Economic Metrics App

Irene Peleteiro Paniagua

Analysis

Problem Identification

The problem I wish to solve is that there is currently no easy way to understand the wellbeing and economy of a country without simply looking at its GDP (or other factors individually). GDP is known to have many problems and could be seen as an outdated metric. Most modern economists now care more about the environmental and social issues rather than simple growth, making sure the population actually benefits from decisions made by governments and firms, and so use metrics relating directly to these issues. For this reason, I want to create a program in which the user can: view and compare countries' social and environmental qualities; see projections, and advice on what should be done to improve them; and use an economic simulator to test out their policies with an 'optimised' plan to compare against. This could allow the public to easily check the current economic picture, and allow policy makers to find the current problems and see possible solutions.

The key features of my solution are as follows:

Feature	**Explanation and Justification**
Uploading data from .csv files from online database (The World Bank, International Monetary Fund and Our World in Data)	To get all my economic data I will use in my solution, I will use these databases as they are trusted and allow free access. Inputting real economic data will allow the users to truly view current economics, which, especially for professional economist, could be crucial. I will export .csv files from the databases and upload them into a local database.
Storing inputted data in local database	All the inputted data will be stored in a local database for a faster execution speed, which will allow the users to seamlessly use the program without waiting for more instructions to load. It will also make development easier, as the software will only have to take data from a single database.
Processing this data	This feature will calculate an index for each metric in each country (e.g.: if life expectancy in the UK is 86 years, it's index will be 0.91 as compared with the other countries), and the averages of these indexes to find the overall country average in environmental and social issues. These averages will make the programming easier, as they will be in a single and simple to process form for displaying. They will also be stored in the database, so the same calculation does not have to be repeated and so saving time and computational resources.
Inputting users' choices	In the program, the user will be able to select which metrics they want (e.g.: only life expectancy and literacy levels) and display a map accordingly. This will allow users to explore the economic data further and achieve the results they want. It may allow professional economists a data source and less experience economists a chance to learn more.

	Users will also be able to select what policies they want in the economic simulator (*more on this in the simulator feature*).
Displaying processed data in colour-coded maps	The data will be displayed in simple colour-coded maps to allow users of all experience levels to understand and interpret the data easily. It will be much easier and more accessible than looking through the raw data.
Algorithm for projections	For the projection part of my program, I will use a projection algorithm to calculate the data and display said data in the same way (colour-coded maps). This algorithm will most likely use linear regression with seasonal variation to provide plausible projections with a simple algorithm to reduce the development time. The projections will allow users to view future economics, and learn more about the possible future of the economy.
Algorithm for the economic simulator	My solution will also have an economic simulator, which will allow users to simulate various scenarios by entering what policies they want. An algorithm (*of my own design using popular economic theories – e.g.: a rise in interest rates by 2% will lower inflation by 2% and increase the cost of living by 1%*). will be used to calculate the results of their choices. The results will then be displayed in the same way as before, a colour-coded map. This will allow users a more interactive way to learn about economics, and more experienced economists might use it to test out their own policy plans.
Algorithm for 'optimised' plan for the economic simulator	The program will also offer an 'optimised' solution along with the economic simulator (*optimised between quotations as it is highly unlikely to be truly optimised*). The algorithm I will use to find this plan is likely to be Dijkstra's algorithm, finding the least expensive 'route' while maximising the citizen's wellbeing *(e.g.: start with an increase in interest rates by 2%, which will lead to a decrease in inflation by 2% but will increase cost of living by 1%, so lower indirect taxes by… until it achieves the best combination of policies)*. This 'optimised' plan will allow inexperienced users further learning experience and might give experienced economists a new perspective.

The features of a computer system that would be required for this solution would be a computer with hardware capable of processing large amounts of data at a good speed, and a suitable operating system for the software to run on. The problem lends itself to a computational solution due to various reasons. There is a large amount of data needed to be processed, so a program – instead of working on it manually – would benefit users as they save time and resources. The program will display a colour-coded map, which is much easier to interpret and understand than the raw data, allowing more inexperienced users to

understand economics more easily and not be intimidated. In general, it will be easier to get economics data from the program than from databases, as it will only keep the relevant and specific data the users are looking for.
Describing it in its simplest form, the solution would process current economic data and produce a visual representation of this data, as well as producing a visual representation for projections based on current data, and allow users to simulate policy making.

Why it is suited to a computational solution

The problem lends itself to a solution using computational methods in a variety of ways. The main reason is the large amounts of data that the program would need to deal with, as well as the various algorithms needed to process it. While it could be possible to do manually, it would be extremely time consuming and inefficient. Moreover, all the data needed is already in online databases which would allow for quick and easy retrieval into a program to process it. The solution also has a visual representation output which can be easily done computationally.

Computational methods the solution lends itself to:

Problem recognition

The computational problems that need to be solved can be split into three: the retrieval, storage and handling of data; the algorithms used to process this data; and the display of the processed data. The retrieval and display of data should be fairly simple, it is the handling of data and algorithms that could be the most challenging. The amount of data could become overwhelming so a good storage system should be considered, and the algorithms complexity could be quite high, so solving these problems should be the priority

Decomposition

As I've said before, the problem can be divided into much smaller components (input, processing, and output) which could then again be further divided. For example, processing could be divided into:

1. Processing environmental data
2. Processing social data
3. Creating predictions based on an algorithm
4. Policy making
    i. Creating predictions based on the user's newly selected policy
    ii. Creating a set of 'optimised' policies based on an algorithm

These steps could be even further divided, creating small functions that would make the problem much more manageable. For this reason, a computational method would benefit the problem's solution.

Pattern recognition

With this problem and the solution I've come up with, the software will need to run the same functions multiple times throughout the program. Patterns can be found throughout the solution I have made, for example, it is likely that the same algorithm used to create the weighted values for a set of data will be used for another set, and, in terms of output, the

same code could be run to create the environmental and social graphs. This pattern repetition could (and will) be utilised with a computational solution.

Abstraction

The original databases I will access have thousands of data values that are irrelevant to the solution. Moreover, for the output of my program, only the summarised data will be displayed in the aim to make economics more accessible – plus, for most of the program, the exact data values are not needed and can be stripped away from the output without major repercussions, only being displayed if the user chooses to find out more. A computational solution could take advantage of this fact, reducing the amount of data needed to be handled at the same time and the amount of data needed to be temporarily stored, improving the program's efficiency.

Stakeholders

The clientele for this product could be quite wide, but should all have some knowledge and interest in economics and the current economic picture. Due to this wide range, I will focus the stakeholders to professional economists who would use this program to aid their work and to the average individual who might wish to know more of current economics.

Government policies depend on data on how the country is faring to find problems and create solutions. For example, the NHS funding increased during the pandemic due to a shortage in space and medics. Policy making lends itself to this software as the problems within the economy could be easily seen with the graphs, and the program will offer advice on possible policies. The stakeholders for the professional economist demographic are Lourdes and Fermin. Fermin is employed in global business operations and Lourdes works in financial data and regulation. They both have good knowledge on economic policies and on the current economic picture, and so can give good feedback on the software.
These stakeholders, professional economists, research and look through economic data to support their decisions and policies, whether at the government, investment, or firm level. They will use my solution to get data on current economics, projected economics and to test out their own policies in the economic simulator. My solution with colour-coded maps will be appropriate as the economists will be able to access the data they want easily and quickly, to both inform themselves and also use the graphics to support their policies (e.g.: projections of a rise in air pollution will support a plan to increase funding for electric car chargers). Most professional economists have to process data and represent it in an easy-to-digest manner anyways, as their plans have to be approved by non-economists, so the solution will directly aid their work.

Another application for this software could be for the public to quickly and easily gain more knowledge on current economics. Individuals could check the graphs to see the current environmental and social standings as well as their goals. They could also see predictions based on the current data, possibly inspiring more to help out solving local problems, and they could see the efficiency of the government's proposed policies by testing them out themselves. The public could benefit from more knowledge by using this software. The

stakeholder for this demographic will be Jenny. Jenny represents the less experienced clients that might wish to use the software to expand their economic knowledge.
These stakeholders, the 'average consumer' (*average as in without much previous economics knowledge*), are likely to only get economic data through news sites and may find economics and interpreting the millions of data points daunting. My solution will offer a simple and easy to understand tool to allow people of any level to interpret economic data without problem. The colour-coded maps are extremely intuitional, and the projections and simulator will offer further opportunities to learn about economic theories through hands-on experience.

Interview Plan

Interview Questions

During the interviews I will ask the questions below. I will also ask follow up questions or ask to elaborate depending on the answers I get. My main objective for these questions is to find what metrics the stakeholders want to see in the graphs as well as what features they want to see overall.

Professional Economist

The questions I will ask Fermin and Lourdes, representing the professional economists, are as follows:

1. When looking to get more information on the current state of the economy, what type of data are you looking for?
2. Currently, how do you get access to economics data?
    - Do you currently rely on news sites to get this data (e.g.: The Economist)?
3. Are you satisfied with the amount and quality of data?
    - If not, what would you wish existed?
4. Are you satisfied with the way you access data currently?
    - If not, what would you wish existed?
5. What would you want to be included in this software (i.e.: the types of graphs, the metrics included in each graph, the countries included)?
6. Would you want projections based on current economics?
7. Would you want to see predictions based on a policy of your choice (e.g.: raise income taxes by 2%)?
8. Would you want the software to offer an 'optimised' policy plan for current economics?
9. Do you have any other ideas/changes you would want me to add?

Questions 1 and 2 will be used to get an idea on how they currently get all their information, including what type of information and where they get it from. This will help me in seeing my competitors and what my possible clients wish for in terms of the data they want to see.

Questions 3 and 4 will try to find the problems with the current data and with the current way of accessing said data. It will also offer the interviewees a chance to offer new ideas and features without thinking of my proposed solution, (it will be a completely unbiased opinion

on what problems they believe should be solved and how they should be solved). It will help me review my current plan and possibly add more features.

Similar to questions 3 and 4, question 5 focuses on what features the stakeholders would want to see, but now with more guidance concerning my proposed solution. This will help confirm features and design choices.

Questions 6, 7 and 8 ask about more specific functions of the software and whether they would like them to be implemented or find them unnecessary.
The final question will allow them to state if they have anything else they'd like to add.

Average consumer

(*Again, an average consumer refers to a consumer with no/not much previous economic knowledge*)
The questions I will ask Jenny, representing the average induvial interested about economics, are as follows:

1. Do you ever look for more information on the economics picture?
    - If so, currently, how do you get access to economics data?
        - Do you currently rely on news sites to get this data (e.g.: The Economist)?
    - Are you satisfied with the amount and quality of data?
        - If not, what would you wish existed?
2. Do you wish there was an easier and quicker way to access economics data on the whole nation's picture?
3. What would you want to be included in this software (i.e.: the types of graphs, the metrics included in each graph, the countries included)?
4. Would you want projections based on current economics?
5. Would you want to see predictions based on a policy of your choice (e.g.: raise income taxes by 2%) to possibly learn more about economics?
6. Would you want the software to offer an 'optimised' policy plan for current economics?
7. Do you have any other ideas/changes you would want me to add?

Question 1 will find whether the average consumer looks for economics data in the first place, and if they do, what they search for and how they find it. This will help me create a few conditions I will need to meet in order to make the software accessible for all levels of knowledge.

Question 2 will simply test if there is actual demand for the software.

Questions 3 to 7 are the same as questions 5 to 9 from the professional economist interview, and achieve the same results as before.

Interviews

Professional economist

Fermin

1. **When looking to get more information on the current state of the economy, what type of data are you looking for?**
   Truthful, reliable, and unbiased data. Inflation rates, specifically changes in commodity prices to see how the economic state will evolve. Real state data as well as housing prices and starts. Also imports and exports data.
2. **Currently, how do you get access to economics data?**
   Through specialised information sites like bespoke investment groups such as Seeking Alpha.
   - **Do you currently rely on news sites to get this data (e.g.: The Economist)?**
     Sometimes, but they don't have the specified data I'm looking for.
3. **Are you satisfied with the amount and quality of data?**
   Yes, I am.
4. **Are you satisfied with the way you access data currently?**
   Yes, I receive daily emails and a weekly report.
5. **What would you want to be included in this software (i.e.: the types of graphs, the metrics included in each graph, the countries included)?**
   I think coloured graphs are a must for the software. I would also like the ability to select the metrics to be shown in the graphs. I like the measures that you showed me for the metrics to be included (*see on 'Features of proposed solution' to see list*). I think the G7 or G20 economies would work well.
6. **Would you want projections based on current economics?**
   Yes.
7. **Would you want to see predictions based on a policy of your choice (e.g.: raise income taxes by 2%)?**
   Yes.
8. **Would you want the software to offer an 'optimised' policy plan for current economics?**
   Yes.
9. **Do you have any other ideas/changes you would want me to add?**
   No.

Lourdes

1. **When looking to get more information on the current state of the economy, what type of data are you looking for?**
   The median salary per age range, inflation rates (and more specifically the price changes in basic products, as well as the differences between types of products such as tourist goods versus resident-bought goods), and the change in the cost of living in different regions.
2. **Currently, how do you get access to economics data?**
   The internet, magazines, and television. Specifically, BBC news and the bank of England website.
   - **Do you currently rely on news sites to get this data (e.g.: The Economist)?**
     Yes.

3. **Are you satisfied with the amount and quality of data?**
   No.
   - **If not, what would you wish existed?**
     More access to the raw data instead of the processed data.
4. **Are you satisfied with the way you access data currently?**
   Yes.
5. **What would you want to be included in this software (i.e.: the types of graphs, the metrics included in each graph, the countries included)?**
   The UK versus the European union and the United States, like the G7 economies. I think it'd be good to see the UK and its regions as well as how it compares to other countries. I'd also like to have other graphs, such as bar charts or pie charts once you 'zoom in' into a region. I'd also like the metrics to be divided by age.
6. **Would you want projections based on current economics?**
   Yes.
7. **Would you want to see predictions based on a policy of your choice (e.g.: raise income taxes by 2%)?**
   Yes.
8. **Would you want the software to offer an 'optimised' policy plan for current economics?**
   Yes.
9. **Do you have any other ideas/changes you would want me to add?**
   No.

Analysis

While the professional economists did have some sites they were getting their data from, it seemed to be either too specific or too processed for their liking. Offering the raw data is part of my planned solution, so it's great to hear that it is what they stakeholders want. Both stakeholders had a few metrics they currently looked for which will aid me when picking what to include. In terms of my current idea, both Fermin and Lourdes liked the colour-coded maps with an ability to 'zoom in' in some way to see the specific data. They also thought that the G7 economies would make a good selection of countries to compare. Both of them liked the idea of projections based on current economics as well as the ability to test out policies and an 'optimised' fiscal plan.
Fermin did point out that he currently received daily emails and weekly reports which might be something I should consider.

Average Consumer

Jenny

1. **Do you ever look for more information on the economics picture?**
   No.
2. **Do you wish there was an easier and quicker way to access economics data on the whole nation's picture?**
   Yeah. Right now, the only information I can get is opinionated or on very specific topics, so I'd like to have a place where I can get the pure objective data.
3. **What would you want to be included in this software (i.e.: the types of graphs, the metrics included in each graph, the countries included)?**
   The wider the range the better, but something like the 'top' economies would be great too. If I could see the whole picture worldwide, then maybe zoom in into Europe, and further zoom in into the UK, that'd be great. In terms of metrics to be included, I'd like to have a representation of the sectors in some way. For the types of graphs, I think maps would work well with the environmental and quality of life data, but I'd also like a pie chart for percentage representations of other data like the sectors. I'd also like a map that shows the averaged data as well as the option to see the specific metrics.
4. **Would you want projections based on current economics?**
   Yes, that would be helpful. It would be good to see the effect of global issues.
5. **Would you want to see predictions based on a policy of your choice (e.g.: raise income taxes by 2%) to possibly learn more about economics?**
   It would be very nice. The software would be much more interactive that way, almost like a game, and I think I'd learn a lot about policies.
6. **Would you want the software to offer an 'optimised' policy plan for current economics?**
   Yes, I think it'd be a great idea from which I could learn a lot from.
7. **Do you have any other ideas/changes you would want me to add?**
   No.

Analysis

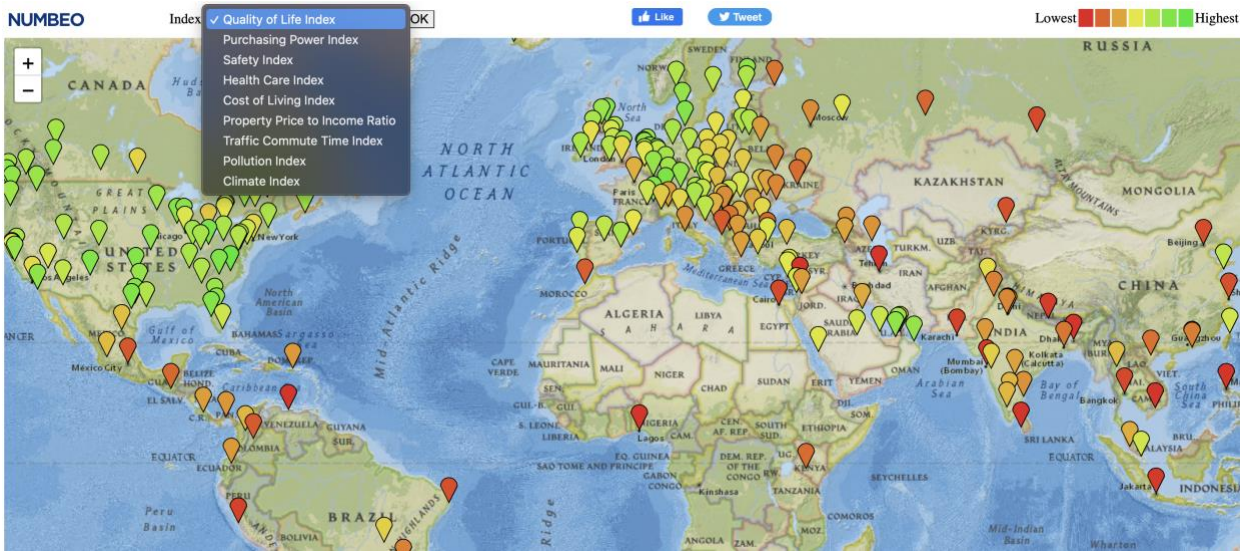
Currently, Jenny doesn't look for economic data and only accesses it through news and more specific and subjective sources, but she would like to have a place where she can easily access the raw and objective data which is what my solution offers. Like the professional economists, she finds the 'top' economies (like the G7 economies) would work well for the software, and she would like the ability to 'zoom in' into the map to see data about a certain country or region. She also said that she'd like other graphs to show the specific data, such as sector output percentage, and that she would want the ability to see an averaged map as well as ones with the individual metrics.
Jenny then added she would like to see the projections based on current economics and have the ability to test out her own policies as well as have access to an 'optimised' fiscal plan.

Research

Existing similar solution

Numbeo

Overview:



Numbeo offers a few colour-coded maps like the one above based on cost of living, property prices and quality of life. These can then be further divided into the specific metrics they consider (the image above shows the drop-down menu you can access to divide quality of life into metrics such as purchasing power, which lets you access another map based on that measure). If you want more information on a single country/city, you can click on its pin to access the actual data (as shown in the image to the left). You can also choose to compare two countries/cities, but the website only displays their data next to each other, which I found a bit useless. I also found there was a lack of information on what weightings they use for each metric, or more information on why they chose these metrics. The website does seem to be up to date and is relatively easy to use, although I found the asthetic of the website a bit antiquated.

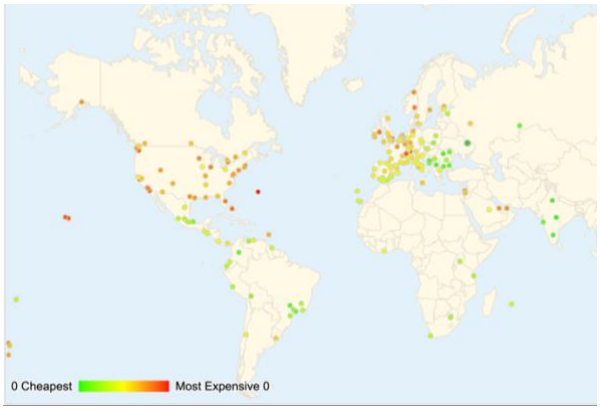
Parts that I can apply to my solution:

The concept of Numbeo is very close to a part of my solution, but there are parts I'd like to expand on. The features I found very useful in Numbeo was the ability to get the average quality of life map as well as the maps for the specific metrics; a feature I will likely implement myself. I found the colour-coding to be quite useful, as well as the access to more information if you click on the specific country/city. I did find the inclusion of some parts of the world map a bit redundant if there is only a few datapoints that can be accesses (e.g.: the whole of Africa

had 6 points). Nevertheless, the website was quite easy to use and had fast loading times, which is something I'd like to achieve in my solution as well.

Expatistan

Overview and parts that I can apply to my solution:

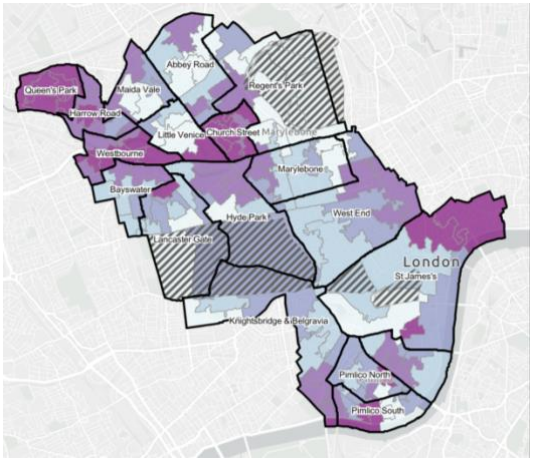


Similar to Numbeo, Expatistan offers world maps based on the cost of living (but not quality of life or property prices). I found Expatistan to have the same ease of use and fast load speed as Numbeo, but with a design more similar to the one I was imagining for my solution (with less harsh colours and a base map without unnecessary detail, simply blank).

Expatitan does have useful comparison data once you click on the country/city (as you can see in the image to the right), which I might look to implement for my solution. Expatistan also has more information on how they actually go about calculating the cost of living index, which I found quite useful. I also liked the fact that Expatistan offered a list of all the countries/cities ranked in order of cost of living prices, as well as a list of recent comparisons and the recent changes in prices (which showed they were constantly updating their website).

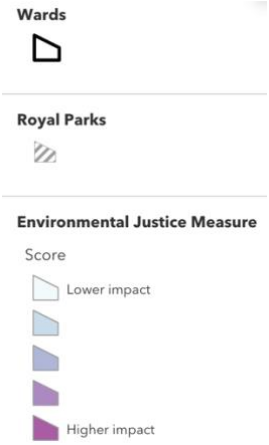
Westminster's Environmental Justice Measure

Overview:



The Westminster's Environmental Justice Measure is a really nice and simple way to look at the environmental issues within Westminster. It displays an incredibly detailed colour-coded map of Westminster representing the scores of each area. The key is easy to read, and the aesthetics are quite pleasing. You can select from a few measures they offer, such as air quality and heat risk, and the website displays the appropriate map. The website also has more information on how the metrics are measured and scored, which was quite useful. The only feature I disliked is that website flowed more as a display on information rather than a more interactive program in which you can find out the individual data you're looking for. The website also only offered the maps of individual metrics (and the overall average EMJ), and there was no way to combine a few of them into an average.

Parts that I can apply to my solution:

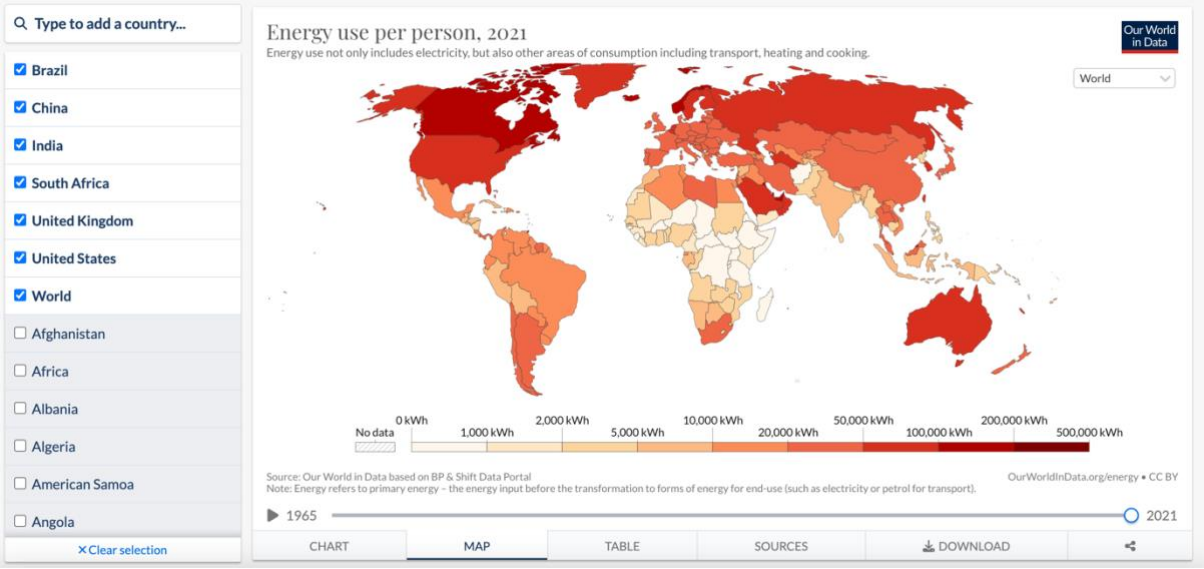


While the concept of the WEJM seems to be a bit different to mine – looking to "highlight differences in how people are impacted by their environment and climate change" and "give residents the information they need to act, influence and reduce negative environmental impacts" rather than simply offering an easier way to access economics data like my solution – the actual graphical part of their program is quite similar to what I had in mind. The aesthetics and metrics used for the WEJM are quite nice and I'll likely use them in my solution as well. There are some metrics, such as the Energy Performance Certificate which might be UK or Europe specific, which means I'll likely have to use another measure, but most of them will hopefully be worldwide measures.

Our World in Data

Overview:

Our World in Data has a wide range of information you can access, all the way from violence and war to food and agriculture. You can select from this wide range of topics the specific dataset you want (such as 'energy use per person') and can see it displayed into different graphical representations. Most of these have an option to see a chart, table, and a colour-coded map of the data. They also offer information on their sources and how they're calculating their data. You can also select the countries and continents to be included, and you can view a timelapse of the metric over a period of time. On top of this you can see different calculations of the data, such as overall versus per capita and the real or nominal change from previous years. The website offers loads of information as well as loads of ways to display it, which I found quite nice, but I find it could be a bit intimidating to someone who doesn't even know what they're looking for.



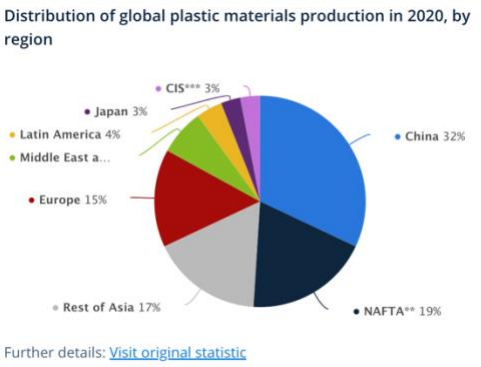
Parts that I can apply to my solution:

I quite liked Our World in Data, as it had a wide range that hit all the boxes. The maps, charts and tables all looked great and were easy to understand. Navigation through the website to find the data you were looking for was relatively easy, the only problem being that you almost

had to already know what you were looking for. Our World in Data is designed as a website, and it delivers in this aspect (and I'll likely use their data), but my aim for my solution is to have a more specific range of data with averages in place so the average user can easily see the main economic picture without having to search for all the individual data points.

Statista

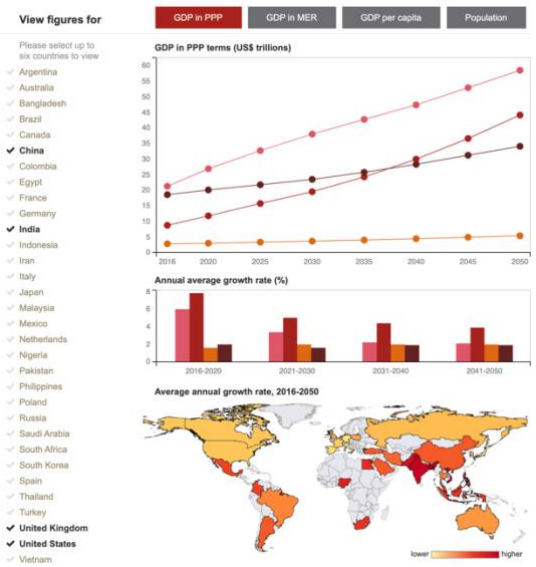
Overview and parts I can add to my solution:



Statista offers a good range of data, seemingly more focused on the production and output aspect of the economy. The website mainly displays the data in a more article-like style rather than interactive like the other competitors, where you can scroll through a particular metric and find titles like "China dominates the plastic industry" which, while still objective, deviates from my goal of pure objectiveness and the user having as much control as they want on the data they're seeing. Statista seems to mainly use pie charts and line graphs, which were useful, but I would've preferred to see maps like the other competitors and my solution.

PwC – The World in 2050

Overview:



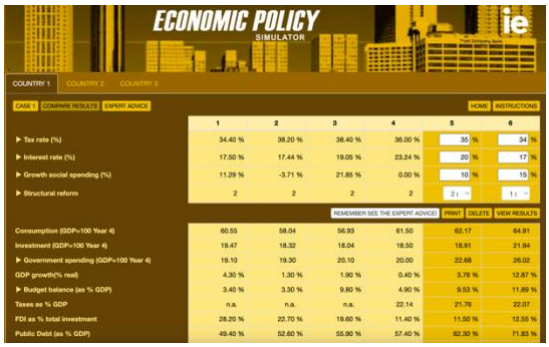
PwC offers projections on what the world will look like in 2050 in the form of a full report with their findings as well as an interactive data tool. The data tool works quite well, displaying the data in line graph, bar chart and map form. You can select what countries to display (although not all countries are available) and what averages on the data you would like (GDP in PPP, MER and per capita). The research does only show projections for GDP and population, which is a bit limiting and I would've liked to see more metrics such as air pollution or literacy levels. The website overall was a bit difficult to navigate, and I couldn't easily find what sources they use (you need to download the report to find that they used IMF's GDP data for 2016 and UN population predictions) and how they went about projecting the growth.

Parts that I can apply to my solution:

I quite liked the concept of PwC – The world in 2050, but I find the execution was a bit lacking. Using the same idea as them, I'd like to not only predict GDP growth but also other metrics, especially environmental concerns to possibly spur more action in tackling climate change.

Economic Policy Simulator (created by ie university)

Overview:



The economic policy simulator is quite a fun idea, and the website works fairly well. The aesthetics could be improved, but the case files for each country you can test out are well written and simulate reality fairly well. You can only control four decisions through 2 years, which I found a bit disappointing, and not much guidance is given on the decisions. The decisions available are to choose a tax rate (%), interest rate (%), growth social spending (%) and structural reform. Tax and interest rates are quite simple to understand and can be used without guidance, but I had no idea what the growth social spending was or how it worked, or how the structural reform could affect the simulation. The 'expert advice' did somewhat help, but was quite lacking.

Parts that I can apply to my solution:

Similar to PwC – The world in 2050, I really like the concept of the Economic Policy Simulator, but I find it's lacking in its execution. I wish you could apply other policies, such as changing planning regulations or tariffs, and that there was more guidance on how to work the simulation.

Features of proposed solution

Initial concept of my solution considering this research

My solution will be split into three parts: representing current economic data, creating and representing projections of economic data, and a small economic policy simulator. (*In bold are the key features of each part*).

The first part will have the following features (only including the G7 economies):

Feature	**Justification and benefits**
Colour-coded map displaying averaged data on environmental issues and colour-coded map displaying averaged data on social issues	All of my stakeholders agreed that the colour-coded maps would be a good addition and easy to understand. Throughout my research I also found plenty of colour-coded maps, so it seems to be a widely used and quite efficient way of representing economic data. I chose to divide it into environmental and social issues as they cover most – if not all – of the possible metrics and seems to be the more popular issues in modern economics (*featured in 'Doughnut Economics' by Kate Raworth*).
Ability to choose what metrics to include for these two maps	Again, all my stakeholders shared the want for an ability to select what metrics they want to be displayed in the maps, and my research also showed many existing solutions also used it.

Ability to 'zoom in' or click a country to get their raw data displayed in graphical form (users can choose from line graph, bar chart and table)	My stakeholders also wanted the ability to look directly at the raw data, so I'll add the ability to view this in different graphical forms. Other existing solutions also had this feature, so I'll implement it.

The second part of my solution will use an algorithm (**linear regression with seasonal variation**) to **create projections on future economics based on current data**. It will then display these in the same way as with the first part of my solution (but likely simplified, as it'll be difficult to project certain metrics).
This feature was approved by all of my stakeholders, and was used in one of the existing solutions (PwC – The world in 2050), so it seems like a good addition to the program. I will represent it in graphical form for the same reasons as before, as it will be beneficial for all users.

The third part of my solution will allow **users to pick from a range of policies** and have an **algorithm create predictions on future economics** based on their policy and current data (*the algorithm will be created by me using popular and generally approved economic theory*). It will then display these in the same way as with the first part of my solution (but likely simplified, as it'll be difficult to project certain metrics). Another **algorithm will also create an 'optimised' set of policies** based on current economics (*the algorithm will likely be Dijkstra's algorithm, finding the least expensive 'route' to maximise citizen welfare*).
Again, these two features were approved by all of my stakeholders, and they all agreed on the potential benefits of it (ability to test out policies, to learn more through first-hand experience, etc.). One of the existing solutions used an economic simulator, but as I discussed then, there was room for improvement so I will implement this idea with all the changes needed.

Throughout my whole solution there will be easily accessible instructions as well as information on how the data is calculated and where the data is sourced from.

The metrics that will be included in the solution are as follows: income inequality, literacy levels, quality of environment (involving air pollution, heat risk, flood risk, tree canopy, proximity to public transport, % of people using public transport), GDP per capita, GDP per PPP, GDP per MER, GNP, GNI, nature of output and type of employment, quality of output, purchasing power, quality of life, life expectancy, social progress, crime levels, cost of living, transport access, gender equality, inflation rate, employment/unemployment rate, government spending, balance of payments, government taxation, interest rates, sector proportions, mean salary per age range. (*These are likely to change as the project goes on*).

Limitations:

The largest limitation to my solution is that I will only use the G7 economies, rather than all worldwide economies. This is largely so I don't need to handle as much data, but also because there isn't much data in developing countries, which could easy cause errors and inaccuracies. It will mean that the users won't be able to access a lot of important economies, such as China and India, but this is a limitation I must set.

