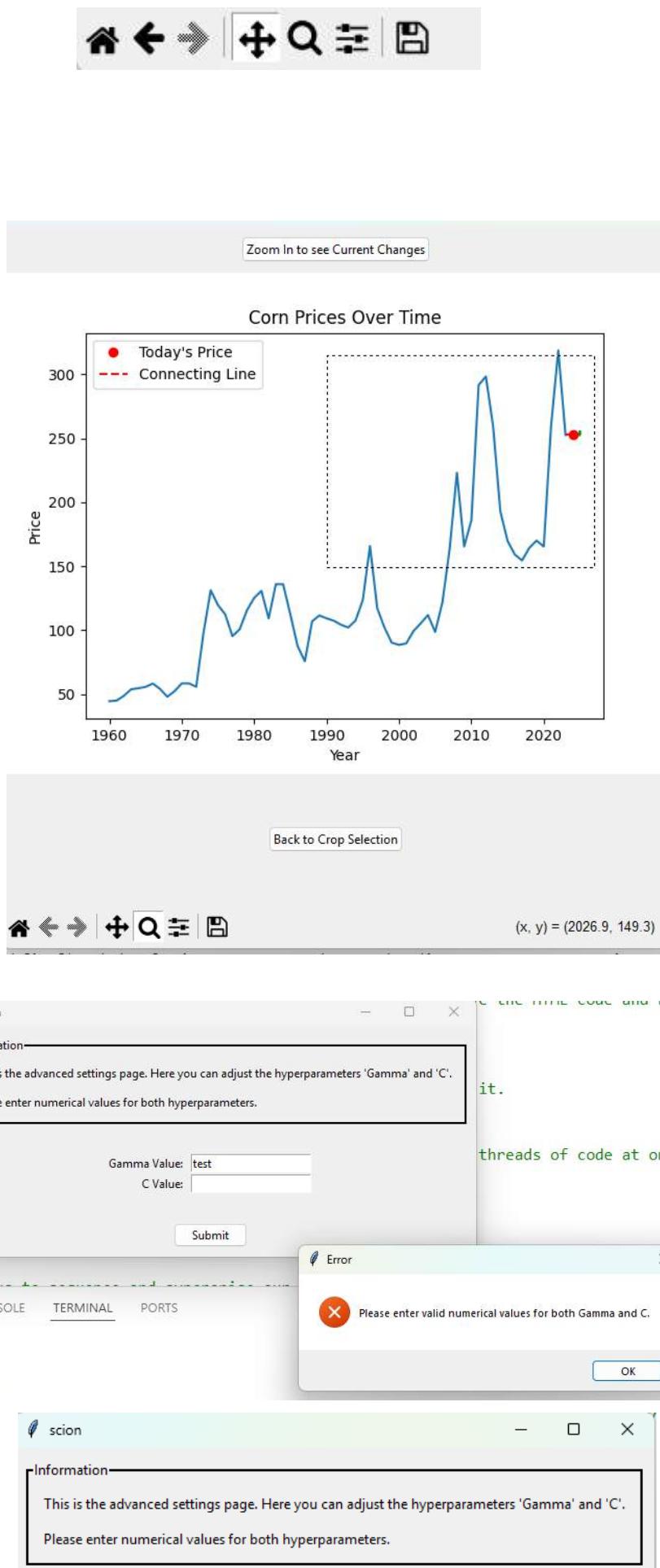
He specifically looked into how it could clearly predict the smaller price movements within each time segment. In one case he produced a detailed graph of the future price of corn commodities which we can see below. Instead of the solution having limited detail it was able to clearly segment itself across a four-day long time segment and make a range of predictions to finally all be combined and plot a graph.

We also made stakeholders test a zooming function and a toolbar that allowed the stakeholders to



directly change the timeframe of the solution. This toolbar allows user's to easily zoom / pan / scale across the entire graph.

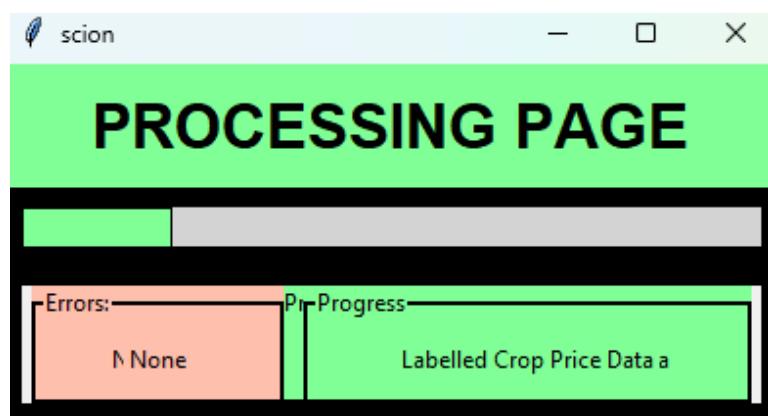
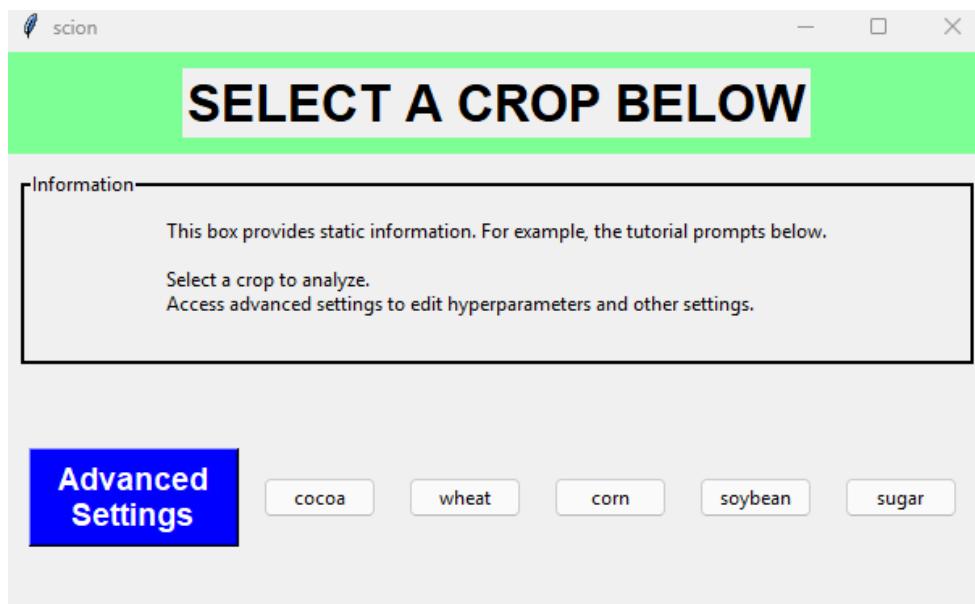
[Zoom In to see Current Changes](#)



This shows the final produced output of our solution talking with Sharath he said it fully met his requirements and worked quickly and effectively as required. Feedback included the actual predicted data being small in comparison to the rest of the chart when we first displayed the program, however we acknowledged and fixed these issues by implementing the necessary zoom in function to see changes in more detail along with the toolbar which allowed users to see clearly how the price changes in real-time.

Mr Manu also wanted to test if the program could handle inputting advanced parameters such as gamma and c values and using that to make a model which will then be used to make predictions. Manu first tested the system's advanced settings box inputting a string and leaving a box empty. The validation within the program detected this and

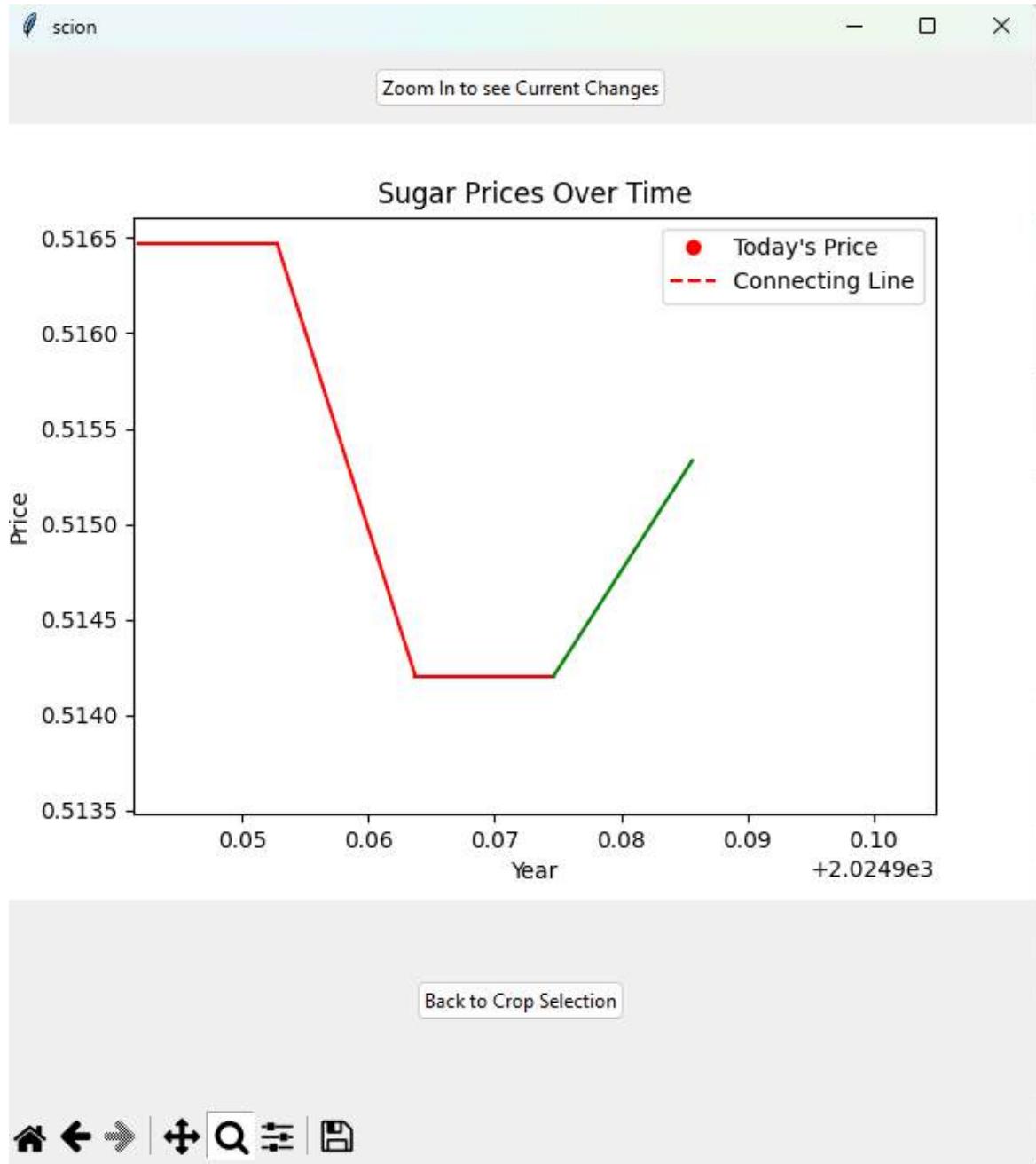
forced the Manu to input correct numerical values for both hyperparameters which in this case was 10 for gamma and 0.1 for C. Once this was complete he was able to successfully click submit and go back to the crop selection page. This page allowed the user to select the crop they want to analyse which in his case was soybean and sugar. Starting off with sugar he then proceeded into the processing page where all the data in the background is processed.



This is the following progress and error indicators he will have seen while using the program. As he is running the software on a machine that exceeds the minimum requirements in total he only spent about 15 seconds on this page. In comparison to average hardware that meets the hardware requirements spending upwards of 30 seconds.

The below image shows the final output of the page as Manu can clearly see how the price of an agricultural commodity will change in the next 16 days. Another Stakeholder requirement was the ability to sequentially process crops.

Manu, our stakeholder, wanted to be able to first analyse one crop and then go to the next crop without exiting our application. The inclusion of the “Back to Crop Selection” button at the bottom of the screen allows Manu to do this easily as he can quickly return to the main screen, specify if necessary any advanced settings and then produce predictions for the following crop.



The above image shows Manu analysing the first crop. The image below shows the next crop Manu analysed and produced predictions for which in this case was Soybean.

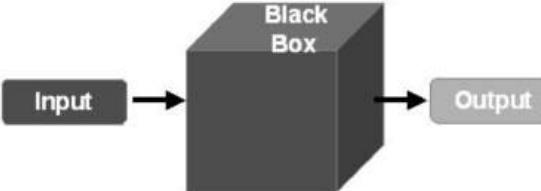
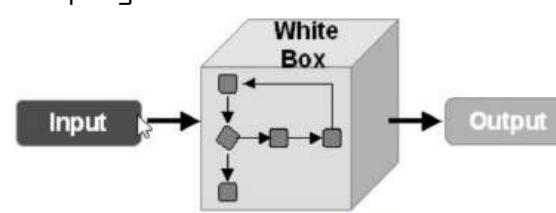


Ultimately all the stakeholders agreed that our solution met the proposed design and requirements that they set out to accomplish.