Another limitation might occur when dealing with the metrics, as it is unlikely I will be able to include all the metrics the users might want due to various complications. Metrics might only be used in certain countries, or have insufficient or inaccurate data. And, of course, I will not be able to use *all* possible metrics – obscure and less known measures such as lead poisoning I'm unlikely to include, as I will probably simply miss it (unless it is already averaged in another measure).

Further meeting with stakeholders

Email I sent to my stakeholders

"Hello [stakeholder name],

Following up on our interview, here is my final proposed solution:

In entering the program, you will be able to access two colour-coded maps based on environmental and social metrics respectively. You will be able to edit these maps and select what metrics to include, which will display the appropriate data. You will also be able to access more data on each country by clicking on it, displaying the individual data for this.
You will also be able to see projections on future economics displayed on the colour-coded maps.
Finally, you will be able to access an economic policy simulator, in which you can select your own policies and see how they will affect the projections. You will also be able to view an 'optimised' policy plan based on the current data.

The metrics that will be included in the program are as follows: income inequality, literacy levels, quality of environment (involving air pollution, heat risk, flood risk, tree canopy, proximity to public transport, % of people using public transport), GDP per capita, GDP per PPP, GDP per MER, GNP, GNI, nature of output and type of employment, quality of output, purchasing power, quality of life, life expectancy, social progress, crime levels, cost of living, transport access, gender equality, inflation rate, employment/unemployment rate, government spending, balance of payments, government taxation, interest rates, sector proportions, mean salary per age range.
If you are missing any, please do tell.

Thank you,
Irene"

The responses

Fermin

“Thank you for the detailed and thoughtful proposal. It incorporates all the feedback and discussions and I am confident the maps will be extremely useful to support planning and decision making.
It would be very useful to have early access to a pilot to test user experience, responsiveness and performance, let me know if this will be possible.”

Lourdes

“I am happy with the proposed solution. I think it's well thought. It provides an impressive amount of indicators and just the right level of granularity.
Looking forward to see the product coming to life.”

Jenny

“This is looking great! The economic policy stimulator sounds interesting, especially when you can customise your own policies and observe the effect - it will definitely make the topic politics and policy making more approachable for the public.”

Requirements

Software and hardware requirements

Hardware

The only hardware requirement is **a computer capable of running this software**, as there will be a large amount of data that will need to be fetched, processed, and displayed. However, most PCs and laptops will be able to handle the program easily, some may just take a few more seconds to output.

Software

Windows, Linux, or Mac operating system – I will need these operating systems, as they all support Python, Numpy and Plotly (which I will use for my solution).

Database System – as I will need to store my data and calculations.

Python interpreter – as I will be using Python for the program.

Numpy – (a Python library) as I will need to easily process large amounts of data with possibly quite complex algorithms.

Plotly – (a Python library) as I will need to create maps, bar charts and line graphs.

Flask – (a Python library) as I will need to offer user interface for the program.

Features requirements

List of all features

(*Essential features in bold, additional features without change*).

- **Colour-coded map displaying averaged data on environmental issues**
- **Colour-coded map displaying averaged data on social issues**
- **Ability to choose what metrics to include for these two maps**
- Ability to view maps for single metrics (that are included in the averaged map)
- **Ability to 'zoom in' or click a country to get their raw data displayed in graphical form (users can choose from line graph, bar chart and table)**
- Using real economic data (from online databases)
- **Ability to view projections on future economics based on algorithm (linear regression with seasonal variance)**
- Colour-coded map displaying averaged data on projected environmental issues
- Colour-coded map displaying averaged data on projected social issues
- **Ability to select from a range of policies in the economic simulator**
- **Algorithm to create predictions based on users' policies (the algorithm will be created by me using popular and generally approved economic theory)**
- **Algorithm to create 'optimised' set of policies (the algorithm will likely be Dijkstra's algorithm, finding the least expensive 'route' to maximise citizen welfare)**

Functionality

(*In brackets how it would roughly look and work in the program*)

Requirements	**Explanation and justification**
Ability to view both colour-coded environmental and social maps (either both visible in the same screen or a button to change from one to the other)	The maps will be in two different categories, environmental and social, so users should be able to either view both at the same time or be able to access both easily. The access to both environmental and social maps is key, since, as discussed before, they are very important issues in today's economics, and all my stakeholder and research also pointed towards them.
Ability to choose what metrics to include in the maps (in a tick-list)	This is one of the main functions of the program, as users should be able to select the metrics they want to see in their maps. This should also be as simple as possible to make it accessible to all users. The ability to select different metrics will allow users more access to different economic data (for research, education, etc.)
A way to reset the maps (button to click on the screen)	There will be a lot of metrics that can be selected, so it should be easy to reset the whole map and go back to the start. This will help users save time and make the program more complete.
A way to click on area/'zoom in' for more information (once clicked, the screen will display the raw data of the metric in table/bar chart/line graph form for the user to choose between)	This will display the other types of charts as a way to get more information on that area and its individual data. There should also be a way to go back to viewing the whole map again. The stakeholders found viewing the raw data on of their key concerns, so the feature is necessary.
A way to access and maneuverer the projection maps (displayed in the same colour-coded map manner)	This is another key feature of the solution, in which users will be able to see how economics will develop. The data will be calculated using linear regression with seasonal variance. Again, this should be as simple as possible so all users can access and use it easily. Both stakeholders and my research agreed that projections would be a good addition to the program
A way to select your policies in the economic simulator (by selecting from a drop-down menu or tick-list the policies they want – e.g.: 'raise direct taxes by 2%' – and displaying the data in the same map form)	The software needs the users' selected policies to calculate and display the future economics. Again, my stakeholders found the simulator would be a great addition and would allow them to have a more hands-on experience with economics.

A way to view the 'optimised' policy plan (by clicking a button on screen to display the 'optimised' set of policies and the result of them in the same map form)	The software will be able to calculate an 'optimised' policy plan with an algorithm, which users might like to use to compare their own plan against the 'optimised' plan and learn from it. Again, my stakeholders agreed it would be a great feature.
A way to access tips and instructions of the economic simulator (a button opens another screen with tips and instructions)	Some users may struggle to use the economic simulator, or might simply want to learn how to be more efficient in it, so tips and instruction should be offered and/or easily accessible.
A menu to access all the features (button to open menu with: current data maps, projections, simulator and help)	This will be implemented so it is easy for users to navigate the software.

Usability

Requirements	**Explanation**
Simple maps, line graphs and bar charts	The software should be as accessible as possible for all users, including those who may have no prior knowledge of economic graphs. For this reason, they should be simple and only include the key information. I will use maps, line graphs and bar charts as my stakeholders and research pointed that they offer an easier way to interpret economic data.
Obvious way to find more information (button on screen at all times and 'find out more' section in menu to access a screen with more information)	This will be screen(s) with short explanations on more information the user might want, such as where the data came from or how the data was calculated, so users can easily access it if they wish to. This will offer professional economists more information.
Obvious way to alter maps (tick-list menu with different metrics always on screen)	This is a key feature in the software, so it should be clear for the user how it can be accessed to improve user experience.
Obvious way to access projections and economic simulator (through menu to be accessed with a button)	They are also key features, so should be easily accessible.
Clear instructions on how the software can be used (button on screen at all times and 'help' section in menu to access a screen with more information)	For all features in the software, but especially the economic simulator since it could be harder to grasp. This will aid less experienced users to have a more seamless user experience.

Hardware and software

Requirements	**Explanation**
Computer specs: *[enough to run the software, amend this later]*	There will be a large amount of data that will need to be fetched, processed and displayed, so the computer should be able to run the software. The computer should also have enough space to store the RDBMS file.
Windows, Linux or Mac operating system	These are the operating systems that support the other software I need; Python, Numpy and Plotly.
Python with Numpy, Plotly and Flask	The software will run using Python with Numpy, Plotly and Flask, so they are needed.

Success Criteria

Criteria	How to evidence	Justification
Program shows simple maps	Screenshot of simple and uncluttered colour-coded maps of environmental and social data.	The software should be as accessible as possible for all users, including those who may have no prior knowledge of economic graphs. For this reason, they should be simple and only include the key information.
A way to select different metrics for the maps	Screenshot of selection menu in the program as well as testing showing that it works.	This is one of the main functions of the program, as users should be able to select the metrics they want to see in their maps. This should also be as simple as possible to make it accessible to all users. The ability to select different metrics will allow users more access to different economic data (for research, education, etc.)
A way to view both environmental and social maps	Screenshot of both maps in the program (and/or testing showing that it works if necessary)	I chose to divide it into environmental and social issues as they cover most – if not all – of the possible metrics and seems to be the more popular issues in modern economics (*featured in 'Doughnut Economics' by Kate Raworth*).
A way to find more information of an area	Screenshot of the data being displayed as well as testing showing that it works.	My stakeholders also wanted the ability to look directly at the raw data, so I'll add the ability to view this in different graphical forms.
Easily accessible instructions on how to use the maps	Screenshot of where the information is placed in the program, which should be in an easily accessible place. Instructions should also be as clear as possible (*may use a tester's opinion as evidence*)	For all features in the software, but especially the economic simulator since it could be harder to grasp. This will aid less experienced users to have a more seamless user experience.
A way to easily access the projections	Screenshot of how the projections can be accessed, which should be simple and easy to find, and a test showing that it works.	This is a key feature in the software, so it should be clear for the user how it can be accessed to improve user experience.

A way to easily maneuverer projections	Screenshot of how the projections look, which should be simple and uncluttered, and a test showing that it works.	Again, a key feature so easy access should be ensured.
Easily accessible instructions on how to use the projections	Screenshot of where the information is placed in the program, which should be in an easily accessible place. Instructions should also be as clear as possible (*may use a tester's opinion as evidence*)	Again, this is a key feature so easy access should be ensured, and instructions should be clear so users of any level could use it.
Simple design for economic simulator and 'optimised' plan	Evidence will be screenshot of this simple design.	The economics behind the simulator might be harder to understand for some users, so the design should be as simple and abstract as possible.
A way to select policies for the economic simulator	Screenshot of selection menu in the program as well as testing showing that it works.	Again, a key feature so easy access should be ensured.
A way to access the 'optimised' economic plan	Screenshot of from where it can be accessed and testing to show that it works.	Another key feature to the program my stakeholders wanted, so access should be easily available.
A somewhat accurate 'optimised' plan	Testing to show how it works and an economist opinion on its accuracy.	While impossible to make a completely accurate 'optimised' plan, my stakeholders (and myself) would want a credible plan.
Easily accessible instructions and tips on how to use the economic simulator	Screenshot of where the information is placed in the program, which should be in an easily accessible place. Instructions should also be as clear as possible (*may use a tester's opinion as evidence*), and tips should be accurate and informative (*may use an economist's opinion for evidence*).	As my stakeholders can have a wide range of economic knowledge, access to clear and informative instructions should be ensured to make the program accessible for everyone.
A menu to access all features	Screenshot of simple and uncluttered menu.	All features should be easily accessible.
Consistent aesthetics throughout	Screenshot of all different features.	This is mainly to have a consistent and cohesive program that all users could use and enjoy.

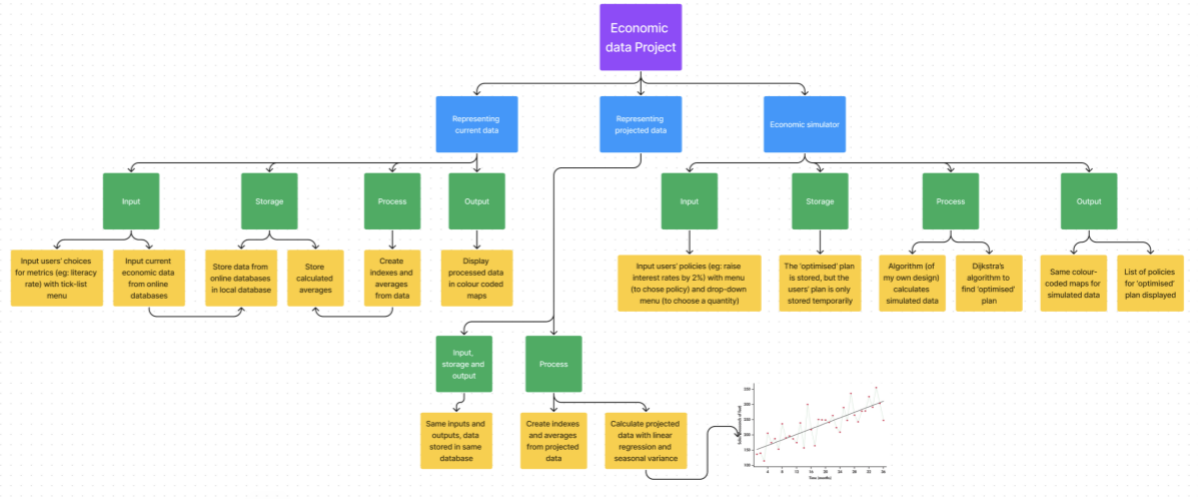
Design

Decomposition

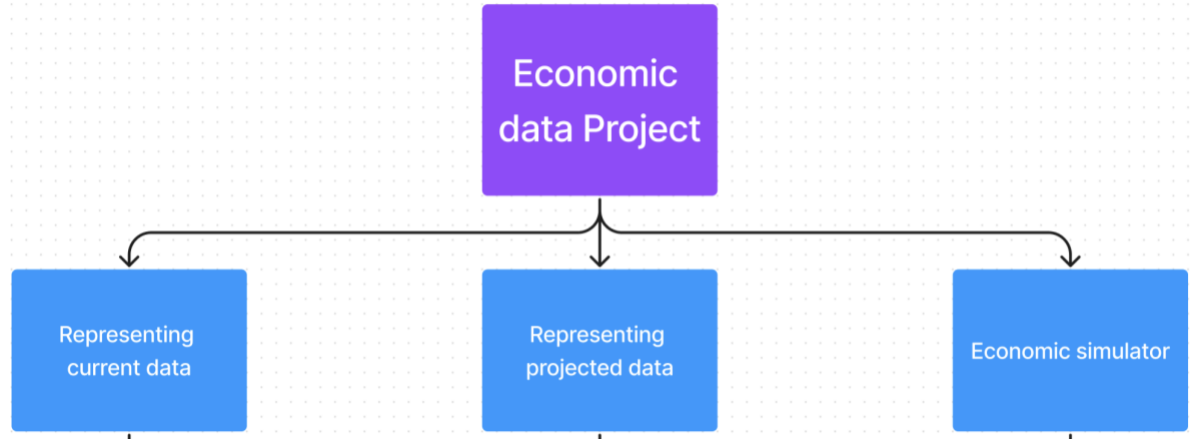
In order to design my solution, I decided to start by diving the idea into smaller and more manageable parts. This will allow me to understand the solution fully, design it more efficiently, and in the end allow me to develop in an easier and quicker way. A modular solution will allow me to use subroutines and classes to tackle each problem individually, making the development and testing stage easier.

I decided to use a hierarchy top-down design to be able to see visually each level, and so allow me to comprehend and design the solution more smoothly.

The following shows the full decomposition:



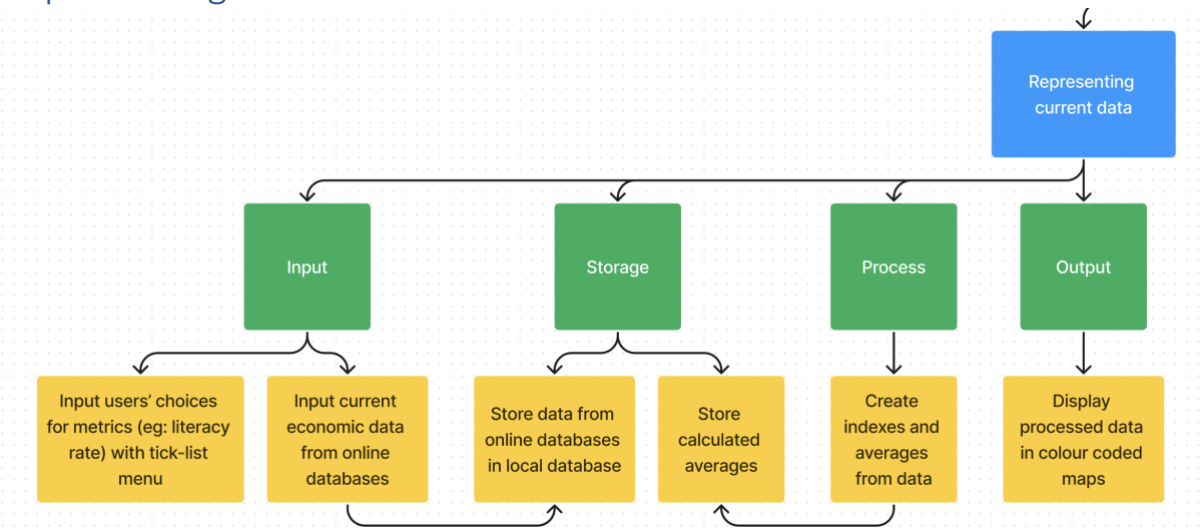
Main level



I started dividing the problem into these three main parts: current data, projected data, economic simulator. Since these are the three main parts of my solution and have distinct subroutines, variables and data structures attributed to them, I decided to tackle each individually. This will allow me to develop each part on its own and fully test it to make sure it works efficiently. It will also allow me to fully think through what each part needs (in terms of input, output, processes, and storage) and figure out how they may link together.

For each of the parts I decided to divide it further into input, output, process and storage.

Representing current data



Summary: represent current economic data in colour-coded maps. Features include user-chosen metrics and access to the raw data.

Representing the current economic data will involve imputing the data from online databases, processing this data into indexes, and creating colour coded maps out of this data. I decided to also add into the diagram how the inputted and processed data links to the storage.
Each of the main functions will be further discussed later, but briefly this is what each stage of the 'current data' section will do:

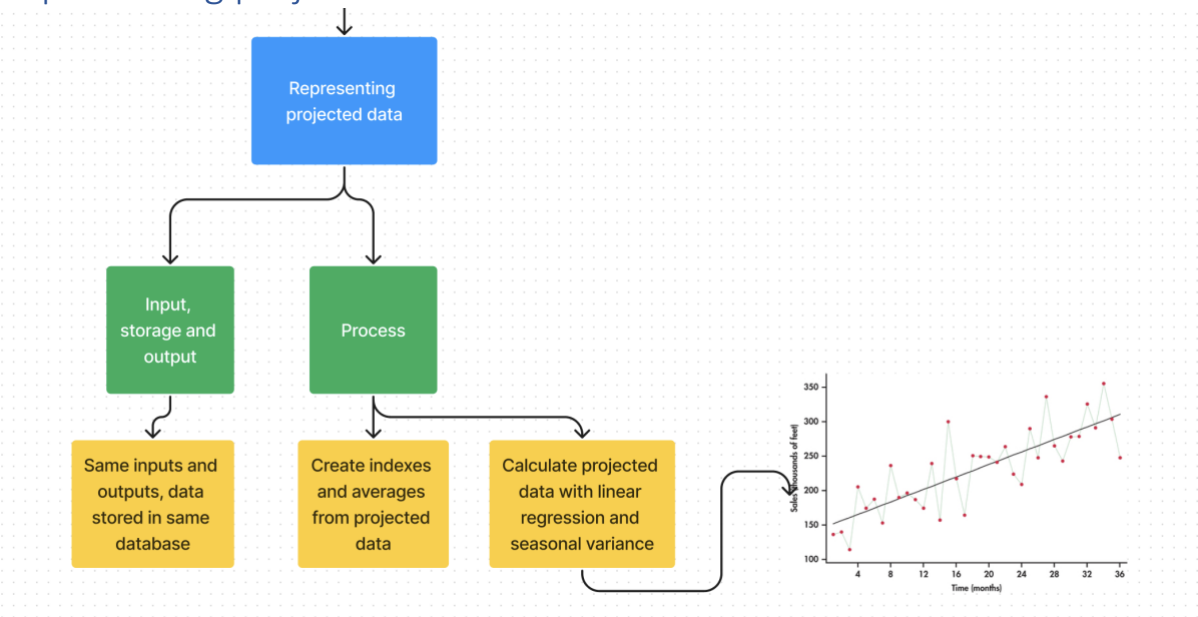
Input: this will involve uploading the data from .csv files from online databases (The World Bank, International Monetary Fund and Our World in Data), as well as getting the user's choices on what metrics to include in the colour coded map. On top of this, it will also input any of the user's clicks on particular areas of the screen, so they are able to access the raw data of their selected country.

Storage: for the 'current data' section, only the indexed data will be stored, which will then be used for the maps (storing the indexed data will avoid continually finding the index for each metric every time the maps are generated). Any need for the raw data will use the original .csv files.

Process: this will simply be calculating the indexes for each metric.

Output: this will use Plotly to both generate the colour-coded maps and table/chart when the user clicks to find the raw data.

Representing projected data



Summary: representing projected data in colour-coded maps. The projections are calculated with linear regression based on real past data, and the user is able to select from 8 quarters into the future. Users are still able to select which metrics to include, but cannot access the raw data.

Representing the projected data will involve quite similar input, storage, and output to the 'current data' section, but will vary the most in its process. Again, each major function will be further explained later, but briefly, what each stage will do:

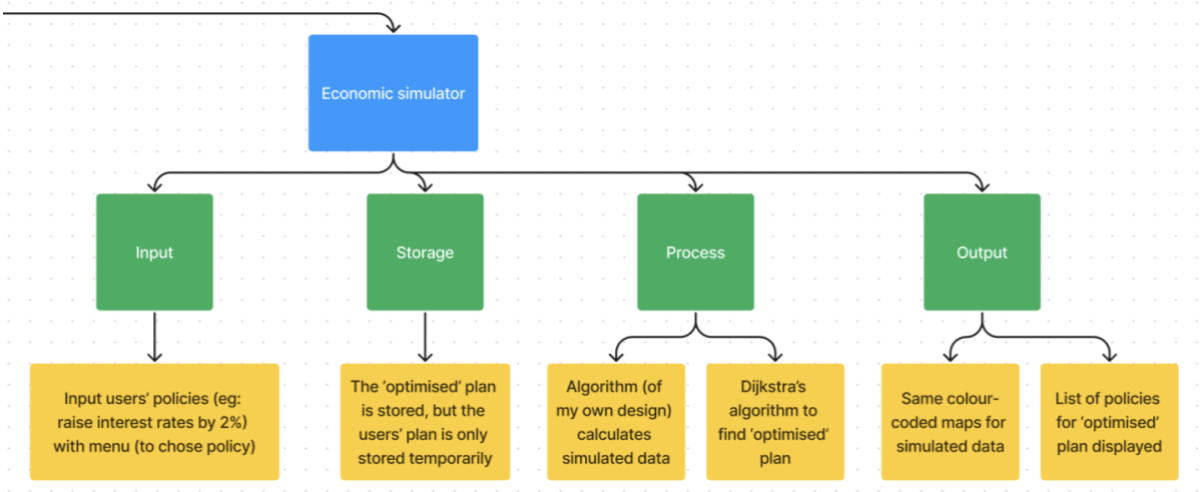
Input: this will involve uploading the data from .csv files from online databases (The World Bank, International Monetary Fund and Our World in Data), as well as getting the user's choices on what metrics to include in the colour coded map for projected data. The projected maps will not include the feature to access the raw data (since there isn't exactly raw data, only projected).

Storage: for the 'projected data' section, only the processed (projected) data will be stored, in its unindexed form. The data will be indexed only once it is needed for the maps. This is mainly due to the projected data also being needed for the economic simulator, and the fact that the algorithm for calculating the projected data takes longer than the algorithm for indexing the data.

Process: this will involve a regression line algorithm to calculate projected data as well as an indexing algorithm (same as the 'current data' one).

Output: this will use Plotly to generate the colour-coded maps of the projected data.

Economic Simulator



Summary: allowing users to simulate economic policy by selecting a set of policies and representing the effects of these in colour-coded maps. Users will also have the ability to access an 'optimised' plan. ***The economic simulator will work on a single country basis.*** *Users will not have the ability to access the raw data.*

The economic simulator will use both the projected data and a set of rules (based on various economic theory) to create predicted data. A version of Dijkstra's algorithm will also be used to create an 'optimised' algorithm. Once again, there will be full explanations for each feature later, but here is a brief discussion:

Input: this will involve inputting the data from the projected data, as well as getting the user's choices on what policies to use and what metrics to include in the colour coded map for projected data. The simulated maps will not include the feature to access the raw data (since there isn't exactly raw data, only predicted).

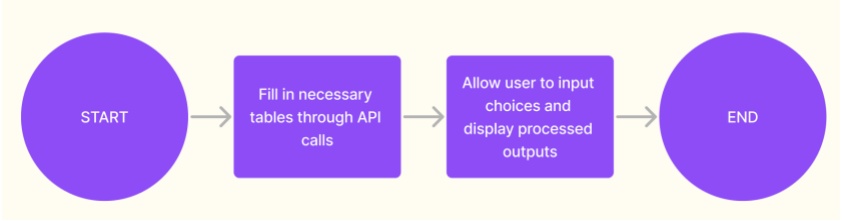
Storage: for the 'economic simulator' section, none of the predicted data will be stored, only the rules for it. This is due to the hundreds of possible combinations of metrics and policies, and the fact that the predicted that is simply the projected data with the economic rules applied.

Process: for the predicted data, it is simply applying the rules to the projected data. For the 'optimised' plan, it will be a Dijkstra's algorithm which find the "most positive" (largest) route to take in order to get the best economic outcome.

Output: this will use Plotly to generate the colour-coded maps of the predicted data, and then a simple output of the 'optimised' plan.

Whole program

This is just a simple visual of how the program as a whole will work, with all the necessary tables being filled before the user is allowed to change the maps. This is mainly to show there are **two stages** to the program, further explained in the algorithms section.



Key variables, data structures and classes

Key data structures

My key data structures will be five tables in a database. These will include a current data table, a projected data table, a 'rules' table for the economic simulator, and two minimum/maximum tables for current and projected data. These tables will be flat files, as there is no need for connections between each of the tables. Each table is further explained below.

Current data table

Current

Country	Life Expectancy Index	Literacy Level Index	...
UK	0.81	0.99	...
US	0.77	0.99	...
Canada	0.82	0.99	...
Germany	0.81	0.99	...
France	0.82	0.99	...
Italy	0.82	0.99	...
Japan	0.85	0.99	...

^ primary key

The 'Current' table will hold the indexed forms of the data. The program will select the relevant metrics and countries from the online databases, and then calculate and store the indexes. The primary key would be the country's name, as it will be completely unique.

As the current data map is likely to be the most used in the program, and the map uses the already indexed data, having a table with only the indexes will increase the efficiency of the program. Without this table, the program would need to calculate new indexes every time a new map was created.

Projected data table

Projected (not indexed)

Country	Quarter	Life Expectancy	...
UK	Q1	81	...
UK	Q2	81	...
...	...	...	...
UK	Q8	83	...
US	Q1	77	...
US	Q2	77	...
...	...	...	...

^ compound primary key ^

The 'Projected' table will hold the projected data for each country for eight consecutive quarters. From the online economic data, the program will calculate a regression line and project values for future quarters in each of the countries, which will be stored in the table. The data will only be indexed once the maps are being created. The key will be a compound key of the country and quarter.

Since the projected data is less likely to be used, and the data itself (not only just the indexes) will be used in other parts of the program, the unindexed data will be stored. This will mean slower execution for the projected maps, but will avoid having to recalculate regression lines for the economic simulator (which is a slower process).

Rules for the economic simulator

Rules

v these will work as multipliers on the projected data

Policy	Quarter	Happiness	...
Raise income taxes	Q1	-0.4	...
Raise income taxes	Q2	-0.2	...
...	...	...	...
Raise income taxes	Q8	0	...
Raise demerit taxes	Q1	0.1	...
Raise demerit taxes	Q2	0.1	...
...	...	...	...

^ compound primary key ^

the policy 'lower' will be the exact opposite to 'raise'
Eg: -0.4, 0.4

The 'Rules' table will hold the changes to each metric caused by the chosen policies for every quarter. These changes will work as multipliers on the projected data, and will be used to make the economic simulator (*further explained later*). It will only store the 'raise' policies, as the 'lower' policies will have the same multiplier, just negative (this will halve the amount of data stored and will avoid data repetition, and so making the program more efficient). The key will be a compound key of the policy and quarter (although the policy attribute will probably be abbreviated into 2-3 letters, such as income taxes to IT).

Minimums and maximums table

MinMax

ID	Life Expectancy	Literacy Level	...
Min	58	30	...
Max	87	100	...

ProjectedMinMax

ID	Life Expectancy	Literacy Level	...
Min	58	32	...
Max	90	100	...

The 'MinMax' table will hold the minimum and maximum values for each metric, and the 'ProjectedMinMax' will hold the minimum and maximum values for each projected metric. Both of these will be unindexed, as it takes more time to find the minimum and maximum data (instead of simply querying for the needed country), and the raw values are needed to calculate indexes for all of the maps. The key will simply be 'min' or 'max'.
(*For the projected data I decided to ignore the quarters, as it is quite unlikely the minimum and maximum values will change much in quarters*).

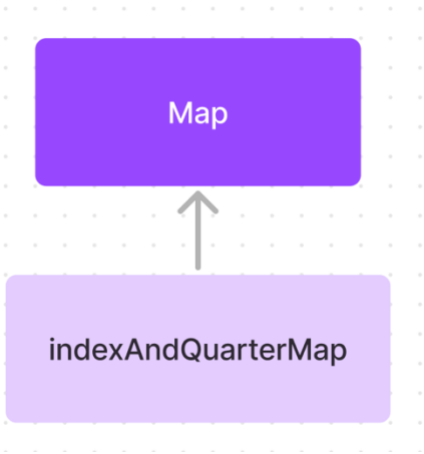
Key classes

The program might have other classes, especially with the 'optimised' plan, the key classes will be used for mapping.
The following is the complete code in pseudocode:

```
CLASS Map
        private mapColour
        private metricsIncluded
        private mapDatabase // from which database the map is from
        public SUBROUTINE new(inColour, inMetricsIncluded, inMapDatabase)
                colour <- inColour
                metricsIncluded <- inMetricsIncluded
                mapDatabase <- inDatabase
        ENDSUBROUTINE

        public SUBROUTINE draw()
                FOR country <- 0 TO country = 6
                        indexes <- mapDatabase[metricsIncluded] //accesses given database and gets the metrics needed
                        average <- avg(indexes) //finds mean of indexes
                        indexColour <- colour(average, mapColour) //translates mean into saturation of blue/green
                        OUTPUT image(country, indexColour) //outputs coloured map of the given country
                NEXT country
        ENDSUBROUTINE

        public SUBROUTINE setMetricsIncluded(inMetricsIncluded)
                metricsIncluded <- inMetricsIncluded
        ENDSUBROUTINE
ENDCLASS


CLASS indexAndQuarterMap INHERITS Map
        // used for the projected and economic simulator
        private quarter
        public SUBROUTINE new(inColour, inMetricsIncluded, inQuarter, inMapDatabase)
                super.new(inColour)
                super.new(inMetricsIncluded)
                super.new(inMapDatabase)
                quarter <- inQuarter
        ENDSUBROUTINE

        public SUBROUTINE draw()
                FOR country <- 0 TO country = 6
                        mapMetrics <- mapDatabase[metricsIncluded] //accesses given database and gets the metrics needed
                        FOR i <- 0 TO i = len(mapMetrics)
                                indexes[i] <- index(mapMetrics[i], ProjectedMinMax[min], ProjectedMinMax[max])
                                // finds index of each metric
                        NEXT i
                        average <- avg(indexes) //finds mean of indexes
                        indexColour <- colour(average, mapColour) //translates mean into saturation of blue/green
                        OUTPUT image(country, indexColour) //outputs coloured map of the given country
                NEXT country
        ENDSUBROUTINE

        public SUBROUTINE setQuater(inQuarter)
                quarter <- inQuarter
        ENDSUBROUTINE
ENCLASS
```

// S for social, E for environmental
SCurrentData <- new Map(blue, userChosenMetrics, CurrentData)
ECurrentData <- new Map(green, userChosenMetrics, CurrentData)
SProjectedData <- new indexAndQuarterMap(blue, userChosenMetrics, ProjectedData, userChosenQuarter)
EProjectedData <- new indexAndQuarterMap(green, userChosenMetrics, ProjectedData, userChosenQuarter)
// another class might be created for the economic simulator, for now this is the design
SEconomicSimulator <- new indexAndQuarterMap(blue, userChosenMetrics, SimData, userChosenQuarter)
SEconomicSimulator <- new indexAndQuarterMap(green, userChosenMetrics, SimData, userChosenQuarter)

Inheritance diagram:



The classes 'Map' and 'indexAndQuarterMap' will both create the maps for the program. The 'Map' class will draw the maps for the current data, and the 'indexAndQuarterMap' will draw the maps for the projected and economic simulator data (as these two require their data to first be indexed, and both utilise quarters, which the current data does not). The 'indexAndQuarterMap' will inherit from the 'Map' class, and use polymorphism for the draw() subroutine as well as add its own setQuarter() function.
The 'Map' class will have the attributes of colour, the metrics to be included, and what database it gets its data from. The child class will have the added attribute of what quarter.

The draw() subroutine finds the indexes required, averages them, finds the colour saturation for that average value, and finally creates an output for the country. It does this for each of the G7 countries.
The child class adjusts the draw() subroutine so it first calculates the indexes of the given data and then continues in the same way.

The setters I added where to change the metrics included and the quarter calculated, as these two variables could change depending on the user input.

The program will instantiate a total of 6 objects, 2 for each of the possible maps (one for social data, in blue, and one for environmental data, in green). They will be used when the program needs to generate the maps, with the program first setting the metrics (and quarters) from the user's input, and then creating the maps.

Justification:
For the maps I decided to use classes instead of subroutines, as that allowed me to create 6 instances to be used throughout the program without needing to constantly handle what the colour, database, metrics, quarters… of each map to be generated. The encapsulation this class allows will make programming other parts much simpler, and will allow for easier testing.

Key variables

The software I plan to create will have few key variables, almost all set by the user's input. The following are the most important variables, how and where they are used, and why they are key:

Key variable	**Explanation**	**Justification**
User's choice of metric(s)	Throughout all stages of the program the user will be able to select which metrics they want included in the maps and calculations, and they may change this at any point to receive different results. For example, a user might choose to view the health (*life expectancy*) and education (*literacy levels*) of all the G7 countries, and they would select this through a tick-list menu.	The user's choice of metrics may completely alter the map they are shown, and my stakeholder's found this to be an ability they wanted. Moreover, it will mean only some data will need to be retrieved from the local databases, therefore reducing the time taken and making the program more efficient.
The minimums and maximums of the data	Again, throughout all stages of the program the software will need to calculate the indexes for the maps. To calculate the indexes, the program will need the worldwide minimum and maximum, instead of using just the G7 data (e.g.: the minimum life expectancy may be 58, while the G7 minimum may be 77, so the indexes will be calculated with 58 as the minimum). (*This will be further clarified in the indexing algorithm*)	As my program only includes the G7, some of the wealthiest economies, some of the metrics may present misleading maps if only using their data (e.g.: while all countries will have high life expectancy, when only using the G7 data, the US will visually appear to have an incredibly low life expectancy while being relatively high in world-wide data). For this reason, storing and using the minimums and maximums of each metric will create more accurate maps.
User's choice of quarter	This will only apply to the projected data and economic simulator, but both will allow the user to select which quarter into the future they wish to look at.	Both the projected data and especially the economic simulator will show different changes to the data depending on the quarter the user is looking at. For example, in the economic simulator, most types of monetary policy will take four quarters to see an effect, making the user's choice of quarter very important.
User's choice of policies	This will only apply to the economic simulator, but the user will be able to select from a drop-down menu which	Again, the user's choice of polices may completely alter the simulation they create,

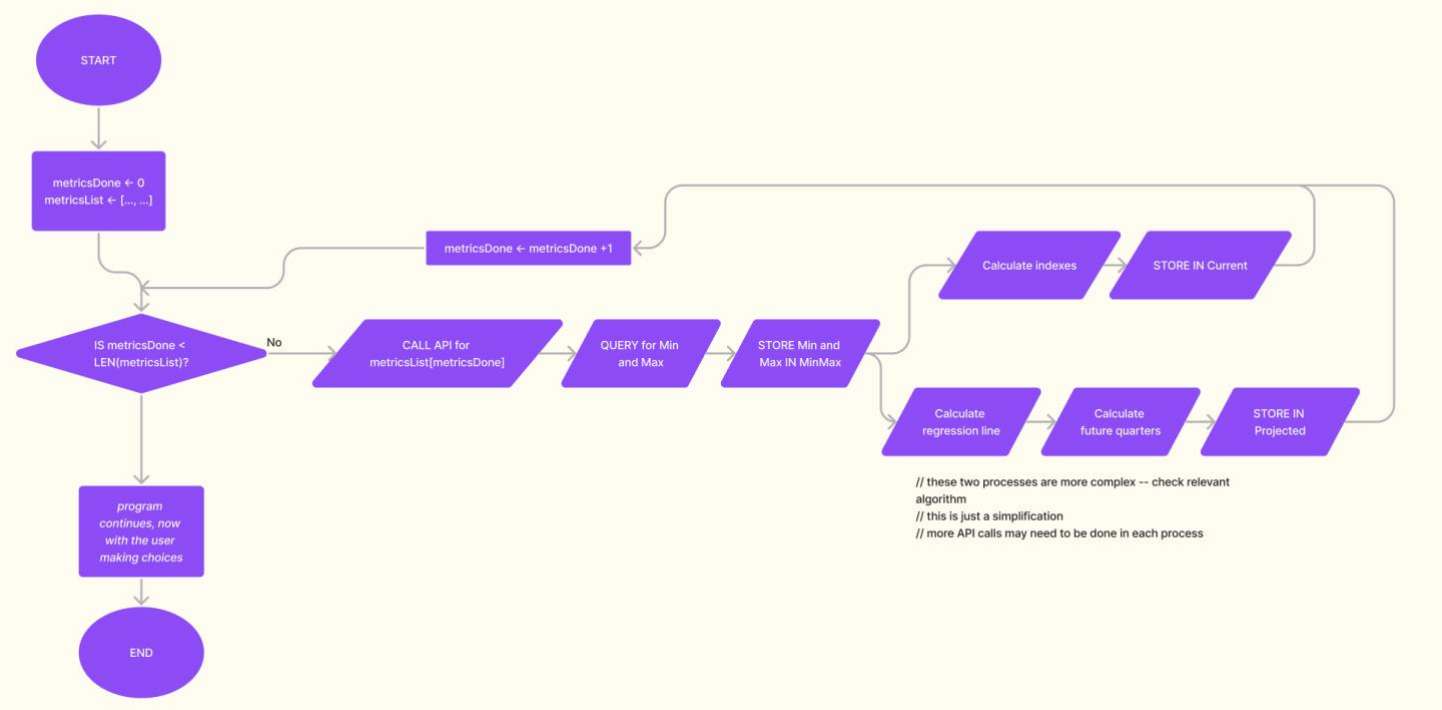
	policies they wish to enact. These choices will create different map simulations.	and my stakeholder's found this ability to explore different scenarios key, therefore the variable is key.

Algorithms

The following are all the major algorithms to be used in the program. They are presented in order of when they'd be used in the program (current data, projected data, economic simulator). I have also included which process(es) of the hierarchy chart they perform.

Program launch

This is used when the software launches, to set up all necessary tables.

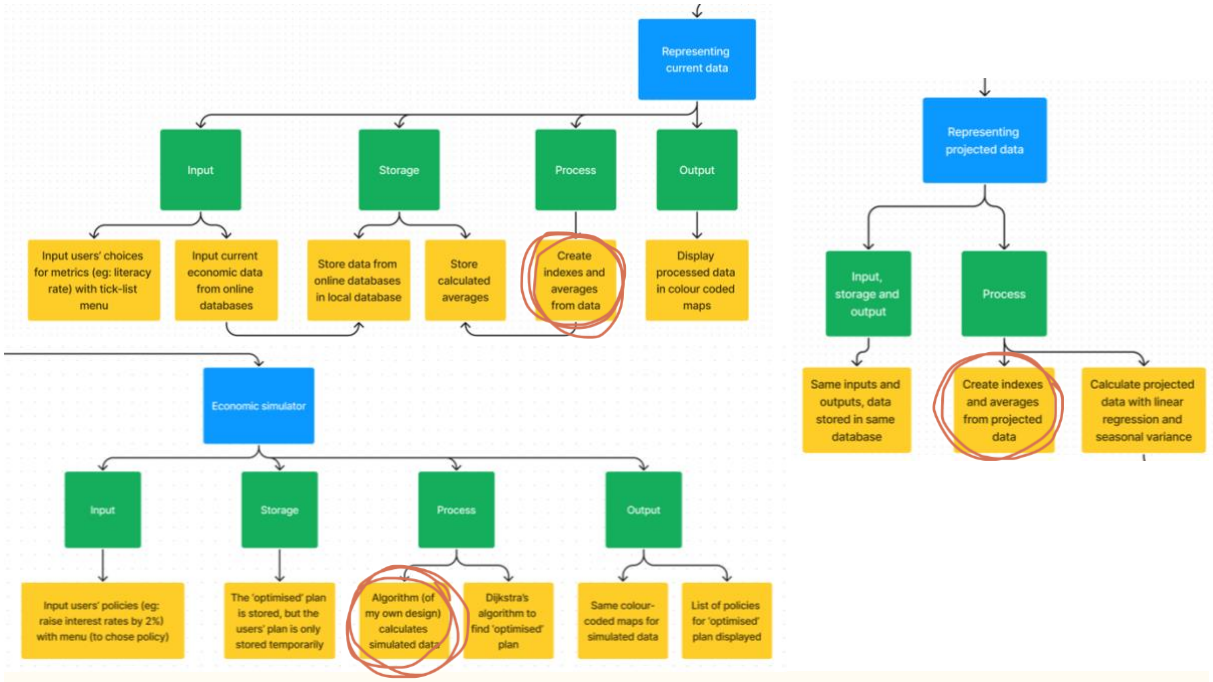


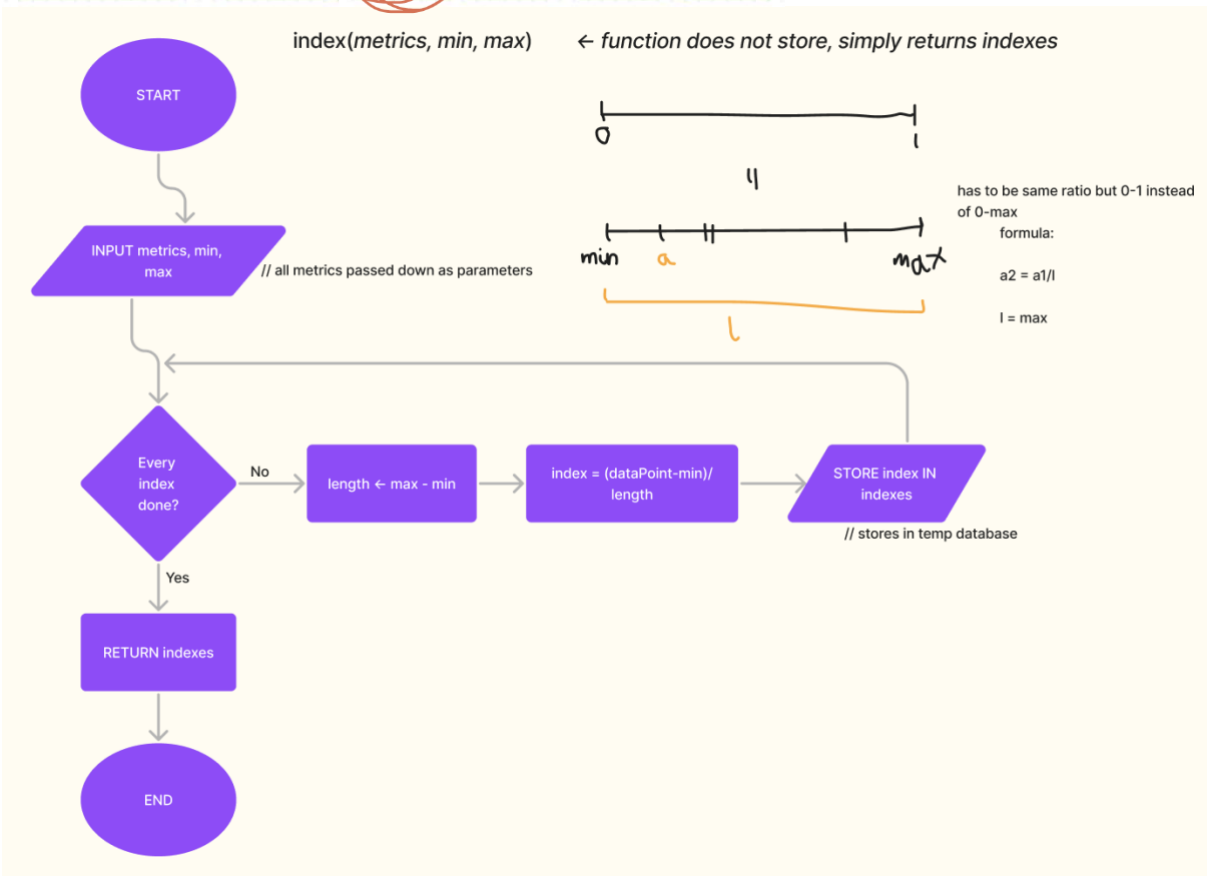
The program launch is used right when the program launches to fill in the tables necessary to have the program ready for the user. The program, for every metric, calls the relevant API for the data and queries for the minimums and maximums of the data. The program then fills out the Current and Projected tables for this metric. After this is completed for every metric and every country, the program moves on to the rest of the processes. *(The economic simulator already has a 'Rules' table set up, and simply uses the projected data as a base, so it does not require any initial processing).*

The actual processes to fill in the Current and Projected tables are more complex, so these algorithms are further explained below.

Indexes

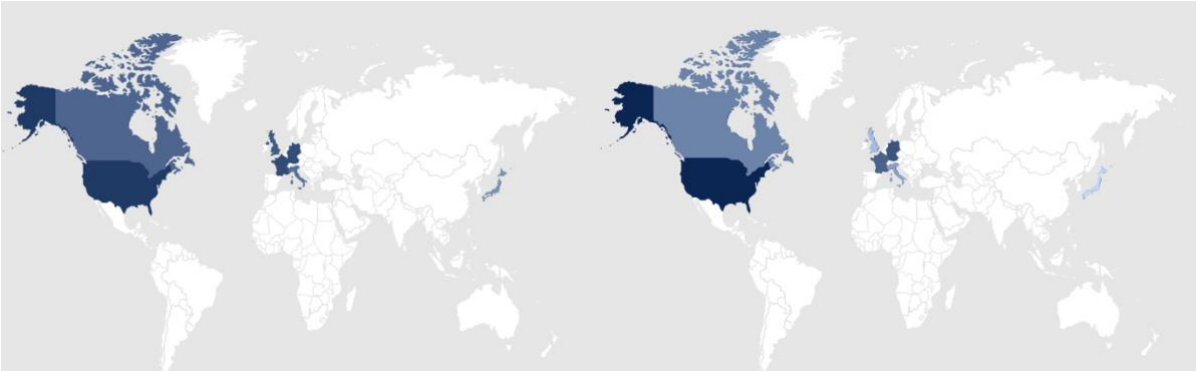
Used in:





The 'Indexes' function will input the metrics to be indexed and the minimums and maximums of the data. It will then index the data to make it a 0-1 value.

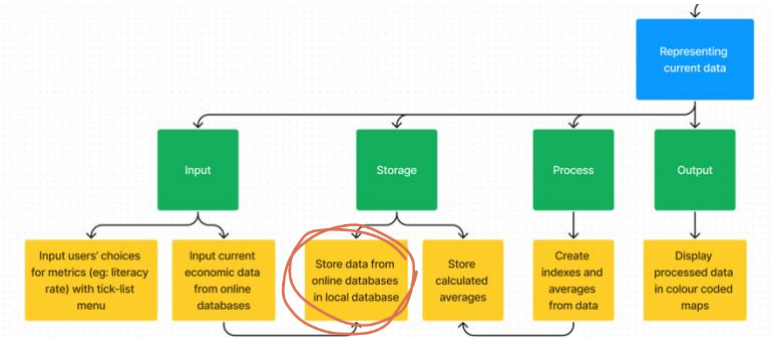
Why the minimums/maximums are needed:

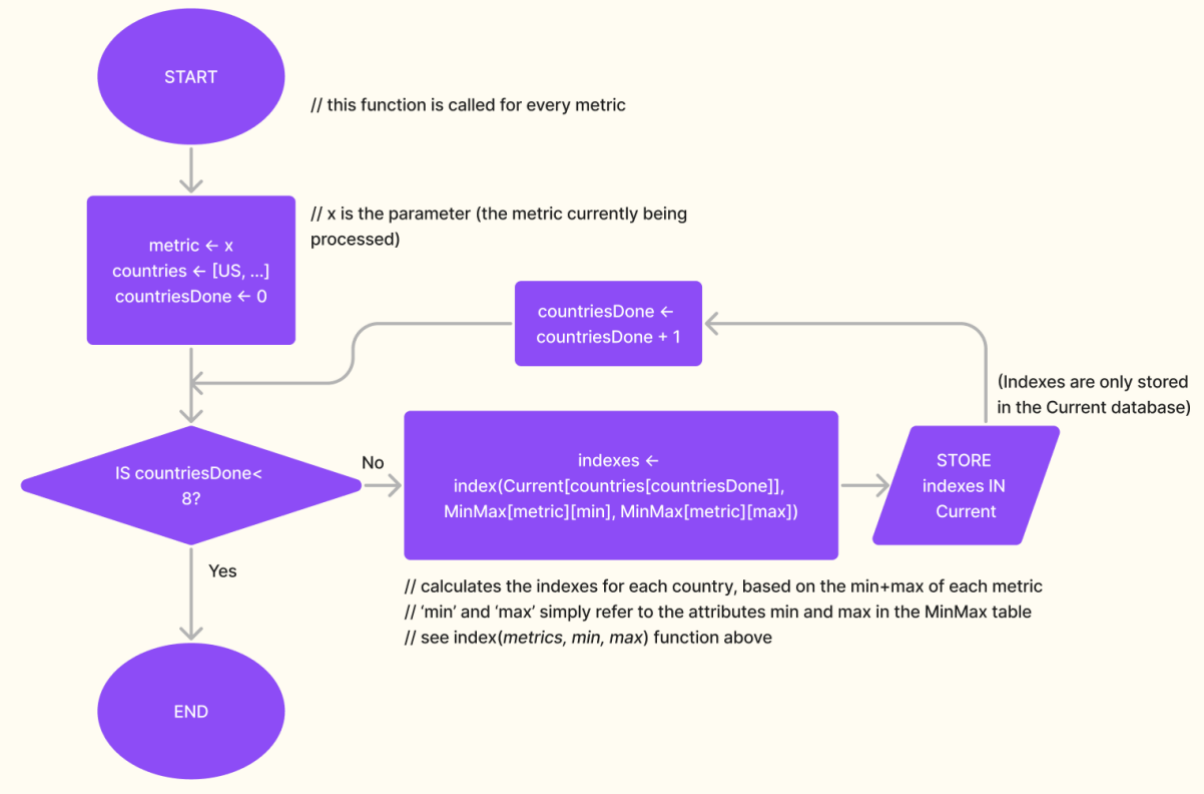


While the actual map might look more like the first one, if the minimums and maximums worldwide are not used, the map the program generates will look more like the second, in which countries of the G7 will be at both extremes, which they are incredibly unlickely to be.

Data simplification for 'Current' table

Used in:

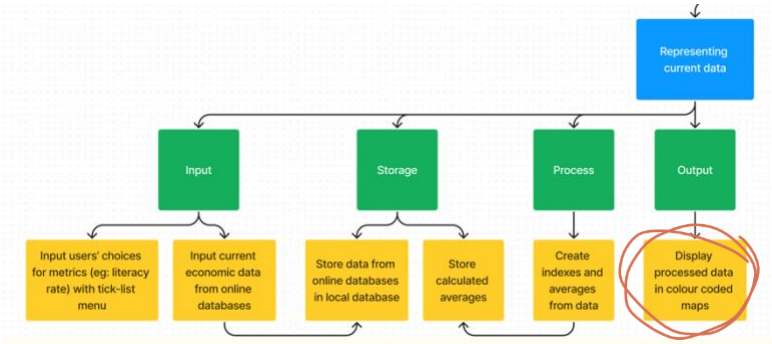


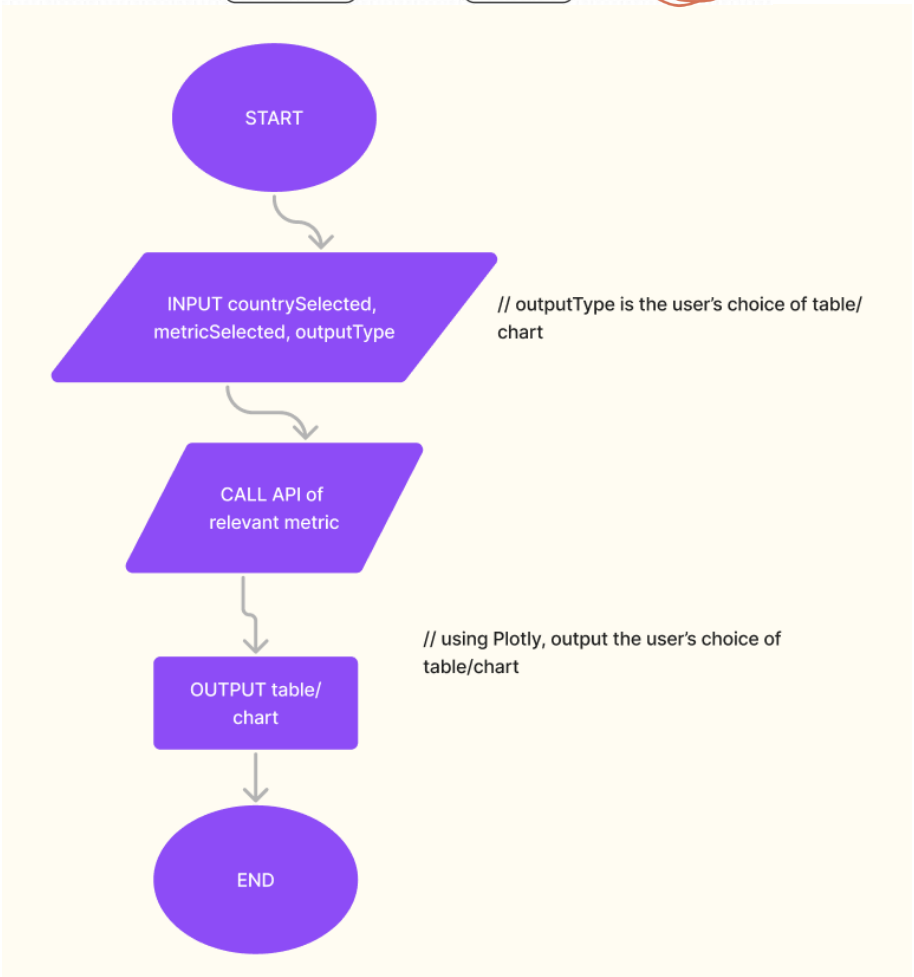


This algorithm simplifies the data for the Current table, based on the data retrieved during the launch of the program. The data is indexed and stored.

Click to see raw data

Used in:

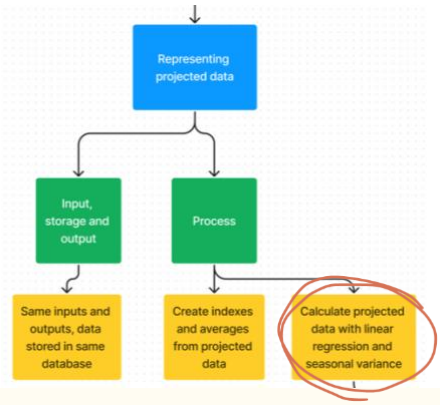


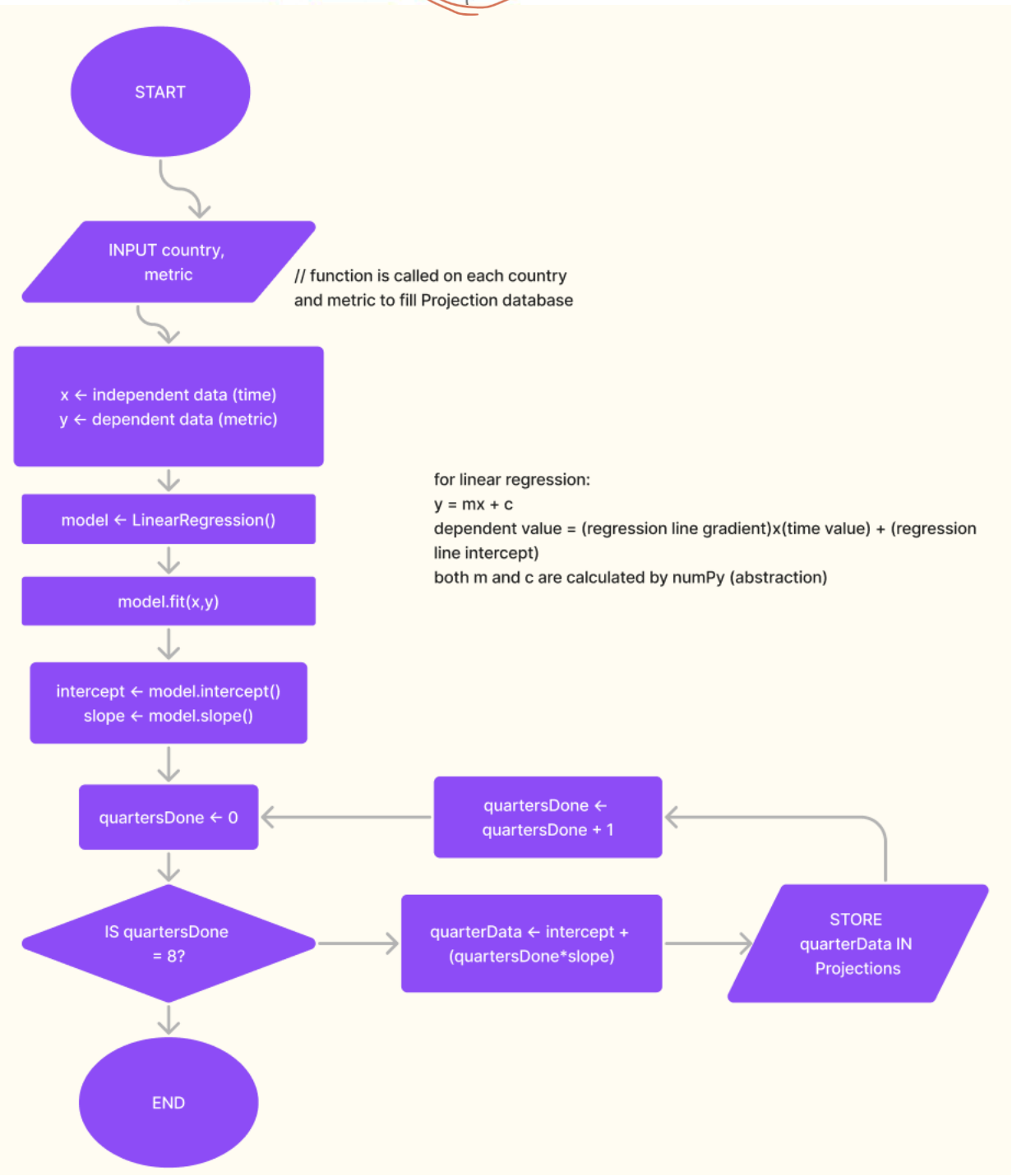


This function will input the relevant data from the online databases and output it based on the user's choice of table/chart. Although this function inputs from the online database again, it will only require querying and no further calculations, therefore it will not be too slow. Moreover, it is probably unlikely this feature will be used frequently.

Linear Regression

Used in:

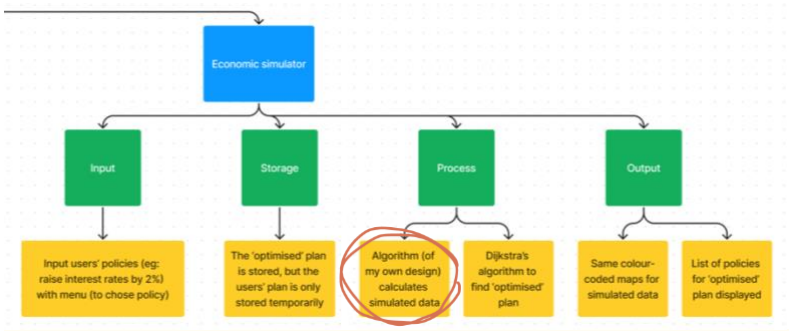


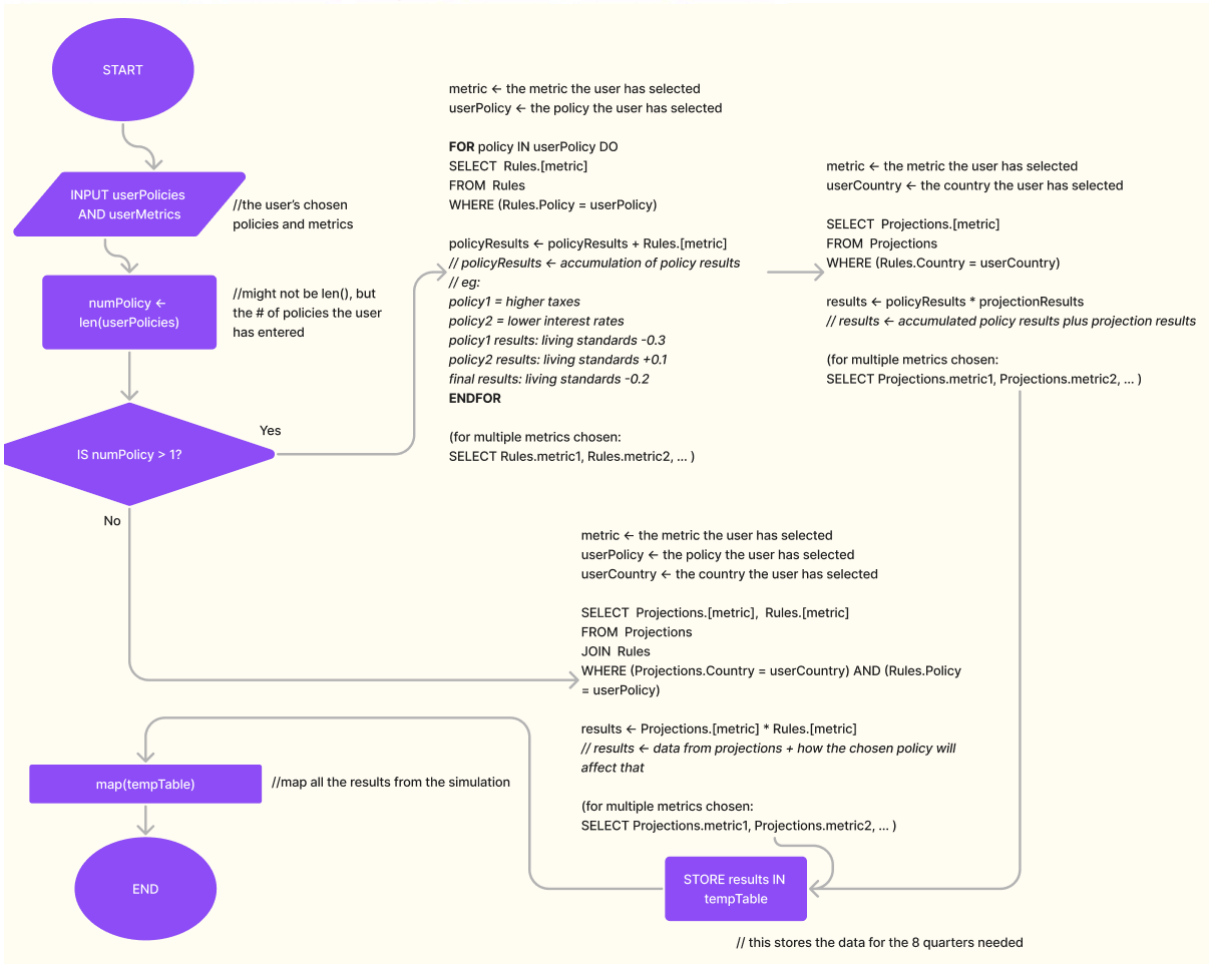


This algorithm first calculates the regression line of each metric (through a Numpy function), then calculates and stores the projected values for each for 8 quarters into the future.
For this subroutine, more data from the online databases will need to be inputted, as the linear regression needs multiple years of data to be able to project. This could mean that the inputting for the current and projected data might happen at different times or in different ways to make both of them the most efficient, but this can be dealt with during development iterations.

Economic Simulator

Used in:





The economic simulator will work by first imputing the user's choice of policy (and their chosen metrics), then calculate the predicted data by applying the 'rules' – the multipliers from the 'Rules' table – to the projected data. These calculated values will then be indexed and mapped. If the user selects more than one policy, the policy multipliers are first added together (*example shown in image*) and then the final value is applied to the projected data.

Complete example of the economic simulator:
Chosen policies: lower income taxes, raise demerit taxes.
Chosen metrics: happiness index.
Chosen quarter: Q1.
Chosen country: US.

Projected data table:

Country	**Quarter**	**Happiness index**
US	Q1	7.04

Rules table:

Policy	**Quarter**	**Happiness index**
Raise income taxes	Q1	-0.02
Raise demerit taxes	Q1	0.005

Projected MinMax table:

ID	**Happiness index**
Min	7.90
Max	4.12

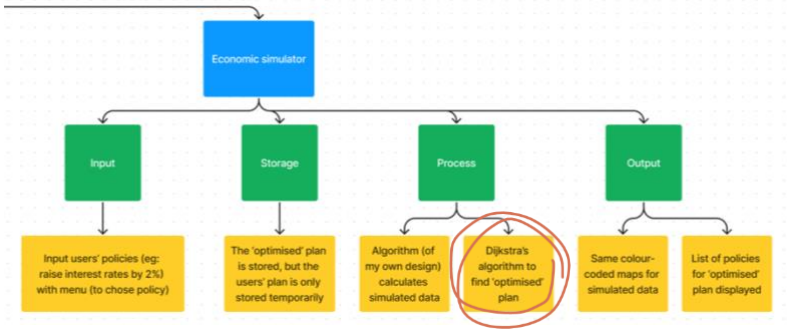
Lower income taxes: 0.02
Raise demerit taxes: 0.005
Total: 0.025

US new score: 7.04*(1+0.025) = 7.216

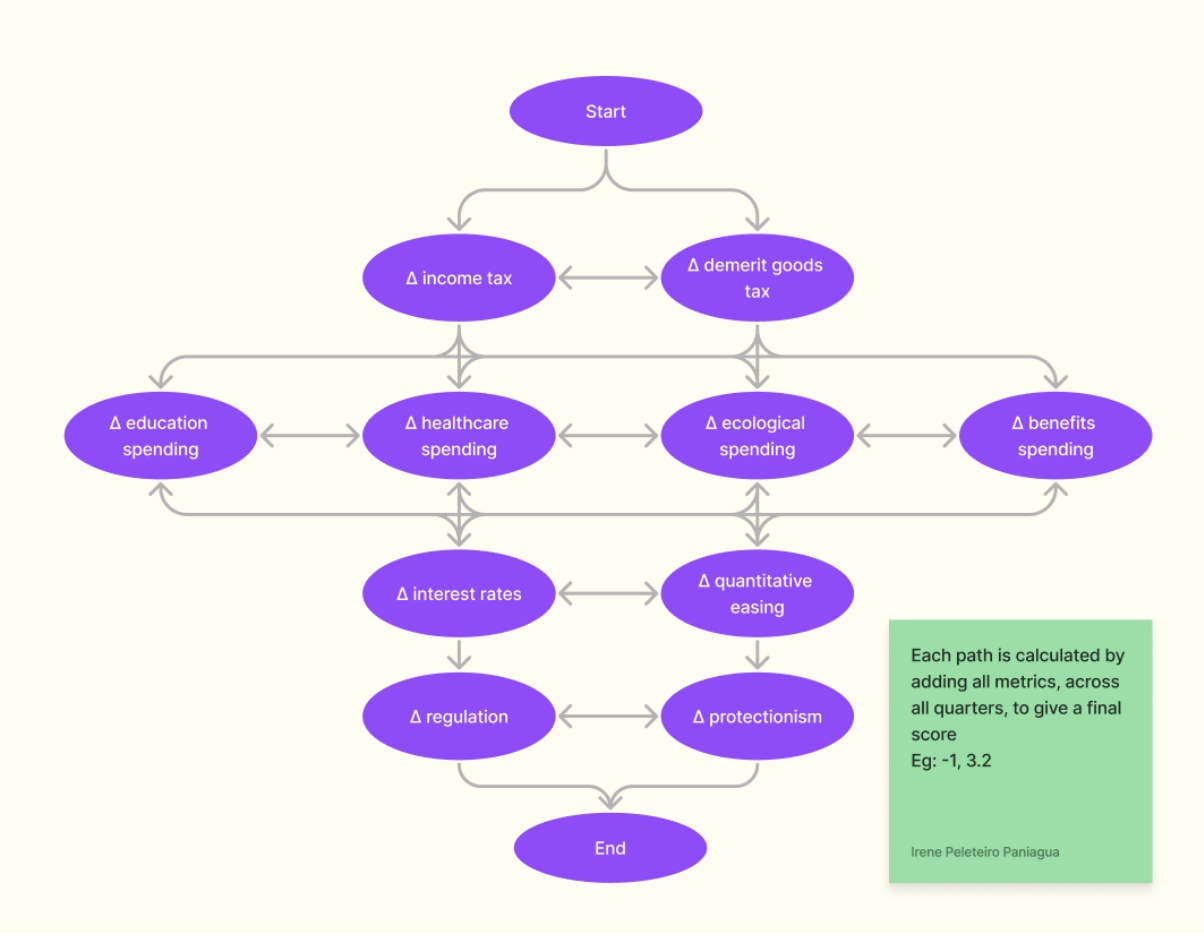
Then the index is calculated and the map is generated.