Evaluation

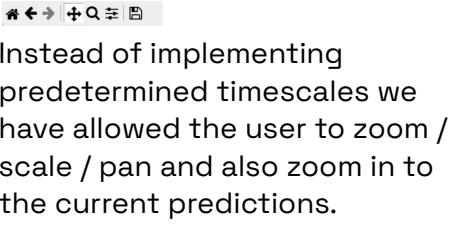
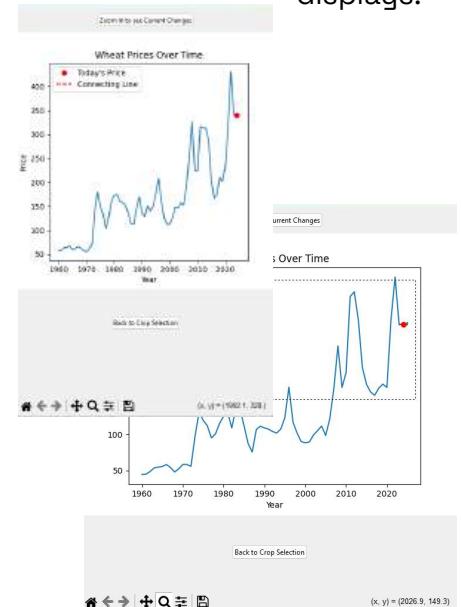
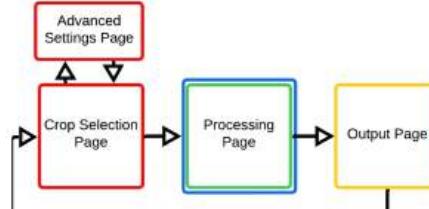
Testing to Inform Evaluation

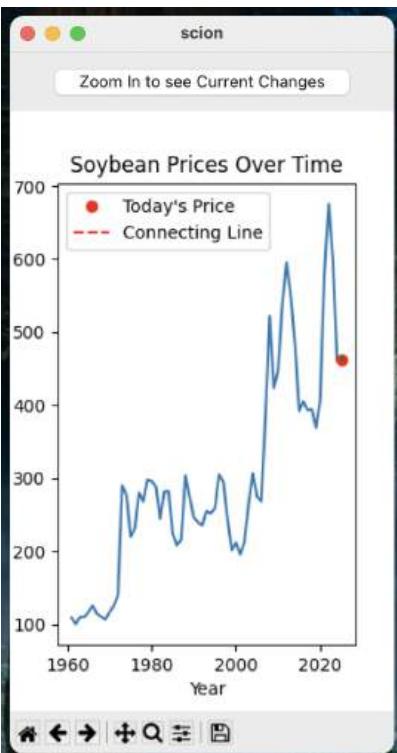
Our tests are conducted in 2 independent sections. To test both functionality of the program and the robustness we navigate each test plan separately.

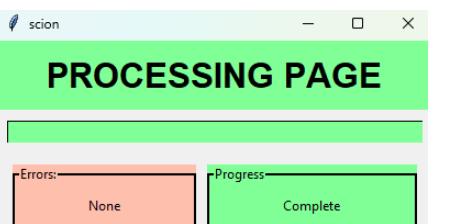
Functionality Testing	Robustness Testing
<p>Functionality testing aims to verify that features work as intended under normal conditions. Ensuring the core functionality of the program operates correctly.</p> <p>Functions are large collections of components. The entire solution can be thought of as a single function that produces an output.</p> <p>Therefore a successful functionality test will produce the desired output that meets the user's requirements. These functions and specific smaller functions were defined in the success criteria during the analysis stage of the project.</p>  <p>This is an example of black-box testing as the tester is unaware about the internal structure of the program and only the expected outputs.</p>	<p>Whereas testing robustness verifies how 'robust' our program is to unexpected inputs or actions. Some of which may be invalid or extreme cases.</p> <p>Specifically robustness testing verifies that the program can handle boundary conditions. Such as extreme inputs and different scenarios. This may involve testing the boundaries of input values and specific edge cases to recreate specific scenarios.</p> <p>A successful robustness test would involve the decision points within the internal structure of the program to successfully classify data and consistently maintain the behaviour of the program.</p>  <p>This is an example of white-box testing as the tester is testing the internal structure of the program and the boundaries as to how the program can handle different inputs and specific scenarios. E.g Errors within the dataset and incorrect inputted hyperparameters.</p>

Functionality Testing:

Evidence of functionality testing is included within the success criteria as it tests the stakeholder requirements which represent how the end-user is expected to use the final solution.

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
1	Timescale changing. Allow the ability to zoom in and pan with the overlaid trend data.	Testing the interactive nature of the graph by switching between multiple viewpoints. Interacting with the original data overlaid on the graph seamlessly.	Switching between multiple segmented timescales using a set of buttons on the interactive graph page.	 <p>Instead of implementing predetermined timescales we have allowed the user to zoom / scale / pan and also zoom in to the current predictions.</p>
2	Compatibility with different devices of different screen sizes.	Change the dimensions of the outputted graph to test the resizing algorithm validating that all the data is shown completely on the screen.	All buttons and interactive elements are visible at all times. This allows us to use this tool on a range of hardware such as tablets, mobile phones and large monitors without needing to build separate interfaces.	The software meets this criteria as it is able to fluidly resize to accommodate different sized displays. 
3	A User Interface with clear differentiable operations.	Stakeholders traversed through each page and tested that all components were both easily accessible and the context behind it easily understood.	Each page serves a separate independent function that is able to do a specific task that users of all skill levels understand.	 <p>All pages were developed separately and the user sequentially traverses through all of them. This means that each page has a single function.</p>

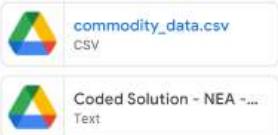
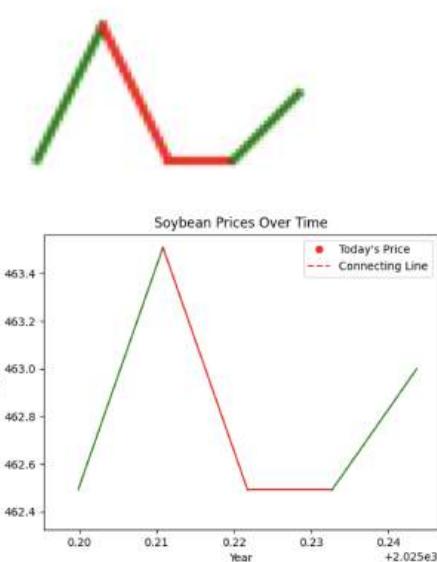
No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
4	Allow for resizing on the device while you are using the software.	The software is able to actively resize when you are using the hardware on multiple devices. This is an active resize so the program is running and executing instructions while the user in parallel resizes the software solution.	The program should be able to effectively scale all the interactive elements and other elements on each page to fit the new dimensions of the program effectively.	 

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
5	Allow for the specific time scale selection of graph data. For instance looking at data across individual days to months and years.	Changing the viewpoint of the original program by using a range of features to traverse the generated graph.	A fully interactive graph that is able to output the historical agricultural commodities data along with overlaid predictions effectively.	 <p>The graph allows for looking into a specific time segment using the following features: zooming / scaling / panning so it partially meets this criteria.</p>
6	A maximum of 2 decision points on each page. A decision point is a point that the user decides which page to next access.	We limited the amount of decisions on each page. This was easy to implement as the previous page navigation design meant that separate actions were kept separate. Such as outputting and processing as they had distinct pages.	Each of the separate independent pages had a maximum number of total decision points and conformed to this stakeholder requirement.	 <p>Looking at the start page there are only the options to either choose advanced settings or pick a crop.</p>
7	Clear colour differentiation between interactable objects and different objects such as trend lines and market data.	Within the processing page all the processes that indicate progress are green and the errors dialog box is in a separate orange frame.	Data that covers separate decision variables are indicated by the separate colouring within the user interface.	 <p>Mutually exclusive elements in this case the error and progress dialog boxes are separated by their differences in colour.</p>

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
8	Customizable colour schemes across all the front-end pages. Aiding the accessibility of our system.	Change the colour scheme of multiple objects within the program and validate that it is accurately displayed within the program.	Customizable colour schemes hidden within the advanced settings page where the user can set custom values for a range of objects and hyperparameters within the solution.	<p style="text-align: center;">Advanced Settings</p> <p>Potential implementations could involve changing the colour schemes within advanced settings for example allowing further customisation of the tool. This feature was not implemented.</p>
9	Clean data retrieved by the API. Using a function that initially checks for missing or anomalous values and then imputes them with valid values.	We have implemented both anomalous and missing value detection within our program to detect errors within our datasets. These errors are “imputed” which means they are replaced with appropriate values.	A dataset that contains missing and anomalous values should be correctly corrected by imputing substituted values in their places. Effectively cleaning the dataset.	<p>See source code. <code>impute_anomalies(dataframe)</code> and <code>impute_missing(dataframe)</code>. These two functions take in a dataframe (in this case the weather and commodity price data) and detect either anomalous or missing values. If any erroneous results are present they are updated and returned as a dataframe.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <code>impute_anomalies(dataset)</code> <code>impute_missing(dataset)</code> </div>
10	Format data retrieved by the API. To successfully pass onto the processing stage.	Formatting involves validating and forcing datasets to conform to the requirements of the program. Whenever data passes through stages in the programming it is validated twice: once when data is sent and again when a module receives any data.	Invalid data isn't allowed to traverse the solution and is instead caught and replaced with valid values in the correct format.	To keep consistency within the formatting. Data types and the form they are presented in is kept consistent, such as the date which is always in the international ISO8601 standard.
11	Keeping API keys secure	OpenMeteo and web scraping don't	API usage isn't misused by the user	Therefore there is no need to keep API keys secure as the API

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
	and not accessible to the user directly. Such as using encryption techniques to store them safely.	require API keys, instead they use the device's IP address to respond to requests.	and all traffic is identifiable by the users IP address which requests data transmission.	responds to the request directly to the device.
12	Using caching to store frequently accessed commodity price datasets on the device locally instead of relying on external sources.	We have a stored commodities prices database within our program which stores historical commodity prices across the five commodities.	Data retrieval is more efficient as only required data is retrieved from external sources such as from APIs and essential datasets are stored within the system.	commodity_data.csv CSV We have a .csv (Comma Separated Value) file that stores the commodity data. It is accessed and updated within the program.
13	Split-screen to see two predictions at once simultaneously and being able to interact with both instances separately at the same time.	We can do this by opening the program twice in two separate windows and running them simultaneously.	Both separate windows are separate instances of the solution so it allows them to run separately on multiple threads within a multi-core processor.	However this is not available as a feature natively within the program. This is possible by running the programs simultaneously in parallel. 
14	Up-to-date data used, relying on high-quality external datasets such as for current commodity prices and weather data	Using APIs instead of relying on a stored database means that we only use "fresh" up-to-date data from reliable sources.	We used the World Bank's pink sheets to access up to date and historical commodity prices.	Annual prices March 2025 (XLS) Data was successfully retrieved from external sources using web scraping techniques including the use of

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
				beautifulsoup4 to do these activities.
15	Intuitive Navigation across the entire system between pages and when outputting data.	We have conformed to the previous requirements by having an easy to navigate system. We successfully implemented the application in separate independent stages.	Each stage provides the user with a specific decision point. For instance a page may have links to an advanced settings page or additional separate pages.	Each page seamlessly connects with the next and there is a framework of how to traverse throughout the entire program as specified below.
16	Transparency with the user about what data is being used and how it is being processed	End-users testing the program should understand the context behind actions or processes taking place at all times while the program is running.	It is expected that there is clear contextual information provided to the user that indicates any background activities that are occurring.	 Dialog boxes that specify program components are included to inform the user as to processes that are occurring.
17	Scalability	To scale the product to be used by multiple users is only constrained by the reliable access to data.	Multiple users at once are able to use and access the program simultaneously without any limitations.	Reliance on APIs for data retrieval means that the only restraint on the scale of our solutions is the availability of open source APIs in storing and transmitting the required data effectively.
18	Responsive Design across all the pages so the user can interact with elements quickly without the system getting either stuck or taking extremely long	The system is able to quickly produce outputs to any requested data for instance producing predictions or generating models.	The system is able to have a consistently low response time for all processes and process data extremely quickly.	All activities to generate a single prediction for a single crop can be done within a single minute which was a requirement set by the stakeholder in order to limit the total time it takes to generate prediction. This prevents activities taking too long and keeps the entire process responsive.

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence																		
	to process a task.																					
19	Help and Support resources provided to the end-user so they are able to understand how to interact with the software.	The end-users are able to understand how to interact with the application and how to make it do specific tasks.	The entire system should be clear how to navigate and produce effective predictions. In this case it should be easy for the user to generate a specific prediction.	Dialog boxes provide context of how to use the application instead of having a dedicated page for interacting with the solution the user is informed at all times how to use the solution.																		
20	Efficient use of storage space such as only storing necessary databases and programs.	The entire system should not use too much space to store both programs and datasets locally on the users device,	Data usage of storing the entire application on the user's device should be minimised. Only necessary essential datasets should be stored on the user's device.	 <p>As the program implemented components within the cloud data is also stored externally and retrieved using APIs.</p>																		
21	Consistency when processing results. With minimal variation when analysing the same dataset with the same hyperparameter variables.	Processing the same dataset with the same initial conditions and hyperparameters should produce the same results in this case future predictions.	There should be consistency within the results of the program as the same initial conditions should produce the same end-result.	<p>The program generated two separate models on the same initial data and were applied to the same forecasted weather data and the same output was produced.</p>  <table border="1"> <caption>Soybean Prices Over Time</caption> <thead> <tr> <th>Year</th> <th>Today's Price</th> <th>Connecting Line</th> </tr> </thead> <tbody> <tr> <td>2020</td> <td>462.4</td> <td>462.4</td> </tr> <tr> <td>2021</td> <td>463.4</td> <td>463.4</td> </tr> <tr> <td>2022</td> <td>462.4</td> <td>462.4</td> </tr> <tr> <td>2023</td> <td>462.4</td> <td>462.4</td> </tr> <tr> <td>2024</td> <td>463.4</td> <td>463.4</td> </tr> </tbody> </table>	Year	Today's Price	Connecting Line	2020	462.4	462.4	2021	463.4	463.4	2022	462.4	462.4	2023	462.4	462.4	2024	463.4	463.4
Year	Today's Price	Connecting Line																				
2020	462.4	462.4																				
2021	463.4	463.4																				
2022	462.4	462.4																				
2023	462.4	462.4																				
2024	463.4	463.4																				

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
22	The program should be able to highlight values that it deems anomalous. Anomalous values are data points outside the normal range.	Datasets with artificially introduced anomalous values are passed into the program as part of both commodity prices and weather data.	The program should be able to both identify the anomalous values and impute them with suitable substitutions maintaining the accuracy of the entire dataset.	The program was able to successfully detect anomalous values and replace that specific value with an average value for that entire column.
23	Anomalous values that are flagged should be handled appropriately and fixed. Such as using techniques such as imputation to replace the values with valid substituted values.	A dataset with artificially introduced anomalous values should be flagged appropriately and the necessary steps to clean the dataset should occur.	Anomalous values that are flagged as anomalous are handled appropriately by the system; this means that it is not possible for anomalous values to be retained within datasets before it is trained on the model.	Anomalous values are substituted effectively and also flagged effectively. This means that datasets are clean and can train the SVM model efficiently without introducing additional bias and reducing the models overall accuracy.
24	Flexibility of the program to handle missing data points across commodity price datasets and weather datasets. As externally sourced data may contain missing values.	Missing values are data points that cannot be used to train the model. In this case these need to be resolved before a complete dataset is passed onto the model.	Datapoints that have missing values should effectively be replaced and processed in training the machine learning model without losing overall the accuracy of the model.	Missing values were appropriately substituted to produce a complete dataset that can be used to train the machine learning model.
25	Ability for the system to handle external errors	Automated debugging means that the program is able to catch errors	If an error occurs it should be caught and then automatically debugged to quickly	Retry-cache was able to be used to automate the debugging processes of reconnecting to the API and

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
	from the API. Automating debugging processes so the system is resilient to external errors due to the API.	before they occur and execute processes to fix them.	resolve it and continue with the normal operations of the system.	requesting the retransmission of packets to the specific IP address without the user needing to restart the application or request.
26	Suitable error messages generated to the end-user in case actions can't be executed.	Simulate a scenario where an error occurs such as a failed API request or a missing dataset file.	The system should provide a clear and concise message directly to the user explaining the issue and how to debug it.	As part of the retry-cache system the program was able to debug itself and indicate that it was reconnecting to the internet in order to retrieve the necessary data.
27	Add advanced settings for users that want more control as part of a custom advanced settings page.	An advanced settings page is provided as part of a page an end-user can access.	Navigate across the advanced settings page by modifying hyperparameter values and generating a custom model using this original data.	The system should appropriately save the changes of any modified advanced settings and apply it when a custom model is trained and generated.
28	Add a start-up page to open when the user first opens the program.	When the program is launched a page is provided which introduces the tool and allows the user to start generating agricultural commodity predictions.	The user is able to interact with the system with a straightforward graphical user interface.	Upon launching the executable file the program displays the start up page made using tkinter directly to the user to interact with the program.
29	The start-up page should convey the meaning of our tool effectively. With labelled pages as to future decision	Launch the application and navigate around the start-up page.	The startup page conveys clearly how to use the tool and what its purpose is. Additionally indicating choices of the crop the user can choose.	The startup page is clear and concise and the user is able to easily select the crop they want to analyse.