‘Optimised’ plan

Used in:



What the graph of policies will roughly look like:



The weightings between each path is calculated by adding the results from a policy on all the metrics, across all quarters. *(This does imply all metrics have equal weighting for an ‘optimal’ plan, which in reality they don’t, but this is a simplified version).*

For example, if a table had two metrics and two quarters as so:

Policy	**Quarter**	**Life expectancy**	**Literacy level**
Raise education spending	Q1	0	0.002
Raise education spending	Q2	0.001	0.003

The weighting of this path would be 0.006.

```
SUBROUTINE optPlan(rulesGraph, start)
        FOR v <- 0 TO v <- LEN(rulesGraph)
                dist[v] <- INFINITY // initial distance from source to vertex v is set to infinite
        NEXT v
        previous[v] <- NULL // previous node in optimal path is unknown
        dist[start] <- 0  // distance of start to itself is 0

        nodesLeft <- rulesGraph //all nodes in the graph are first unoptimised
        WHILE nodesLeft > 0 DO
                maxD <- 0
                FOR i <- 0 TO LEN(dist)
                        IF dist[i] > maxD
                        maxD <- dist[i] // finds node with the greatest distance from start
                        ENDIF
                NEXT i
                nodesLeft.DELETE(largestD)
                FOR neighbour <- 0 TO LEN(neighbours of largestD) // each neighbour of removed node
                        altRoute <- maxD + dist_between(maxD, neighbour) // finds if alt route is < direct route
                        IF altRoute > dist[neighbour]
                                dist[neighbour] <- altRoute
                                previous[neighbour] <- maxD
                        ENDIF
                NEXT neighbour
        ENDWHILE
        RETURN previous // whole route
ENSUBROUTINE
```

The economic simulator will be a simple Dijkstra's algorithm, just instead of minimising the distance of the whole route, it will maximise it (i.e.: it will maximise the benefit gained from the policy). The chosen route will then be returned as the 'optimised' plan.

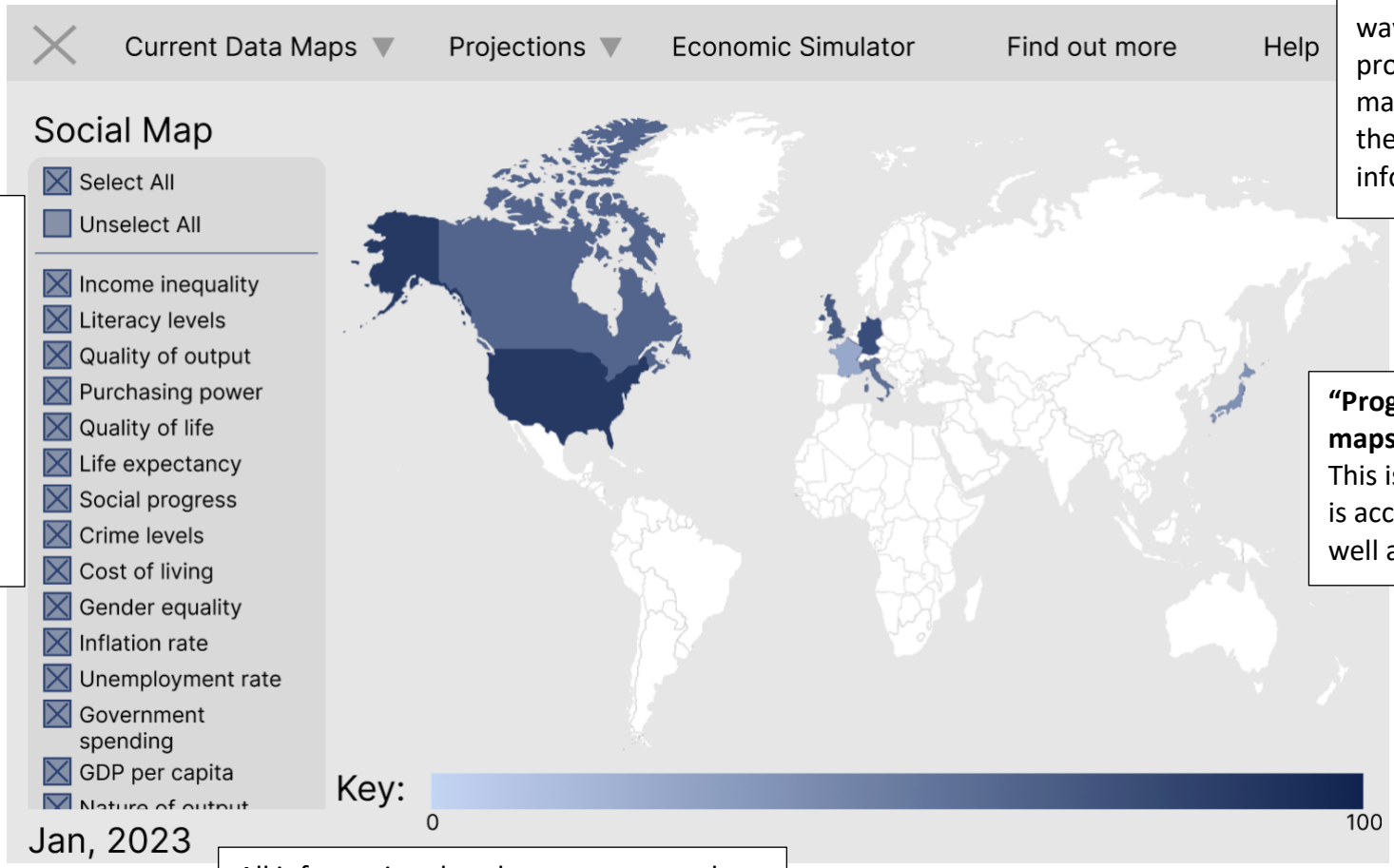
User Interface

Current data

This is the page users will be greeted with, the social map representing current data (the environmental data has the same design, just in green – see below). The following is a short analysis of how the design will answer my success criteria:

"A way to access/view […]"
This menu provides an easy way to access all parts of the program: both current data maps, both projected maps, the economic simulator, more information and instructions.

"A way to select different metrics for the maps"
This has to be simple for all users and is a key feature as it allows users to interact with the maps and select what metrics they want to include.



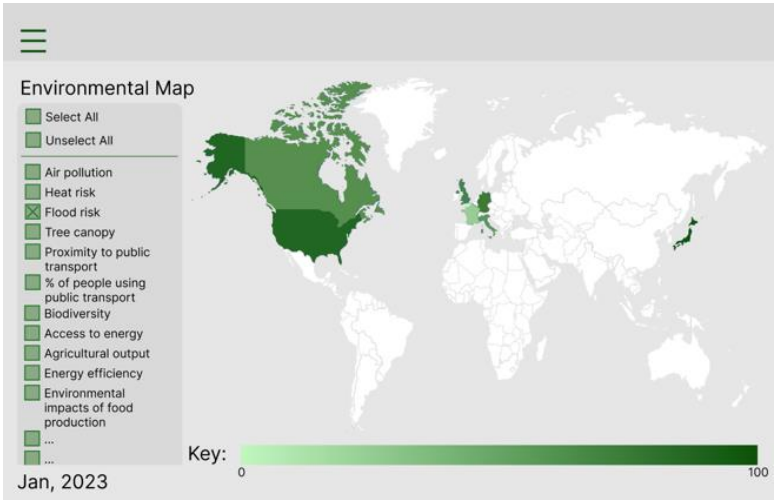
"Program shows simple maps"
This is to ensure the program is accessible for all users (as well as aesthetically pleasing).

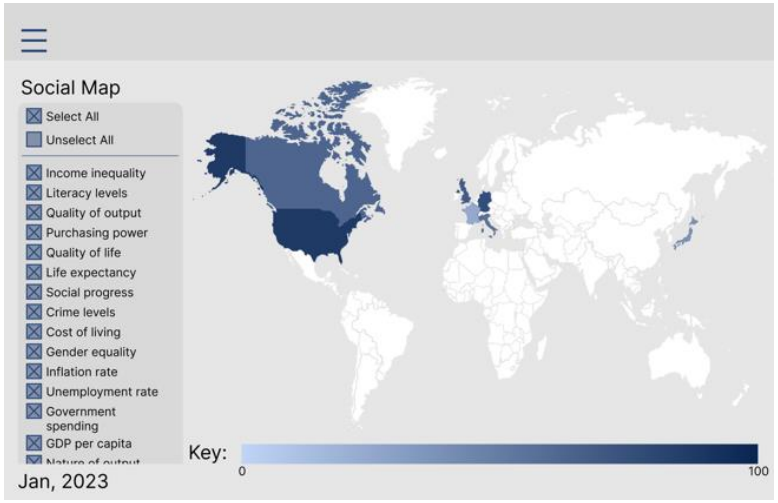
All information that the user may need (the date of the data and the key for the maps) is shown

Further justification of design choices

Map:

I decided to make the map the main focus point, as it is one of the most important features of the program. Although the G7 are really spread out and can look quite small, I decided to include the whole world map to give some perspective to users.





I decided to go for green for the environmental map (as it the colour for nature), and blue for the social map (as it is still a vibrant colour while not being harsh such as red). The to get more information, users will be able to click on the country they want.

Key:

The key may not be from 0-100, but it will generally show darker colours as "better" and lighter as "worse" (a high life expectancy will be on the darker end).

Metrics:

As one of the main features my stakeholders wanted, I decided to make it a large part of the screen. I also added the ability to select/unselect all to make the program easier to use. To select and unselect, users can simply click on the box.

Menu:

From clicking on the menu icon, the users will have access to all other parts of the program as well as a 'Find out more' section containing more info of where the data is from, how it is calculated, etc. and a 'Help' section to give more guidance on each part of the program. This will make the program accessible to all types of users.

Example screens:

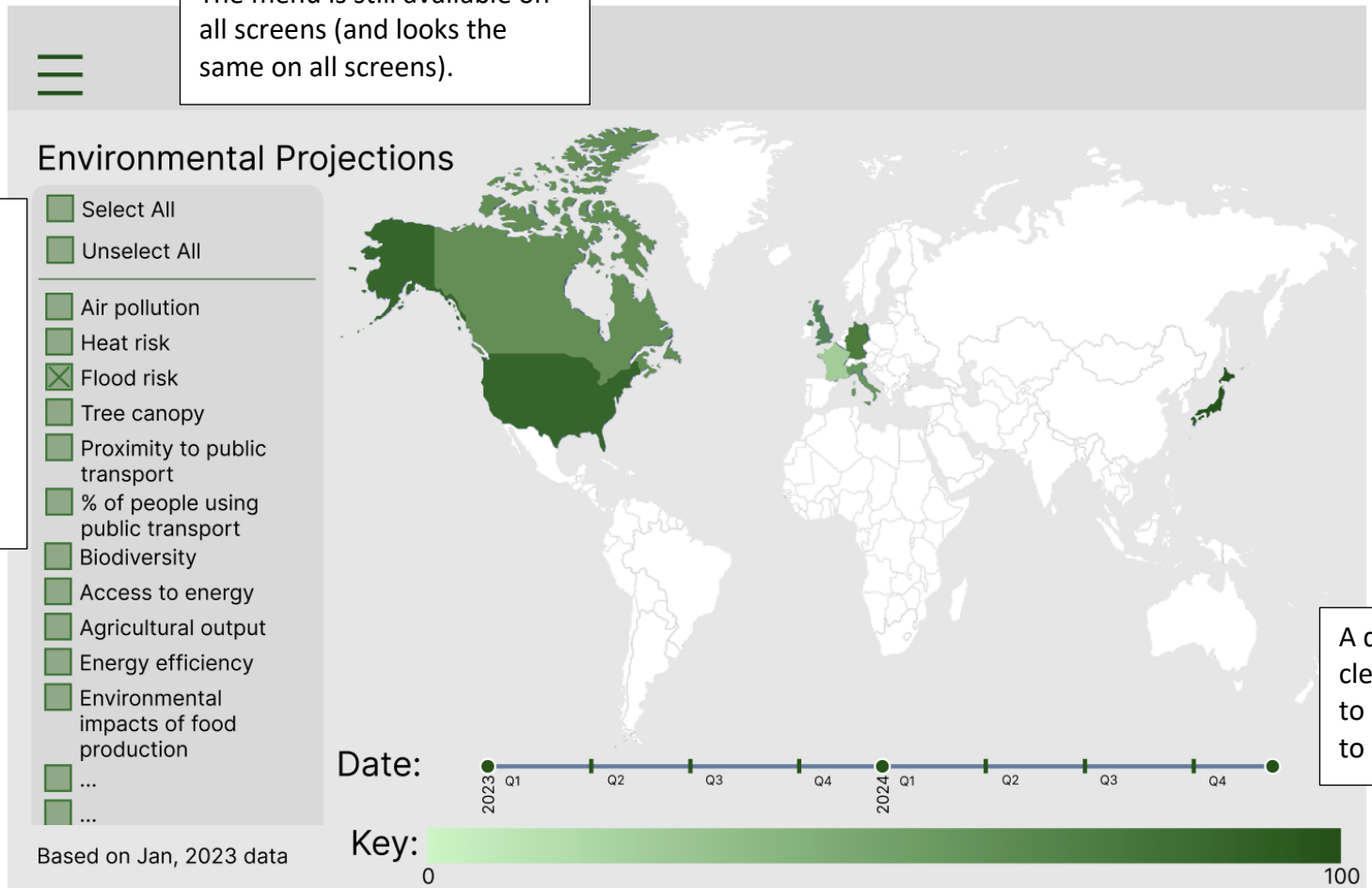
These are two examples of what the software will look like, as well as displaying how the menu would look.

Projected data

The projected data will have the addition of the ‘Date’ line, from which users will be able to select which of 8 quarters they wish to look at. The following is a short analysis of how the design will answer my success criteria:

“A way to access/view […]”
The menu is still available on all screens (and looks the same on all screens).

“A way to easily maneuverer projections”
The design for the projections has to be especially simple to allow users of all skill to access it without problem.



A dateline is added and clearly visible, allowing users to see the projections for up to 8 quarters in the future.

Economic simulator

For the economic simulator, users will immediately be able to select what policies they want, and on what country they want it on. To run it, they will simply press the ‘run’ button. Just like in the Projected data, users will be able to select what metrics they want included as well as what quarter they want to see. Users will also be able to load the ‘optimised’ plan by pressing its button.

I decided to include a ‘before’ and ‘after’ for the maps, to show users the changes their policies have achieved.
The ‘before’ map would be calculated in the same way as the current data maps.

“A way to select policies for the economic simulator”
The first thing users will see when accessing the economic simulator is the way to add their own policies and run them, which will reduce any confusion with how the program works.

Users can easily select what country they want for the simulation.

Users can still select the metrics they want for the maps. Users can also select the quarter they want to view on the dateline.

“A way to access the optimised plan”
The button for the optimised plan is easily accessible for users.

“Simple design for the economic simulator and optimised plan”
Like the projections, this part of the software might be a bit confusing for users, so a simple design will facilitate its use for users of all levels.

“Consistent aesthetics throughout”
The design across all features is consistent to ensure the software is cohesive and users can enjoy it.

Review with stakeholders

Since such a large part of the project is making sure the visual representation is accessible to users, I wanted to review with my stakeholders that they found the interface fitted their request. I sent them the following email along with a link to an interactive version of the interface and the explanations above.

Email I sent to my stakeholders

"Hello [stakeholder name],

I have created a prototype for the user interface for my project on visual representation of economic data. First, I'd like to remind you what the final proposed solution was:

In entering the program, you will be able to access two colour-coded maps based on environmental and social metrics respectively. You will be able to edit these maps and select what metrics to include, which will display the appropriate data. You will also be able to access more data on each country by clicking on it, displaying the individual data for this.
You will also be able to see projections on future economics displayed on the colour-coded maps.
Finally, you will be able to access an economic policy simulator, in which you can select your own policies and see how they will affect the projections. You will also be able to view an 'optimised' policy plan based on the current data.

From this solution I have started designing how the software will work and now have an interactive prototype of the user interface for you to test out. I'd love to hear your thoughts on the general aesthetic of the program, on whether the positioning and sizing of all the features is appropriate and on any changes that you think should be made. I have also attached a document describing the success criteria I am to hit with this design, so please also inform me if you find I am hitting all of these to a sufficient level.

Thank you,
Irene"

Responses

Fermin

"The look & feel is great, I love the layout, the graphics and the colours used, they clearly show the areas to focus on.

Navigation can be improved, as some options are still not available in the prototype, it would be helpful to shade them or somehow mark them so that you know what works and what is work in progress.

The economic simulator looks very promising, looking forward to more of that when its fully operational."

Jenny

“Very aesthetic & clean page - everything is clear, and it is easy to navigate.
In my opinion you hit all the success criteria you intended to achieve.

Here’s some EBIs (even better if):

- Great if user can zoom in & out so smaller countries/ particular region can be seen as needed
- Introduce country labels (user can select it to be displayed on the map or not)
- Bold titles & sub-titles to create contrast
- For projections, it would be cool if the word on the timeline (e.g. 2024, Q1) disappears when mouse is not on it, and reappear when you place the mouse on it or near it
- Might not be something you focus on at this stage but great to have a dark mode version”

Lourdes

“I find the choice of maps and different indicators as long as the top menus and drill down options very top point. It is useful to be able to analyse data using different keys and visualisations. I like the colours and the design of the maps.”

Analysis

Overall, my stakeholders were happy with the design and found it was able to complete all the success criteria I set. Most problems they had stemmed from the prototype itself, as I just created the necessary interactions to be able to *view* all the important user interface (and nothing that would truly carry out its processes). They also gave me some extra features they found could be useful, which I will keep in mind when developing the software.

Testing

For any type of software development testing will be crucial, as it will help me try and uncover any undetected errors and aid in fixing them before the program is given to users. I will test every feature on its own before linking them to any other, ensuring that the program, once whole, will work without bugs.

My testing will be destructive and will try to break the program – this way there is a lower chance of the program not working for the users. This will be done through various testing methods.

Testing methods

The testing thought the development prototype will be carried out on every feature individually, and will have a set of input data to be processed and a set of expected results to compare the output against.

While coding the test I will carry out are white box tests, and I will not document them as this will simply ensure the logic and structure of each feature is working. Once I am done with every feature, I will carry out a black box test, in which I will test if the function of each feature is carried out correctly (these I *will* document, with screenshots of the input, output and expected result, as well as summarise the test). If there are any errors, I will fix them before letting my stakeholders test out the feature.

Once the whole system is done, I will carry out an alpha test and ensure it is working, then let my stakeholders carry out beta testing to ensure it meets all of their requirements.

Testing of each feature

Below is a description of how I will test each major feature individually, as well as the data I will use.
Key for testing data tables: normal data, *boundary data*, **erroneous/invalid data**.

Importing data from online databases through API calls

This is a key part for the program, as it was incredibly important for my stakeholders (and the general purpose of the software) to have real economic data. This data needs to be imported from online databases, so I will need to make sure the data is correct and up to date.

Input	Expected result	Output
Find the value of GDP in the UK	[Value seen on online database is the same as the result]	
Find the value of happiness index in France	[Value seen on online database is the same as the result]	
Find the value of GDP in China	"Invalid selection. This country is not available"	
Find the value of RPI in the US	"Invalid selection. This metric is not available"	

The input will likely be simply checking the 'Current' table, and the expected result for invalid inputs may not be exactly in the same wording or may simply not be accessible to users. (RPI is invalid as the metric is not included).

Goal of testing:
Ensure the 'Current' table is importing the correct data values and no other data values.

Colour-coded map displaying averaged data

Again, this is a key feature of the program, so these tests will ensure it is working perfectly.

Input	Expected result	Output
Create a map with indexes: 0.1, 0.2, 0.4, 0.5, 0.6, 0.8, 0.9	Map of the G7, each with different and correct saturations	
Create a map with indexes: 0, 0, 0.1, 0.5, 1, 1, 1	Map of the G7, with correct saturations (2 at the lightest saturation, 3 at the darkest)	
Create a map with indexes: 0.1, 0.2, 0.4	Map of the G7, with correct saturations and the ones without an index left without any colour	
Create a map with indexes: 0, -1, 100, 19, 'a', 1, 1	"Invalid selection. These indexes are not between 1 and 0"	
Create a map with indexes: 0, 0, 0, 0, 0, 0, 0, 0	"Invalid selection. The map is made for 7 countries, not more"	

Again, the wording for the invalid results may not be the exact same, or may simply not be accessible for users to choose.

Goal of testing:
Ensure the map class and display functions are able to create the correct maps for the G7 countries, and is able to handle any invalid map combinations.

Ability to choose what metrics to include for these two maps

In order to interact and alter the maps, users need to be able to choose what metrics to include, so this will test the correct metrics are being chosen and averaged as they should be.

Input	Expected result	Output
Select GDP and show display map	Map of the G7, with correct saturations based on the GDP data	
Select GDP and happiness index and display map	Map of the G7, with correct saturations based on an average of the GDP and happiness index data	
Select all metrics and display map	Map of the G7, with correct saturations based on an average of all data	

Select no metric and display map	"Invalid selection. At least one metric must be selected"	
Select RPI metric and display map	"Invalid selection. This metric is invalid"	

Again, the wording for the invalid results may not be the exact same, or may not be available to users.

Goal of testing:
Ensure the metric selection works as it should (can select any available metric in any combination) and displayed the correct maps.

Ability to view projections on future economics based on linear regression

The projection algorithm should be able to output correct extrapolated data based on past data, so I will first ensure the results are correct. I will then also make sure it is projecting the correct maps.

Input	Expected result	Output
Select GDP and find the data for the 4th quarter into the future	[Value seen on 'Projected' table is the same as manually calculated value]	
Select happiness index and calculate the **indexed** data for the 6th quarter into the future	[Value outputted by the program is the same as manually calculated value]	
*Select all metrics and calculate the **averaged indexed** data for the 3rd quarter into the future*	[Value outputted by the program is the same as manually calculated value]	
Select no metric and calculate the averaged indexed **data for the 3rd quarter into the future**	"Invalid selection. At least one metric must be selected"	
Select all metrics and calculate the averaged indexed **data for the 10th quarter into the future**	"Invalid selection. Only the first 8 quarters into the future are available"	
Select GDP and display map for 1st quarter into the future	Map of the G7, with correct saturations based on an GDP index	
Select GDP and happiness index and display map for 2nd quarter into the future	Map of the G7, with correct saturations based on averaged indexes	
Select all metrics and display map for 5th quarter	Map of the G7, with correct saturations based on averaged indexes	

Select no metric and display map for 3rd quarter	"Invalid selection. At least one metric must be selected"	
Select all metrics and display map for 10th quarter	"Invalid selection. Only the first 8 quarters into the future are available"	

The first tests making sure the correct data is calculated will be more like white box tests, but I will document them as they are an integral part to the projections. I will most likely use my own values to ease calculation.

Goal of testing:
Ensure the projected data is being calculated in the correct way and can use any metric and any quarter. Ensure the correct maps are displayed from this data.

Ability to select from a range of policies in the economic simulator and Algorithm to create predictions based on users' policies

In order to interact with the economic simulator, users need to be able to choose what policies they want, so this will test the correct policies and the corresponding 'Rules' are being selected.

Input	Expected result	Output
Select "Raise interest rates by 2%" with GDP, and check the correct data is selected	[Value seen on 'Rules' table is the same as the result]	
Select "Raise interest rates by 2%" with GDP and happiness index, and check the correct data is selected	[Values seen on 'Rules' table are the same as the results]	
Select "Raise interest rates by 2%" and "Increase demerit taxes by 2%" with GDP, and calculate the changes to the metric	[Result is the same as the manually calculated value based on the data from 'Rules' table]	
Check the above tests work for various quarters into the future	[Results are the same as values on 'Rules' table / manually calculated values]	
Select "Raise interest rates by 2%" with all metrics, and check the correct data is selected	[Values seen on 'Rules' table are the same as the results]	
Select all policies with GDP, and calculate the changes to the metric	[Result is the same as the manually calculated value based on the data from 'Rules' table]	
Select no metric and check selected data	"Invalid selection. At least one metric must be selected"	

Select no policy metric calculate the changes to the metric	"Invalid selection. At least one policy must be selected"	
Select no country and check selected data	"Invalid selection. One country must be selected"	
Display the maps for the above tests	Map of the selected country, with correct saturation	

Like with the projected data, this will involve a bit of white box testing to ensure the data is being fetched and processed in the correct way before it is displayed as a colour-coded map.

Goal of testing:
Ensure the inputs (policies, metrics, country and quarter) all work for fetching the data and processing it. Ensure the correct maps are displayed from this data.

Algorithm to create 'optimised' set of policies

To allow users to explore what an 'optimised' policy plan might look like, a version of Dijkstra's algorithm will be employed. This testing will make sure this is working up to an acceptable standard so users can learn from it.

Input	Expected result	Output
Run the algorithm for a single metric (GDP)	[Given set of policies is the same as manually calculated set of policies]	
Run the algorithm for GDP and happiness index	[Given set of policies is the same as manually calculated set of policies]	
Run the algorithm for all metrics	[Given set of policies is the same as manually calculated set of policies]	
Select no metric and display map	"Invalid selection. At least one metric must be selected"	
Select RPI metric and display map	"Invalid selection. This metric is invalid"	

These tests will be both white and black box testing, as I will ensure the decisions made by the computer are correct for every step of the algorithm as well as the final results.

Goal of testing:
Ensure the 'optimised' plan is processed as it should be.

Testing checklist for User Interface

Once the program is complete, this is the checklist I will use to make sure the user interface meets all the *key* requirements:

Action	Working?
Metrics can be selected and unselected	
All metrics can be selected and unselected at once	
Metrics alter the maps displayed accordingly	
Colour-coded maps are displayed	
Key information about the maps (data and key) are displayed	
Users can switch between maps through the menu	
All maps have the appropriate colours	
All maps that need a dateline have them	
The dateline can be used to select a quarter and alters the maps displayed accordingly	
Users can access an 'information' section	
Users can access a 'help' section	
Users can select a country for the economic simulator	
Users can select and unselect policies for the economic simulator	
Users can view an 'optimal' plan for the economic simulator	

Furthermore, the following is a checklist for more features my stakeholders wished for, but are not *key* for the function of the program:

Action	Working?
Users can click on a country to see its raw data	
Users can interact with this raw data and view it in different formats	
Labels for the countries can be switched on/off	
Users can zoom in on the map	
Dateline labels only appear when the mouse is on it	

Validation

For this program, there are multiple points where validation may occur, mainly when importing and storing the data as well as user input validation, however, for the reasons I will explain below, validation is not a major part of the program, and is mainly done through design choices (rather than testing through code).
Parts of this section have been updated after the completion of the code to give more insight.

For the user input, I will ensure it is validated by restricting the possible inputs. Users will only be able to use checklists, radio buttons and drop-down menus, ensuring the user input can only be the allowed values. There will be no way for the input to be invalid.

For the importing of data, I will ensure data is validated by cleaning it in various ways. The first stage will be a pre-processing of the files to be imported, as I select the metrics I want and the specific measure to represent them (e.g.: p0p100_gini_pretax for income inequality). This initially ensures data is of the correct data type since I will not choose any qualitative measures and only use float data. Then, the data I need from the tables is selected and stored int the Current table in the way needed, which I tested multiple times to ensure there were no transmission errors. Moreover, the data will be directly imported from World Bank and Our World in Data (using their python libraries), so there should be no issues, but I still tested a few metrics to ensure there were no mistakes. Overall, the fact that the imported data was able to be stored in the Current table without causing any errors is indication that all the data is valid.

For the storage and processing of data, it is all based on the imported data, which as explained above will be validated, so there should be no need for further validations. When storing for the Current data, the data is mainly being selected and moved before indexing, all of which assume certain qualities of the table and data (e.g.: World Bank tables have country codes ITA, GBR, etc. and years YR2000, YR2001, etc.), so validation is guaranteed, as the format and data qualities are ensured during importing (especially since World Bank and Our World in Data are quite consistent). For projections and the economic simulator, it once again will make certain assumptions, but validation is ensured since they will be based on the Current table and my own Rules table.

Development plan

For my development process, I will use an iterative development. This will allow me to make sure every stage and function is working correctly and that the users' requirements are being considered at every stage by having my stakeholders review each prototype. The following is my plan for each prototype of development, but it may change as I start coding.

Between every prototype, I will send my stakeholders the prototype to play around with and I will fix any bugs that they've found and try my best to enact their feedback.

Prototype 1

This prototype will build the basic structure for the program and the current data section.

- Create 'Current' table and 'MinMax' table
- Create function to import data from online databases through API calls
- Create indexing algorithm
- Create map class and function to display them
- Create UI to select and unselect metrics

This first prototype may not have all metrics available, and just a smaller selection to avoid complication while developing the structure. This prototype may also only show one map: social, environment, or just a mix of the available metrics.

Prototype 2

This prototype will continue building the user interface and will add the projected data section.

- Create 'Projected' table and 'ProjectedMinMax' table
- Create linear regression algorithm
- Adjust API calls to also apply to projected data
- Create child map class to display projected maps
- Create UI to select quarters on the dateline
- Set up the menu

This prototype may still not have all metrics, but will have more available. Both social and environment maps will be available.

Prototype 3

This prototype will continue building the user interface and will add the economic simulator.

- Create 'Rules' table
- Create function for economic simulator
- Create UI to select policies and country
- Create Dijkstra's algorithm
- Create child map class to display economic simulator maps
- Create UI to run and view 'optimal' plan
- Adjust menu to access all of the above

This prototype will now have all metrics.

Prototype 4

This prototype will finalise the program, but it is likely not all of these functions will be added.

- Adjust menu to access information and help sections (*throughout development I will write to my stakeholders all the information and help they need for each prototype, so these sections will already be written*)
- Create function to click on map to get raw data
- Adjust UI so labels on country can be turned on/off
- Adjust UI so users can zoom in on maps
- Adjust dateline labels so they only appear when the mouse is on it

Development and testing

Prototype 1

During the development of the software, I decided to make a few changes to the design of the software. The following are explanations of my reasoning behind each choice, and how it will affect the design. Some of these will be better understood within context (as they are solutions to bugs/design miscalculations), and so will be expanded upon when discussing the testing.

Changes in Prototype 1

Current table

Change: indexes of latest year → indexes of the years 2000-2019

The previous design of the current table only stored the index of the latest year. This was mainly due to the fact I had planned to use queries instead of simply importing whole .csv tables (*explained later*). I thought that querying for multiple years would take too long and so opted for a single year, and that it would take up too much space. Nevertheless, with the current system I can had the indexes for the years 2000-2019 without an impact of performance, allowing users to access multiple years. It also makes the next stage, projections, a simpler set of calculations. If this is something my stakeholders do not want however, I will revert back to the old plan.

MinMax table

Change: holding minimums and maximums → removed

In the previous design I opted to store the minimums and maximums of the data in case of any necessary calculations after the tables had been completed. With the new design however, the minimums and maximums are directly integrated into the indexes of the Current table, and so shouldn't be needed for anything else. Just in case though, I verified there is easy lookup for the minimums and maximums in the data if necessary.

Libraries

Changes:

Previous: Plotly, Flask

Current: owid, wbgapi, pandas, json, plotly.express, dash (+ html, dcc, callback, Output, Input, Patch), dash_bootstrap_components

The number and type of libraries have changed a lot with development, so I will go through these by the section of code they are used in.

Importing data

Change: API calls → using the owid and wbgapi libraries

When first trying to import the data, I tried using IMF's API calls. It was a very difficult and long process (which my inexperience in APIs didn't aid), and it had an even more complicated process to find the codes for the all the metrics I needed. Therefore, I looked for an alternative and found that Our World in Data and World Bank both had libraries with very similar importing procedures. Both allowed you to search through their catalogue to find your metric, and then had simple functions to import that metric. They had different table formats, but I managed to merge them into the single Current table with ease.

Data management

Change: no library → pandas

I didn't know the exact format the data would be imported in, so I didn't prepare to use any library to manage it, but I now use the pandas library to create my Current.csv table.

Map creation

Change: Plotly → Plotly.express, json

This change was mainly due to me finding an easier way to do exactly what I needed for the maps using a more specified section of Plotly in combination with a geojson map. This allowed me to keep a pretty good user interface for the maps (which allows zoom control and hovering over countries to see their exact data value), have highly accurate maps and have an easy way to alter the maps through my code. It also allowed easy integration with the user interface.

For the maps there is another change for this stage. Since I have a single map so far, I focused more on ensuring the UI worked well and will create the map class in the next iteration.

User interface

Change:

Flask → dash (+ html, dcc, callback, Output, Input, Patch), dash_bootstrap_components

This change was mostly due to the fact that the Plotly documentation recommended the use of Dash as a way to interact with the colour-coded maps. It allowed easy integration, and the website still maintains an aesthetic close to the prototype.

Whole Program Algorithm

Change: data importing is done every time the program runs → data importing only needs to be done yearly

From the available data I was only able to get up to 2019, and since all the data is yearly there is no point in waiting for the data to import every time when the Current table can be created once and accessed at any point afterwards. Moreover, the importing is quite a lengthy process already, and it is not all the metrics yet, so it'd slow the user down by too much.

Indexing Algorithm

Change: simple minimum and maximum distribution → normal distribution

My original plan for the indexing algorithm was a simple linear index. However, due to the data having very large outliers (as either minimums or maximums), it heavily distorted the indexes. Therefore, I opted to use a normal distribution, with the indexes showing how far away the data was from the mean (the range being 2 standard deviations from the mean).

Code Analysis and Iterative Testing

The following will be an analysis of all the major parts of my code, by section, with the testing and error resolution I did throughout development. It also goes through the iterative tests I carried out for each section.

Explanation of the code is made through summaries, in-text comments (in green), and code analysis (in text-boxes over code). The errors and bugs I resolved will be highlighted in red.