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
	points within the program.			
30	Simplicity and minimalism is kept throughout all the visuals and graphics in the application. By only outputting necessary data to the user.	A new user should be able to navigate through the entire program effectively without needing additional external help.	The overall framework and design is kept minimalistic and only displays necessary data elements without clutter.	Only outputting necessary data and elements means that the entire process maintains a minimalist approach.
31	Multitasking capability to be able to access and predict the future commodities movements for multiple crops at once simultaneously .	Data inputting should allow multiple crops to be processed simultaneously without them interfering with each other.	The system should be able to process and predict agricultural commodities for multiple crops concurrently without performance being excessively degraded.	By allowing the solution to run on separate threads on a multi-core processor it is able to run in parallel with each other as seen with the application of this when analysing and predicting crops in split screen.
32	Appropriate Indexing of the commodity price database within the locally and externally sourced one.	Test the retrieval of commodity price data all at once this means that data will have to be effectively retrieved from both the current database and the externally sourced database.	If all the values are indexed appropriately data retrieval occurs both quickly and with high accuracy due to the proper indexing.	Consistency between multiple separate independent data elements means that it is necessary that they work in conjunction with each other.
33	Interrupting the system while processing should not	Close the system while processing activities are occurring such as during the	Verify that the original database is uncorrupted and data integrity is preserved. This means that the	While data processing activities were occurring we tested a forced interruption upon further processing after restarting the program it was

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
	corrupt the contents of the database. Adding additional file handling anti file corruption measures so data is handled appropriately.	processing stage where the processing screen is active.	solution has an added level of robustness as it's able to consistently repair data and avoid corrupting data sources.	able to produce valid predictions without being impacted by the previous crash that occurred. This means overall the system is resilient to crashes and can consistently produce predictions.
34	Process data concurrently when available on the user's system. Such as parallel processing.	Parallel processing speeds up data processing in comparison to processing data sequentially in chronological order.	Concurrent data processing makes data processing tasks more efficient and avoids system bottlenecks by maximising the usage of multiple cores or threads within the hardware.	When available on the user's system python automatically switches to allow for the usage of multiple cores to maximise the efficiency of the program and to produce predictions as fast as possible.
35	Provide a wide range of potential commodities that the user can select and analyse. Agricultural commodities range from cocoa, arabica coffee to corn, sugar and wheat.	Select and analyse a range of the provided agricultural commodities within the system and produce multiple predictions for the commodities price movements.	The solution should allow end-users to select and analyse multiple agricultural commodities. The output of these analysis activities being the change in the agricultural commodities price in the short-term.	Once selected at the start up page multiple crops which were corn, cocoa, wheat, soybean and sugar. All these agricultural commodities were selected at the start-page and at the end produced a short-term prediction using historical links between the price of an agricultural commodity and climate data.
36	The model takes seasonal harvest changes into account so isn't a static model it is actively able	The performance of the model overall should be able to understand the patterns that occur throughout each season and apply that onto unseen	Reflecting seasonal changes will mean that the original model's training data is complete and trained effectively to generate these predictions.	Testing corn commodities across all the seasons using a backtest which involved comparing the predictions generated by the program with the actual predictions. The model was able to successfully make accurate predictions on

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
	to improve its previous predictions and model.	forecasted data to make an accurate prediction.		the unseen testing data.
37	Continuous refinement of the machine learning model by training it with newly accessible data to make more accurate predictions.	Retrain the same model by consecutively retrying the same initial requested prediction. This means that the model will be trained again and refined.	A model with newly incorporated data should be more accurate than a model that is only trained on historical data points. This additional data means that the model is more accurate.	This involves comparing the predictions the original model made with predictions the new model made with the same input data. In this case the new model had about an increased 5% accuracy than the previous model in predicting the price of corn commodities.
38	Resource optimization so the hardware it is running on locally is fully utilised effectively and optimised to produce the most accurate and fast as possible.	Executing the entire program on the minimum hardware using a virtual machine to simulate the hardware and monitoring the overall GPU and CPU use of the hardware.	The model should be able to efficiently use the hardware within the virtualized machine maximising the total hardware utilisation without overloading the entire system.	 <p>Measured CPU load and GPU load showed that the entire system was utilised efficiently without hardware allowed to idle. In this case more powerful systems will be able to effectively use the program without crashing or processing tasks taking extremely long.</p>
39	Ethical data processing by following GDPR across the entire software solution.	Simulate user data inputted into the system ensuring all data is anonymized and stored securely. Following GDPR regulations appropriately.	All data should be processed in compliance with GDPR. With no data being transmitted without the user's consent.	Data should be anonymized before it is processed if it contains user identifiable data. This complies with GDPR. As the IP address anonymises the user's identity data can be processed as normal.
40	Transparency with the user about what data is being imported from external sources and	At all stages of processing or retrieval activities the user should be informed as to processes	Transparency should be retained throughout the entire program informing the user data sources where data is retrieved from and	Dialog boxes along with elements within the processing page inform the user as to processes currently executing. This means that the user is able to effectively and thoroughly understand what processes are

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
	how it is being processed.	occurring in the background.	how it is processed on the users device.	occurring in the background.
41	Follow data exchange standards when passing exchanging data across the entire software solution.	Data exchange happens within the program when data is exchanged across each stage. This means that validation needs to occur at each stage to ensure that data is exchanged within standardised formats.	It is expected that when data traverses the system it should be adhering to standardised formats such as using CSV, Dataframes data structures. This means that data processing activities can happen without errors occurring due to the format of data.	Parsing and using breakpoints to debug the data exchange across components we can verify that data adheres to the original standard. Additionally data validation techniques aided in the necessity of data being in a set format.
42	Adhering to API Rate Limits to avoid abusing their system.	Send multiple API requests to OpenMeteo for weather datasets and web scrape the commodity prices until the API rate limit is reached.	The system should respect the limits and timeouts implemented by the API and avoid exceeding them. Additionally the system should be able to handle errors produced by the API.	OpenMeteo's rate limits are extremely lenient which means under normal load the user is unlikely to hit the maximum amount of data that the API can retrieve. Additionally as downloading commodity prices from the internet takes some time there is not a large bandwidth usage so the website didn't block specific IP address traffic. Therefore operations in retrieving data from external processes could occur seamlessly.
43	Cross-Platform compatibility with multiple devices such as laptops and mobile devices irrespective of their operating system.	Execute the software on multiple different devices such as on a windows, macOS laptop and mobile devices such as tablets and smartphones.	The software solution should run smoothly on all the devices without excessive lagging and the same maintained functionality.	On high performance hardware such as laptops the software was able to run smoothly with no performance issues. Additionally on mobile phones the system took longer to process the same data however performed well. The implementation of a touchscreen interactive graph allowed users to efficiently utilise the software.

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
44	Consistent Data sampling frequency to make sure all the datasets contain all the necessary data to generate a model.	Input an invalid incomplete dataset into the machine learning model initially. In this case weather and commodity prices may not be matched.	The system is able to either independently standardise these two datasets automatically or instead recognise there is a discrepancy between the dimensions and highlight this error to the user.	Before the SVM models are generated while testing two datasets with a different amount of data points. The solution is able to recognise this and highlight that the frequency of the two datasets don't match and aren't valid.
45	Be able to effectively execute operations in the background while the user is not actively using the application.	Minimise the program application and open another application on the device.	While the program is minimised the resource usage should remain efficient without impacting the entire system.	The system while minimised showed its resilience to multitasking by allowing multiple programs to run simultaneously while data is processed in the background.
46	The solution should effectively be able to have a high uptime without crashing or experiencing preventable errors such as not having access to data.	Execute the software continuously for 1 hour while simulating potential scenarios that the program could come across such as corrupted data sets and internet connectivity issues.	The software should be able to handle these errors and run continuously for an hour by automatically handling them.	The system successfully was able to process data consistently by automating debugging operations and effectively patching errors that may occur during normal operations.
47	Implement a progress bar to show the progress of any processing activities directly to the user.	During the processing stage where all the data is processed, a progress bar shows the respective progress of all the operations.	The progress bar updates in real time in parallel to the operations being processed.	The progress bar effectively shows the progress that is occurring in the background. This was done in parallel to the program processing data in the background by utilising separate python threads which run simultaneously to each other.

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
48	Build using a modular design using several individual and independent reusable components.	A modular design requires multiple individual components. Each component functions independently and can be reused throughout the entire system.	The entire solution is a collection of modular components that have been pieced together to produce the solution. This also aids the maintainability of the program as individual components can be replaced or debugged.	The final solution uses completely separate and independent components that have been split into stages. Each stage is made up of multiple components and allows tasks like data validation to take place effectively.
49	UI of the visualisation of the outputted graph should be consistent with other trading applications.	The output graph should be easy to use and similar to popular applications that stakeholders use such as Robinhood and Trading212. These applications' UI's have been detailed in the design section.	The interactive user interface should be intuitive and easy to use which means that it will be consistent with popular trading applications like Trading212 and Robinhood.	The final output graph was easy to use on all devices and specifically allowed for touch-based input methods such as on a smartphone device with a small screen.
50	Trend interface should be clearly overlaid on top of historical and present agricultural commodity data.	Test viewing both historical and present agricultural commodity data on a single graph. Additionally overlay a prediction on top of previous data to see both elements simultaneously.	The predicted trend should be overlaid on the future and should be distinguishable from the previous historical data.	The future forecasted trend produced by the model was clearly overlaid on top of the respective data and could easily be distinguished and understood from other data points.
51	Automatic API Data retrieval timeout by being able to retry requesting data in case the API hasn't provided it	Test the APIs retry mechanism to request the retransmission of packets or to try retry connecting to the internet.	If no data is received after waiting for requested data the system should suitably retry the request of packets to be retransmitted or aim to try to reestablish	Using the retry-mechanism within retry-cache the program was able to successfully retry connecting to the external API automatically or request retransmission of data.

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
	within an appropriate time frame.		connection with the external API.	
52	Directly output errors that occur to the user in a clear format.	Test that the system is able to output errors to the user when necessary so they are informed.	All error messages are handled by the error component within the program. This means that errors should be displayed to the user if they occur.	Through the error logs dialog box within the processing page stakeholders were able to see and understand any errors that arose while the program was executing tasks.
53	Data should be retrieved from the API and passed onto the model within 10 seconds.	Measure the time taken for data to be requested and to be passed directly into the training stage to train the model.	Data should be retrieved front the API and passed onto the model within 10 seconds.	Data retrieval occurred extremely fast as both the climate API provided by OpenMeteo and the commodity prices only took a maximum of 5 seconds to retrieve from and preprocess to pass onto the model.
54	The prediction should be produced within 60 sections of being executed. This includes time to retrieve and compile data.	Start an activity to predict the future forecast of a specific crop and time the total time it takes to complete these activities.	The total time from when the end-user requests a prediction and the final output of the program should happen within a 60 second timeframe.	The entire system from start to finish took only a maximum of 20 seconds. This included data retrieval of datasets, compiling the model and producing the required predictions.
55	Automate the number of epochs the model is trained over. A single epoch is one training cycle over the training data.	Input a dataset to start training the initial machine learning model and monitor the total amount of times the model is trained on the training dataset.	The system's model should be trained adequately over the training data; this means it isn't overtrained or undertrained on the initial dataset.	Instead of implementing a system that automates the number of epochs the model is trained instead all models are trained on the dataset the same number of times. Instead the model's accuracy is impacted by hyperparameters which are set to the most optimal values and then used in conjunction to train the model accurately.
56	Display the time-period	Test that the output of the	The time-series graphs are accurately	The time-period graph accurately represents the

No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
	graphs for the price of an agricultural commodity and accurately.	entire solution has correctly allocated a correct time segment and the correct individual date values for the x axis.	displayed and end-users can distinguish between historical and present price data. This means that users are able to interpret graphs easily.	historical, present and forecasted price data. Stakeholders said they were easily able to distinguish and understand all the data points.
57	Retrieve external agricultural commodity price data and climate data using an API successfully.	Send an API request for weather data alongside a web scraping request for agricultural commodity price data and validate that the response is valid and in the correct format.	Both systems should successfully be able to produce valid datasets. One for historical and present commodity prices and another of forecasted and historical weather data.	The APIs and external sources of data were able to successfully produce data that is valid and in the correct format for the system to understand and further process and analyse the data.
58	Compatibility of both weather and commodity price data to be used in conjunction with each other to produce a model that can predict the future prices of agricultural commodities.	Retrieve both processed weather and commodity price data from the system before it is trained on by the model and validate that it is valid.	It is expected that both price data is compatible by containing sufficient data points that are accurate and contain no missing or anomalous values.	During this stage data compatibility is ensured by a validation stage which ensures that only datasets that are able to be processed are actually processed. This means that any model that produces predictions within the solution has been verified to be using representative data that is accurate.
59	Validate data types and the format of data before it is passed into the model	Input weather and commodity price data with varying formats and data types into the model.	Only data types that are of the correct format should be passed into the model for training. In this case invalid data is unused and discarded while also generating an error to indicate it.	Data validation to verify that data is the correct format such as a dataframe or csv for a dataset. This means that the model only receives data in the correct format that it is able to process itself.