Importing from Our World In Data

```
from owid import catalog
import pandas as pd


#print(catalog.find('life'))
# the above line is used to find the different metrics in the catalog

df = pd.DataFrame(catalog.find('life', namespace='demography').load())
# df holds all the data from OWID regarding life expectancy


df = df["life_expectancy_0"]
# here I am narrowing down the data to only hold the specific metric I need
```

Importing Our World in Data's catalogue, and pandas to manage tables

'demography' uniquely identifies the table I want. Different metrics have different ways of identifying them

For Our World in Data, the above is the process to import the data. I first search through the catalogue, using their function catalog.find(keyword), to search for the metric I want. Once I have more information on the exact metric I want, I load the table into a pandas Dataframe. From there, I select the single metric I want from the table, in this case "life_expectancy_0".

I did have to face an error with my method:

```
# OWID tables got updated - there are now multiple 'inequality' tables, so I need to specify
  further
# | this is the command I was using before, which now gave the error:
# |     ValueError: only one table can be loaded at once (tables found:
  |     world_inequality_database, world_inequality_database,
  v     world_inequality_database_distribution)
# df = pd.DataFrame(catalog.find('inequality').load())
df = catalog.find('world_inequality_database', version='2023-01-27').load()
```

This is what the table looks like once imported:

```
country,year,life_expectancy_0
Afghanistan,1950,27.7
Afghanistan,1951,28.0
Afghanistan,1952,28.4
Afghanistan,1953,28.9
Afghanistan,1954,29.2
Afghanistan,1955,29.9
Afghanistan,1956,30.4
Afghanistan,1957,30.9
Afghanistan,1958,31.5
Afghanistan,1959,32.0
Afghanistan,1960,32.5
Afghanistan,1961,33.1
Afghanistan,1962,33.5
Afghanistan,1963,34.0
Afghanistan,1964,34.5
Afghanistan,1965,35.0
```

This table has a multi-index (a composite primary key with 'country' and 'year'), and a single field (life expectancy)

Tests:
For this test, the inputs are inside the code. I did not add validation for invalid inputs as users cannot access them (shown later on).
I chose the year 2000 to test a single result, any other year would've done the same test.
The dfLife (data frame) in the inputs is the life expectancy table. Any other table from Our World in Data would've worked as well.

Input	Expected result	Output
Find the value of Life Expectancy in the UK `print(dfLife['United Kingdom'][2000])`	Online: 77.9	77.9
Find the value of Life Expectancy in Country `print(dfLife['Country'][2000])`	"Invalid selection. This country is not available"	KeyError: 'Country'
Find the value of RPI in the US `print(dfRPI['United States'][2000])`	"Invalid selection. This metric is not available"	NameError: name 'dfRPI' is not defined

These tests ensure the data is being imported correctly.

Importing from World Bank

```
import wbgapi as wb
import pandas as pd

# below line is used to find metrics in the WB catalog that include 'GDP'
#print(wb.series.info(q='GDP'))

dfAll = wb.data.DataFrame('NY.GDP.MKTP.CD')
```

Importing World Bank's library to access the data, and pandas to manage tables

'NY.GDP.MKTP.CD' uniquely identifies the table I want.

For World Bank, the above is the process to import the data. I first search through the catalogue, using their function series.info(q=Keyword), to search for the metric I want. Once I have more information on the exact metric I want, I load the table into a pandas Dataframe. World Bank tables have a single metric, so I don't have to narrow it down.

This is what the table looks like once imported:

allGDP.csv

```
economy,YR1960,YR1961,YR1962,YR1963,YR1964,YR1965,YR1966,
CAN,39239090176.0,41469849373.2046,45131300479.4114,48565
DEU,,,,,,,,,,,403898488000.0,448298200000.0,488766454000.
FRA,46834000000.0,50775000000.0,56906000000.0,63794000000
GBR,26155000000.0,27765000000.0,29017000000.0,30915000000
ITA,13014645760.0,14474595328.0,16263192576.0,18628194304
JPN,15950643462144.0,19263102386176.0,21860286726144.0,25
USA,543300000000.0,563300000000.0,605100000000.0,63860000
```

This table has a single index (a primary key 'economy'), and fields for all the years it covers.

Tests:

For this test, the inputs are inside the code. I did not add validation for invalid inputs as users cannot access them (shown later on).

I chose the year 2000 to test a single result, any other year would've done the same test.

The dfLife (data frame) in the inputs is the life expectancy table. Any other table from Our World in Data would've worked as well.

Input	Expected result	Output
Find the value of GDP in Italy `print(df.loc["ITA", "YR2000"])`	1.24 trillion	1241512900000.0
Find the value of GDP in Country `print(df.loc["Country", "YR2000"])`	"Invalid selection. This country is not available"	KeyError: 'Country'
Find the value of RPI in the US `print(dfRPI.loc["USA", "YR2000"])`	"Invalid selection. This metric is not available"	NameError: name 'dfRPI' is not defined

Formatting (World Bank Data into Our World in Data format)

Since the Our World in Data data was in the format I wanted, I had to format World Bank's data to the same style.

Our World in Data has a multi-index (a composite primary key with 'country' and 'year'), and fields for each metric. World Bank has a single index (a primary key 'economy'), and fields for all the years it covers – each table is a single metric.

This algorithm formatted World Bank data into the Our World in Data format:

```
# used to format the WB dataframes to the way the OWID dataframes are
def formatWB(df_in):
    countries = ["Italy", "United Kingdom", "United States", "France", "Japan", "Germany",
                 "Canada"]
    tuples = []
    for country in countries:
        for year in range(2000,2020):
            tuples.append((country,year))


    # creates multilevel index for each country with the years 2000-2022
```

This creates a list of tuples which will be used as the multi-index (it will look like this: [(Italy, 2000), (Italy, 2001), …]

```
    index = pd.MultiIndex.from_tuples(tuples, names=["country", "year"])

    dataList = []
    for country in ['ITA','GBR', 'USA','FRA', 'JPN', 'DEU', 'CAN']:
        for year in range(2000,2020):
            year = 'YR'+str(year)
            dataList.append(df_in.loc[country, year])

    # turns data and index into dataframe
    df_out = pd.Series(dataList, index=index)
    return df_out
```

This creates a list of all the data in the order required for the dataframe.

The data is reformatted by creating a multi-level index that is the same as Our World in Data, and then creating a list of all the data in the same order as the index. This is then combined to make the dataframe.

Data Selection

```
# the following further reduces the data (so only the G7 and the years I need are used) and places all of this on a new dataframe
# this is used for Our World in Data
def selectCountries(df_in):
    df_out = df_in.loc[("Italy",2000):("Italy",2023)]
    for country in countries:
        temp = df_in.loc[(country,2000):(country,2023)]
        df_out = pd.concat([df_out, temp])
    return df_out
```

Initialises dataframe

All G7 countries

```
# for World Bank, the country selection is done as the data is imported
df = wb.data.DataFrame('NY.GDP.MKTP.CD',['ITA','GBR', 'USA','FRA', 'JPN', 'DEU', 'CAN'])

# for importing the whole table, the following is done
dfAll = wb.data.DataFrame('NY.GDP.MKTP.CD')
# ensuring only countries are used for the indexing, not regions
regions = ['WLD', 'HIC', 'OED', 'AFE', 'AFW', 'ARB', 'CEB', 'EAP', 'EAR', 'EAS', 'ECA', 'ECS',
           'EMU', 'EUU', 'IBT', 'IDA', 'IDB', 'IDX', 'LAC', 'LIC', 'LMC', 'LMY', 'LTE', 'MEA',
           'MIC', 'MNA', 'NAC', 'PRE', 'PST', 'SAS', 'SSA', 'SSF', 'TEA', 'TEC', 'TLA', 'TMN',
           'TSA', 'TSS', 'UMC', 'IBD']
dfAll = dfAll.drop(index=regions)
```

Regions error

As I was developing the indexing algorithm for the World Bank data, I kept getting incredibly large values for the minimum and maximums, even when applying the normal distribution (explained below). Taking a closer look at the data, I realised it included a lot of regions made up of multiple countries (eg: Europe, World), which was driving all my values up and distorting the indexing. To prevent this, I delete all the regions with .drop().

The above ensures the dataframes only consists of the G7, with different methods for the two types of data imports.

Tests:

For this test, the inputs are inside the code. I did not add validation for invalid inputs as users cannot access them (shown later on).
I chose the year 2000 to test a single result, any other year would've done the same test.
The dfLife (data frame) in the inputs is the life expectancy table. Any other table from Our World in Data would've worked as well.

For this test I am verifying only the G7 are stored in the table:

Input	Expected result	Output
Find the value of GDP in Italy `print(dfGDP['United Kingdom'][2000])`	1.24 trillion	1241512900000.0
Find the value of Life Expectancy in China `print(dfLife['China'][2000])`	"Invalid selection. This country is not available"	KeyError: 'China'

Indexing Algorithms

The following is Our World in Data's indexing algorithm:

```
def indexOWID(dfFull, dfCut):       # dfFull contains all countries, dfCut contains only G7
    countries = ["Italy", "United Kingdom", "United States", "France", "Japan", "Germany",
                 "Canada"]

    dfFull = dfFull.swaplevel()     # levels are swapped around (now year, country) so it can
                                                be iterated through
    for year in range(2000,2020):
        dfInfo = dfFull.loc[year].describe()   # gets summary statistics of the data
        mean = dfInfo.loc['mean']            # finds mean for that year
        stdDev = dfInfo.loc['std']           # finds standard deviation for that year
        minval = mean - 2*stdDev                  # creates a minimum -2 stds away from mean
        maxval = mean + 2*stdDev                  # creates a maximum +2 stds away from mean
```

Indexing design fix

The old code was the following:

```
maxval = max(dfFull.loc[year])
minval = min(dfFull.loc[year])
```

```
        if minval < 0:
            minval = 0                          # ensures minval is above 0

            dfCut = dfCut.swaplevel()           # levels are swapped (now country, year) so
                                                                    the years can be searched for

        if minval > min(dfCut.loc[year]):
            minval = min(dfCut.loc[year])
        if maxval < max(dfCut.loc[year]):
            maxval = max(dfCut.loc[year])       # ensures the G7 do not surpass the maximum

        dfCut = dfCut.swaplevel()       # levels are swapped again (now year, country) so
                                                                    iteration can continue
        length = maxval - minval
        for country in countries:
            dfCut.loc[country,year] = (dfCut.loc[country,year]-minval)/length
    return dfCut
```

Negative values error

Beforehand, I had no validation of the minimums and maximums. This meant that some indexes were above 1 (eg: USA's GDP was above 2 standard deviations) or below 0 (eg: France's income inequality was below 2 standard deviations). Through this code I ensure the G7 data is within the range and so ensures indexes are between 0 and 1.

Values in table are replaced with new indexes

The following is World Bank's indexing:

```
def indexWB(dfFull, dfCut):   # dfFull is not formatted, dfCut is
```

```
countries = ["Italy", "United Kingdom", "United States", "France", "Japan", "Germany",
                        "Canada"]

dfInfo = dfFull.describe()                  # gets all the summary statistics of the
                                                                    dataframe
```

For World Bank data, the getting the summary statistics can be done outside of the loop. This is because the table containing all data has not been formatted, and so the columns are the years, and since .describe() gives information by column, it can be used as such.

```
for year in range(2000,2020):
    mean = dfInfo["YR"+str(year)].loc['mean']    # finds mean for that year
    stdDev = dfInfo["YR"+str(year)].loc['std']   # finds standard deviation for that year
    minval = mean - 2*stdDev                  # creates a minimum -2 stds away from mean
    maxval = mean + 2*stdDev                   # creates a maximum +2 stds away from mean
```

Indexing design fix: See in previous code

```
    if minval < 0:
        minval = 0                          # ensures minval is above 0
    elif minval > min(dfCut.loc[year]):
        minval = min(dfCut.loc[year])
    dfCut = dfCut.swaplevel()
    if maxval < max(dfCut.loc[year]):
        maxval = max(dfCut.loc[year])          # ensures the G7 do not surpass the maximum
```

Negative values error: See in previous code

```
    dfCut = dfCut.swaplevel()
    length = maxval - minval
    for country in countries:
        dfCut.loc[country,year] = (dfCut.loc[country,year]-minval)/length
return dfCut
```

These two indexing algorithms perform the same steps but are adjusted for the two styles of importing tables. The algorithm starts by finding the summary statistics for each year in the data. This allows me to access the mean and standard deviation for that year. The minimum is then 2 standard deviations below mean, and the maximum 2 standard deviations above (encompassing 95% of data, and so getting rid of extreme values). From then I verify the G7 is withing these values, and calculate their indexes (which replace the old data in the table). This is repeated for the years 2000-2019.

Input	Expected result	Output

Find the index value of GDP in the UK `print(dfGDP['United Kingdom'][2000])`	0.16 (2dp)	0.16252631147551808
Find the index value of Life Expectancy in China `print(dfLife['China'][2000])`	"Invalid selection. This country is not available"	KeyError: 'China'
Find the value of RPI in the US `print(dfRPI['United States'][2000])`	"Invalid selection. This metric is not available"	NameError: name 'dfRPI' is not defined

Map Creation

```
import pandas as pd
import json
import plotly.express as px

countriesMap = json.load(open("countries.geojson", 'r'))
```

This is simply a file containing all the information needed to draw the map (and extract of it can be seen below).

```
df = pd.read_csv("Current.csv")
df = df.set_index('year')
df = df.loc[2019]
df = df.replace("United States", "United States of America")
# the geojson file has a different name, so changing it is necessary

for country in countriesMap['features']:
    country['id'] = country['properties']['ADMIN']
    # adding a new property 'id' to the main list of properties of every country
    # the id is simply the name of the country, as they are unique
    # this is used so the id (country name) can be accessed from the first level of the geojson
      file

colorscale = ["rgb(200, 240, 255)", "rgb(50, 50, 255)"]   # blue colour scale
ccmap = px.choropleth(df, locations='country', geojson=countriesMap, color='GDP',
        color_continuous_scale=colorscale, height=500)
```

Here I am reading my Current table and changing the index to 'year'. For now I'm only focusing on 2019, which can be changed later on.

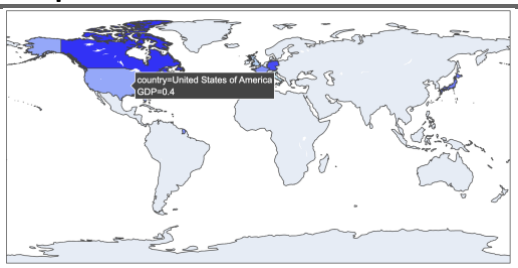
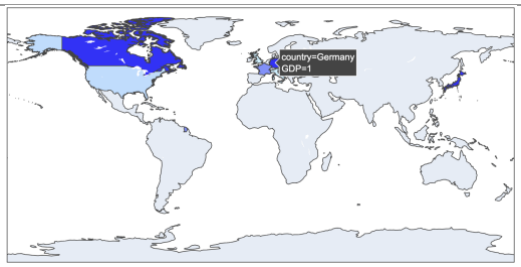
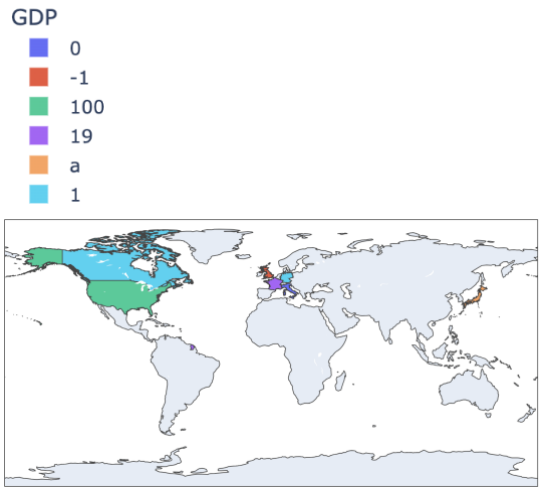
This creates the map using the geojson file, the data and the colour scale.

This code uses a geojson file of the world map and the data from the Current table to create a colour coded map though Plotly.express. Currently, I have not created a map class as I only have to worry about a single map. The class will be created in the next iteration.
Below is what the geojson file looks like:

```
{
"type": "FeatureCollection",
```

Each 'feature' is a country, each with an ID and the set of coordinates to draw it.

"features": [
{ "type": "Feature", "properties": { "ADMIN": "Aruba", "ISO_A3": "ABW" }, "geometry": { "type": "Polygon", "coordinates": [[[-69.996937628999916, 12.577582098000036], [-69.936390753999945, 12.531724351000051], [-69.924672003999945, 12.519232489000046], [-69.915760870999918, 12.497015692000076], ...]] } },

Input	Expected result	Output
Create a map with indexes: 0.1, 0.2, 0.4, 0.5, 0.6, 0.8, 0.9 .csv file: **"country,year,GDP** **Italy,2000,0.1** **United Kingdom,2000,0.2** **United States,2000,0.4** **France,2000,0.5** **Japan,2000,0.6** **Germany,2000,0.8** Canada,2000,0.9"	Map of the G7, each with different and correct saturations	
Create a map with indexes: 0, 0, 0.1, 0.5, 1, 1, 1 *.csv file:* **"country,year,GDP** **Italy,2000,0** **United Kingdom,2000,0** **United States,2000,0.1** **France,2000,0.5** **Japan,2000,1** **Germany,2000,1** Canada,2000,1"	Map of the G7, with correct saturations (2 at the lightest saturation, 3 at the darkest)	
Create a map with indexes: 0.1, 0.2, 0.4 *.csv file:* **"country,year,GDP** **Italy,2000,0.1** **United Kingdom,2000,0.2** United States,2000,0.4"	Map of the G7, with correct saturations and the ones without an index left without any colour	
Create a map with indexes: 0, -1, 100, 19, 'a', 1, 1	"Invalid selection. These indexes are not between 1 and 0"	 The geojson file does support this, but the data will be only indexes thanks to the process of creating the Current table.
Create a map with indexes: 0, 0, 0, 0, 0, 0, 0, 0	"Invalid selection. The map is made for 7 countries, not more"	The map does allow more countries (all the one accessible in the geojson file), bur since the data is cut beforehand, more than 7 countries will never occur.

In the next prototype I will include validation withing the map class to ensure incorrect maps are not possible.

User Interface Creation and Map Integration

```
import dash
from dash import html, dcc
import dash_bootstrap_components as dbc

# initialising a multipage app with the stylesheet theme SPACELAB
app = dash.Dash(__name__, use_pages=True, external_stylesheets=[dbc.themes.SPACELAB])



menu = dbc.Nav(
        # creating a menu to access every page
        [
            dbc.NavLink(            # navlink with the name and path assigned in each file
                [
                    html.Div(page["name"], className="ms-2"),
                ],
                href=page["path"],
                active="exact",    # highlights active page in menu
            )
            for page in dash.page_registry.values()      # this is done for every page in
                                                                    the registry
        ],
        vertical=False,
        pills=True,
        className="bg-light",
            # ^ this is the design style of the menu
)



app.layout = dbc.Container([
   # title of the page
   dbc.Row([
      dbc.Col(html.Div("Colour Coded Maps",
                  style={'fontSize':50, 'textAlign':'center'}))
   ]),
```

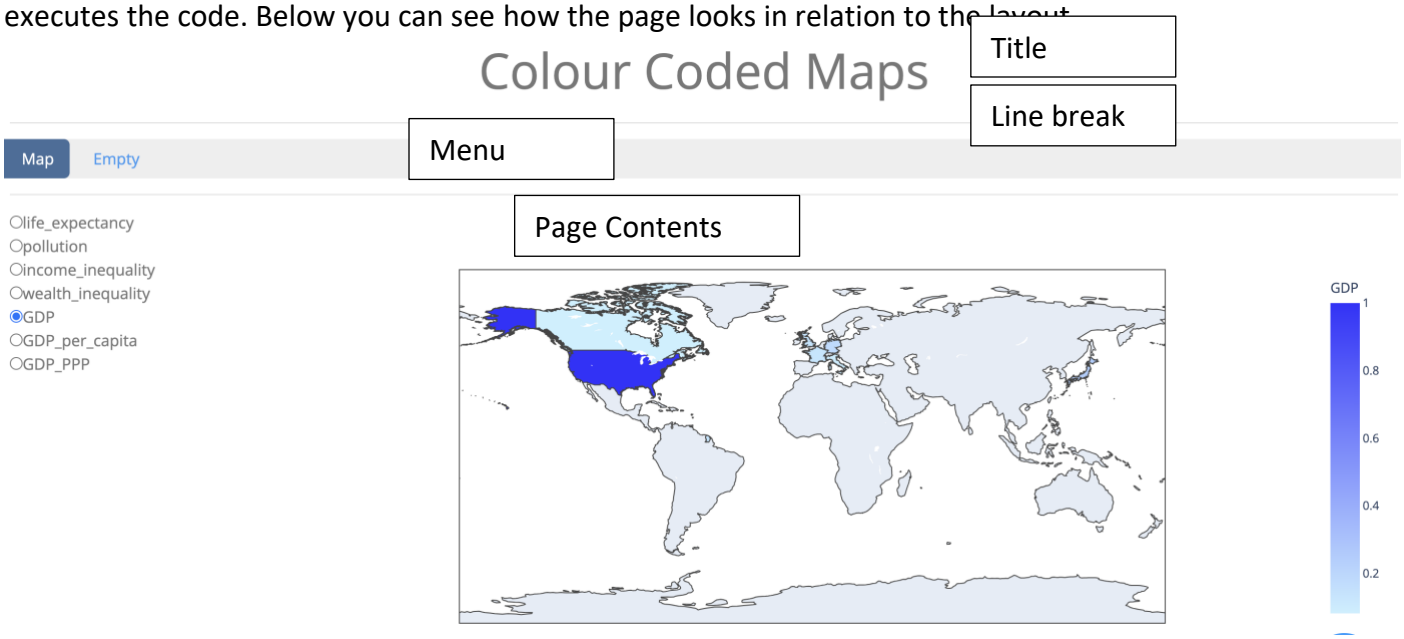
This initialised the app with the style SPACELAB (which is close to the design aesthetic) and with the ability to hold and access multiple pages.

For every page in the page registry (which are added at the start of the other page files), this code creates a path and adds it to the menu. The navigation link of each page (name and path) is initialised on their files.

```
    html.Hr(),  # line break
    dbc.Row([
        menu
    ]),
    html.Hr(),

    dbc.Row([
        dash.page_container
    ])
], fluid=True)


if __name__ == "__main__":
    app.run(debug=True)
```

The app looks like this:
Title
Line break
Menu
Contents of page

This code creates an app and adds all the pages to it with a menu. It then creates the layout of the page and executes the code. Below you can see how the page looks in relation to the layout.



This is the main page for now, which displays the maps:

```
import pandas as pd
import json
import plotly.express as px
import dash
from dash import html, dcc, callback, Output, Input, Patch
import dash_bootstrap_components as dbc

# registering this page as the homepage (path='/') to the app registry
dash.register_page(__name__, path='/', name="Map")
```

Adds this page to the page registry so it can be accessed by the app.

```
[ map creation code ]

userOptions = df.columns.values.tolist()
del userOptions[0]    # removing 'country' from the list of options, as we only want user to
                                choose between metrics

layout = html.Div(
   [
      dbc.Row(
         [
            dbc.Col(
               [
                  dcc.RadioItems(options=userOptions,
                                 value='GDP',
                                 id='metric-choice')
               ], xs=4, sm=3, md=2, lg=2, xl=2, xxl=2   #this is used to set the size
                                                        ratios for different screen sizes
            ),

            dbc.Col(
               [
                  dcc.Graph(id='map',
                            figure= ccmap)
               ], xs=8, sm=9, md=10, lg=10, xl=10, xxl=10
            )
         ]
      )
   ]
)

@callback(
   Output(component_id='map', component_property='figure'),
   Input(component_id='metric-choice', component_property='value')
)
```

Makes a list of the metrics so users can choose between them.

The initial value selected is GDP, but can change into any other metric.

Divides page into 2: a list of the metrics and the corresponding map. The metrics are currently radio buttons (can select only one at a time), but will change to a checklist in the next iteration (can select as many as you like).

The output of the page is the map (id='map' and component='figure') and the input are the metrics (id='metric-choice' and component='value').

Speed fix

Since the program was taking a very long time when switching between metrics, I looked into what could be done to speed it up. Patch did this quite easily, and worked incredibly well.

```
def update_graph(value):
    patchedFig = Patch()
    dataList = df[value].tolist()
    patchedFig['data'][0]['z'] = dataList
    patchedFig['layout']['coloraxis']['colorbar']['title']['text'] = value
    patchedFig['data'][0]['hovertemplate'] = 'country=%{location}<br>'+value+'=%{z}<extra></extra>'
                # inside the geojson file, the hovertemplate is the following:
                            'country=%{location}<br>GDP=%{z}<extra></extra>'
                                # to change the metric, I simply need to change the name of the metric
                                 (from GDP to whatever 'value' is being chosen)

    return patchedFig
```

Hover values error

As I was testing the hover values of the map, I realised they weren't being updated by the change in metric and were instead stuck with the initial GDP values. I had to search though the geojson file to know how to access the hover values (patchedFig['data'][0]['hovertemplate'], seen better below), and then update it so it showed the new metric that is chosen.

This code divides the page in a 2:10 ratio (at the largest window sizes) so the user can select the metrics and see the corresponding map being displayed. The @callback sets up the input and outputs (selecting the metrics and displaying the map), so they can be updated with the function update_graph(). This function takes in the currently selected metric, 'value', creates a list of the data of this metric and updates the data in the map, the title of the colour bar and the hover values.

```
Figure({
    'data': [{'coloraxis': 'coloraxis',
              'geo': 'geo',
              'geojson': {'features': [{'geometry': {'coordinates': [[[-69.9969376
                                                                       2899992, 12.57
                                                                       7582098000036]
                                                                       , [-69.9363907
                                                                       5399994, 12.53
                                                                       172435100005],
                                                                       [-69.924672003
                                                                       99994, 12.5192
                                                                       32489000046],
                                                                       [-69.915760870
                                                                       99992, 12.4970
                                                                       15692000076], … ]],
                                                    'type': 'Polygon'},
                                        'id': 'Aruba',
                                        'properties': {'ADMIN': 'Aruba', 'ISO_A3': 'ABW'},
                                        'type': 'Feature'},
                               'type': 'FeatureCollection'},
              'hovertemplate': 'country=%{location}<br>GDP=%{z}<extra></extra>',
```

['data'][0]['hovertemplate'] gives access to what is displayed when hovering.

```
              'locations': array(['Italy', 'United Kingdom', 'United States of America', 'France',
                                  'Japan', 'Germany', 'Canada'], dtype=object),
```

```
            'name': '',
            'type': 'choropleth',
            'z': array([0.09406971, 0.13362617, 1.        , 0.12763076, 0.23937138, 0.18185447,
                        0.08155498])}],
```

['data'][0]['z'] gives access to a list of all the indexes for that metric.

['layout']['coloraxis']['colorbar']['title']['text'] gives access to the colour bar name.

```
    'layout': {'coloraxis': {'colorbar': {'title': {'text': 'GDP'}},
                             'colorscale': [[0.0, 'rgb(200, 240, 255)'], [1.0,
                                            'rgb(50, 50, 255)']]},
               'geo': {'center': {}, 'domain': {'x': [0.0, 1.0], 'y': [0.0, 1.0]}},
               'legend': {'tracegroupgap': 0},
               'margin': {'t': 60},
               'template': '...'}
})
```

Stakeholder feedback

Along with sending the prototype to my stakeholders and seeing what they thought of it so far, I also wanted to ask them a few extra questions to aid my development in later prototypes. I will also send them the following description of the prototype, so they know what to expect:

"This is the first prototype of the program. It only includes 7 metrics so far and doesn't have the option to select more than one, which will be added in the next iteration. For now, I have not divided the metrics by section, and there is only one map available to view. The data shown on this map is from 2019, the latest data I could access. In the map you can pan, zoom, and hover over countries to see their index score."

Stakeholder Questions

These are the questions I will ask my stakeholders:

1. I currently hold data for the years 2000-2019. This means there is an increase in the file size you need, but it also gives you access to data from more years. Is this something you wish, or should the software only hold the latest data (2019)?
2. To access the raw data, there are two possibilities. Either the raw data is stored alongside the program, increasing the overall size, or the data is imported when accessed, creating delays in the access to data. Which would you prefer?
3. Is this UI an accessible design for you? If not, what could be included?
4. Are the pan and zoom-in functions of the map sufficient? If not, what could be included?
5. Is it clear when you switch between metrics? If not, what could be included?
6. Is the loading time for the maps acceptable?
7. Anything else to add?

Questions 1 and 2 are two design queries I had, concerning the size of the program versus extra data/faster loading, and what the users valued more. For questions 3-6, I am ensuring that the website is working in a way that is satisfactory for my users. Question 7 is an open question for them to add any other thoughts.

Professional Economist

Fermin:

1. **I currently hold data for the years 2000-2019. This means there is an increase in the file size you need, but it also gives you access to data from more years. Is this something you wish, or should the software only hold the latest data (2019)?**
   It is very valuable to have a longer set of data to evaluate the trends. The incremental cost of a larger file size is irrelevant when compared with the benefits of having trended data.
2. **To access the raw data, there are two possibilities. Either the raw data is stored alongside the program, increasing the overall size, or the data is imported when accessed, creating delays in the access to data. Which would you prefer?**
   Performance is very important, repones time is a key factor in the user experience, so a larger file size is again irrelevant.
3. **Is this UI an accessible design for you? If not, what could be included?**

Yes, it accessible. Having said that, I would appreciate some more contextual help. For example, you should add a clarification that the maps only show the data of the G7 countries.

4. **Are the pan and zoom-in functions of the map sufficient? If not, what could be included?**
   Yes, they cover the main needs.
5. **Is it clear when you switch between metrics? If not, what could be included?**
   I think a title would help make the switches clearer.
6. **Is the loading time for the maps acceptable?**
   Yes, it is a reasonable response time.
7. **Anything else to add?**
   No.

Lourdes:

1. **I currently hold data for the years 2000-2019. This means there is an increase in the file size you need, but it also gives you access to data from more years. Is this something you wish, or should the software only hold the latest data (2019)?**
   I definitely think it would be more useful to have data for different years to create a more complex analysis and more accurate trends.
2. **To access the raw data, there are two possibilities. Either the raw data is stored alongside the program, increasing the overall size, or the data is imported when accessed, creating delays in the access to data. Which would you prefer?**
   A larger overall file size is acceptable.
3. **Is this UI an accessible design for you? If not, what could be included?**
   Yes, but it would be great if you had different options for font and colours for people that may have visual impairments.
4. **Are the pan and zoom-in functions of the map sufficient? If not, what could be included?**
   Yes.
5. **Is it clear when you switch between metrics? If not, what could be included?**
   It is clear but it could be improved by adding some contextual details.
6. **Is the loading time for the maps acceptable?**
   Yes, I think it is okay.
7. **Anything else to add?**
   No.

Average Consumer

Jenny:

1. **I currently hold data for the years 2000-2019. This means there is an increase in the file size you need, but it also gives you access to data from more years. Is this something you wish, or should the software only hold the latest data (2019)?**
   Definitely more years. A single year is one only piece of data, having multiple years will allow me to see trends in the data.
2. **To access the raw data, there are two possibilities. Either the raw data is stored alongside the program, increasing the overall size, or the data is imported when accessed, creating delays in the access to data. Which would you prefer?**

Data stored within the program. I want to play with the program and see multiple data at first, rather than searching for one specific piece of data. If there is a delay each time I select a country, it makes it very difficult to compare data across years or countries by just looking at it, which means I am more likely to use an alternative program.

3. **Is this UI an accessible design for you? If not, what could be included?**
   Yes, it's accessible. But it would be better if you also include the raw data for life expectancy, pollution & GDP, as well as the index you currently have.
4. **Are the pan and zoom-in functions of the map sufficient? If not, what could be included?**
   Yes – I can zoom in & out using the zoom button and also the touch pad/mouse. These 7 countries take up a small amount of space in the world map so I think it would be a good idea to add an automatic zoom function the countries, where one button for each country is added to the zoom button and these buttons will be shown once the user put their mouse onto the zoom button.
5. **Is it clear when you switch between metrics? If not, what could be included?**
   I think it's sufficient, but would be better if a title is added on top of the map.
6. **Is the loading time for the maps acceptable?**
   Yes, it is very instantaneous.
7. **Anything else to add?**
   No.

Analysis

My stakeholders were happy to have larger overall file size if it meant more features and greater performance, so I will keep the 2000-2019 data and a copy of the raw data. This means there will be a new CurrentRaw table and a CurrentIndexed table, both with 2000-2019 data. The stakeholders were also happy with the user interface, just wished for a title when changing metrics and an overall more contextualised interface. The increased performance thanks to Patch is acceptable, so there's no need to attempt improving it more.

Prototype 2

With the second prototype there wasn't any major changes to the design, just a few minor adjustments to make the code work with the current libraries I am utilising. I will briefly detail this unforeseen addition and justify why it is needed below, then explain the rest of the code.

Changes in Prototype 2

Library to import map classes

<u>Change:</u> no plan for class imports → importing using sys

There was no previous design for how and where the map classes would be stored, but my aim was to have them in a different file and be imported into the app pages, as to have consistency with the map classes across the whole code. This was not working, however, as I could not store my mapClasses.py file within the same folder as the pages due to the way Dash apps function. Therefore, I found a way around this using sys and creating a path for the code to use so it can access the files outside the immediate folder. (The […] are the rest of the path which I will not show).

```
import sys
sys.path.append('[...]/projectPrograms')
from mapClasses import *
```

New function for initial data processing

<u>Change:</u> simply procedural code → new function

While adding the new metrics for the Current and Projected table I noticed there was quite a lot of repeated code that could be made simpler with the use of a function. This made my code neater and meant I don't have to keep rewriting the same code for every new metric I add. The code can be seen below.

```
# used to reduce repeated code when importing data from OWID
def finiliseOWID(df_in, metricName, tableName):
    df = df_in[metricName]
    dfAll = df.sort_index()
    df = selectCountries(dfAll)
    df = indexOWID(dfAll, df)
    df = df.rename(tableName)
    return df

# I will only apply this function to the metrics from iteration 2 and onwards, to show the growth of the code



df = pd.DataFrame(catalog.find('global_carbon_budget', version='2023-09-28').load())
dfCarbon = finiliseOWID(df, 'consumption_emissions_per_capita',
```

The function inputs the name of the metric required within the table and what the final name for the metric should be.

```
                    'global_carbon_budget')
```

Code Analysis and Iterative Testing

The following will be an analysis of all the major parts of my code, by section, with the testing and error resolution I did throughout development. It also goes through the iterative tests I carried out for each section.

Explanation of the code is made through summaries, in-text comments (in green), and code analysis (in text-boxes over code). The errors and bugs I resolved will be highlighted in red.

New imports

The current metrics being imported now are: 'pollution', 'global_carbon_budget', 'greenhouse_gas_emissions', 'renewable_electricity_capacity', 'life_expectancy', 'income_inequality', 'wealth_inequality', 'GDP', 'GDP_per_capita', 'GDP_PPP', and 'child_mortality'.