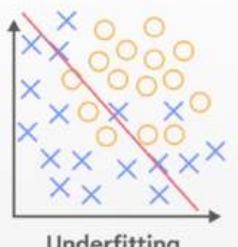
No.	Component Tested	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
60	Create a set of pages that all users will interact with that contain all the essential functionality of the software solution.	Traverse the system by producing a prediction and following how the solution flows to get to the final output page.	Pages have been segmented into distinct sections. These distinct sections have their own functionality so pages that do different tasks are kept independent from each other.	<p>Pages themselves should contain their own essential functionality. Pages have been split into the following.</p> <ol style="list-style-type: none"> 1. Input Page 2. Advanced Settings Page 3. Processing Page 4. Output Page <p>Each page has a single task that it focuses on and allows the entire system to be kept intuitive and easy to navigate across.</p>
61	Add different pages for different types of users such as advanced users who want additional functionality.	Access additional functionality by going onto the advanced settings page for advanced users and access custom hyperparameter values and additional custom settings that the user can change.	Advanced user's also want to have access to this tool therefore it is expected for users to also have quick access to advanced functionality such as selecting hyperparameter values.	As the tool is able to cater to a wide audience as shown by the diverse range of stakeholders and therefore end-users the solution contains an advanced settings page which allows for unlocking additional functionality with the program such as creating custom models which user tuned hyperparameters insteads of the solution automatically fine-tuning models during the training process.
62	Automatically validate all the data that is passed into the model. Through using validation checkpoints at each stage when data is exchanged.	Input data that do not conform to the systems requirements, verify that the system automatically validates that data is correct.	As data analysis is a key component of our solution data exchange needs to be validated by setting clear checkpoints to validate and correct data as it traverses the entire system.	The solution successfully validated data at multiple stages within the program. When data was either recoverable or unable the system additionally took measures to further process data automatically without errors occurring.

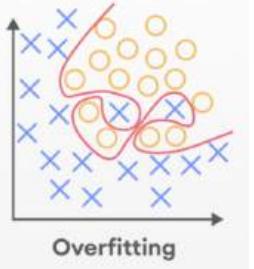
Robustness Testing:

Within each component of the program we are testing the following categories of input data.

Valid	Input data that falls within the acceptable range and is an expected input.
Boundary	Input data that falls at the edge of the acceptable boundaries.
Extreme	Input data that is valid, but is above the typical range.
Invalid	Input data that is outside the acceptable range and should be rejected.

Additionally as part of testing the robustness of the entire system and individual components we have implemented stress testing to test the overall ability of the program to handle conditions that are outside the normal operations. This allows us to identify system limits and bugs that occur if the program is used at an extreme level.

No	Component Tested	Type Of Test	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
1	Extremely low hyperparameter values	Boundary	Input extremely low hyperparameter values to train the model with this is at the lower bound of acceptable values for hyperparameters Gamma and C. Gamma = 0.001 and C=1	The model should successfully train and produce a reasonably accurate model that can generate predictions.	<p>The model was able to produce predictions. However the low values for hyperparameters meant that the model wasn't able capture patterns effectively. The model was underfitting the training data.</p>  <p style="text-align: center;">Underfitting</p>
2	Extremely high hyperparameter values	Extreme	Input extremely low hyperparameter values to train the model with this is at the lower bound of acceptable values for	The model should successfully train and produce a reasonably	The model was able to produce predictions. However the high values for hyperparameters meant that the model

No	Component Tested	Type Of Test	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
			hyperparameters Gamma and C. Gamma = 1000 and C=1000.	accurate model that can generate predictions.	wasn't took longer to capture patterns effectively. The model was overfitting the training data. 
3	Testing invalid hyperparameter values on the advanced settings page	Invalid	Input hyperparameter values that are invalid and can't be used to train the model. In this case we tested setting Gamma and C to negative numbers and non-numeric characters.	The system should validate all hyperparameter values after they are submitted and reject any that are not of the correct format.	The system was able to validate that these values don't meet the system's requirements and reject them as invalid values.
4	Generate predictions of the future market movements of a range of agricultural commodities.	Valid	Use a trained SVM model along with forecasted weather data to generate multiple predictions as to the future price of multiple different agricultural commodities.	Each crop should have a prediction produced for the dates that were included as part of the forecasted weather data.	Predictions are generated successfully and are consistent when the model classifies the same data again.
5	Test the system's ability to capture invalid coordinates of the incorrect format.	Invalid	The format of the coordinates is the WGS84 system. This means that coordinates outside of this format that are invalid and follow other standards should be rejected.	The system should be able to validate coordinates before they are used and ensure they are of the WGS84 system's format.	The system should indicate that coordinates are of the incorrect format and are invalid and prompt the user to use the correct format.

No	Component Tested	Type Of Test	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
6	Data retrieval at the boundary of acceptable dates.	Boundary	The external API should be able to request data at the edge of acceptable values for dates to retrieve data from.	External APIs such as openMeteo should have records that extend to their entire range so data should be effectively retrieved.	The API was able to successfully implement data retrieval for the dates without needing to generate an error or retrieve an empty dataset.
7	Attempting to retrieve data from invalid dates	Invalid	Test that the API or system is able to handle when invalid dates that the external API does not contain data for is entered.	External APIs such as openMeteo only store data within the necessary range therefore a suitable error should be produced indicating that the system does not contain the data.	The solution was able to recognise that the date is of an incorrect format or value and produced an error indicating that no data could be successfully retrieved.
8	Test that the web scraping tool is able to handle a website with a changing UI and internal HTML structure.	Extreme	Apply the web scraping algorithm that uses beautifulsoup4 to retrieve data on the same website, but of different architectures. Such as a different layout and source code HTML.	Commodity prices are retrieved from external sources using a web scraping tool which parses the entire HTML and finds the data. Changes in the HTML framework of a web page should not hinder the retrieval operations.	As the actual elements and data the website contained should remain consistent the program should still be able to successfully retrieve data from it.

No	Component Tested	Type Of Test	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
9	Data retrieval of historical weather data across large time frames.	Extreme	Retrieve weather data for multiple regions across a large time range such as 50 years.	The solution should be able to both request and retrieve a large dataset without crashing.	Weather datasets are extremely large as they contain a large number of datapoints over a large time range. However the system was able to successfully retrieve the entire historical weather dataset.
10	Retrieve weather data from the external API.	Valid	Request historical and current weather data for a region with valid coordinates.	The system should be able to retrieve and store the data appropriately.	The data was able to be quickly retrieved from the external weather API openmeteo and provided to the user.
11	Test that data is able to be imputed properly even if there is only a limited amount of initial data points.	Boundary	Provide weather data with only a limited amount of initial data points and validate that the missing or anomalous value is able to be correctly substituted by the program.	As the program is resilient to datasets that contain missing or anomalous values the program should be able to detect and fix this error.	The system was able to both detect the anomalous or missing value and substitute it with a valid value making the entire dataset more accurate of the actual conditions.
12	The volume spread across multiple regions are incomplete.	Invalid	The volume spread across multiple regions does not add up to 100% therefore may not be representative of the entire market.	If any arithmetic errors occur the error should be caught and suitably handled advising the user that the operation that couldn't be completed.	Arithmetic errors were caught and an error was displayed to the user indicating that the operation couldn't be completed. If the operation did complete then no errors were indicated to the user.
13	Anomalous values within	Extreme	Provide a dataset with anomalous outliers	The program should be able	Outliers are detected if they are outside the

No	Component Tested	Type Of Test	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
	the dataset that are outliers within the normal range are automatically flagged.		within it and verify that the system is able to flag them to the user.	to detect and flag the outliers.	interquartile range by a large degree. This means that the sensitivity of flagging outliers can be changed.
14	Anomalous or missing values are able to be imputed with suitable substitutions.	Extreme	Provide a dataset with both anomalous and missing values and validate that they are able to be successfully imputed with a suitable value.	Imputing values involves replacing the erroneous value with the average of that entire column so the dataset is both complete and representative of the original data.	The system was able to impute the values effectively without generating any errors in the process.
15	Enter invalid hyperparameter values as part of the advanced settings page	Invalid	Input invalid hyperparameter values for gamma and c as part of the advanced settings page such as negative numbers or non-numeric characters.	The system should be able to identify that the submitted data is of the wrong format and display an error indicating this.	The system displayed that the input that was submitted was invalid and specified the correct format to structure the user's response.
16	Labelling of commodity prices to train the Support Vector Machine on.	Valid	Input a dataset of commodity prices and validate that data is successfully classified and an appropriate label is defined for it.	The dataset should be both successfully labeled and classified.	The entire commodity prices dataset was appropriately labelled.
17	Perform an exhaustive grid search cross validation to find the optimal hyperparameters	Stress	A large amount of hyperparameters are provided and the system will have to evaluate the most efficient combination	The grid search cross validation should take longer as there are a larger amount of	The grid search cross validation process took longer however it successfully outputted the most optimal hyperparameter

No	Component Tested	Type Of Test	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
	rs over a large grid of values.		from a large range of values.	values and an optimal value should be outputted.	values for gamma, c and the kernel after testing all the combinations of these values as a grid.
18	Test the ability for the system to handle constant internet disruptions while data retrieval operations are occurring.	Stress	Simulate internet disconnection issues during data retrieval tasks of the weather data and the commodity prices.	The system should be able to either try to reconnect to the original system and retransmit data or inform the user.	Through the implementation of the retry-cache module when doing network related data retrieval the system was able to constantly retry reconnecting with the original data source such as an external API.
19	Train the Support-Vector Machine over a large dataset.	Stress	Provide a large dataset with multiple rows and columns for training on the SVM.	The system should train the model on all the available data points without crashing.	The model training took longer than the normal average training time however successfully was trained on the entire dataset without crashing.
20	Simulate 500 users concurrently interacting with our solution producing predictions.	Stress	Simulate 500 multiple independent users accessing and using the system at the same time.	The system should be able to handle multiple users using the program at the same time simultaneously .	The system successfully handled multiple separate simulated users using the program at the same time without crashing.
21	Data retrieval component stress testing.	Stress	Simulate 1000 independent requests for historical weather data for the API.	The external system should be able to handle the high volume of requests and respond to them quickly.	The API and the individual user's system was able to handle a large number of simultaneous requests however a large volume of requests may hit the API's limit causing

No	Component Tested	Type Of Test	Input Data / Action	Expected Outcome	Actual Outcome + Evidence
					data retrieval to take longer.
22	User Interface switching at extremely fast speeds.	Stress	The user should rapidly switch between pages within the solution.	The system should maintain its responsiveness and handle rapid page switching without the entire program crashing.	The system was able to handle rapid switching without any issues arising. Additionally processing pages allowed the user to prevent switching occurring while data is being processed which made the system robust.

The solution as detailed by the success criteria met the original stakeholder's requirements extremely well.

Interviewing stakeholders they said their key strengths were the functionality of the system as it was able to provide consistent short-term predictions accurately on the agricultural commodity market. The user experience was extremely clear as all skill levels of users could access and use it without needing external help and support. Additionally the program was adaptable as it was able to predict not just a single agricultural commodity, but multiple with high levels of accuracy.

Potential shortfalls of the solution may be that there isn't extensive transparency explaining the processes in the background as they are extremely complex so end-users may only have a surface level understanding as to how the solution works as an entire system. Additionally the program has an extensive reliance on external data sources such as for commodity prices and climate data. If these sources were either made redundant or the format of their output was changed the data retrieval components would have to be adjusted. This is mentioned additionally in the maintenance section of the evaluation of the entire program.

Usability Features and Usability Testing

Evidence of testing can be seen in the following videos where we go through the initial prototype which we built at the beginning of the development cycle and then the final testing of the completed solution that meets all the stakeholder's criteria.

Video Testing Information:

The attached video contains the two main components of the program and evidence of it being tested. The initial prototype testing and also the entire final solution running. The initial prototype testing shows initially what the program was and after stakeholder feedback and iterations as detailed within the development section was iterated into the final solution. In this video the entire final solution video is shown to be running with a run through as to the main features that stakeholders required. In this case the final solution testing shows how to produce a prediction for multiple crops and how an average end-user would use the product. This means that components and features that the user is likely to use are included

Maintenance

Future Maintenance and Potential Future Limitations of my solution:

Our tool is very fluid in the sense that it will work effectively without the need for regular maintenance. As we are using both an API to retrieve the weather data and extracting the historical and current commodity prices from an external website. The solution is able to synchronise all of the data and verify that it always has correct and contains the most up-to-date values.

This synchronisation is done by retrieving the current date so the program at all times is able to understand the range of data it has and missing values. This means that as the commodity prices and weather data is updated on the respective sources our tool will also update them so there isn't a need to effectively maintain our solution.

The main issue regarding maintenance is the changing interfaces for an API, commodity prices website and the modules. Discussing the API and commodity prices if there are drastic changes to the way data is produced in response to the same response our solution may not be able to handle it. This is because of the dependency to the previously unchanged format. For instance if the weather and commodity price data is always retrieved as panda's dataframes then either source is updates to produce data in another format such as an excel(.XLS / .XLSX) file type

as we have built the entire program as to handle only the same consistent panda's dataframe format we will not be able to access this. In this case we have used validation techniques within our program to catch a data dimensional error when it occurs. If a file that is loaded into the program is of the wrong file type or shape then the program immediately catches the error before progress begins and displays it to the user appropriately. This means that in the future if there is to be drastic changes within the file type of the retrieved data then the user can easily understand this and directly update the accessible python source code to convert data in the correct format.

Implementing modules has meant that we can implement external features within our program such as the ability to use a graphical user interface through tkinter. Issues we faced while using tkinter was its GPU utilisation for rendering elements which runs in parallel with the program being run both on the CPU. This meant that there needed to be necessary synchronisation between both systems so on extremely-powerful laptops this might be an issue. Modules are constantly updated so issues like the one above might occur in a range of situations. The changing nature of modules will also have to be taken into account as within our program we received this error.

```
FutureWarning: Downcasting object dtype arrays on .fillna,  
.ffill, .bfill is deprecated and will change in a future version.  
Call result.infer_objects(copy=False) instead. To opt-in to the  
future behavior, set  
`pd.set_option('future.no_silent_downcasting', True)`  
commodityPricesDataframe['Percentage Change'] =  
commodityPricesDataframe.iloc[:, 1].pct_change() * 100
```

This was in respect to a function that calculates percentage change. As the processes are constantly updated and changed. Using this specific pandas module when the next version is released and is widely adopted our program will be unable to run. Therefore there is also a necessity to keep the version of a module we are accessing and using constant. This is possible with modules that can be fully downloaded such as numpy and pandas as they don't need to access live information for the internet for these tasks. Instead they contain a collection of static code that can be used and downloaded offline instead of using the more recent one we can use our saved historical backup. However as said before when we require data that needs to be up-to-date such as the mentioned commodity prices and forecasted and historical weather data it is necessary to conform to these new standards and build the software again around them.

Post development issues may arise if in the future agricultural markets become redundant to climate impacts as technology improves and massive investment goes into alternatives such as indoor vertical farming, intensive farming techniques and use of chemical fertilisers, insecticides and pesticides. The agricultural market may get hardened to climate impacts. In that case the suitability of a system that uses climate data to predict agricultural commodities may have limited use in that sector. Other applications of our tool therefore could involve finding correlation between the climate and other data points. Stakeholders suggested a disease prevention tool as there is a clear link between malaria carrying female mosquitos in high prevalence in times of high rainfall and high temperature. The same principles apply by finding correlation between the weather and another factor. Using an input(weather) and an output(the outcome e.g crop yield , malaria deaths etc) to build a trend to justify assumptions and help make informed decisions whatever the challenge may be.

Final Evaluation

Cocoa beats Bitcoin to emerge as top performer of 2024, but leaves a bitter aftertaste

The soaring cost of cocoa — with prices reaching a record high of \$12,500 per tonne, a staggering 180% gain — has left a bitter aftertaste for the chocolate industry. Leading brands are grappling with rising input costs and are increasingly passing the burden to consumers.

Recently in the news the extreme volatility of agricultural commodities was highlighted as it showed the extreme amount of fluctuations that these crops occur. In 2024 alone the price of cocoa increased by over 180% in comparison to the cryptocurrency bitcoin which increased by over 160%. This moreover shows the real-world implications of this tool. Stakeholders such as Sharath were impressed by the program's ability to make predictions accurately and extremely quickly.

In the future we can try to use more variables in addition to the current ones to make more accurate predictions. This involves using more powerful hardware resources such as utilising external processing in the cloud allowing us to process more data and in more detail adding to the overall success of our solution. Stakeholders highlighted that this tool provided extreme detail outside of the prediction solutions shown at the research stage of this project. This included vesper which in comparison with the final solution isn't able to match the ability to produce custom models using advanced settings which many advanced users want access to. Future releases may focus on continued support and maintenance of the solution along with scalability of the program to accommodate more data and effectively increase the accuracy of the entire program. Additionally future versions

could also implement more educational resources to comprehensively show how data is being processed more transparently maximising the understanding of users using the tool. Stakeholders also enjoyed the extensive cross platform nature of the solution as it was able to run on multiple devices effectively without the need for different separate interfaces which enhanced the user experience and overall functionality.

Ultimately the continuous improvement over multiple iterations meant that the entire solution met the stakeholder's requirements and were able to effectively predict the price of agricultural commodities using climate patterns.

Appendix

Code Listings

```
#We install pandas to store the dataframes. Dataframes are used to store our weather and commodity price datasets.
```

```
import pandas as pd  
import numpy as np
```

```
#CSV is imported as we are also working with .CSV files and pandas dataframes interchangeably.
```

```
import csv
```

```
#Sklearn is an open-source machine learning library in python and is what we use within our program to implement an SVM.
```

```
from sklearn.svm import SVC
```

```
#We import a scale module from sklearn to make the weather(features) dataset have a mean of 0 and a standard deviation of 1.
```

```
from sklearn.preprocessing import scale
```

```
#Grid-search cross validation module is also used to do the grid search cross validation if our program doesn't do it manually.
```

```
from sklearn.model_selection import GridSearchCV
```

```
#Datetime is used to handle dates, they are all in the form ISO8601 so can be easily accessed and understood.
```

```
from datetime import datetime, timedelta
```

```

#These are modules required by openMeteo as we are accessing an external
database through an API.
import openmeteo_requests
#These two modules are used to retry connecting to an API if the user
disconnects from the server.
import requests_cache
from retry_requests import retry

#Web scraping modules are included here beautifulsoup4 is used to retrieve the
HTML code and requests to access elements within a website.
from bs4 import BeautifulSoup
import requests

#Once we have the link to an external dataset we will use os to download it.
import os

#When we implemented a loading screen we used subprocess to run multiple
threads of code at once.
import subprocess

#Abstract syntax tree used for the loading screen.
import ast

import time#Allows us to sequence and synchronise our functions
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox

import matplotlib.pyplot as plt

from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
NavigationToolbar2Tk)

#Added support for advanced error handling to filter through and catch errors
before they are shown to the user.
import warnings
import logging

global worldBank_url
worldBank_url = "https://www.worldbank.org/en/research/commodity-markets#1"

global cropIndex # This is a global crop index with all the possible crops we have
available.
cropIndex = ['cocoa','soybean','corn','wheat','sugar']

#Suppresses logs about the systems CPU Architecture. e.g Apple Silicon(RISC)
logging.getLogger('IMK').setLevel(logging.WARNING)

```

```

# This is an error alert function and will be what is called when an error is caught
within the program.
#This function allows the program to make decisions when errors occur.
def errorAlert(error,type):
    print('error encountered')
    print(error)

    #The type of the error defines the next steps to be taken after the error has
been detected.
    #In this case there are two types: a soft and a hard reset.
    #A soft reset will return back to the start page, whereas a hard reset will initialize
the entire program.
    if type == 'soft reset':
        print('soft reset - returning to start page')
    else:
        type == 'hard reset'
        print('hard reset - resetting entire program')

# %% [markdown]
# Retrieve Historical Weather Data

# %%
# crop + Coordinates indicate the top three major producers of that crop around
the world.errorAlert
# These three major producers are represented as three distinct coordinates with
a latitude and a longitude component
#
#The volume spread indicates what percentage of the produce that each location
produces.

cocoaCoordinates = [[6.053110308791517, -5.632365625597256],
                    [5.723258880120529, -2.0289499687551404],
                    [-1.4526661026550038, -80.30948687187554]]
cocoaVolumeSpread = [0.660, 0.187, 0.153]

soybeanCoordinates = [[-19.64619384124217, -54.26514130936858],
                      [42.29404827977364, -92.81193554839746],
                      [-35.06747636366788, -58.071092994544266]]
soybeanVolumeSpread = [0.487, 0.360, 0.153]

sugarCoordinates = [[-22.591058675268453, -48.380706513546365],
                    [27.512949904632368, 80.63046034857008],
                    [16.17804967961032, 103.56698522101857]]
sugarVolumeSpread = [0.471, 0.404, 0.125]

wheatCoordinates = [[33.91053065183956, 113.67123697952496],
                    [47.904101854372406, 2.055730082984269],

```

```

[27.166528613792373, 80.59817535483508]]
wheatVolumeSpread = [0.357, 0.353, 0.290]

cornCoordinates = [[42.685090443052125, -93.32174099559683],
                  [40.954928136698115, 117.03116790283333],
                  [-11.258015767551417, -54.79968906997016]]
cornVolumeSpread = [0.474, 0.377, 0.149]

def verifyCoordinateValidity(latitude,longitude):#Returns validity of the
coordinates
    try:
        if -90 < latitude < 90:
            validLatitude = True
        else:
            validLatitude = False #Invalid latitude. The value is not between -90 and 90
    except:
        validLatitude = False #Invalid input. The value is invalid as it is not a numerical
value.

    try:
        if -180 < longitude < 180:
            validLongitude = True
        else:
            validLongitude = False #Invalid latitude. The value is not between -180 and
180
    except:
        validLongitude = False #Invalid input. The value is invalid as it is not a numerical
value.

    if validLatitude == True and validLongitude == True:#The entire program will
return either a valid / invalid statement that corresponds to the coordinate
components.
        return 'valid values'
    else:
        return 'invalid values'

def openMeteo_historical_climate(latitude, longitude):

    #This is a validation technique used to retry the connection constantly if the
connection were to disconnect for any reason.
    cache_session = requests_cache.CachedSession('.cache', expire_after = 3600)
    retry_session = retry(cache_session, retries = 5, backoff_factor = 0.2)
    openmeteo = openmeteo_requests.Client(session = retry_session)

    url = "https://climate-api.open-meteo.com/v1/climate"#This is the url we use to
directly access openMeteo's weather API.

```

```

#Parameters define data we input into the historical weather API.
params = {
    #The location (coordinates) are expressed with the latitude and the longitude
    #coordinates.
    "latitude": latitude,
    "longitude": longitude,

    #The start and end-date have been set to predetermined values.

    "start_date": "1960-01-01",
    "end_date": "2023-11-01",
    "models": "MRI_AGC3_2_S",
    "daily": ["temperature_2m_mean", "cloud_cover_mean",
    "relative_humidity_2m_mean", "precipitation_sum"]
}

responses = openmeteo.weather_api(url, params=params)#We pass the
parameters that we defined straight into and through the API.

response = responses[0] # Process first location. Add a for-loop for multiple
locations or weather models which we can do using multiple parameter tables.
#Our implementation processes locations one at a time however all the data can
theoretically be retrieved at once.
#This is so we can clearly distinguish different pieces of data which would be
difficult if it was all stored within a single dataframe.

#These are identifiers which highlight the location and timezone of where the
crop is grown.
#These identifiers can be used to debug the program to find out what data is
included within the dataframe.
debugCoordinates = (f"Coordinates {response.Latitude()}°N
{response.Longitude()}°E")
debugElevation = (f"Elevation {response.Elevation()} m asl")
debugTimezone = (f"Timezone {response.Timezone()}
{response.TimezoneAbbreviation()}")
debugTimezoneDifference = (f"Timezone difference to GMT+0
{response.UtcOffsetSeconds()} s")

# Process data in a daily format. By doing so we can extract the four metrics we
are looking for.
#These metrics are temperature, cloud_cover, humidity and precipitation
daily = response.Daily()
daily_temperature_2m_mean = daily.Variables(0).ValuesAsNumpy()
daily_cloud_cover_mean = daily.Variables(1).ValuesAsNumpy()
daily_relative_humidity_2m_mean = daily.Variables(2).ValuesAsNumpy()
daily_precipitation_sum = daily.Variables(3).ValuesAsNumpy()

```

```

#We need to generate a sequence of dates. To do this we repeat an interval
between each date.
daily_data = {"date": pd.date_range(
    start = pd.to_datetime(daily.Time(), unit = "s", utc = True),
    end = pd.to_datetime(daily.TimeEnd(), unit = "s", utc = True),
    freq = pd.Timedelta(seconds = daily.Interval()),
    inclusive = "left"
)}


#These populate the dataframe with the necessary dates with the four metrics
we just retrieved from the API.
daily_data["temperature_2m_mean"] = daily_temperature_2m_mean
daily_data["cloud_cover_mean"] = daily_cloud_cover_mean
daily_data["relative_humidity_2m_mean"] = daily_relative_humidity_2m_mean
daily_data["precipitation_sum"] = daily_precipitation_sum

daily_dataframe = pd.DataFrame(data = daily_data)

#We return the entire historical weather dataset as a single panda's dataframe.
return daily_dataframe


def retrieve_historical_climate(crop):#This entire function can retrieve all the
historical weather data for a crop and accordingly merge all its datasets.
    global historicalWeather1, historicalWeather2, historicalWeather3 #We make
these dataframes have a global scope so they can be accessed across the entire
program.

    #We use a for loop to loop through all three crop locations and extract the
location and the respective coordinates.
    #This process occurs three times as we need to get the 1st, 2nd and 3rd
historical weather dataset. This is because there are three major locations where
each crop is grown.
    for i in range (3):
        latitude, longitude = globals()[crop + 'Coordinates'][i] #Using globals() instead
of using multiple if statements for each crop we can write efficient code. This
means that we can directly call the crop's coordinates.
        coordinateValidity = verifyCoordinateValidity(latitude, longitude)#We use
validation to verify that the coordinates are correct and valid.

        if coordinateValidity == 'valid values':#If the specific coordinates are valid we
can generate each historical weather dataframe one at a time.
            globals()['historicalWeather' + str(i + 1)] =
openMeteo_historical_climate(latitude, longitude) #Generates 3 different
dataframes, one for each country

        #If data is incorrect we catch it before the error is allowed to propagate and
call the error function.
        elif coordinateValidity == 'invalid values':

```

```

        errorAlert('Crop information database contains invalid longitude and latitude
coordinates.','hard reset')
    else:
        errorAlert('Coordinates could not be validated. No data has been requested
from the API OpenMeteo.','hard reset')

    historicalWeather1.to_csv('historicalWeather1.csv', sep=',', index=False,
encoding='utf-8') #Saves the dataset as a .csv file. This allows for physical
debugging as we have can see the .csv file in human read-able format as a csv is
tabular.
    historicalWeather2.to_csv('historicalWeather2.csv', sep=',', index=False,
encoding='utf-8') #.csv files and dataframes are interchangeable so it is efficient
to save all weather datasets as CSVs.
    historicalWeather3.to_csv('historicalWeather3.csv', sep=',', index=False,
encoding='utf-8')

#Converts .csv back into a pandas dataframe. This uses the utf-8 encoding.
historicalWeather1 = pd.read_csv("historicalWeather1.csv", header=0)
historicalWeather2 = pd.read_csv("historicalWeather2.csv", header=0)
historicalWeather3 = pd.read_csv("historicalWeather3.csv", header=0)

#This is where we manually multiply all the datasets with the respective
percentage volume which that location produces.
#We multiply every column within the dataset by the respective volume spread
that location produces.
#
#This is possible due to the consistent indexing and variable naming scheme
within our program as the index of the coordinate is equal to the index of the
percentage spread.
#
historicalWeather1["temperature_2m_mean"] =
(historicalWeather1["temperature_2m_mean"] * globals()[crop +
'VolumeSpread'][0])
historicalWeather1["cloud_cover_mean"] =
(historicalWeather1["cloud_cover_mean"] * globals()[crop + 'VolumeSpread'][0])
historicalWeather1["relative_humidity_2m_mean"] =
(historicalWeather1["relative_humidity_2m_mean"] * globals()[crop +
'VolumeSpread'][0])
historicalWeather1["precipitation_sum"] =
(historicalWeather1["precipitation_sum"] * globals()[crop + 'VolumeSpread'][0])

historicalWeather2["temperature_2m_mean"] =
(historicalWeather2["temperature_2m_mean"] * globals()[crop +
'VolumeSpread'][1])
historicalWeather2["cloud_cover_mean"] =
(historicalWeather2["cloud_cover_mean"] * globals()[crop + 'VolumeSpread'][1])