There was also a slight adjustment to the indexing function for Our World in Data.

```
def selectCountries(df_in):
    countries = ["United Kingdom", "United States", "France", "Japan", "Germany", "Canada"]

    df_out = df_in.loc[("Italy",2000):("Italy",2023)]
    for country in countries:
        temp = df_in.loc[(country,2000):(country,2023)]
        df_out = pd.concat([df_out, temp])
    return df_out
```

While importing the new metrics I realized this function was relying on another local variable named 'countries' and added this to avoid any problems if there were to be future changes in the other function.

Duplicate Indexes error

With the addition of the new 'countries' variable, I failed to realise that as it was defined beforehand (with 'Italy' still in the list), Italy was being added to the tables twice. However, I didn't see this until later and tried multiple other ways to fix it with varying success.

I'll briefly detail these, as they helped me further my understanding of Pandas:

```
# df = df.drop_duplicates(keep='first')
```

This systematically worked, but was actually getting rid of duplicate values, not indexes, and so was getting rid of unnecessary values (eg: the US having GDP index 1.0 for most years).

```
# df.index = df.index.drop_duplicates(keep='first') (Length mismatch: Expected axis has 192 elements, new values have 168 elements)
```

This had the right logic, as it identified and kept only one copy of the duplicate indexes, but could not just simply be applied to the data.

```
# df = df.groupby(df.index).first()
```

This was an attempt at having the duplicate values simply be averaged, but the command is not supported by my version of Pandas.

Apart from this error and the new function, importing the new metrics was quite easy thanks to my use of functions.

```
df = pd.DataFrame(catalog.find('renewable_electricity_capacity', version='2023-06-26').load())
dfRenewable = finiliseOWID(df, 'total_renewable_energy', "renewable_electricity_capacity")
```

One adjustment was made for the child mortality rate, as it didn't have the index I needed, so I quickly adjusted this and continued as normal.

```
df = pd.DataFrame(catalog.find('child_mortality', version='2020-12-19').load())
df = df.set_index(['country', 'year'])
dfChildMor = finiliseOWID(df, 'probability_of_death_under_5', "child_mortality")
```

Tests:
For this test, the inputs are inside the code. I did not add validation for invalid inputs as users cannot access them as explained before.
I chose the year 2000 to test a single result, any other year would've done the same test.
The dfRenewable (data frame) in the inputs is the renewable electricity capacity table. Any other table from Our World in Data would've worked as well.

Input	Expected result	Output
Find the value of renewable electricity capacity in the UK `print(dfRenewable['United Kingdom'][2000])`	Online: 2.8 GW	2893.0 MW
Find the value of renewable electricity capacity in China `print(dfRenewable ['Country'][2000])`	"Invalid selection. This country is not available"	KeyError: 'Country'
Find the value of RPI in the US `print(dfRenewable ['United States'][2000])`	"Invalid selection. This metric is not available"	NameError: name 'dfRPI' is not defined

These tests ensure the data is being imported and processed correctly.

Map Classes

When initialising the map, it takes in the colour of the map and the name of its dataframe.

```
class CCMap:
    def __init__(self, mapColour, mapDataframeName):
        if mapColour == 'blue':
            self.mapColour = ["rgb(200, 240, 255)", "rgb(50, 50, 255)"]
            self.mapMetrics = ['life_expectancy', 'income_inequality', 'wealth_inequality',
                               'GDP', 'GDP_per_capita', 'GDP_PPP', 'child_mortality']
```

The colour is used to set the colour axis for the map and the metrics that should be included in the map (environmental for green, social for blue).

```
    elif mapColour == 'green':
        self.mapColour = ["rgb(179, 242, 180)", "rgb(38, 128, 13)"]
        self.mapMetrics = ['pollution', 'global_carbon_budget', 'greenhouse_gas_emissions',
                                        'renewable_electricity_capacity']
    else:
        return 'inavlid colour'
```

There is also a validation to make sure only 'blue' and 'green' can be passed.

```
    self.mapDataframeName = mapDataframeName
    self.indexName = 'year'
    self.mapDate = 2019
```

For current maps the index name for the date is 'year'.

```
def getMetrics(self):
    return self.mapMetrics
```

getMetrics() is used later on to ensure the metrics being used are consistent across all files.

```
def getDataframe(self):
    df = pd.read_csv(self.mapDataframeName)
    df = df.set_index(self.indexName)
    df = df.replace("United States", "United States of America")
    self.mapDataframe = df
    return self.mapDataframe
```

Direct access to the dataframe is needed for the callback function with Patch (will elaborate later on), so this ensure the same version of the dataframe is being used.

```
def createMap(self):
    countriesMap = json.load(open("countries.geojson", 'r'))

    df = pd.read_csv(self.mapDataframeName)
    df = df.set_index(self.indexName)
    df = df.loc[self.mapDate]
    df = df.replace("United States", "United States of America")

    metrics = df.columns.values.tolist()
    for metric in metrics:
        if metric not in self.mapMetrics and metric != 'country':
            df = df.drop(metric, axis=1)

    self.mapDataframe = df
```

```
        for country in countriesMap['features']:
            country['id'] = country['properties']['ADMIN']
            # adding a new property 'id' to the main list of properties of every country
            # the id is simply the name of the country, as they are unique
            # this is used so the id (country name) can be accessed from the first level of the
                    geojson file

        colorscale = self.mapColour
        ccmap = px.choropleth(df, locations='country', geojson=countriesMap, color='GDP',
                              color_continuous_scale=colorscale, range_color=[0,1], height=500)

        userOptions = df.columns.values.tolist()
        del userOptions[0]    # removing 'country' from the list of options, as we only want
                                user to choose between metrics

        return ccmap, userOptions


class projectedCCMap(CCMap):
    def __init__(self, mapColour, mapDataframe):
        CCMap.__init__(self, mapColour, mapDataframe)
        self.indexName = 'quarter'
        self.mapDate = "2020 QR:0"

# vv Not worked on yet vv
class econSimCCMap(projectedCCMap):
    chosenCountry = 'United States of America'
```

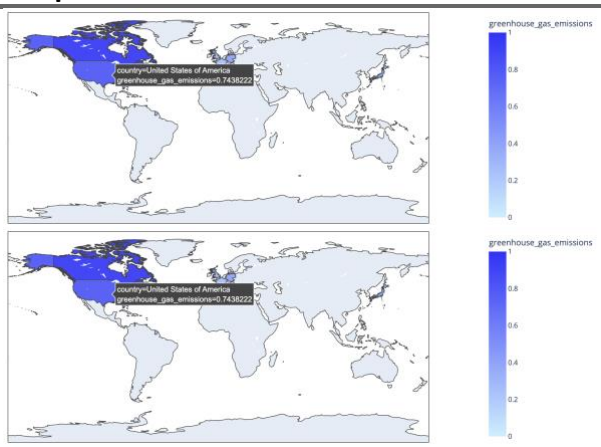
Changed the coloraxis to have a constant range between 0 and 1 instead of adjusting for every map, as it makes comparison easier

Same way to create a map, but it is now a behaviour of the class.

Projected child class inherits the same attributes and behaviours except for the date index name, now 'quarter', and the initial mapDate.

The above code has adapted the previous map code included in my 'currentMaps.py' page into a class to ensure consistency across code, reduce the amount of repeated code and generally make the code easier to read and understand. This code is stored in a separate file that is then imported to the necessary pages to be used in the Dash app.

These tests simply ensure the map classes are working in the exact same way as the previous code (which, since already verified to be working correctly, will ensure the map classes are accurate).
I haven't added a lot of validation to the map classes due to the fact that the importing and processing of the data will already get rid of the errors discussed beforehand.

Input	Expected result	Output
Create a map with greenhouse gas emissions data from 2019.	Map of the G7, each with different and correct saturations. Previous code: ---------- Map Classes:	
Create a map with RPI data from 2019	"Invalid selection. There is no RPI metric in the data."	No option for this shown in user interface.
Create a map with GDP data from 2050	"Invalid selection. Data ranges from year 2000-2019."	No option for this shown in user interface.

New page layout

```
layout = html.Div(
    [
        dbc.Row(
            dcc.Dropdown(options=['Environment', 'Social'],
                         id='map-choiceC',
                         value='Social',
                         clearable=False
            )
        ),
        html.Hr(),

        dbc.Row(
            dcc.RadioItems(options=years,
                         id='date-choiceC',
                         value=2019,
                         inline=True,
                         labelStyle={'padding':'10px'}
            )
        ),

        dbc.Row(
            [
                dbc.Col(
                    [
                        dcc.Checklist(options=currentMaps['SCurrentData'].getMetrics(),
                                      id='metric-choiceC')
```

Dropdown to choose between 'Environmental and 'Social' maps.

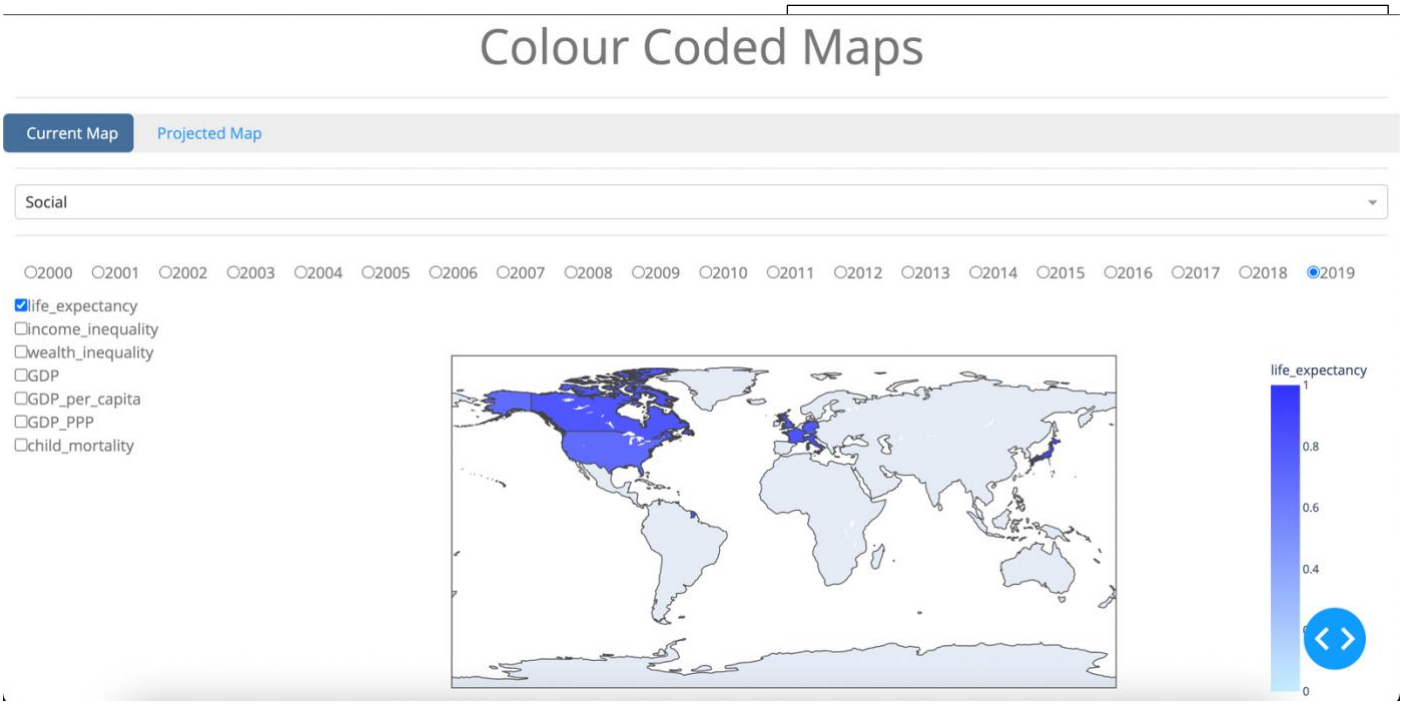
Radio buttons to choose between years (between quarters in the projected page).

The choice of metrics has now been changed from radio buttons to a checklist.

```
                ], xs=4, sm=3, md=2, lg=2, xl=2, xxl=2
            ),

            dbc.Col(
                [
                    dcc.Graph(id='currentMap',
                              figure= ccmap)
                ], xs=8, sm=9, md=10, lg=10, xl=10, xxl=10
            )
        ]
    ),
    html.Hr(),
  ]
)
```

This code create a page with a row for a dropdown to choose between maps, a second row to choose between years, and a final row made of two columns: one with a checklist to select between metrics and another with the map.

This is how the page looks:



Main callback

```
  [ Map Type callback discussed below ]


@callback(
   Output(component_id='currentMap', component_property='figure'),
   [Input(component_id='metric-choiceC', component_property='value'),
    Input(component_id='date-choiceC', component_property='value'),
    Input(component_id='map-choiceC', component_property='value')]
)
```

```
def update_graph(metricChoice, dateChoice, mapChoice):
    patchedFig = Patch()


    df = currentMaps['SCurrentData'].getDataframe()
    df = df.loc[dateChoice]


    if mapChoice == 'Environment':
        userOptions = currentMaps['ECurrentData'].getMetrics()
        metrics = df.columns.values.tolist()
        for metric in metrics:
            if metric not in userOptions and metric != 'country':
                df = df.drop(metric, axis=1)
    elif mapChoice == 'Social':
        userOptions = currentMaps['SCurrentData'].getMetrics()
        metrics = df.columns.values.tolist()
        for metric in metrics:
            if metric not in userOptions and metric != 'country':
                df = df.drop(metric, axis=1)


    if len(metricChoice) == 0:
        mean = []
    elif len(metricChoice) == 1:
        mean = df[metricChoice].values.tolist()
        fix = []
        for i in range(len(mean)):
            fix.append(mean[i][0])
        mean = fix
    elif len(metricChoice) > 1:
        dataList = df[metricChoice[0]]
        for i in range(len(metricChoice)-1):
            i += 1
            dataList = pd.concat([dataList, df[metricChoice[i]]], axis=1)
        mean = dataList.mean('columns')
        mean = mean.tolist()


    patchedFig['data'][0]['z'] = mean
```

Function inputs the values defined in the callback.

Selects the new year from the database based on user choice.

Selects the metrics to be considered in the table and drops the ones that should be ignored.

Selecting multiple metrics errors

Previously, only single metrics could be selected, so there were no problems with the callback. With the addition of the checklist, a bit more of validation was needed.

`if len(metricChoice) == 0:` is used so the code goes on even if no metric is selected by displaying an empty map.

For `elif len(metricChoice) == 1:`, the code gets the list of values, but since these come as a 2d list, fix is used to reformat it.

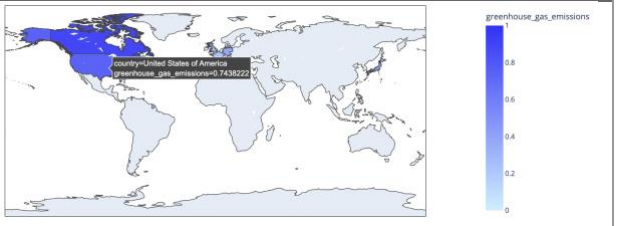
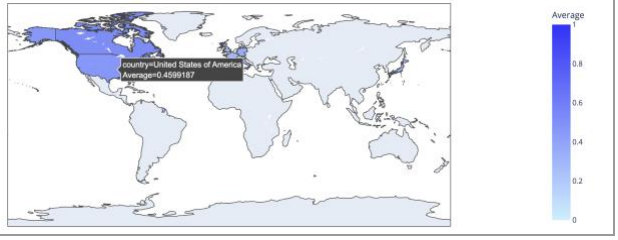
Creates a dataframe for the required metrics, find the average for each column and turns it into a list.

```
if len(metricChoice) == 0:
    pass
elif len(metricChoice) == 1:
    patchedFig['layout']['coloraxis']['colorbar']['title']['text'] = metricChoice[0]
    patchedFig['data'][0]['hovertemplate'] = 'country=%{location}<br>'+ metricChoice[0] +'=%{z}<extra></extra>'
else:
    patchedFig['layout']['coloraxis']['colorbar']['title']['text'] = 'Average'
    patchedFig['data'][0]['hovertemplate'] =
            'country=%{location}<br>Average=%{z}<extra></extra>'

    # inside the geojson file, the hovertemplate is the following:
      'country=%{location}<br>GDP=%{z}<extra></extra>'
    # to change the metric, I simply need to change the name of the metric (from GDP to
      whatever 'value' is being chosen)

return patchedFig
```

Updates the text of the hover template depending on the number of metrics selected.

I attempted to display the different values included in the average for the hover text, but this was simply not possible with the way the figure data is structured.

The @callback sets up the input and outputs (selecting the metrics, the date and the map type, and displaying the map), so they can be updated with the function update_graph(). This function takes in the currently selected metric, 'metricChoice', the currently selected year, 'dateChoice', and the currently selected type of map, 'mapChoice'. It then updates the data in the map, the title of the colour bar and the hover values.

Input	Expected result	Output
Create a map with greenhouse gas emissions data from 2019.	Map of the G7, each with different and correct saturations. The hover should display the correct name.	
Create a map with greenhouse gas emissions and pollution data from 2019.	Map of the G7, each with different and correct saturations. The hover should display the correct name.	

Create a map with RPI data from 2019	"Invalid selection. There is no RPI metric in the data."	No option for this shown in user interface.
Create a map with GDP data from 2050	"Invalid selection. Data ranges from year 2000-2019."	No option for this shown in user interface.

Map type callback

```
@callback(
    Output(component_id='metric-choiceC', component_property='options'),
    Input(component_id='map-choiceC', component_property='value')
)
def setMetricChoices(mapChoice):
    if mapChoice == 'Environment':
        return currentMaps['ECurrentData'].getMetrics()
    elif mapChoice == 'Social':
        return currentMaps['SCurrentData'].getMetrics()
    else:
        return []

@callback(
    Output(component_id='metric-choiceC', component_property='value'),
    Input(component_id='metric-choiceC', component_property='options')
)
def setInitialValue(metricOptions):
    value = metricOptions[0]
    return [value]
```

Updates the metrics displayed according to the type of map chosen (environmental metrics for 'Environment', and social for 'Social').

Sets the initial value of the metrics selected to the first value in the list (this occurs when switching between map types).

Input	Expected result	Output
Select 'Social' in the dropdown.	The 'Social' metrics being displayed, with the first one selected.	
Select 'life-expectancy' after selecting 'Environmental' for the map type.	No option for 'life-expectancy' shown if 'Environmental' is selected.	No option for 'life-expectancy' shown if 'Environmental' is selected.

Projected algorithm

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression

# get values from .csv file
# turn them into array
# create models for the data
# predict the values for next 8 quarters in the future

df = pd.read_csv("Current.csv")
df = df.set_index(['country','year'])

data = {}

countries = ["United Kingdom", "United States", "France", "Japan", "Germany", "Canada", "Italy"]
metrics = ['pollution', 'global_carbon_budget', 'greenhouse_gas_emissions', 'renewable_electricity_capacity', 'life_expectancy',
'income_inequality', 'wealth_inequality', 'GDP', 'GDP_per_capita', 'GDP_PPP', 'child_mortality']

for country in countries:
    countryData = {}
    for metric in metrics:
        dfCut = df.loc[country, metric]
        countryData[metric] = dfCut.tolist()
    data[country] = countryData


projectedData = {}

for metric in metrics:
    metricList = []
    for country in countries:
        x = np.array([i for i in range(2000, 2020)]).reshape((-1, 1))
        y = np.array(data[country][metric])

        model = LinearRegression()
        model.fit(x, y)

        #   times by 100 since range needs to be integers, will need to divide at the end
        for quarter in range(201900, 202125, 25):
            quarter = quarter/100
            print(quarter)
            projection = model.predict([[quarter]])
```

These are the (simplified) steps the projected algorithm follows. I decided against using functions for this, as for loops carry out the same job in a way that makes it easier to see step-by-step.

This turns the Current.csv dataframe into a dictionary which looks as such: {'United Kingdom':{'pollution':[…, …, …], …}, …}, with the list being all the years in order.

Creates a model for every metric based on the 2000-2019 data.

Predicts the next 8 quarters with the model and stores it to a list.

```
            print(projection)
            metricList.append(projection[0])

    projectedData[metric] = metricList


projectedIndex = []
for country in countries:
    for quarter in range(201900, 202125, 25):
        quarter = quarter/100
        year = int(quarter//1)
        quarter = int((quarter%1)*4)
        projectedIndex.append((country, str(year)+" QR:"+str(quarter)))


projectedIndex = pd.MultiIndex.from_tuples(projectedIndex, names=["country", "quarter"])

projectedDf = pd.DataFrame(data= projectedData, index=projectedIndex)
projectedDf.to_csv('Projected.csv')
```

Creates an index for the dataframe.

Turns index and data into a dataframe and stores it as a .csv file.

The algorithm starts by turning the Current.csv dataframe into a dictionary so I can iterate through it easily. With this I create a model for each metric and country, which I use to predict the next 8 quarters of data. I then create a list of tuples for a multi-index and combine this and the data into a new Projected table which is then stored as a .csv file.

Tests:

Input	Expected result	Output
Find the index value of pollution in the UK *(seen in the Projected.csv file)*	0.20 (2dp)	0.2016252199436117
Find the index value of Life Expectancy in China *(seen in the Projected.csv file)*	"Invalid selection. This country is not available"	No 'China' in Projected table
Find the value of RPI in the US *(seen in the Projected.csv file)*	"Invalid selection. This metric is not available"	No 'RPI' in Projected table
Find the value of Life Expectancy in the US in 2050 *(seen in the Projected.csv file)*	"Invalid selection. This metric is not available"	No '2050' in Projected table

Stakeholder feedback

Along with sending the second prototype to my stakeholders and seeing what they thought of it so far, I also wanted to ask them a few extra questions to aid my development in later prototypes. I will also send them the following description of the prototype, so they know what to expect:

"This is the second prototype of the program. It includes 11 metrics so far and now has the ability to select multiple of them. They are also now divided by type of map and can be accessed by selecting between 'Environmental' and 'Social' in the dropdown. You can now also select the year you would like to view by clicking on the radio button you want (the years span 2000-2019). You can still pan, zoom, and hover over countries to see their index score. You can now also access the Projected map, for up to 8 quarters into the future, which works in the same way as the Current map"

Stakeholder Questions

These are the questions I will ask my stakeholders:

1. Currently, the projected map shows the future 8 quarters. However, these don't display very drastic changes in the data. Would you rather have projections of years into the future, or keep the 'more accurate' only 8 quarters into the future?
2. Is this UI an accessible design for you? If not, what could be included?
3. Is the loading time for the maps still acceptable?
4. Anything else to add?

Questions 1 is a design query I had, concerning what dates should be projected. For questions 2-3, I am ensuring that the website is working in a way that is satisfactory for my users. Question 4 is an open question for them to add any other thoughts.

Professional Economist

Fermin:

1. **Currently, the projected map shows the future 8 quarters. However, these don't display very drastic changes in the data. Would you rather have projections of years into the future, or keep the 'more accurate' only 8 quarters into the future?**
   I would prefer to have the years projected, since the type of data does not significantly change quarter-to-quarter.
2. **Is this UI an accessible design for you? If not, what could be included?**
   Yes.
3. **Is the loading time for the maps still acceptable?**
   Yes.
4. **Anything else to add?**
   No.

Lourdes:

1. **Currently, the projected map shows the future 8 quarters. However, these don't display very drastic changes in the data. Would you rather have projections of years into the future, or keep the 'more accurate' only 8 quarters into the future?**
   I'd rather have the 'more accurate' 8 quarters into the future.
2. **Is this UI an accessible design for you? If not, what could be included?**
   Yes.
3. **Is the loading time for the maps still acceptable?**
   Yes.
4. **Anything else to add?**
   No.

Average Consumer

Jenny:

1. **Currently, the projected map shows the future 8 quarters. However, these don't display very drastic changes in the data. Would you rather have projections of years into the future, or keep the 'more accurate' only 8 quarters into the future?**
   Keep the 'more accurate' 8 quarters – I think it's more useful to see projection in the near future, since future (e.g. 10 years) wouldn't be very accurate or useful.
2. **Is this UI an accessible design for you? If not, what could be included?**
   Yes.
3. **Is the loading time for the maps still acceptable?**
   It's acceptable for selecting metrics on the same map, but switching between the pages is a bit slow.
4. **Anything else to add?**
   No.

Analysis

My stakeholders were quite divided when it came to choosing between the 'more accurate' 8 quarters and the years. I will see if there's a possibility to include both, but if not, I will go with the majority and only include 8 quarters into the future. They were still happy with the accessibility for the UI and the loading times, so I will continue to update the web app in the same style.

Intermediate Stakeholder Questions

During the early development of prototype 3, I encountered a problem regarding the new metrics, and I wished to have the opinion from my stakeholders to decide what to do. The problem is that a lot of the metrics I wanted to add had data missing, but only for particular countries/years, which meant there was a lot of data that could still be used. Here are examples of this problem:

```
# dfGovEffectiveness # missing 2001 from all countries, rest is great
# dfGovExpEduPerGDP # no data for Canada 2003, 2004, 2006
# dfIncomeTax # no data for Japan
# dfTimeForTaxes # no data for 2000-2004, for US and Japan no data for 2000-2013
# dfPoliticalStability # quite good, no 2001 data
# dfTradeTransportQuality  # only data for 6 years, probably taken every 2 years
```

Nevertheless, I wanted to get my stakeholders opinion on having no data for certain countries/years depending on the metric, and so asked them the following question:

- This concerns only the new metrics to be added. Would you rather have more metrics which may have some data missing (examples above shown), or less metrics, but have all of them with no data missing?

Responses

Professional economists

Fermin: I think it is best to include those metrics, as long as the missing data is clearly indicated in the notes.

Lourdes: I would still want to see these metrics with the available data, even if it is not complete. For the data that is available, the view would be more holistic. For those years with data missing, I would like the website to make it clear there is data missing.

Average Consumer

Jenny: Less metrics but all of them with no data missing. This way it's easier to compare the data if needed. If I only see data for one country when selected a category, I would be less interested in exploring / using the website further.

Analysis

Since most of my stakeholders agreed to still include the metrics, even if some data was missing, I decided to go ahead with it. I will also include a waring if a metric with missing data is selected.

Prototype 3

During the development of the software, I decided to make a few changes to the design of some aspects of prototype 3. The following are explanations of my reasoning behind each choice, and how it will affect the design. Some of these will be better understood within context (as they are solutions to bugs/design miscalculations), and so will be expanded upon when discussing the testing.

Changes in Prototype 3

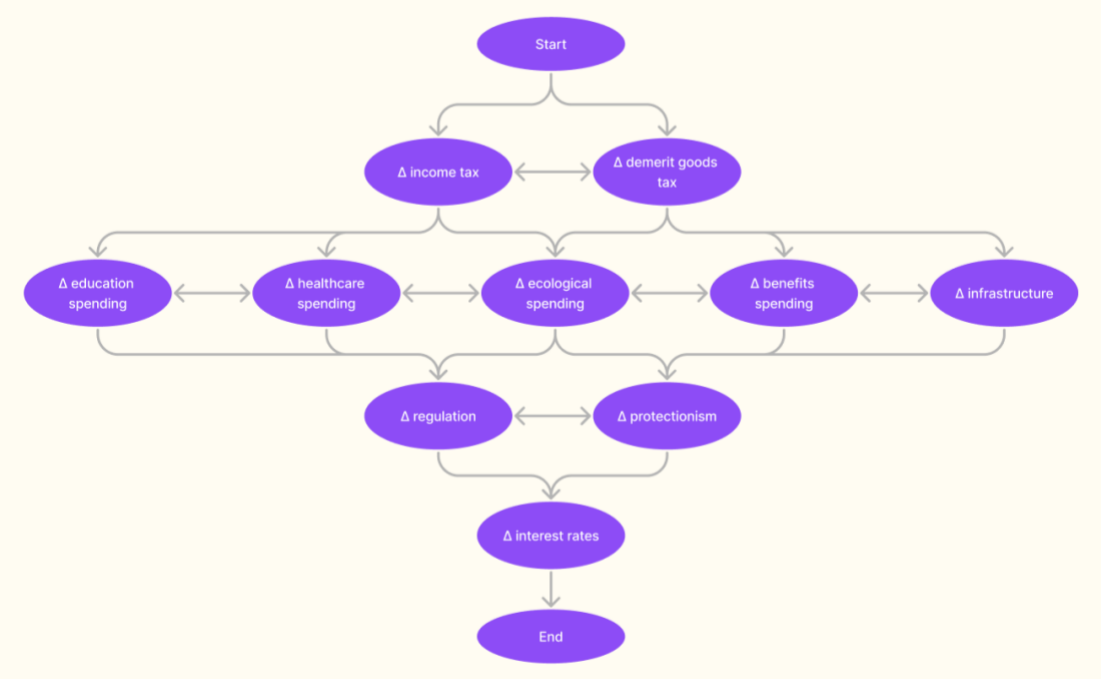
Combination of Prototype 3 and 4

<u>Change:</u> different prototypes → all changes in one

Looking at my plan for prototype 4, I realized a lot of the features had already been completed, and so I decided to combine the last two prototypes. On top of creating the economic simulator, I will add the option to see the raw data and will add the 'help' and 'information' sections. The addition of the raw data will be covered as part of the 'updates' to previous code, as it only affects the 'Current' section. The extra sections will be covered later on.

Optimum Algorithm

<u>Change:</u> previous graph → new graph



This new graph was created due to two main reasons: avoiding errors and simplicity. With the version of Dijkstra's algorithm maximising distance, there cannot be any loops within the graph. To avoid this, I created the new graph with different 'levels' and had the paths between levels be directed. The levels represent the following: government taxes, government spending, other major government policies, central bank policies. The other reason for the change, is the added simplicity. With the new way the levels are presented, it is much easier to understand the policies the algorithm is choosing to take.

Change: quantitative easing → infrastructure
I also decided to change one of the policies, going from quantitative easing to infrastructure. This is due to quantitative easing being a lot more dependent on a lot more factors than infrastructure on whether it will be effective or not. Infrastructure is also a very important factor, and quantitative easing aims to achieve the same effect as interest rates, so the switch overall allows for more choices by the user.

Change: only one visit to each policy → multiple visits allowed
Due to the nature of the new graph, the algorithm is able to select policies on each 'level' more than once, which is kept track of. Once the optimum algorithm is shown, the number of times each policy is visited is applied. This means that some policies will be raised/lowered by more than 5%, which is more accurate to real life.

Change: only positive paths considered → both positive and negative paths are considered
The 'Rules' table is made to consider only **raising** the chosen policies (by 5%), but the economic simulator should also be able to consider lowering the chosen policies, since real policies aren't constricted to only raises. To do this, the economic simulator algorithm can choose to take the positive (unadjusted 'Rules' table, so raising the policy) or the negative path (lowering the policy). This will be further explained while explaining the code of the algorithm.

Change: same rules for all countries → different rules according to slopes
Previously, the optimum plan would use the same set of rules for all countries, which is not very realistic considering they each have a different starting point. I could've resolved this by creating a different set of rules for each economy, but that would be incredibly time consuming, so I decided to use a more automatic method. It will be seen more clearly later on, but the idea is the following: if projections are negative, policies will help out more and so have a stronger effect, and if projections are positive, policies will not add as much.

Map

Changes: only one country selected → all countries displayed
For the economic simulator I had designed to select only one country and apply the rules to the singular country. This had the benefit of focusing on a single economy and seeing how changing the policies selected could maximise that economy. However, this would mean creating 7 different maps to choose between, which would unnecessarily raise the size and complexity of the program. Instead, the user can see how the same set of policies affect different economies, providing a different viewpoint for the users.

Raw data display

Changes: only seen when hovering above country → option to display data
To display the raw data, I initially planned to display the raw data when hovering over the country. However, while trying to adjust the hover template to display the averages, I found out it was extremely difficult to add more data to the hover templates. Therefore, I decided to allow the user to switch between seeing the raw data and the indexed data, so they still had access to it.

Information page

<u>Changes:</u> simply text → using the library dash_table

To display all the information about the metrics, I decided to use a table, as it would more clearly display all the information without me having to write everything in the information page with different headings and paragraphs.

Updates to previous prototypes

The following will be an analysis of the updates I added to previous parts of my code, by section, with the testing and error resolution I did throughout development. It also goes through the iterative tests I carried out for each section.

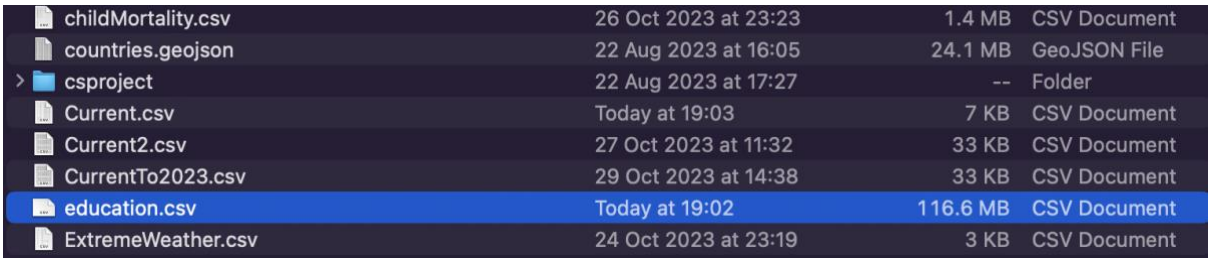
Explanation of the code is made through summaries, in-text comments (in green), and code analysis (in text-boxes over code). The errors and bugs I resolved will be highlighted in red.

Importing new metrics

The number of metrics I have imported have risen to a final 42, which are the following:
'life_expectancy', 'alcohol_use_disorders', 'fertility_rate', 'income_inequality', 'wealth_inequality', 'child_mortality', 'political_stability', 'interest_rates', 'body_mass_index', 'hours_to_file_taxes', 'income_tax', 'education_pct_government_expenditure', 'government_effectiveness', 'labour_participation', 'migration', 'rural_living_pct_popoluation', 'health_pct_government_expenditure', 'government_expenditure_pct_GDP', 'R&D_expenditure_pct_GDP', 'current_account_pct_GDP', 'inflation', 'bullying_victimisation', 'drug_use', 'mental_health', 'pedestrian_road_injuries', 'literacy_rate', 'unemployment', 'pollution', 'forest_area', 'forest_depletion_pct_GNI', 'global_carbon_budget', 'environmental_occupational_risks', 'greenhouse_gas_emissions', 'renewable_electricity_capacity', 'trade_transport_quality', 'unsafe_water', 'GDP', 'GDP_per_capita', 'GDP_PPP', 'GNI', 'GNI_per_capita', 'GNI_PPP'

While importing all these new metrics, there were some errors I encountered and two new functions I created to reduce duplicated code. I will talk through some of the general errors first, as well as some adjustments made to metrics, and then explain the two new functions.

The education metric



One of the most important metrics for the software was an education metric, preferably literacy rate as it is the most detailed. However, I found that the literacy rates in both Our World in Data and World Bank were severely lacking data, and so searched for other metrics. I found an 'education' metric in OWID, but found I could barely load it due to its massive size, so opted for other metrics. The singular metric with somewhat consistent data was 'education_lee_lee' from which I got the total enrolment into primary schooling (far from the detail included in literacy rates, but the only metric I could use).

Mental health metric

For the mental health metric, I wanted to consider all types of disorders, and so combined them into one metric as explained below:

```
# vvvvvvvvvvvvvvvvvvvv fixing mental health import
df = pd.read_csv('mentalHealth.csv')
df = df.set_index(['country', 'year', 'sex', 'age'])
```

The dataframe is already imported, so I am simply reading it and setting up the indexes.