```

```

historicalWeather2["relative_humidity_2m_mean"] =
(historicalWeather2["relative_humidity_2m_mean"] * globals()[crop +
'VolumeSpread'][1])
historicalWeather2["precipitation_sum"] =
(historicalWeather2["precipitation_sum"] * globals()[crop + 'VolumeSpread'][1])

historicalWeather3["temperature_2m_mean"] =
(historicalWeather3["temperature_2m_mean"] * globals()[crop +
'VolumeSpread'][2])
historicalWeather3["cloud_cover_mean"] =
(historicalWeather3["cloud_cover_mean"] * globals()[crop + 'VolumeSpread'][2])
historicalWeather3["relative_humidity_2m_mean"] =
(historicalWeather3["relative_humidity_2m_mean"] * globals()[crop +
'VolumeSpread'][2])
historicalWeather3["precipitation_sum"] =
(historicalWeather3["precipitation_sum"] * globals()[crop + 'VolumeSpread'][2])

#As all the historical weather dataframes have been processed we can just add
them using an additional function which adds the respective numbers.
combined_dataframe = historicalWeather1 + historicalWeather2 +
historicalWeather3

#The date which is a string has also been concatenated so we need to fix that.
combined_dataframe["date"] = historicalWeather1["date"]

#Let's format the data into yearly averages as we have too many daily weather
datapoints.
combined_dataframe["date"] = pd.to_datetime(combined_dataframe["date"])# Convert "date" to datetime type
combined_dataframe["year"] = combined_dataframe["date"].dt.year# Extract
the year

combined_dataframe_yearly_averages =
combined_dataframe.groupby('year').mean()# Calculate the yearly averages of
each column using a dataframe manipulation function.
combined_dataframe_yearly_averages =
combined_dataframe_yearly_averages.reset_index() #Resets the index of
combined_dataframe_yearly_averages

#Define the title labels of our new dataframe.
combined_dataframe_yearly_averages =
combined_dataframe_yearly_averages[['year','temperature_2m_mean',
'cloud_cover_mean', 'relative_humidity_2m_mean', 'precipitation_sum']]

#We save this combines dataframe to a csv so we are able to call it within all
components of the program as it is now a global file.

```

```

combined_dataframe_yearly_averages.to_csv('combined_historicalWeather.csv',
sep=',', index=False, encoding='utf-8')

#We can return the dataframe as a single dataframe that contains all the
combined historical weather values that have been processed to be in yearly time
segments.
return combined_dataframe_yearly_averages

# %% [markdown]
# Retrieve Forecasted Weather Data

# %%
def openMeteo_forecasted_climate(latitude, longitude, startDate):
    #We only input the start date as our program will always use the longest
    #possible forecast length automatically.
    #The longest possible forecast length is 16 days after the current date.

    # Setup the Open-Meteo API client with cache and retry on error.
    cache_session = requests_cache.CachedSession('.cache', expire_after = 3600)
    retry_session = retry(cache_session, retries = 5, backoff_factor = 0.2)
    openmeteo = openmeteo_requests.Client(session = retry_session)

    #openMeteo has a maximum forecast 16 days after the start date. This is 15 days
    #as we are including today's date as it is a 0 indexed counting system.

    date_placeholder = datetime.strptime(startDate, '%Y-%m-%d')# Parse the
    #ISO8601 startDate to extract what the day is.
    endDate = date_placeholder + timedelta(days = 15) #Add 15 days to the current
    #date
    endDate = endDate.strftime('%Y-%m-%d')# Format the result back to ISO 8601
    #and set that as the endDate of the weather forecast.

    #all required weather variables to retrieve are listed here
    url = "https://api.open-meteo.com/v1/forecast"
    params = {
        #The coordinate components
        "latitude": latitude,
        "longitude": longitude,
    }

    #We want data in the highest quality possible so we will retrieve data in hourly
    #steps.
    #Hourly steps means that there will be more data points per day however we
    #only have a maximum of 16 days so our system will be able to handle it.
    #Name the four metrics we want predictions for.

```

```

"hourly": ["temperature_2m", "cloud_cover", "relative_humidity_2m",
"precipitation", ],

#Define the start and the end data.
"start_date": startDate,
"end_date": endDate
}

#Pass the defined parameters into the forecasting openMeteo API.and save the
responses.
responses = openmeteo.weather_api(url, params=params)

# Process first location. Add a for-loop for multiple locations or weather models.
#We process all the responses sequentially so it is clear as to what location is
related to what response.
response = responses[0]

#We save the identifiers for the program in case errors occur and we need to
debug exactly what has happened and where forecasted weather data has been
retrieved for.
debugCoordinates = (f"Coordinates {response.Latitude()}°N
{response.Longitude()}°E")
debugElevation = (f"Elevation {response.Elevation()} m asl")
debugTimezone = (f"Timezone {response.Timezone()}"
{response.TimezoneAbbreviation()}")
debugTimezoneDifference = (f"Timezone difference to GMT+0
{response.UtcOffsetSeconds()} s")

# Process hourly data. The order of variables needs to be the same as requested.
hourly = response.Hourly()
hourly_temperature_2m = hourly.Variables[0].ValuesAsNumpy()
hourly_relative_humidity_2m = hourly.Variables[1].ValuesAsNumpy()
hourly_precipitation = hourly.Variables[2].ValuesAsNumpy()
hourly_cloud_cover = hourly.Variables[3].ValuesAsNumpy()

#Create an hourly time column for the data.
hourly_data = {"date": pd.date_range(
    start = pd.to_datetime(hourly.Time(), unit = "s", utc = True),
    end = pd.to_datetime(hourly.TimeEnd(), unit = "s", utc = True),
    freq = pd.Timedelta(seconds = hourly.Interval()),
    inclusive = "left"
) }

#Save each component that has been retrieved from the forecasted weather API
into a single hourly data dataset.
hourly_data["temperature_2m"] = hourly_temperature_2m
hourly_data["cloud_cover"] = hourly_cloud_cover
hourly_data["relative_humidity_2m"] = hourly_relative_humidity_2m

```

```

hourly_data["precipitation"] = hourly_precipitation

hourly_data = pd.DataFrame(data = hourly_data) # Convert dataset into a
panda's dataframe

return hourly_data #Return the hourly dataframe.

def retrieve_forecasted_climate(crop): #This entire function can retrieve all the
forecasted weather data for a crop and accordingly merge all its datasets.
    global forecastedWeather1, forecastedWeather2, forecastedWeather3
    #We make these dataframes have a global scope so they can be accessed
across the entire program.

    currentDate = datetime.today()#Retrieve the current data in the correct ISO8601
format for the API.
    currentDate = currentDate.strftime('%Y-%m-%d')

    #We use a for loop to loop through all three crop locations and extract the
location and the respective coordinates.
    #This process occurs three times as we need to get the 1st, 2nd and 3rd
forecasted weather dataset. This is because there are three major locations where
each crop is grown.
    for i in range (3):
        latitude, longitude = globals()[crop + 'Coordinates'][i]#Using globals() instead
of using multiple if statements for each crop we can write efficient code. This
means that we can directly call the crop's coordinates.
        coordinateValidity = verifyCoordinateValidity(latitude, longitude)#We use
validation to verify that the coordinates are correct and valid.

        #If the specific coordinates are valid we can generate each forecasted
weather dataframe one at a time.
        if coordinateValidity == 'valid values':
            globals()['forecastedWeather' + str(i + 1)] =
openMeteo_forecasted_climate(latitude, longitude, currentDate) #Generates 3
different dataframes, one for each country
        elif coordinateValidity == 'invalid values':
            errorAlert('Crop information database contains invalid longitude and latitude
coordinates.', 'hard reset')
        else:
            errorAlert('Coordinates could not be validated. No data has been requested
from the API OpenMeteo.', 'hard reset')

        forecastedWeather1.to_csv('forecastedWeather1.csv', sep=',', index=False,
encoding='utf-8') #Saves the dataset as a .csv file. From numpy to pandas
dataframe.
        forecastedWeather2.to_csv('forecastedWeather2.csv', sep=',', index=False,
encoding='utf-8')

```

```

forecastedWeather3.to_csv('forecastedWeather3.csv', sep=',', index=False,
encoding='utf-8')

#Saves the dataset as a .csv file. This allows for physical debugging as we have
can see the .csv file in human readable format as a csv is tabular.

#.csv files and dataframes are interchangeable so it is efficient to save all
weather datasets as CSVs. This also means that it is globally accessible across the
entire program.

forecastedWeather1 = pd.read_csv('forecastedWeather1.csv', header=0)
forecastedWeather2 = pd.read_csv('forecastedWeather2.csv', header=0)
forecastedWeather3 = pd.read_csv('forecastedWeather3.csv', header=0)

#This is where we manually multiply all the datasets with the respective
percentage volume which that location produces.

#We multiple every column within the dataset by the respective volume spread
that location produces.

#
#This is possible due to the consistent indexing and variable naming scheme
within our program as the index of the coordinate is equal to the index of the
percentage spread.

#
forecastedWeather1["temperature_2m"] =
(forecastedWeather1["temperature_2m"] * globals()[crop + 'VolumeSpread'][0])
forecastedWeather1["cloud_cover"] = (forecastedWeather1["cloud_cover"] *
globals()[crop + 'VolumeSpread'][0])
forecastedWeather1["relative_humidity_2m"] =
(forecastedWeather1["relative_humidity_2m"] * globals()[crop +
'VolumeSpread'][0])
forecastedWeather1["precipitation"] = (forecastedWeather1["precipitation"] *
globals()[crop + 'VolumeSpread'][0])

forecastedWeather2["temperature_2m"] =
(forecastedWeather2["temperature_2m"] * globals()[crop + 'VolumeSpread'][1])
forecastedWeather2["cloud_cover"] = (forecastedWeather2["cloud_cover"] *
globals()[crop + 'VolumeSpread'][1])
forecastedWeather2["relative_humidity_2m"] =
(forecastedWeather2["relative_humidity_2m"] * globals()[crop +
'VolumeSpread'][1])
forecastedWeather2["precipitation"] = (forecastedWeather2["precipitation"] *
globals()[crop + 'VolumeSpread'][1])

forecastedWeather3["temperature_2m"] =
(forecastedWeather3["temperature_2m"] * globals()[crop + 'VolumeSpread'][2])
forecastedWeather3["cloud_cover"] = (forecastedWeather3["cloud_cover"] *
globals()[crop + 'VolumeSpread'][2])

```

```

forecastedWeather3["relative_humidity_2m"] =
(forecastedWeather3["relative_humidity_2m"] * globals()[crop +
'VolumeSpread'][2])
forecastedWeather3["precipitation"] = (forecastedWeather3["precipitation"] *
globals()[crop + 'VolumeSpread'][2])

#As all the historical weather dataframes have been processed we can just add
them using an addition function which adds the respective numbers.
combined_dataframe = forecastedWeather1 + forecastedWeather2 +
forecastedWeather3

#As given by openMeteo's documentation future forecasts may contain missing
data.(empty values)
#Instead of physically removing datapoints we can just extract complete
datapoints and equate that to the original dataset.
#This effectively deletes empty datapoints as they aren't included within the
new dataframe.
combined_dataframe =
combined_dataframe.loc[(combined_dataframe['temperature_2m'] != None) &
(combined_dataframe['cloud_cover'] != None) &
(combined_dataframe['relative_humidity_2m'] != None) &
(combined_dataframe['precipitation'] != None)]

combined_dataframe['date'] = pd.to_datetime(combined_dataframe['date'],
format='mixed') # Ensure 'date' is datetime type

#Defines the split of the data into four time segments. Each of which has a
length of 4 days.
combined_dataframe['time_segment'] = pd.cut(combined_dataframe['date'],
bins=4, labels=['Days 1-4', 'Days 5-8', 'Days 9-12', 'Days 13-16'])

# Calculate mean values for each grouped time segment. This means that we
have four different datapoints instead of a single datapoint.
combined_dataframe = combined_dataframe.groupby('time_segment',
observed=False).mean()

combined_dataframe.to_csv('combined_forecastedWeather.csv', sep=',',
index=False, encoding='utf-8')#Save it as a csv so we can use this dataframe for
debugging and access it from the rest of the program.

return combined_dataframe

# %% [markdown]
# Retrieve Current Agricultural Commodity Price

# %%
def get_dataset_link(mainWebsite_url): # Returns the path of commodity price
dataset from the World Bank Group Commodity Markets page.
#url: The URL of the World Bank Commodity Markets page.

```

```
#We use validation here with the implementation of a try: except: statement  
#This means that we can catch the error before it crashes the entire program.
```

```
#Reasons that we can't access the website may be a firewall or a lack of an  
internet connection so we need to communicate these to our users.
```

```
#an alertDialog within the except clause prints the error why the user can't  
access the web page and executes an action to fix it.
```

```
try:
```

```
    page = requests.get(mainWebsite_url)  
    page.raise_for_status() # Verifies the webpage is reachable.  
    html = BeautifulSoup(page.text, 'html.parser') # Extracts the html
```

```
    # Find the specific link containing "Annual prices"  
    xlsx_link = html.find("a", href=True, string="Annual prices")
```

```
    if xlsx_link:
```

```
        return xlsx_link["href"]
```

```
    else :
```

```
        alertDialog('Could not access commodity market data','soft reset')
```

```
except requests.exceptions.RequestException as error: # Prevents the code  
from crashing if the user can't access the webpage and prints the reason.
```

```
    alertDialog(f"Error accessing web page>>> {error}","hard reset")
```

```
def download_dataset(fileURL):
```

```
    try:#Validation is also used here as we are downloading the entire file chunk by  
    chunk as you can see.
```

```
        #If for any reason while downloading it fails the error is caught by the except  
        clause and prints the error.
```

```
        response = requests.get(fileURL, stream=True)  
        response.raise_for_status()
```

```
        with open('commodityPrices.xlsx', 'wb') as f:#Downloads the entire file  
            for chunk in response.iter_content(chunk_size=8192):  
                f.write(chunk)
```

```
        return os.path.abspath('commodityPrices.xlsx')#Returns the absolute location  
where the data is stored.
```

```
    except requests.exceptions.RequestException as error: #Error handling  
        alertDialog(f"Error downloading file>>> {error}","hard reset")
```

```
def retrieve_commodity_prices(): #Retrieves current and historical agricultural  
prices.
```

```
    commodityDatasetLink = get_dataset_link(worldBank_url)  
    commodityPricesPATH = download_dataset(commodityDatasetLink)
```

```

#Our entire program uses dataframes to process data due to openMeteo's
dependency on them and the embedded data manipulation that can be easily
used on them.

commodityPricesDataframe = pd.read_excel(commodityPricesPATH, sheet_name
= 'Annual Prices (Nominal)')#We read the retrieved dataset as a pandas dataframe.

for i in range(5):#We drop 5 rows of empty spaces to make the titles be the clear
column headers of the dataframe.
    commodityPricesDataframe = commodityPricesDataframe.drop(index=[i])

commodityPricesDataframe = commodityPricesDataframe.iloc[:,[0,11,24,28,35,45]]#We extract only the relevant commodities which we are using to
make predictions to be the entire dataset

commodityPricesDataframe =
commodityPricesDataframe.drop(commodityPricesDataframe.index[[1, 2]], axis=0)

return commodityPricesDataframe

# %% [markdown]
# Classify Weather Data

# %%
def
classify_commodityPrice_Dataframe(crop,commodityPricesDataframe):#Labels to
the selected crop's change in price as either positive or negative.

#This for loop matches the crop with it's location within the croplIndex array.

#The index within the crop in the croplIndex array is the same as the index within
the commodity prices dataframe.

for i in range(len(croplIndex)):
    if croplIndex[i] == crop:
        cropRow_DataframeReference = i
        break

commodityPricesDataframe = commodityPricesDataframe.iloc[:, [0,i + 1]]#Extract
only the crop column you need and the dates.

pd.set_option('display.max_rows', None)#Display the maximum amount of rows
and columns so all available data is able to be accessed,
pd.set_option('display.max_columns', None)

```

```

commodityPricesDataframe =
commodityPricesDataframe.iloc[1:].reset_index(drop=True)
commodityPricesDataframe.iloc[:, 1] =
pd.to_numeric(commodityPricesDataframe.iloc[:, 1])

#Filters out unnecessary errors produced by the program
warnings.filterwarnings('ignore', category=FutureWarning,
message="Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated")

# Calculate percentage change
commodityPricesDataframe['Percentage Change'] =
commodityPricesDataframe.iloc[:, 1].pct_change() * 100
commodityPricesDataframe['Percentage Change'] =
commodityPricesDataframe['Percentage Change'].astype(float).fillna(0)

# Round percentage change to an integer
commodityPricesDataframe['Percentage Change'] =
commodityPricesDataframe['Percentage Change'].round(0)

commodityPricesDataframe['Percentage Change'] =
(commodityPricesDataframe['Percentage Change'] / 10).round(0) * 10

# Convert percentage change to correct multiplier format
commodityPricesDataframe['Percentage Change'] = 1 +
(commodityPricesDataframe['Percentage Change'] / 100)

commodityPricesDataframe['Percentage Change'] =
commodityPricesDataframe['Percentage Change'].astype(str)
return commodityPricesDataframe#Returns the processed dataframe.

# %% [markdown]
# SVM

# %

def trainingSupportVectorMachine(historicalWeather, historicalPrice,
hyperparameters):
    X = historicalWeather # The weather is the feature and contains the four climate
    metrics
    y = historicalPrice # The labelled prices are the labels and is what we are trying
    to train our SVM to accurately predict.

    if hyperparameters == 'no hyperparameters': # If there are no given
    hyperparameters set it to default ones.
        C_value, gamma_value, kernel_value = 10000, 0.01, 'rbf'

```

```

else:
    C_value, gamma_value, kernel_value = hyperparameters['C'],
    hyperparameters['gamma'], hyperparameters['kernel']#This allows advanced users
    to have access to custom hyperparameters when they train their model.

    X.drop('year', axis=1, inplace=True)
    y = y.iloc[:, 2:]
    y=y.values.ravel() #Turns dataframe into a 1 dimensional array as we only have a
    single column we can convert it into a single row.

    if len(X) != len(y):#This involves validation as the model checks if there is missing
        data. This will happen if one dataset is larger than another dataset so therefore
        there are missing datapoints.
        errorAlert('missing data. The length of historical weather and price data do not
        match. ','hard reset')

    # Scale the data so it has a mean of 0 and a standard deviation of 1. This gives us
    previously described performance increases.
    X_scaled = scale(X)

    # Train the SVM on ALL data
    clf_svm = SVC(C=C_value, gamma=gamma_value, kernel=kernel_value)
    clf_svm.fit(X_scaled, y)

    return clf_svm#Return the trained SVM model.

# %%
#Outputs the optimal hyperparameters as a dictionary and creates a processing
txt file.
def findOptimal_hyperparameters(crop):
    param_grid = [
        {'C':[1, 10, 100, 1000, 10000], #regularisation parameter C.
        'gamma': [ 1, 0.1, 0.01, 0.001],#Different gamma values
        'kernel': ['rbf', 'poly']}]]#We can pick different kernel functions ,but RBF is works
    for most.

    X = retrieve_historical_climate(crop)
    commodityPricesDataframe = retrieve_commodity_prices()
    y = classify_commodityPrice_Dataframe(crop ,commodityPricesDataframe)
    y = y.iloc[:, 2:]

    y = y.values.ravel()

    X.drop('year', axis=1, inplace=True)

    if len(X) != len(y):

```

```

        errorAlert('missing data. The length of historical weather and price data do not
match.','hard reset')

X_scaled = scale(X)

# Run GridSearchCV in a subprocess and capture output. Virtual environment!
command = ["python3", "-c",
           "from sklearn.model_selection import GridSearchCV; "
           "from sklearn.svm import SVC; "
           "#import numpy as np; "# Import numpy if needed for X_scaled and y
           "param_grid = {'C':[0.01, 0.1, 1, 10, 100, 1000], "
           "'gamma':[10, 1, 0.1, 0.01, 0.001, 0.0001], "
           "'kernel':['rbf', 'poly']}; "
           "optimal_params = GridSearchCV(SVC(), param_grid, cv=10,
scoring='accuracy', verbose=2);"
           "# Assuming X_scaled and y are numpy arrays converted into a list. As
subprocess is a virtual environment.
           f"X_scaled = np.array({X_scaled.tolist()});" # Convert to list for string
representation
           f"y = np.array({y.tolist()});"
           "optimal_params.fit(X_scaled, y);"
           "print(optimal_params.best_params_)"
           ]

#Run the command above and then dump all the outputs line by line into a text
file.
with open('processingOutput.txt', 'w') as outfile:
    subprocess.run(command, stdout=outfile, stderr=subprocess.STDOUT,
text=True)

with open('processingOutput.txt', 'r') as file:
    for line in file:
        pass
    optimalHyperparameters = line
    optimalHyperparameters = optimalHyperparameters.strip()

#Extract the optimal hyperparameters from the last line of the outputted text
file and return it to the user.
#We use ast.literal_eval to interpret the string in the text file into a dictionary.
optimalHyperparameters = ast.literal_eval(optimalHyperparameters)

return optimalHyperparameters

# %%
#This is the prediction layer. We take the entire model and the crop and produce a
prediction based on forecasted weather data.

```

```

def predict(crop,model):

    #We retrieve and standardise the forecasted weather
    futureWeather_dataFrame = retrieve_forecasted_climate(crop)

    #Delete the first date column as it is unnecessary and not a numerical value.
    futureWeather_dataFrame =
    futureWeather_dataFrame.drop(futureWeather_dataFrame.columns[0], axis=1)

    #Standardise and scale the data.
    futureWeather_dataFrame_scaled = scale(futureWeather_dataFrame)

    #We input the forecasted weather data into the model producing the
    predictions which we return as an array.
    prediction = model.predict(futureWeather_dataFrame_scaled)

    # Convert to array of floating-point numbers
    prediction = prediction.astype(np.float64)

    yearlyChange = np.array(prediction)
    fourDayChange = 1 + (((yearlyChange - 1) / 365) * 4)

    prediction = fourDayChange

    return prediction

# %%
def impute_anomalies(dataframe):

    #This is the k value and defines how large the bounds will be.
    #A larger k value will mean the bound will cover less of the entire dataset.
    #This means that the function will be less sensitive to anomalies.
    k = 2

    for column in dataframe.columns:

        #This checks if the column only contains numerical numbers as we can't
        impute string values.
        if pd.api.types.is_numeric_dtype(dataframe[column]):

            #Find the upper and the lower quartiles.
            Q1 = dataframe[column].quantile(0.25)
            Q3 = dataframe[column].quantile(0.75)

            IQR = Q3 - Q1 #Find the interquartile range

            #Define the upper and lower bounds which data have to be within.

```

```

lower_bound = Q1 - k * IQR
upper_bound = Q3 + k * IQR

#.loc is used to access and modify specific elements within the specific
column.
#This means there is no need to create an inefficient for loop to loop
through all the values and columns.

#The condition selects only elements that fall outside of the bounds we
defined.
#These specific elements are then replaced with the mean of the column.
dataframe.loc[(dataframe[column] < lower_bound) | (dataframe[column] >
upper_bound), column] = dataframe[column].mean()

return dataframe

def impute_missing(dataframe):

    for column in dataframe.columns:
        #This checks if the column only contains numerical numbers as we can't
        impute string values.
        if pd.api.types.is_numeric_dtype(dataframe[column]):

           #.loc is used to access and modify specific elements within the specific
            column.
            #This means there is no need to create an inefficient for loop to loop
            through all the values and columns.

            #These specific elements are then replaced with the mean of the column.
            dataframe[column] =
            dataframe[column].astype(float).fillna(dataframe[column].mean())

    return dataframe
# %%
custom_cValue = None
custom_gammaValue = None

def customHyperparameters(c,gamma):
    global custom_cValue
    global custom_gammaValue
    custom_cValue = c
    custom_gammaValue = gamma

# %%
def initialiseParameters():
    custom_cValue = None
    custom_gammaValue = None

```

```

# %% [markdown]
# Graphical User Interface

# %%
def clear_page():
    #Clears the entire page of widgets. So the entire page is empty.
    for widget in root.winfo_children():
        widget.destroy()

def crop_selection_page():
    #We open the crop selection page as the starting page for the user to pick from
    #a selection of crops.
    clear_page()

    # Header
    header_frame = tk.Frame(root, bg="#80ff97")
    header_frame.pack(fill="x")
    tk.Label(header_frame, text="SELECT A CROP BELOW", font=("Arial", 24,
    "bold")).pack(pady=10)

    # Information Box
    info_frame = tk.LabelFrame(root, text="Information", bd=2, relief="solid")
    info_frame.pack(fill="x", padx=10, pady=10)
    info_text = "This box provides static information. For example, the tutorial
    prompts below.\n\n"
    info_text += "Select a crop to analyze.\n"
    info_text += "Access advanced settings to edit hyperparameters and other
    settings.\n"
    tk.Label(info_frame, text=info_text, justify="left",
    wraplength=500).pack(padx=10, pady=10)

    # Buttons Frame
    button_frame = tk.Frame(root)
    button_frame.pack(fill="x", padx=10, pady=10)

    # Advanced Settings Button
    advanced_button = tk.Button(button_frame, text="Advanced\nSettings",
    command=lambda :advanced_settings(), fg="white", bg="blue",
    font=("Arial", 14, "bold"), width=10)
    advanced_button.pack(side="left", padx=5, pady=5)

    # Crop Buttons
    for crop in ("cocoa", "wheat", "corn", "soybean", "sugar"):
        ttk.Button(button_frame, text=crop, command=lambda c=crop:
        processing_page(c), width=10).pack(side="left", padx=10, pady=50)

```

```

def processing_page(crop):
    # This is the processing page where all the data is being processed in the
    background.
    # The user will get updates as to the progress and if any errors occur during the
    entire process.
    clear_page()

    # Header of the processing page...
    header_frame = tk.Frame(root, bg="#80ff97")
    header_frame.pack(fill="x")
    tk.Label(header_frame, text="PROCESSING PAGE", font=("Arial", 24, "bold"),
    bg="#80ff97").pack(pady=10)

    # Progress Bar that shows the progress of the entire solution.
    progress_frame = tk.Frame(root)
    progress_frame.pack(fill="x", padx=10, pady=10)
    progress_canvas = tk.Canvas(progress_frame, height=20, bg="lightgray",
    highlightthickness=0)
    progress_canvas.pack(fill="x")
    progress_bar = progress_canvas.create_rectangle(0, 0, 0, 20, fill="#80ff97")#The
    rectangle represents the progress bar we can change the rectangle's width to
    represent the change in progress over time.

    # makes the labelled error and progress frames that are side by side to each
    other.
    error_progress_frame = tk.Frame(root)
    error_progress_frame.pack(fill="x", padx=10, pady=10)
    error_frame = tk.LabelFrame(error_progress_frame, text="Errors:", bd=2,
    relief="solid", bg="#ffbfad")
    error_frame.pack(side="left", fill="both", expand=True, padx=5)
    progress_info_frame = tk.LabelFrame(error_progress_frame, text="Progress",
    bd=2, relief="solid", bg="#80ff97")
    progress_info_frame.pack(side="right", fill="both", expand=True, padx=5)

    # Error Block
    error_label = tk.Label(error_frame, text="None", bg="#ffbfad")
    error_label.pack(padx=10, pady=10)

    # Progress Block
    progress_label = tk.Label(progress_info_frame, text="", bg="#80ff97") # Empty
    label initially
    progress_label.pack(padx=10, pady=10)

    # We can use this function to update the text within the progress block
    def update_progress_text(text):
        progress_label.config(text=text)

    #We can use this function to update the text within the error block.