```
df = df.xs('Both', level=2, drop_level=False)
df = df.xs('All ages', level=3, drop_level=False)
df = df.reset_index(level='sex')
df = df.reset_index(level='age')
df = df.drop(['sex', 'age'], axis=1)

df = df.sort_index()
dfCut = selectCountries(df)

# add together
share_anxiety_disorders,share_bipolar_disorders,share_depressive_disorders,share_eating_disorders,share_schizophrenia_disord
ers
df['sum'] =
df[['share_anxiety_disorders','share_bipolar_disorders','share_depressive_disorders','share_eating_disorders','share_schizophrenia_
disorders']].sum(axis=1)
dfCut['sum'] =
dfCut[['share_anxiety_disorders','share_bipolar_disorders','share_depressive_disorders','share_eating_disorders','share_schizophre
nia_disorders']].sum(axis=1)

df = indexOWID(df['sum'], dfCut['sum'])

dfMentalHealth = df.rename('mental_health')
# ^^^^^^^^ mental health import
```

These two lines 'drop' (delete) all data except where sex='Both' (both genders are considered) and age='All Ages' (all ages are considered).

I then drop the attributes sex and age, as they are no longer needed.

This creates a 'sum' attribute which sums all the attributed selected.

This code means I can now have a more holistic metric for mental health.

Incomplete reads

http.client.IncompleteRead: IncompleteRead(4234425 bytes read, 4402785 more expected)

This was a network connection error, which I was able to resolve easily by retrying, but it was something that had never occurred before.

Our World in Data timeouts

EOF occurred in violation of protocol (_ssl.c:1129)

While importing form World Bank, I run into the error above. I found out it was an error caused by too many requests sent at a time for World Bank. There was no way around it, so I had to search for metrics and check through them in small batches. It thankfully had no impact on the importing of all metrics (as it took some time to process the data before importing the next metric), but it still lengthened the process of finding metrics by a lot.

Functions

The two below functions reduce the amount of repeated code by grouping all the previous functions.

```
# used to reduce repeated code when importing data from OWID
```

```
def finiliseOWID(df_in, metricName, tableName):
    df = df_in[metricName]
    dfAll = df.sort_index()
    df = selectCountries(dfAll)
    df = indexOWID(dfAll, df)
    df = df.rename(tableName)
    return df
# I will only apply this function to the metrics from prototype 2 and onwards, to show the growth of the code

# same as finilisedWB, used to reduce repeated code
def finiliseWB(code, reName):
    regions = ['WLD', 'HIC', 'OED', 'AFE', 'AFW', 'ARB', 'CEB', 'EAP', 'EAR', 'EAS', 'ECA', 'ECS', 'EMU', 'EUU', 'IBT', 'IDA', 'IDB', 'IDX',
'LAC', 'LIC', 'LMC', 'LMY', 'LTE', 'MEA', 'MIC', 'MNA', 'NAC', 'PRE', 'PST', 'SAS', 'SSA', 'SSF', 'TEA', 'TEC', 'TLA', 'TMN', 'TSA', 'TSS',
'UMC', 'IBD']
    dfAll = wb.data.DataFrame(code)
    dfAll = dfAll.drop(index=regions)
    df = wb.data.DataFrame(code,['ITA','GBR', 'USA','FRA', 'JPN', 'DEU', 'CAN'])
    df_out = formatWB(df)
    df_out = df_out.rename(reName)
    df_out = indexWB(dfAll, df_out)
    df_out = df_out.sort_index()
    return df_out



# I will only apply this function to the metrics from prototype 3 and onwards, to show the growth of the code
```

Concatenate error

Previously, I was returning df instead of df_out, since I had returned df for the finaliseOWID() function. This was a simple mistake, which resulted in the following error:

cannot concatenate object of type '<class 'tuple'>'; only Series and DataFrame objs are valid

The two functions execute the exact same code as before, and so are used to reduce the repeated code.

Raw data imports and display

For the importing the raw data, I decided to simply re-use the same code without indexing, as it meant less coding for the same result, and it wouldn't have any impact on the performance of the program.

Here is an example of the adjustments made to the code:

```
def finiliseOWID(df_in, metricName, tableName):
    df = df_in[metricName]
    dfAll = df.sort_index()
    df = selectCountries(dfAll)
    df = df.rename(tableName)
    return df
```

No indexing step.

This new code created another .csv file, now with the raw data, named ‘CurrentRaw’.
For the app, I created two new objects out of the CCMaps class with the ‘CurrentRaw’ data instead of the ‘Current’ data.

```
SCurrentRawData = CCMap('blue', 'CurrentRaw.csv')
ECurrentRawData = CCMap('green', 'CurrentRaw.csv')

currentRawMaps = {'SCurrentRawData': SCurrentRawData, 'ECurrentRawData': ECurrentRawData}
```

I then created a set of radio items so the user can choose between seeing the indexed and the raw data. This is then used in the callback to switch between the ‘Current’ and ‘CurrentRaw’ tables.

```
    dbc.Row(
        dcc.RadioItems(options=['Indexed Data', 'Raw Data'],
                       id='data-choiceC',
                       value='Indexed Data',
                       inline = True,
                       labelStyle={'padding':'10px'}
        )
    ),
```

```
@callback(
   Output(component_id='currentMap', component_property='figure'),
   [Input(component_id='metric-choiceC', component_property='value'),
    Input(component_id='date-choiceC', component_property='value'),
    Input(component_id='map-choiceC', component_property='value'),
    Input(component_id='data-choiceC', component_property='value')]
)

def update_graph(metricChoice, dateChoice, mapChoice, dataChoice):
   patchedFig = Patch()

   if dataChoice == 'Indexed Data':
      df = currentMaps['SCurrentData'].getDataframe()
   elif dataChoice == 'Raw Data':
      df = currentRawMaps['SCurrentRawData'].getDataframe()
```

Inputting the user’s choice of data

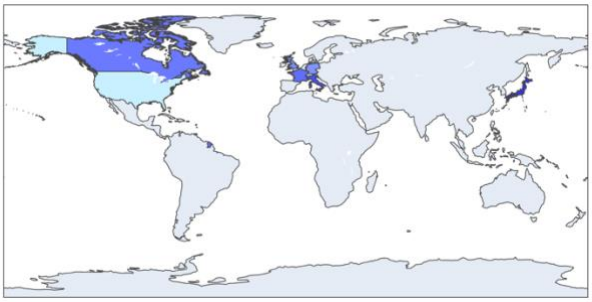
If the raw data is selected, get ‘CurrentRaw’, else get the ‘Current’ data.

```
[... rest of the callback ...]
```

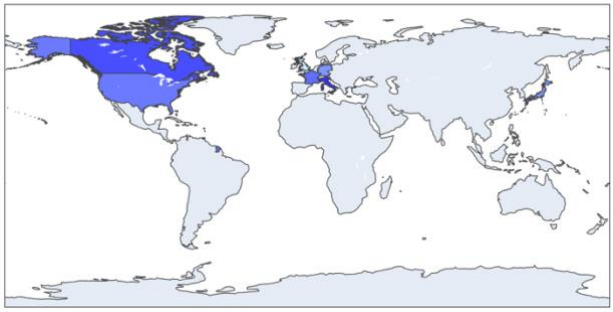
Raw data table error

Since the indexed data is based on the raw data, their two maps should be quite similar. However, as I tested how the application was working, I realized the data being shown wasn't lining up as it should.

Indexed data:



Raw data:



Some of the countries had been flipped around. I realized the countries were being assigned in alphabetical order, since the indexed data had been sorted by index, and that the raw data had not been sorted. I applied a simple piece of code to fix this:

```
df = pd.read_csv('CurrentRaw.csv')
df = df.set_index(['country', 'year'])
df = df.sort_index()
df.to_csv('CurrentRaw.csv')
```

warning when any of these metrics were selected. Here, I am creating a component so a warning can get displayed if any of these metrics are selected.

```
        html.Div(id='warningE', style= {'textAlign':'center'}),
```

The warning is originally empty.

```
@callback(
    Output(component_id='warningE', component_property='children'),
    Input(component_id='metric-choiceE', component_property='value')
)

def changeWaring(metricChoice):
    if len(metricChoice) > 0:
        for metric in metricChoice:
            if metric in
['income_tax','hours_to_file_taxes','governement_effectiveness','education_pct_government_expenditure','interest_rates','trade_tran
sport_quality','political_stability','fertility_rate','literacy_rate']:
                newWarning = 'At least one of the metrics selected is missing some data.'
                break
            else:
                newWarning = ''
        return newWarning
```

The warning display is dependent on the chosen metric.

If any of the chosen metrics are in the list of metrics that are missing data, the warning is updated to 'At least one of the metrics selected is missing some data.'. If not, it is left empty.

```
else:
    return 'At least one metric must be selected.'
```

If no metrics are selected, it displays a message for the user (telling them that at least one metric needs to be selected in order to display the data).

This is how the warning looks on the application:
(fertility_rate is missing data)

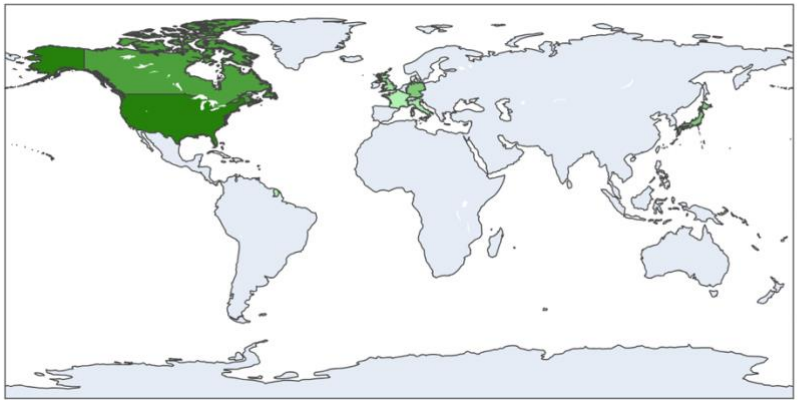
Changing colour of maps

While developing I realized that the maps had not been updating colour depending on whether environmental/social was chosen, so I adapted the following code to also update the colour:

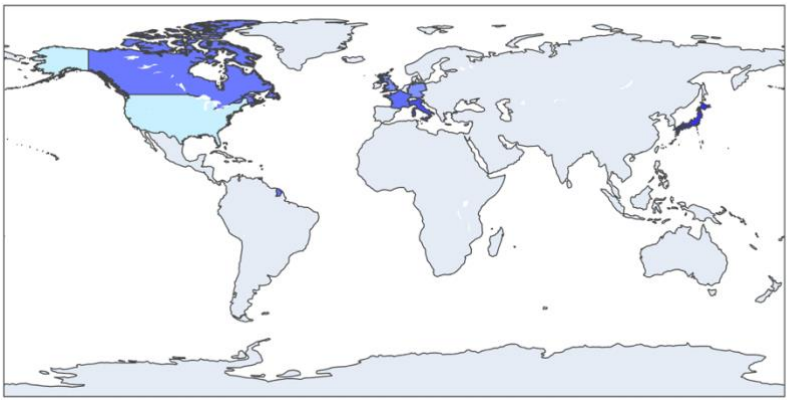
```
if mapChoice == 'Environment':
    userOptions = currentMaps['ECurrentData'].getMetrics()    # list of metrics is same for raw and indexed
    metrics = df.columns.values.tolist()
    for metric in metrics:
        if metric not in userOptions and metric != 'country':
            df = df.drop(metric, axis=1)

        patchedFig['layout']['coloraxis']['colorscale'] = ((0.0, "rgb(179, 242, 180)"), (1.0, "rgb(38, 128, 13)"))

elif mapChoice == 'Social':
    userOptions = currentMaps['SCurrentData'].getMetrics()
    metrics = df.columns.values.tolist()
```

This access the colour scale of the patched figure and changes the colours accordingly (green for environmental, blue for social).

```
for metric in metrics:
    if metric not in userOptions and metric != 'country':
        df = df.drop(metric, axis=1)
patchedFig['layout']['coloraxis']['colorscale'] = ((0.0, "rgb(200, 240, 255)"), (1.0,  "rgb(50, 50, 255)"))
```



This is how the map looks when environmental is selected. It makes it even more clear when you're selecting different sets of metrics.



And this is how it looks when social is selected.

Code Analysis and Iterative Testing

The following will be an analysis of all the major parts of my code, by section, with the testing and error resolution I did throughout development. It also goes through the iterative tests I carried out for each section.

Explanation of the code is made through summaries, in-text comments (in green), and code analysis (in text-boxes over code). The errors and bugs I resolved will be highlighted in red.

Rules table

policy	date	life_expectancy	pollution	income_inequality	wealth_inequality	GDP	GDP_per_capita	GDP_PPP
Income Tax	2019 QR:0	0	0	0	0	0	0	0
Income Tax	2019 QR:1	0	0	0	0	0	0	0
Income Tax	2019 QR:2	0	0	0	0	-0.1	-0.1	-0.1
Income Tax	2019 QR:3	0	0	0	0	-0.3	-0.3	-0.3
Income Tax	2020 QR:0	0	0	0.5	0	-0.1	-0.1	-0.1
Income Tax	2020 QR:1	0	0	0	0	-0.1	-0.1	-0.1
Income Tax	2020 QR:2	0	0	0	0	-0.1	-0.1	-0.1

The above is a section of the 'Rules' table, which I created myself following general economics rules (with the aid of professionals and the internet). All of the policies are being raised by 5% (e.g.: increase income tax by 5%) and the table shows the percentage by which the metrics change (e.g.: income inequality changes by 0.5% in 2020 QR:0).

The 'Rules' table was made with the optimum plan algorithm in mind, so I came up with a list of 'normal' and 'opposite' metrics. Normal metrics have a positive income if they increase (such as life expectancy) and opposite metrics have a positive impact if they decrease (such as income inequality). For the example 'income inequality changes by 0.5% in 2020 QR:0 with an increase income tax by 5%', the tax will affect the metric positively (so is positive in the table), but will truly decrease the metric (so is considered an opposite metric). Opposite metrics are later on considered for the economic simulator.

Economic Simulator

These are two new methods added to the EconSimCCMap class:

```
def getRules(self):
    policies = [ # list of policies ]
    # different from self.policies (attribute) due to the ordering of the Rules table
    df = pd.read_csv('Rules.csv')
    df = df.set_index(['policy', 'date'])
    for metric in self.oppositeMetrics:
        data = []
        for policy in policies:
            for date in self.dates:
                temp = -1 * df.loc[(policy, date)][metric]
                if temp == 0:
```

```
                temp = 0
            data.append(temp)
        df[metric] = data
          return df

def getEmpty(self):
    df = pd.read_csv('RulesEmpty.csv')
    df = df.set_index(['total', 'date'])
    df = df.replace("United States", "United States of America")
    return df
```

This code adapts some of the metric in the 'Rules' table. As previously discussed, some of the metrics are 'opposite', so should actually lower when having a positive impact. Here I am flipping these metrics, so they can be adjusted appropriately.

This empty table will be used in the callback of the economic simulator.

The rest of the code is part of the page for the economic simulator.

```
    dbc.Row(
      [
      dbc.Col(
          [
            dbc.Row(
              html.H1("Select the below policies to raise them by 5%",
                        style={'fontSize':15})
            ),

            dbc.Row(
              dcc.Checklist(options=[ # list of policies ],
              id='policy-choiceE',
              value=['Income_Tax']))
            ]
          ),

        dbc.Col(
          [
            dbc.Row(
              html.H1("Select the below policies to lower them by 5%",
                        style={'fontSize':15})
            ),

            dbc.Row(
              dcc.Checklist(options=[ # list of policies ],
              id='neg-policy-choiceE',
              value=['Income_Tax']))
          ]
        )
```

This creates a checklist of all the policies so users can choose to "raise them by 5%". This will be used in a later callback.

This creates a checklist of all the policies so users can choose to "lower them by 5%". This will be used in a later callback.

```
        ]
    ),
```

These two checklists are used so the user can chose their range of policies. I opted for only allowing users to change a policy by raising/lowering it by 5%, as allowing different percentages would become extremely difficult to keep track of (applying it to the rules wouldn't be too difficult, but it would require 20 more inputs to handle). This is how the checklists look on the application:


```
@callback(
    Output(component_id='econSimMap', component_property='figure'),
    [Input(component_id='metric-choiceE', component_property='value'),
     Input(component_id='date-choiceE', component_property='value'),
     Input(component_id='map-choiceE', component_property='value'),
     Input(component_id='policy-choiceE', component_property='value'),
     Input(component_id='neg-policy-choiceE', component_property='value')]
)
```

The two new inputs are for the list of policies to "raise" (policy-choiceE) and the list of policies to "lower" (neg-policy-choiceE).

```
def update_graph(metricChoice, dateChoice, mapChoice, policyChoice, negPolicyChoice):
    patchedFig = Patch()

    df = econSimMaps['SEconSim'].getDataframe()
    df = df.loc[dateChoice]
```

The callback starts in the same way as the others.

```
    dfRules = econSimMaps['SEconSim'].getRules()

    totalRules = econSimMaps['SEconSim'].getEmpty()
    for policy in policyChoice:
        totalRules = totalRules + dfRules.loc[policy]
    for policy in negPolicyChoice:
        totalRules = totalRules - dfRules.loc[policy]
```

Firstly, we get the table of 'Rules' and an empty table to hold the data. Then, according to the chosen policies, the data in 'Rules' is applied to the empty table.

```
df['date'] = df.index
df = df.reset_index(drop=True)
df = df.set_index(['country', 'date'])     # setting a different index so the data can be
                                                                    accessed and modified
for country in [ # list of countries ]:
    df.loc[(country, dateChoice)] = df.loc[(country, dateChoice)]*
                                                        (1 + totalRules.loc[('Total', dateChoice)]/100)
```

For all the countries, the policies selected are applied to the projected data by turning the percentage into a multiplier (0.5% → 1.005) and timing it with the data.

```
tempC = []          # resetting the index to be only 'date', not ['country', 'date']
tempD = []
for i in df.index:
    tempC.append(i[0])
    tempD.append(i[1])
df['date'] = tempD
df['country'] = tempC
df = df.reset_index(drop=True)
df = df.set_index('date')


        [ rest of the callback ]
```

Index error

To apply the rules to the data, I first needed to change the index from 'date' to ['country', 'date'] to be able to iterate through it. However, the rest of the code relies on a dataframe with only 'date' as the index. To fix this, I employ this code to save columns out of the index and then set the index 'date'.

This code uses the user's chosen policies and uses the 'Rules' table to create a dataframe of all the changes that should occur. It then applies these changes to the projected data of each country. This allows users to change their policies however they see fit and see how they fare depending on the country and metrics.

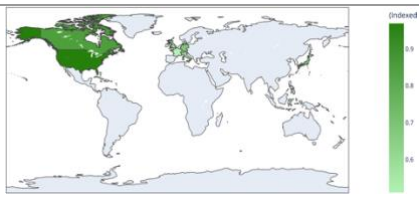
Tests:

For this test, some of the inputs are inside the code. I did not add validation for invalid inputs as users cannot access them as previously explained.

The policies/metrics/dates used are not specific, and any others could have fulfilled the point of the tests.

Input	Expected result	Output
Select "Raise interest rates by 5%" with GDP, and check the correct data is selected `print(dfRules.loc['Interest_Rates']['GDP'])`	0.0,0.0,-0.1,-0.3,-0.1,-0.1,-0.1, -0.1,-0.1,-2.0,-2.0, -3.0	date 2019 QR:0 0.0 2019 QR:1 0.0 2019 QR:2 -0.1 2019 QR:3 -0.3 2020 QR:0 -0.1 2020 QR:1 -0.1 2020 QR:2 -0.1 2020 QR:3 -0.1

		2021 QR:0 -0.1 2025 -2.0 2035 -2.0 2050 -3.0
Select “Raise interest rates by 5%” with GDP and life expectancy, and check the correct data is selected	0.0,0.0,-0.1,-0.3,-0.1,-0.1,-0.1, -0.1,-0.1,-2.0,-2.0,-3.0 *(interest rates have no effect on life expectancy)*	date 2019 QR:0 0.0 2019 QR:1 0.0 2019 QR:2 -0.1 2019 QR:3 -0.3 2020 QR:0 -0.1 2020 QR:1 -0.1 2020 QR:2 -0.1 2020 QR:3 -0.1 2021 QR:0 -0.1 2025 -2.0 2035 -2.0 2050 -3.0
Select “Raise interest rates by 5%” and “Increase regulation by 5%” with GDP, and check the correct data is selected	-2.0, -2.0, -1.0, -0.1, -0.3, -0.1, -2.0, -0.1, -0.1, -2.1, -0.1	date 2019 QR:0 -2.0 2019 QR:1 -2.0 2019 QR:2 -1.0 2019 QR:3 -0.1 2020 QR:0 -0.3 2020 QR:1 -0.1 2020 QR:2 -2.0 2020 QR:3 -0.1 2021 QR:0 -0.1 2025 -0.1 2035 -2.1 2050 -0.1
Select “Raise interest rates by 2%” with all metrics, and check the correct data is selected	-0.1,-0.2,0.2,0.5,0.0,0.1, 0.7,0.8,0.1,13.5,8.0,9.0	date 2019 QR:0 -0.1 2019 QR:1 -0.2 2019 QR:2 0.2 2019 QR:3 0.5 2020 QR:0 0.0 2020 QR:1 0.1 2020 QR:2 0.7 2020 QR:3 0.8 2021 QR:0 0.1 2025 13.5 2035 8.0 2050 9.0
Select all policies with GDP, and calculate the changes to the metric (total of all dates)	9.4	9.400000000000002 *This is due to an error with the way .sum() works, but is resolved in the actual code.*
Select no metric and check selected data	“Invalid selection. At least one metric must be selected”	No data is outputted.

Select no policy metric calculate the changes to the metric	"Invalid selection. At least one policy must be selected"	No data is outputted.
Select no country and check selected data	"Invalid selection. One country must be selected"	No data is outputted.
Display the maps for the above tests	Map of the selected country, with correct saturation	 *(saturations are accurate)*

These tests ensure the economic simulator based on users' policies is working correctly

Optimum Plan Algorithm

For the optimum plan, I decided to use the slopes of the projected data for each country and metric to make the optimum plans for each country more accurate. I added this to the projected data algorithm to store the slopes (to be used later in the optimum plan algorithm):

Only the code for the additions will be shown, anything related to the projected algorithm can be seen in its own section.

```
slopesData = {}

for metric in metrics:
    slopesMetricList = []
    for country in countries:
        x = np.array([i for i in range(2000, 2020)])
        y = np.array(data[country][metric])

        length = 0
        while length < len(y):
            if str(y[length]) == 'nan':
                y = np.delete(y, length)
                x = np.delete(x, length)
            else:
                length = length + 1
        if len(x) == 0:
            for i in range(15):
                metricList.append('NaN')
            slopesMetricList.append('NaN')
            continue

        x = x.reshape((-1, 1))

        model = LinearRegression()
        model.fit(x, y)

        slope = model.coef_
        slopesMetricList.append(slope[0])
```

For all the values in the y axis (the actual data), these values are deleted if they are NaN.
(The slope cannot be calculated with NaN values).

If all of the values were NaN, len(x) should be 0 and so a slope will not be able to be calculated. The slope is instead stored as NaN.

```
    slopesData[metric] = slopesMetricList

# this is for the economic simulator
slopesDf = pd.DataFrame(data= slopesData, index= countries)
slopesDf.index.name = 'country'
slopesDf.to_csv('Slopes.csv')
```

This is an extract of how the 'Slopes' table looks like:

country	life_expectancy	fertility_rate	GDP	GDP_per_capita	GDP_PPP
United Kingdom	0.0014374755846529700	0.002413779468968420	-0.001291971864106840	-0.008290309644418220	0.001714446648143970
United States	-0.0015976297252795400	-0.0010149024441975400	0.0	-0.003801052684065420	0.0
France	0.0020884250379659900	0.0022170500266978000	-0.00016896994080743600	-0.003525764501530850	0.001763854246971050
Japan	0.0018255507933722100	0.0014431458236170100	-0.008284517120916420	-0.013582884370687900	-0.001963523594060180
Germany	0.0004808163815308250 0	0.0032368726856419400	-0.00015759907224177900	-0.0006777000124930250	0.002067680977187710
Canada	0.001021862922533780	0.0012917081533471400	0.0012464253515615700	0.000350812911897669	0.002006349773535560
Italy	0.002076160480764860	0.0010264505506797900	-0.0008659075973560320	-0.005520702643730260	0.0007582412872340050

The slopes data is then used in the optimum plan algorithm:

```
import pandas as pd

dfRules = pd.read_csv("Rules.csv")
dfRules = dfRules.set_index(['policy','date'])

dfSlopes = pd.read_csv("Slopes.csv")
dfSlopes = dfSlopes.set_index('country')

def optPlan(dfR, dfS, c):
    dfS = dfS.loc[c]
    # dont need to check for NaN values, since
    # 1. Slopes are only NaN if the WHOLE metric for the country is NaN
    # 2. and so it'd be NaN * NaN (no change)
    # (The slopes will never override data)


    policies = ['Income Tax', 'Demerit Goods Tax', 'Education Spending', 'Healthcare Spending', 'Environment Spending', 'Benefits Spending', 'Interest Rates', 'Regulation', 'Protectionism', 'Infrastructure']

    data = {}

    for policy in policies:
```

NaN values

While applying the Slopes data to the optimum plan, I worried that the NaN values would create errors. However, the comments here explain why it cannot occur.

Creates a dictionary out of the sums of the policies for all dates with the slopes applied.

```
    temp = dfR.loc[policy].sum(axis='index')
    temp = temp * (1 - (5*dfS))
    s = temp.sum()
    data[policy] = s
```

Underscore errors

Some of the policies are written without _ while other have no _. I simply kept track of it as I wanted to avoid unnecessary errors, but later code mostly uses _.

```
# WARNING,, THE KEYS ARE NOT UPDATED TO HAVE _
Start = {'Income_Tax': data['Income Tax'], 'Demerit_Goods_Tax': data['Demerit Goods Tax']}
Income_Tax = {'Demerit_Goods_Tax': data['Demerit Goods Tax'], 'Education_Spending': data['Education Spending'],
'Healthcare_Spending': data['Healthcare Spending'], 'Environment_Spending': data['Environment Spending'], 'Benefits_Spending':
data['Benefits Spending'], 'Infrastructure': data['Infrastructure']}

# the rest of the nodes are initialised accordingly, these are a few examples

Interest_Rates = {'End': 0}
End = {}
```

For the nodes I decided to use dictionaries and the 'data' dictionary to assign the distances.

```
nodes = {'Start': Start, 'Income_Tax': Income_Tax, 'Demerit_Goods_Tax': Demerit_Goods_Tax, 'Education_Spending':
Education_Spending, 'Healthcare_Spending':Healthcare_Spending, 'Benefits_Spending': Benefits_Spending,
'Environment_Spending': Environment_Spending,
'Infrastructure': Infrastructure, 'Regulation': Regulation, 'Protectionism': Protectionism, 'Interest_Rates': Interest_Rates, 'End': End}
```

A dictionary of all the nodes.

```
start = 'Start'
distance = {}

for node in nodes:
    if node == start:
        distance[node] = 0
    else:
        distance[node] = -100000
```

Initialising the nodes to all have a large negative distance to start with.

```
priorityQueue = ['Start', 'Income_Tax', 'Demerit_Goods_Tax', 'Education_Spending', 'Healthcare_Spending', 'Benefits_Spending',
'Environment_Spending', 'Infrastructure', 'Regulation', 'Protectionism', 'Interest_Rates', 'End']
previous = {}
```

The priority queue will keep track of what nodes should be visited first.

'previous' keeps track of all the visited nodes and how many times they were visited (the optimum plan).

```
while len(priorityQueue) > 0:
    currentNode = priorityQueue.pop(0)
    for neighbour in nodes[currentNode]:


        newDist = distance[currentNode] + nodes[currentNode][neighbour]
        if newDist > distance[neighbour]:
            distance[neighbour] = newDist
            sortedDist = dict(sorted(distance.items(), key=lambda item: item[1],
                                                                    reverse=True))
            temp = []
            for node in sortedDist:
                if node in priorityQueue:
                    temp.append(node)
            priorityQueue = temp
            if neighbour not in previous:
                previous[neighbour] = 1
            else:
                previous[neighbour] += 1

        newDist = distance[currentNode] - nodes[currentNode][neighbour]
        if newDist > distance[neighbour]:
            distance[neighbour] = newDist
            sortedDist = dict(sorted(distance.items(), key=lambda item: item[1],
                                                                    reverse=True))
            temp = []
            for node in sortedDist:
                if node in priorityQueue:
                    temp.append(node)
            priorityQueue = temp
            if ('Negative' + neighbour) not in previous:
                previous["Negative " + neighbour] = 1
            else:
                previous["Negative " + neighbour] += 1

for node in distance:
    distance[node] = round(distance[node], 1)

return previous, distance
```

For every node in the priority list, and for every neighbour of that node, the 'positive' and 'negative' distance of that policy is calculated to find the best path from the current node to the neighbour nodes.

If the new distance is greater than the previous one, the distance is updated, and the priority queue is reordered in descending order by distance. The neighbour is then added to the 'previous' list.

Same process for the negative paths.

Sums error

When using operations with pandas and dictionaries, the distances were being calculated to be slightly off the actual number (eg: 9.39999999999 instead of 9.4), so this simply rounds the distance to be accurate.

```
econSimData = []

countries = ["Italy", "United Kingdom", "United States", "France", "Japan", "Germany", "Canada"]
for country in countries:
    route, distances = optPlan(dfRules, dfSlopes, country)

    countryData = {}
    for path in route:
        if 'Negative' in path:
            temp = path.partition(' ')[2]
            countryData[temp] = countryData[temp] - route[path]
        elif path not in econSimData and path != 'End':
            countryData[path] = route[path]

    for policy in countryData:
        econSimData.append(countryData[policy])



temp = {}
temp['amount'] = econSimData       # just to have the column named 'amount' in the dataframe
econSimData = temp


policies = ['Income_Tax', 'Demerit_Goods_Tax', 'Education_Spending', 'Healthcare_Spending', 'Environment_Spending',
'Benefits_Spending', 'Infrastructure', 'Regulation', 'Protectionism', 'Interest_Rates']

econSimIndex = []
for country in countries:
    for policy in policies:
        econSimIndex.append((country, policy))


econSimIndex = pd.MultiIndex.from_tuples(econSimIndex, names=["country", "policy"])

projectedDf = pd.DataFrame(data= econSimData, index=econSimIndex)
projectedDf.to_csv('EconSimOptPlan.csv')
```

The optimum plan is calculated for all countries.

And it is formatted so I can use it in the application.

Creates a dataframe out of the optimum plans.

Tests:

I heavily adapted these tests so they could accurately assess the optimum plan algorithm. They are the tests I carried out thought development.

The policies/metrics/dates used are not specific, and any others could have fulfilled the point of the tests. For some of these tests I only said 'accurate results' as they were too long to show.

Input	Expected result	Output

Slopes data is accurately applied to the policies in Italy	{'Income_Tax': -73.9, 'Demerit_Goods_Tax': 12.8, 'Education_Spending': 153.2, 'Healthcare_Spending': 206.2, 'Benefits_Spending': 154.8, 'Environment_Spending': 221.5, 'Infrastructure': 109.9, 'Regulation': 32.5, 'Protectionism': 1.0, 'Interest_Rates': -72.2} *The above data was rounded when manually calculating.*	{'Income_Tax': -73.8696768672307, 'Demerit_Goods_Tax': 12.815221052784405, 'Education_Spending': 153.15614544058096, 'Healthcare_Spending': 206.2319346498429, 'Benefits_Spending': 154.76600089341065, 'Environment_Spending': 221.5435864005716, 'Infrastructure': 109.89831446448285, 'Regulation': 32.52521176361404, 'Protectionism': 1.027562362041052, 'Interest_Rates': -72.21623289744703}
Slopes data is accurately applied to the policies of all countries	*Accurate results*	*Results was all accurate*
Get the distance from Income Tax to Education Spending in Italy	153.2	153.15614544058096
Get the distance from Income Tax to Interest Rates in Italy	"There is no path between Income Tax and Interest Rates"	Error

Run adapted Dijkstra's algorithm for the below simplified graph:
`A = {'B': -10, 'C': 5}`
`B = {'C': 2, 'D': -1}`
`C = {'B': 2, 'D': -3}`
`D = {}`
	previous = {'B': 2, 'Negative B': 1, 'C': 2, 'D': 1, 'Negative D': 1}
	
	distance = {'A': 0, 'B': 14, 'C': 12, 'D': 15}
		previous = {'B': 2, 'Negative B': 1, 'C': 2, 'D': 1, 'Negative D': 1}
		
		distance = {'A': 0, 'B': 14, 'C': 12, 'D': 15}

Run adapted Dijkstra's algorithm for actual policies	*Accurate results*	*Results was all accurate*
Run function for China	"Invalid selection. There is no data for China"	KeyError: 'China'

These tests ensure the 'optimised' plan is processed as it should be.

Optimum Plan Display

These are another two new methods added to the EconSimCCMap class:

```
def getOptPlan(self):
    df = pd.read_csv('EconSimOptPlan.csv')
```

Adapts the data from the 'OptPlan' table to a format easier to use in the callback.

```
    df = df.set_index(['country', 'policy'])

    optPlan = {}
    for i in df.index.tolist():
        optPlan[i] = df.loc[i]['amount']

    return optPlan

def getOptPlanText(self, country):
    df = pd.read_csv('EconSimOptPlan.csv')
    df = df.set_index(['country', 'policy'])

    policies = [ # list of policies ]

    optPlanText = []

    for policy in policies:
        temp = policy + ': ' + str((df.loc[(country, policy)]['amount'] * 5)) + '%       \n'

        optPlanText.append(temp)
    optPlanText = ''.join(optPlanText)

    return optPlanText
```

Creates the text that will be shown on the page of the optimum plan (displaying the policies for each country).

First, the below elements were added to display the optimum plans for each country:

```
    dbc.Row(
        dcc.Dropdown(options=["Italy", "United Kingdom", "United States", "France", "Japan",
                                                                    "Germany", "Canada"],
                     id='country-choice',
                     value='Italy',
                     clearable=False
        )
    ),

    dbc.Row(
        dcc.Textarea(id='optPlanText', value= '', disabled=True, style=
                                                    {'textAlign':'center', 'height': 260})
    ),
```

A dropdown allows users to choose between countries and the text displays the according optimum plan.

And this callback utilises getOptPlanText() function to update and display the optimum plans.