```

```

def update_error_text(text):
    error_label.config(text=text)

# Function to update the progress on the progress bar this makes the bar larger.
def update_progress(percent):
    progress_canvas.coords(progress_bar, 0, 0, progress_canvas.winfo_width() *
(percent / 100), 20)

def processingTasks():

    try:#Validation has occured as we are making sure that historical weather and
    price data has been retrieved successfully/
        historicalWeather = retrieve_historical_climate(crop)
        commodityPrices = retrieve_commodity_prices()

        #We update all the progress indicators to show this.
        update_progress_text('Retrieved Weather + Price Data')
        update_progress(10)

        progress_canvas.update_idletasks()#This prevents tkinter from running
        anything past this point before the previous tasks have been completed /
        rendered.

    except:
        update_error_text('ERROR - Retrieving Data see the python terminal')#We
        catch the errors and display it to the user both in tkinter and in the python
        command line interface.
        errorAlert('Unfortunately we were unable to retrieve either historical weather
        or commodity prices from the internet. Check your internet connection or
        firewall','soft reset')

    try:#We validate whether the commodity prices have been successfully
    labelled.
        labelledPrices = classify_commodityPrice_Dataframe(crop,
        commodityPrices)#Label the commodity prices

        #Update the progress to show this
        update_progress(20)
        update_progress_text('Labelled Crop Price Data')

        progress_canvas.update_idletasks()
    except:

```

```

update_error_text('ERROR Labelling commodity prices')#We catch the errors
and display it to the user both in tkinter and in the python command line
interface.

errorAlert('Unfortunately the commodity prices were not able to be either
accessed or labelled.','soft reset')

update_progress(30)
update_progress_text('Training the model...')
findOptimal_hyperparameters(crop)

#If the user has not set custom C and Gamma hyperparameter values then we
can use the grid-search cross validation to find these optimal values and deliver it
to the user.

if custom_cValue == None and custom_gammaValue == None:
    trainedModel =
trainingSupportVectorMachine(historicalWeather,labelledPrices, 'no
hyperparameters') #The program will optimise it itself with custom
hyperparameters.

else:
    #If the user has set custom C and Gamma hyperparameter values then we
    will invoke this else statement. Within this we will use these values and pass it
    within a dictionary.

    custom_hyperparameterDictionary = {
        'C': custom_cValue,
        'gamma': custom_gammaValue,
        'kernel': 'rbf'
    }

#We can then pass these values in the dictionary into training the model.
trainedModel = trainingSupportVectorMachine(historicalWeather,
labelledPrices, custom_hyperparameterDictionary)

update_progress_text('Predicting the future...')
update_progress(80)

#We want the predictions to have a global scope so we can access it across
the entire program.

global predictions

predictions = [] #We set the predictions to make sure it is empty before we
add the new predictions to the dataset.

predictions = predict(crop, trainedModel)

update_progress_text('Complete')

```

```

update_progress(100)

root.after(0, processingTasks())#We run the processingTasks function 100ms or
0.1s after the entire program to let tkinter render all their content correctly.
output_page(crop)

def output_page(crop):#This is the output page where we output the graph and
the predictions onto a suitable interactive page.

    clear_page()
    import datetime

    commodity_prices_df = classify_commodityPrice_Dataframe(crop,
retrieve_commodity_prices())

    global predictions
    predictions = predictions.astype(np.float64)#Produces a numerical array of all
the predictions from a string to a float datatype.

    # Creates the plot
    fig, ax = plt.subplots()
    ax.plot(commodity_prices_df.iloc[:, 0], commodity_prices_df.iloc[:, 1]) # Plotting
of the year and the price...
    ax.set_xlabel("Year")
    ax.set_ylabel("Price")

    ax.set_title(f"{crop.capitalize()} Prices Over Time") #Title of the graph (crop
name is here) Prices Over Time

    # Retrieve all the dates we need...
    today = datetime.date.today()
    current_year = today.year

    # Get the last datapoints from the dataset.
    last_year = int(commodity_prices_df.iloc[-1, 0])
    last_price = commodity_prices_df.iloc[-1, 1]

    # Plot today's data (assuming the price is the same as the last year's price) We
are assuming this dataset is the most up-to-date dataset we have access too.
    ax.plot(current_year, last_price, 'ro', label="Today's Price")

    # Draw a line connecting the last point and today's point
    ax.plot([last_year, current_year], [last_price, last_price], 'r--', label="Connecting
Line") # Draw a line between the points

    last_date = datetime.date.today() # We set the start to be todays date.

```

```

last_price = commodity_prices_df.iloc[-1, 1] # Retrieve the last price

for i in range(min(4, len(predictions))): # Plot up to 4 predictions

    next_date = last_date + datetime.timedelta(days=4) #Add four days to get to
    the next point.

    next_price = last_price * predictions[i] # Calculate the next price using the
    predictions array.

    # Determine line color based on price change green is positive red is negative
    # change.

    if next_price > last_price:
        line_color = 'g-'
    else:
        line_color = 'r-'

    # Convert dates to years for plotting
    last_year = last_date.year + (last_date.timetuple().tm_yday / 365.25) #
    Accounts for leap years as 1/4 = 0.25
    next_year = next_date.year + (next_date.timetuple().tm_yday / 365.25) #We
    need to do this as our entire dataset has a yearly axes and we can't plot months
    and days.

    ax.plot([last_year, next_year], [last_price, next_price], line_color) #Plot the
    connecting line with the correct line colour.

    last_date = next_date
    last_price = next_price

def zoom_to_predicted_prices(): #Allow time scale viewing to see specific
    segments of the graph.

    start_date = datetime.date.today() #We set this to today's current date
    beforehand so we can use this same variable.

    dates = []
    prices = []

    last_price = commodity_prices_df.iloc[-1, 1] # Get the last price from the
    dataframe using the iloc dataframe function.

    current_date = start_date # Initialize current_date with start_date

    for i in range(len(predictions)): # Iterate through the predictions
        dates.append(current_date) # Add the current date to the dates list
        prices.append(last_price * predictions[i]) # Calculate and add the predicted
        price

```

```

    current_date = current_date + datetime.timedelta(days=4) # Increment the
current_date by 4 days

    years = [d.year + (d.timetuple().tm_yday / 365.25) for d in dates]# Convert
dates to years for plotting as our x axis is in years.

    # Set x,y zoom
    ax.set_xlim(min(years) - 0.15, max(years) + 0.15) # Zoom to see x axis years
    ax.set_ylim(min(prices) * 0.99, max(prices) * 1.01) # Zoom to see y axis
predicted prices

    canvas.draw() # draw the canvas again to reflect the zoom.

    zoom_button = ttk.Button(root, text="Zoom In to see Current Changes",
command=zoom_to_predicted_prices)#Button to zoom in
zoom_button.pack(pady=10)

    ax.legend()

    # Embeds the plot in the tkinter window
    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas.draw()
    canvas.get_tk_widget().pack()

    # Adds the matplotlib toolbar
    toolbar = NavigationToolbar2Tk(canvas, root, pack_toolbar=False)
    toolbar.update()
    toolbar.pack(side=tk.BOTTOM, fill=tk.X)

    # Creates and packs the back button
    back_button = ttk.Button(
        root,
        text="Back to Crop Selection",
        command=lambda: [crop_selection_page()],
    )
    back_button.pack(pady=50)

#This will be the advanced settings page where users are able to customise the
hyperparameters.
def advanced_settings():
    clear_page()

    # Information Box
    info_frame = tk.LabelFrame(root, text="Information", bd=2, relief="solid")
    info_frame.pack(fill="x", padx=10, pady=10)
    info_text = "This is the advanced settings page. Here you can adjust the
hyperparameters 'Gamma' and 'C'.\n\n"
    info_text += "Please enter numerical values for both hyperparameters."

```

```

tk.Label(info_frame, text=info_text, justify="left",
wraplength=500).pack(padx=10, pady=10)

# Input Frame
input_frame = tk.Frame(root)
input_frame.pack(pady=20)

# Gamma Value
tk.Label(input_frame, text="Gamma Value:").grid(row=0, column=0, sticky="e")
gamma_entry = tk.Entry(input_frame)
gamma_entry.grid(row=0, column=1, padx=5)

# C Value
tk.Label(input_frame, text="C Value:").grid(row=1, column=0, sticky="e")
c_entry = tk.Entry(input_frame)
c_entry.grid(row=1, column=1, padx=5)

def submit_values():
    try:
        gamma = float(gamma_entry.get())
        c = float(c_entry.get())

        customHyperparameters(c, gamma)
        crop_selection_page()
    except ValueError:#Here we use validation to make sure the user inputs integer
values for both gamma and c.
        messagebox.showerror("Error", "Please enter valid numerical values for both
Gamma and C.")

# Submit Button
submit_button = ttk.Button(root, text="Submit", command=submit_values)
submit_button.pack(pady=10)

if __name__ == "__main__":
    initialiseParameters()
    root = tk.Tk()
    root.title("scion")
    crop_selection_page()

    root.mainloop()

```

Bibliography

References

Examples of commodities (image)

Source:<https://learn.robinhood.com/articles/626haurrOd1BFJ3CkfH7xq/what-is-a-commodity/>

Supply and demand (image)

Source:https://en.wikipedia.org/wiki/Supply_and_demand#/media/File:Supply-demand-equilibrium.svg

Changes in Soybean and Cocoa Commodity Prices (Accurate as of 06/09/24) and respective soybean and cocoa graphs.

Source:<https://tradingeconomics.com/commodity/cocoa>

Example weather data showing all four weather metrics from 1960-2024. The weather metrics displayed are relative humidity, precipitation sum, mean cloud cover and mean temperature.

Source:<https://open-meteo.com/>

All the available historical weather data climate models available on openMeteo.

Source:<https://open-meteo.com/>

An example use-case of IBM's Environmental Suite showing their crop prediction tool as well as a weather summary.

Source:<https://www.ibm.com/products/environmental-intelligence-suite/monitoring>

Spire Global's soil moisture insights across regions of the world they cover.

Source:<https://spire.com/weather-climate/soil-moisture-insights/>

Spire Global's detailed overview of their satellites located in space.

Source:<https://earth.esa.int/eogateway/missions/spire/description>

Forecasted agricultural commodities created by Vesper.

Source:<https://vespertool.com/product/outlook/>

An example of how Trading212 graphs current and historical market prices.

Source:<https://helpcentre.trading212.com/hc/en-us/articles/4494804742301-How-to-activate-and-manage-chart-indicators>

An example forecast of potential future agricultural commodities prices.

Source:<https://www.morningstar.co.uk/>

Robinhood mobile application screenshot showing their stock market price graphing user interface.

Source:<https://newsroom.aboutrobinhood.com/introducing-robinhood-in-the-uk/>

Source:<https://robinhood.com/gb/en/>

An example Vesper prediction that shows their prediction user interface and the output predictions.

Source:<https://vespertool.com/product/outlook/>

Source:<https://vespertool.com/>

A screenshot of Robinhood's desktop application Robinhood Legends.

Source:<https://robinhood.com/us/en/legend/>

Information about the implementation of a support vector machine model.

Hyunsoo Yoon, Cheong-Sool Park, Jun Seok Kim, Jun-Geol Baek, Algorithm learning based neural network integrating feature selection and classification, Expert Systems with Applications, Volume 40, Issue 1, 2013, Pages 231-241 and ISSN 0957-4174

Source:<https://doi.org/10.1016/j.eswa.2012.07.018>.

An example image that shows an implementation of a future stock prediction model.

Source:<https://huggingface.co/foduicom/stockmarket-future-prediction>

Outlier data analysis visualisation image showing the inter quartile range, bounds and the relevant "outliers" of the data.

Source:<https://blog.alliedoffsets.com/beyond-the-norm-how-outlier-detection-transforms-data-analysis>

Standard deviation formula. This formula was invented by Karl Pearson in 1890.

Source:<https://www.geeksforgeeks.org/karl-pearsons-coefficient-of-correlation-methods-and-examples/>

Support Vector Machine explanation by StatQuest.

Source:<https://youtu.be/efR1C6CvhmE?si=tuTMd0AsOnDbHRhA>

An implementation of a kernel function followed by the creation of a hyperplane in between data.

Source:<https://medium.com/@abhishekjainindore24/svm-kernels-and-its-type-dfc3d5f2dc8>

All the structure diagrams and wireframes were created using LucidChart and Google Drawings.

Source:<https://www.lucidchart.com/pages/>

Source:<https://docs.google.com/drawings/>

How to embed matplotlib within tkinter.

Source: <https://www.youtube.com/watch?v=IVTC8CvScQo>

OpenMeteo allowed us to have open-source access to historical and future weather data without an API-key.

Source:

```
@software{Zippenfenig_Open-Meteo,
author = {Zippenfenig, Patrick},
doi = {10.5281/zenodo.7970649},
license = {CC-BY-4.0},
title = {Open-Meteo.com Weather API},
year = {2023},
copyright = {Creative Commons Attribution 4.0 International},
url = {https://open-meteo.com/}
}
```

The World Bank Group allowed us to have access to historical and current price data through their website. These datasets are constantly updated.

Source:

<https://www.worldbank.org/en/research/commodity-markets>

The Max Planck Institute provided the model that was used for climate analysis.

MPI_ESM1_2_XR

Source: Accessed via OpenMeteo.

More Information:

<https://gmd.copernicus.org/articles/12/3241/2019/>

Cocoa Agricultural Commodities beats Bitcoin to emerge as the top performing stock of 2024.

Source: CNBC News

<https://www.cnbc.com/2024/01/18/cocoa-agricultural-commodities-beats-bitcoin-to-emerge-as-the-top-performing-stock-of-2024-with-record-gains-outpaces-bitcoin.html>

Disclosure

The information and program does not constitute financial advice or recommendation and should not be considered as such. Predictions produced by the program is not financial advice and not regulated by the Financial Conduct Authority. Always do your own research and seek independent financial advice when required. The value of agricultural commodities may fluctuate. They can fall as well as rise and you may not get back the original amount you invested. This tool is for informational purposes only; we cannot be held liable for any actions you take as a result of outputs from this tool.