```
@callback(
    Output(component_id='optPlanText', component_property='children'),
```

```
    Input(component_id='country-choice', component_property='value')
)
def changeWaring(countryChoice):
    optPlanText = econSimMaps['SEconSim'].getOptPlanText(countryChoice)
    return optPlanText
```

This code allows users to change between countries and see what the different optimum plans are.

```
@callback(
    Output(component_id='optPlanMap', component_property='figure'),
    [Input(component_id='metric-choiceO', component_property='value'),
     Input(component_id='date-choiceO', component_property='value'),
     Input(component_id='map-choiceO', component_property='value')]
)
```

There are no new inputs as this page simply displays the optimum plan for all the countries.

```
def update_graph(metricChoice, dateChoice, mapChoice):
    patchedFig = Patch()

    df = econSimMaps['SEconSim'].getDataframe()
    df = df.loc[dateChoice]

    dfRules = econSimMaps['SEconSim'].getRules()


    countries = [ # list of countries ]
    totalRules = econSimMaps['SEconSim'].getOptEmpty()
    optPlan =  econSimMaps['SEconSim'].getOptPlan()
    for country in countries:
        for policy in econSimMaps['SEconSim'].policies:
            sumOfRules = dfRules.loc[policy] * optPlan[(country, policy)]
            total = totalRules.add(sumOfRules, fill_value=0)
            totalRules.loc[country] = total


    df['date'] = df.index
    df = df.reset_index(drop=True)
    df = df.set_index(['country', 'date'])       # setting a different axis so the data can be
                                                                    accessed and modified
    for country in countries:
        df.loc[(country, dateChoice)] = df.loc[(country, dateChoice)]*(1 +
```

The callback starts in the same way as the others.

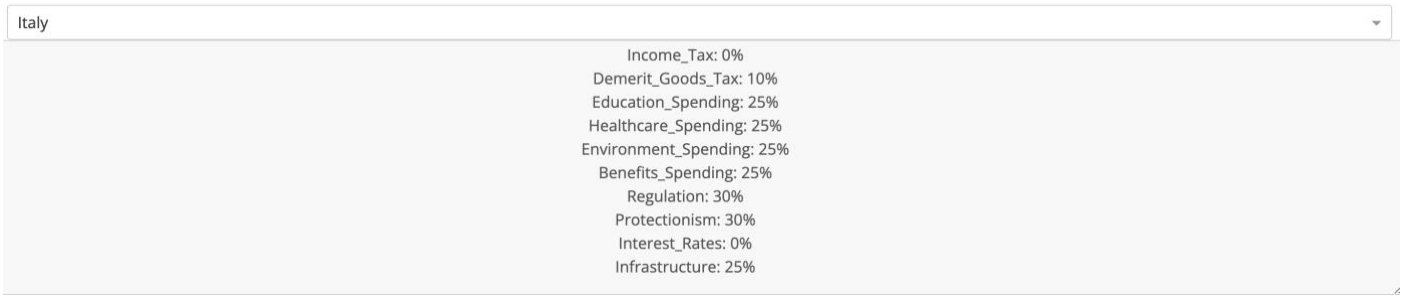
The callback gets the optimum plans and calculates the new data for each country.

The sum of the changes of all the policies is calculated by adding policies to the originally empty table ‘totalRules’.

```
                                                    totalRules.loc[(country, dateChoice)]/100)


    tempC = []                  # resetting the index to be only 'data'
    tempD = []
    for i in df.index:
        tempC.append(i[0])
        tempD.append(i[1])
    df['date'] = tempD
    df['country'] = tempC
    df = df.reset_index(drop=True)
    df = df.set_index('date')

     [ rest of the callback ]
```

This code uses the already calculated optimum plan of each country to create a dataframe of all the changes that should occur. It then applies these changes to the projected data of each country.

This is how the text of the optimum plan looks on the page:



Help and Information

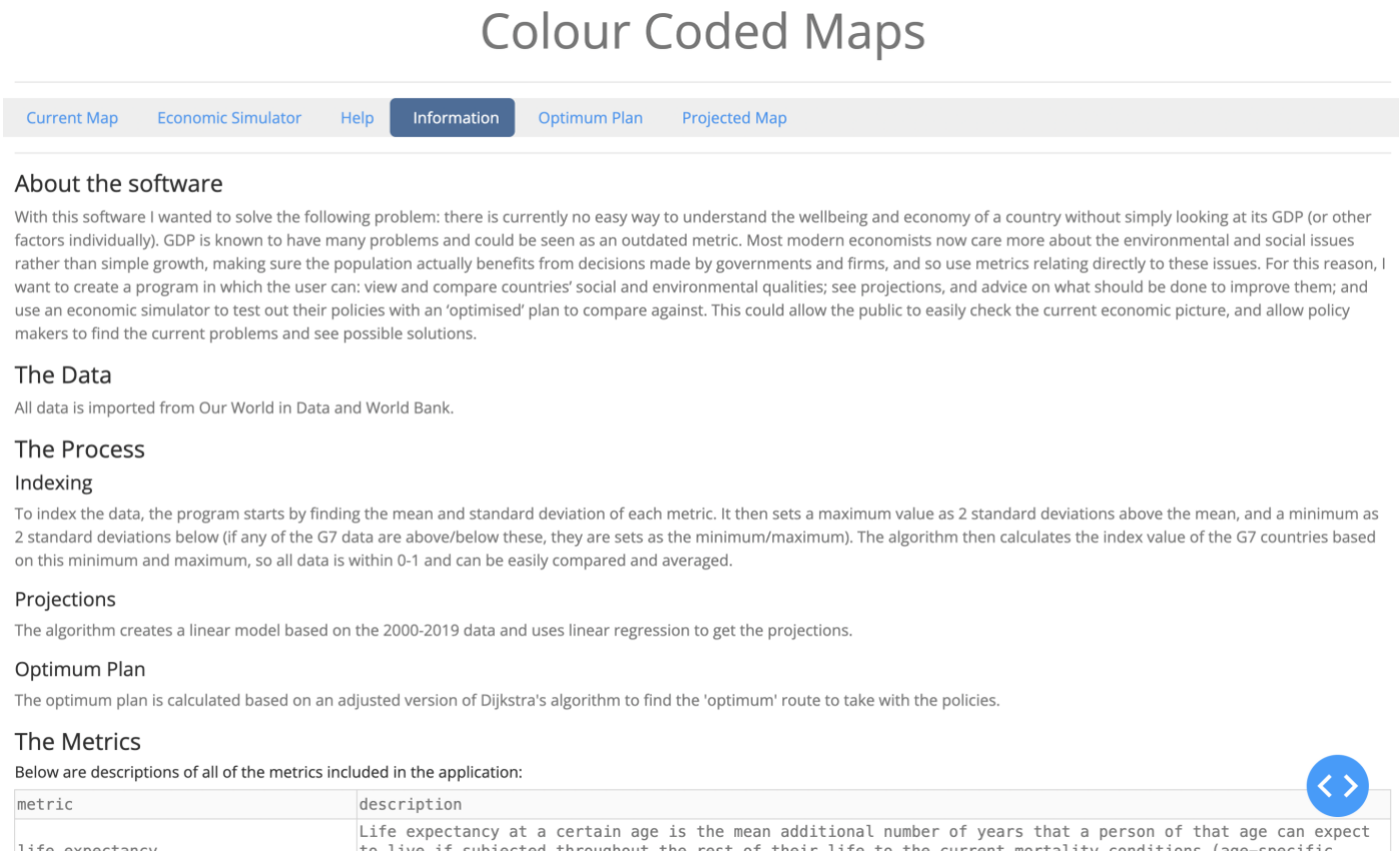
For the information section I decided to use a table to display the descriptions of the metrics. The rest of the information is displayed in text.

```
df = pd.read_csv('Info.csv')


layout = html.Div(
    [
        dbc.Row([
            html.H4('About the software'),
            html.P( # explanation of the aim of the software )
        ]),

        dbc.Row([
            html.H4('The Data'),
            html.P('All data is imported from Our World in Data and World Bank.')
        ]),
```

```
        dbc.Row([
            html.H4('The Process'),
            html.H5('Indexing'),
            html.P( # explanation of the indexing process ),
            html.H5('Projections'),
            html.P('The algorithm creates a linear model based on the 2000-2019 data and uses linear regression to get the projections.'),
            html.H5('Optimum Plan'),
            html.P("The optimum plan is calculated based on an adjusted version of Dijkstra's algorithm to find the 'optimum' route to take with the policies.")
        ]),

        dbc.Row([
            html.H4('The Metrics'),
            html.H6('Below are descriptions of all of the metrics included in the application:'),

            dash_table.DataTable(
                data=df.to_dict('records'),
                columns=[{'id': c, 'name': c} for c in df.columns],
                style_cell={'textAlign': 'left'},
                style_data={
                    'whiteSpace': 'normal',
                    'height': 'auto',
                }
            )
        ])

    ]
)
```

Creates a table out of the data in the dataframe ‘Info’ which includes all the metrics and their definitions.

It looks like this on the webpage:



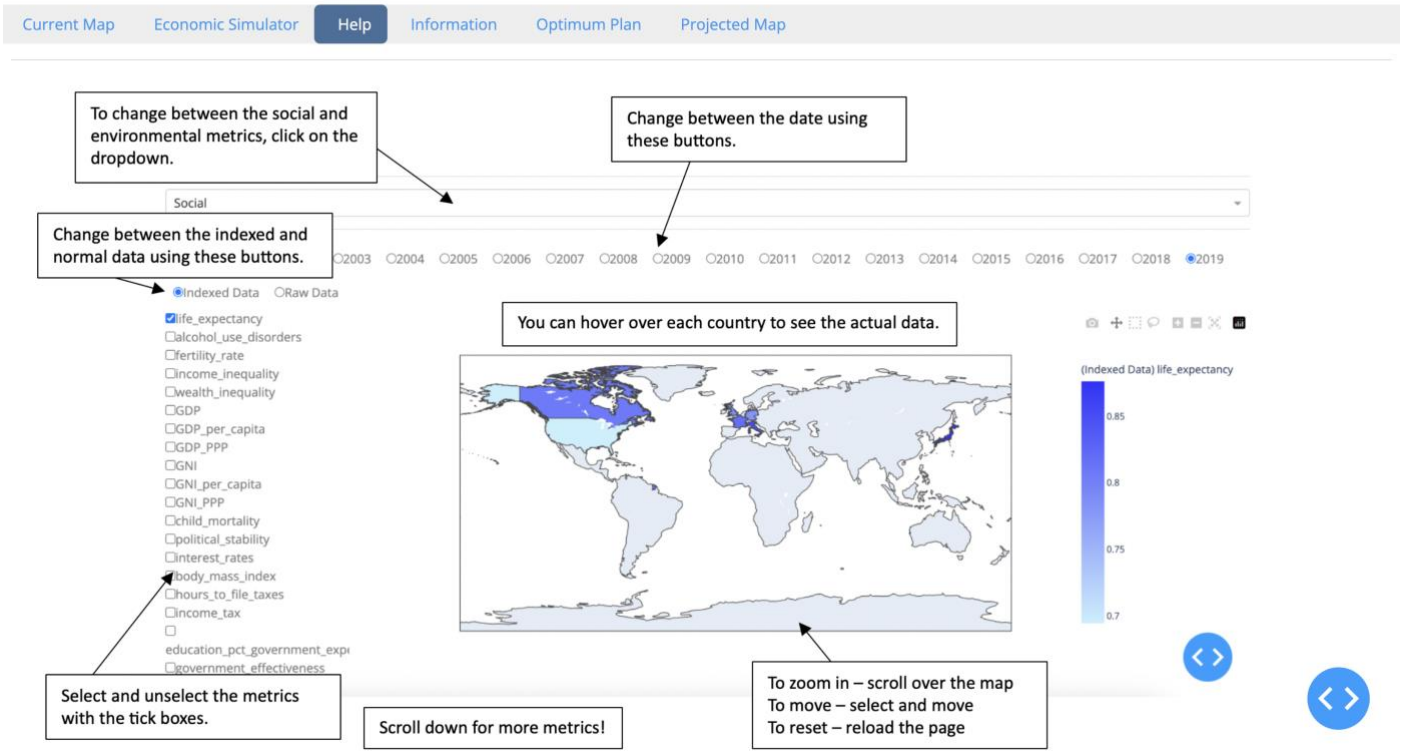
For the help section, I decided the most effective way to help users would be through visual aid, so I created some images detailing how to use the program and included them in the help page.

The code shows a single image, as it is incredibly repetitive.

```
image_path = 'assets/helpForCurrent.png'


layout = html.Div(
    [

        dbc.Row(
            html.Img(src=image_path)
        )

    ]
)
```

This is how it looks on the page:

Colour Coded Maps



Final User Interface Tests

Now that the program is complete, this is the checklist I will use to make sure the user interface meets all the *key* requirements:

Action	Working?
Metrics can be selected and unselected	Y
All metrics can be selected and unselected at once	N
Metrics alter the maps displayed accordingly	Y
Colour-coded maps are displayed	Y
Key information about the maps (data and key) are displayed	Y
Users can switch between maps through the menu	Y
All maps have the appropriate colours	Y
All maps that need a dateline have them	Y
The dateline can be used to select a quarter and alters the maps displayed accordingly	Y
Users can access an 'information' section	Y
Users can access a 'help' section	Y
Users can select a country for the economic simulator	N (Adapted)
Users can select and unselect policies for the economic simulator	Y

Users can view an ‘optimal’ plan for the economic simulator	Y

(The failed requirement will be covered in the evaluation section – the main reason for the failure was time and that it is not a key feature).

Furthermore, the following is a checklist for more features my stakeholders wished for, but are not *key* for the function of the program:

Action	**Working?**
Users can click on a country to see its raw data	Y (Adapted)
Users can interact with this raw data and view it in different formats	Y
Labels for the countries can be switched on/off	Y
Users can zoom in on the map	Y
Dateline labels only appear when the mouse is on it	N

(Again, the failed requirement will be covered in the evaluation section – and again, the main reason for the failure was time and that it is not a key feature).

Evaluation

Final Stakeholder Feedback

As part of my evaluation, I wanted to get my final stakeholder feedback on now all of the available features in the program. With these questions I want the make sure my stakeholders are happy with all the key features, and I want to get their opinions on the features I wasn't able to deliver. There are also some questions for features/additions that could be implemented in later versions at the request of the stakeholders.
Since there will be a lot of questions, they will be divided by the section of the program to make them clearer.

Questions

The questions I will ask all of my stakeholders are as follows:

Part A – Current Maps

1. Is the user interface clear, and do you know how to use everything that is on display?
    - Is there any feature that you do not see how to access?
2. Is the display of the raw data as its own map acceptable?
3. Do you mind that there is no way to access another visual representation of the raw data, as had been requested?

Part B – Projected Maps

1. Are the dates provided acceptable?
2. Is there any problem with the fact that there is not a large change in the projected data (as data takes long to change)?
3. Would you rather a more complex economical algorithm was used to project the data, rather than a simple regression?
    - If this was available, would you want the option of multiple projection algorithms?
4. The projected data does not re-index the data based on the projections of all countries, but simply projects the indexes of the G7. Is this alright (*there is unlikely to be any major changes)*?

Part C – Economic Simulator

1. Do you mind that policies can only be changed by +-5%?
2. Are the available policies acceptable?
    - Are there any major policies missing?
    - Are any of the policies unnecessary?
3. Are the available dates acceptable?
4. Is there any problem with the initial values of the checklist being set to both +5% and -5% on income tax (i.e.: no change)?
5. Is there any problem with the fact that there is at times not a large change in the data (as policies might only apply later on)?
    - Could this be made clearer?

Part D – Optimum Plan

1. Is the layout clear?
2. Do you mind that there is little change between the different plans for the G7?

3. Would you like an explanation of the plan included within the page (otherwise in the information section)?
4. Would you like an explanation of the algorithm to also be included within the page (otherwise in the information section)?

Part E – Help and Information

1. Is the information page well structured?
2. Is there any major missing information section?
3. Are the metric descriptions sufficient?
    - Would you like references for them?
4. Is the style of the help section appropriate and clear?
5. Does the help section have anything missing?

Part F – General/Other questions

1. Are the colours of the two maps visually appealing?
    - Do they fit the 'social' and 'environmental' themes well?
2. Do you mind that the menu is in alphabetical order (rather than ordered by section)?
3. Is the style appropriate and consistent?
4. Are there enough metrics?
    - Are there any major metrics missing?
    - Are any of the metrics unnecessary/inappropriate?
    - Is there a problem with the number of environmental metrics compared to the number of social metrics (*I attempted to make it somewhat even, but there were a lot more social than environmental within the databases used)?*
5. Is the warning for missing data clear?
    - Could it be adjusted in some way?
6. Do you mind that there is no select/unselect all?
7. Is the key clear, even when it changes to fit the data selected?
8. Is the information provided in the hover text enough?

Part G – Overall program

1. Are you overall happy with the whole program?
2. Have all your key requirements been met?
3. Do you believe the program solves the original problem (accessibility to economic data)?
4. Is there anything missing?
5. Any other comments:

Analysis

Question A.1 aims to see how the users feel about the user interface in the current maps. The current maps have most of the features the other maps do, so it is a good basis to see if the features are laid out clearly and are easily accessible.

Question A.2 will ensure users are happy with the way the raw data is displayed, and A.3 aims to determine if the extra representation of the raw data was something the stakeholders really wanted. It may still be added in later versions, but I want to make sure my stakeholders are still content.

Question B.1 ensures the users are happy with the dates provided, and B.2 asks if they are happy even if the provided dates do not show a great change in the data.
Question B.3 is for future versions, to see what kind of features the stakeholders would want. Since the projection algorithm is extremely simple, I think my stakeholders would want the option for a more complex one (perhaps considering economic trends and theories).
The aim of Question B.4 is also to see if the software needs to be updated in future versions. Currently, the indexes are simply extrapolated for the projections as it is much simpler (and so takes less time and storage), but it might be something my stakeholders want in the future.

Question C.1 is something I had considered applying, but ended up not doing due to the complexity of the required callbacks (a different input for the 14 policies available). This questions whether the stakeholders mind that only +-5% is allowed, and so whether they want more freedom in later versions.
Question C.2 and C.3 make sure the stakeholders are content with the provided policies and dates.
Question C.4 is simply a preference question, it can be easily changed according to what the stakeholders wish, and question C.5, similar to B.2, ensures the users are happy with the clarity of the changes in the data and how they are shown.

Question D.1 aims to see how the users feel about the layout of the policies in the optimum plan. I aimed to make it as clear as possible, but I wanted to see what the stakeholders thought of it.
D.2 is something that cannot be easily changed. Due to the nature of the optimum plan algorithm, the fact that the G7 have similar starting points and that the same rules are applied, the plans for all the countries are extremely similar. Along with B.3, it might be something that with later versions and more complex methods could be fixed.
Both D.3 and D.4 question whether more information should be placed in the optimum plan page as well as in the information section.

Questions E.1 and E.2 make sure the stakeholders are happy with the structure and clarity of the help and information sections, since any users should be able to easily access the provided information.
E.2 and E.5 ensure there is nothing major missing in the two sections, and question E.3 makes sure the layout of the metric descriptions is clear.

Questions F.1-3 aim to see if the stakeholders are overall happy with the style choices. The colour of the maps and the menu order is more subjective, but I want to ensure style is at least consistent
With question F.4 I want to make sure the users are happy with the metrics provided, one of the key parts of the software.
My stakeholders agreed to keep metrics with missing data as long as there was a warning, so question F.5 asks whether they are happy with the warning provided.
I was not able to add the select/unselect all feature due to time constraints, so I want to ask if the stakeholders are too bothered by the lack of this feature with F.6.
Questions F.7 and F.8 ensure the users are happy with the way the colour-key and hover data works.

All of the questions in part G ask whether the users are happy with the overall state of the software, and feel that their requirements have been met.

Responses

Due to the length of the questions, I've decided to have the responses all under the same question, to reduce the number of repeated questions.
A lot of the responses where simple 'yes'/'no' which makes sense considering how specific the questions are. I got the general thoughts of my stakeholders while asking the questions, and extra comments were added effectively.

Part A – Current Maps

1. **Is the user interface clear, and do you know how to use everything that is on display?**
   Fermin: Yes.
   Lourdes: Yes.
   Jenny: Yes.
   - **Is there any feature that you do not see how to access?**
     Fermin: No.
     Lourdes: No.
     Jenny: No.
2. **Is the display of the raw data as its own map acceptable?**
   Fermin: Yes, I actually find the separate map quite helpful.
   Lourdes: Yes, I did not mind either way.
   Jenny: It's okay, but I would rather have them in the same instance, with the raw data in the hover.
3. **Do you mind that there is no way to access another visual representation of the raw data, as had been requested?**
   Fermin: No.
   Lourdes: Yes, I would rather have the access to other representations, but it is not a key requirement.
   Jenny: With the current software, I actually wouldn't expect to be able to access these, so the lack of them is no problem at all.

Part B – Projected Maps

1. **Are the dates provided acceptable?**
   Fermin: Yes.
   Lourdes: Yes.
   Jenny: Yes.
2. **Is there any problem with the fact that there is not a large change in the projected data (as data takes long to change)?**
   Fermin: No.
   Lourdes: No, it is simply the way the data is projected, no adjustments should be made to erroneously exaggerate it.
   Jenny: No.
3. **Would you rather a more complex economical algorithm was used to project the data, rather than a simple regression?**
   Fermin: No.
   Lourdes: No.

Jenny: No.

- **If this was available, would you want the option of multiple projection algorithms?**
  Fermin: Yes, but it is not a high priority, more of a want than a need.
  Lourdes: Yes, it'd be nice to compare.
  Jenny: Yes.

4. **The projected data does not re-index the data based on the projections of all countries, but simply projects the indexes of the G7. Is this alright (*there is unlikely to be any major changes)*?**
   Fermin: Yes, it's fine.
   Lourdes: Yes.
   Jenny: Yes.

Part C – Economic Simulator

1. **Do you mind that policies can only be changed by +-5%?**
   Fermin: No.
   Lourdes: It would have been nice to have more options, but it is not necessary.
   Jenny: Yes, it would be more fun if you can customise the amounts, but there might not be major changes so could be unnecessary.
2. **Are the available policies acceptable?**
   - **Are there any major policies missing?**
   - **Are any of the policies unnecessary?**

   Fermin: It seem like a pretty comprehensive list.
   Lourdes: The available policies are good, but I'd rather their display names did not have underscores.
   Jenny: They're good.
3. **Are the available dates acceptable?**
   Fermin: Yes.
   Lourdes: Yes.
   Jenny: Yes.
4. **Is there any problem with the initial values of the checklist being set to both +5% and -5% on income tax (i.e.: no change)?**
   Fermin: No.
   Lourdes: No, I don't mind at all.
   Jenny: No.
5. **Is there any problem with the fact that there is at times not a large change in the data (as policies might only apply later on)?**
   Fermin: No, it makes sense that, as you explained, some policies simply take time to take effect.
   Lourdes: No.
   Jenny: It's not very satisfying, but it is true to the real world, so it is realistic.
   - **Could this be made clearer?**
     Fermin: You could put a disclaimer about the policies.
     Lourdes: No need.
     Jenny: You can add a sentence explaining this.

Part D – Optimum Plan

1. **Is the layout clear?**
   Fermin: Yes.
   Lourdes: Yes.
   Jenny: Yes.
2. **Do you mind that there is little change between the different plans for the G7?**
   Fermin: No.
   Lourdes: No.
   Jenny: No.
3. **Would you like an explanation of the plan included within the page (otherwise in the information section)?**
   Fermin: Yes.
   Lourdes: Yes.
   Jenny: Yes, I think it would make it more accessible.
4. **Would you like an explanation of the algorithm to also be included within the page (otherwise in the information section)?**
   Fermin: Yes.
   Lourdes: No. While an explanation of the plan itself is useful for the economical application, there is no need to have the algorithm explained within the page.
   Jenny: I think it's fine to simply have it in the information section, but maybe add a sentence in the page saying that you can 'find out more' there.

Part E – Help and Information

1. **Is the information page well structured?**
   Fermin: Yes.
   Lourdes: Yes.
   Jenny: Yes.
2. **Is there any major missing information section?**
   Fermin: No.
   Lourdes: No.
   Jenny: No.
3. **Are the metric descriptions sufficient?**
   - **Would you like references for them?**

   Fermin: Yes, and references would be good too.
   Lourdes: Yes, and yes to the references too.
   Jenny: Yes. I don't mind having/not having references.
4. **Is the style of the help section appropriate and clear?**
   Fermin: Yes, the way it is laid out is easy to understand and apply.
   Lourdes: Yes, it's actually really helpful.
   Jenny: Yes, I like the help section.
5. **Does the help section have anything missing?**
   Fermin: No.
   Lourdes: No.
   Jenny: No.

Part F – General/Other questions

1. **Are the colours of the two maps visually appealing?**
    - **Do they fit the 'social' and 'environmental' themes well?**

   Fermin: The differences in the intensity are not easy to appreciate at times. The map colours fit the themes well.
   Lourdes: I think they are okay, I don't like the colours too much, but I don't know what a better option would be.
   Jenny: They fit the theme well, and are okay visually.
2. **Do you mind that the menu is in alphabetical order (rather than ordered by section)?**
   Fermin: No.
   Lourdes: Yes, I'd rather they were ordered by section.
   Jenny: It would make more sense ordered by section.
3. **Is the style appropriate and consistent?**
   Fermin: Yes.
   Lourdes: Yes.
   Jenny: Yes.
4. **Are there enough metrics?**
    - **Are there any major metrics missing?**
    - **Are any of the metrics unnecessary/inappropriate?**

   Fermin: Yes, the list of metrics is very comprehensive. If anything, I would consider simplifying it and having a shorter list.
   Lourdes: Yes, the metric list is good.
   Jenny: Yes.
    - **Is there a problem with the number of environmental metrics compared to the number of social metrics (*I attempted to make it somewhat even, but there were a lot more social than environmental within the databases used)?***
      Fermin: No.
      Lourdes: No, it makes sense considering the environmental metrics area is yet to develop.
      Jenny: If there is simply not enough environmental metrics, then no.
5. **Is the warning for missing data clear?**
    - **Could it be adjusted in some way?**

   Fermin: Yes.
   Lourdes: Yes.
   Jenny: You could make it red, but it's good.
6. **Do you mind that there is no select/unselect all?**
   Fermin: I would prefer to have it, but it is not a key requirement.
   Lourdes: It would have been nice, but it is not a main concern.
   Jenny: Yes, it would be more efficient having it.
7. **Is the key clear, even when it changes to fit the data selected?**
   Fermin: Yes, I find it makes more sense for the key to adjust, rather than not.
   Lourdes: Yes.
   Jenny: Yes.
8. **Is the information provided in the hover text enough?**
   Fermin: Yes.
   Lourdes: Yes.
   Jenny: Yes.

Part G – Overall program

1. **Are you overall happy with the whole program?**
   Fermin: Yes.
   Lourdes: Yes.
   Jenny: Yes.
2. **Have all your key requirements been met?**
   Fermin: Yes.
   Lourdes: Yes.
   Jenny: Yes.
3. **Do you believe the program solves the original problem (accessibility to economic data)?**
   Fermin: Yes.
   Lourdes: Yes.
   Jenny: Yes.
4. **Is there anything missing?**
   Fermin: No.
   Lourdes: No.
   Jenny: No.
5. **Any other comments:**
   Fermin: I would prefer the map to be in a different format in which only the countries that actually contain data appear, so that it is easier to see the smaller countries in Europe. By eliminating the countries in grey, the user interface could be cleaner and less cluttered.
   Lourdes: I find the term 'indexed' and 'raw' not very intuitive for the names of data. I think the names 'normalised' and 'absolute' values would explain what they are in a more effective way.
   Jenny: No.

Analysis

With the current maps, my stakeholders were happy with the usability and did not mind the difference in the display of the raw data. They were quite divided when it came to the access to other visual representations, some of them not needing or wanting it, while one still wanted it. It should be considered in later iterations of the software, but doesn't seem to be a main concern.

They were completely happy with the projected maps, and did not feel the need for more complex projection algorithms (although, if available, would like to see different projections).

For the economic simulator, they were once again happy with everything provided as well as the usability and looks. The only thing that could be improved is being allowed to change policies by more than +-5%.

With the optimum plan, further explanations could be included in the page, but the layout and actual process is clear.

The information and help sections were the most liked by my stakeholders, which is great for my goal of the software being accessible to as many users as possible.

Overall, the stakeholders were happy with the program as a whole and saw all their key requirements met.

Here are a few more comments from the stakeholders of possible adjustments:

- Map colours and intensities could be improved.
- The menu could be ordered by section.
- A select/unselect all would be beneficial.
- The map could be adjusted to highlight only the G7.
- The wording could be improved to it more intuitive ('normalised'/'absolute' instead of 'indexed'/'raw').
- Some metric names could be shortened/simplified.

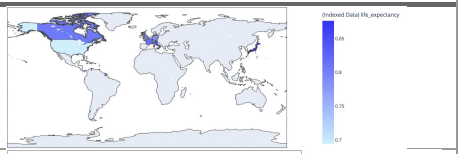
Testing to inform final evaluation

In this section I will run multiple tests, mainly to inform the success criteria, based on testing the functionality and robustness of the program. Some of these tests will be repeated in the success criteria section as a way to evidence their completion.

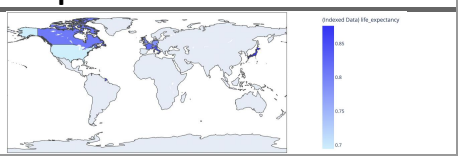
Function

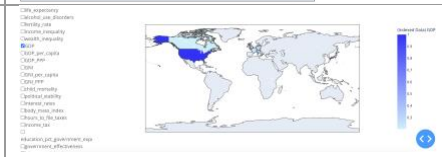
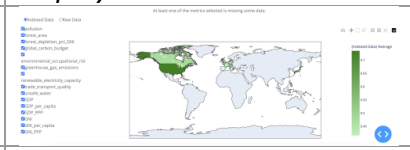
The following will run tests based on the functionality requirements set in the Analysis section. *Reasoning for these requirements may be viewed in the Analysis section.*

Ability to view both colour-coded environmental and social maps

Input	Expected result	Output
Select 'Social' in the map selection	Social map is displayed in blue colour	
Select 'Environmental' in the map selection	Environmental map is displayed in green colour	
Select 'Economic' in the map selection	"Invalid selection. Only Social and Environmental maps are available"	Users do not have availability to select maps other than Social or Environmental (drop-down menu)

Ability to choose what metrics to include in the maps

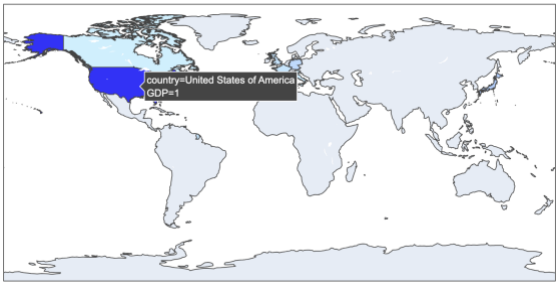
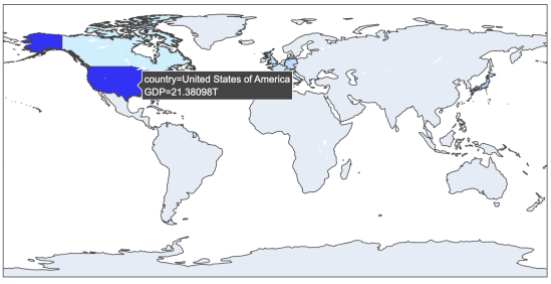
Input	Expected result	Output
Select 'Social' in the map selection	Social metrics are displayed	

Select 'Environmental' in the map selection	Environmental metrics are displayed	
Select 'GDP' in the Social map	Map changes accordingly, displaying the correct data	 *GDP is selected and corect data is displayed.*
Select all metrics in the Environmental map	Map changes accordingly, displaying the correct data	
Select no metrics in the Social map	"At least one metric must be selected to display data."	 *Error is displayed.*
Select 'Happiness Index' in the metric selection	"Invalid selection. This metric is not available."	Users do not have availability to select metrics that are not available (checklist)

A way to reset the maps

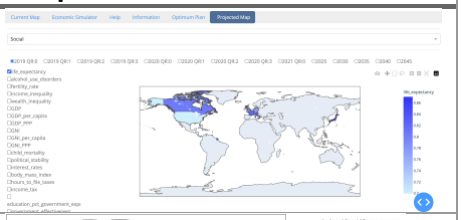
This function was not achieved, but the app can simply be reloaded to reset the maps.

A way to click on area/'zoom in' for more information

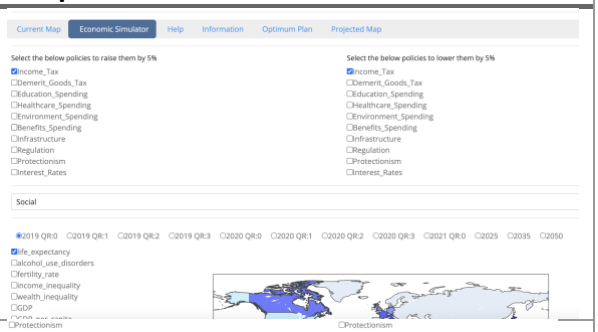
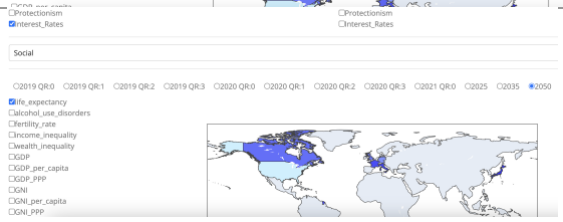
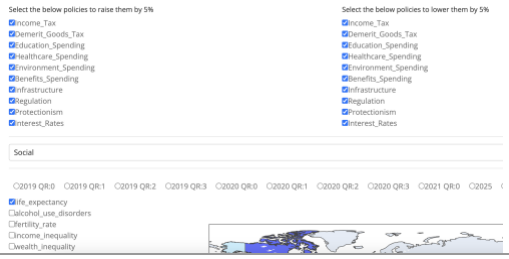
Input	Expected result	Output
Hover over the US	Exact value/data is accessed	
Select 'Raw Data'	Correct raw data is displayed	
Access table / bar chart / etc.	Table / bar chart / etc. is displayed with the correct data	*This function was not achieved, for reasons explained later on in the documentation.*

Hover over China	No effect	There is no data displayed.

A way to access and maneuverer the projection maps

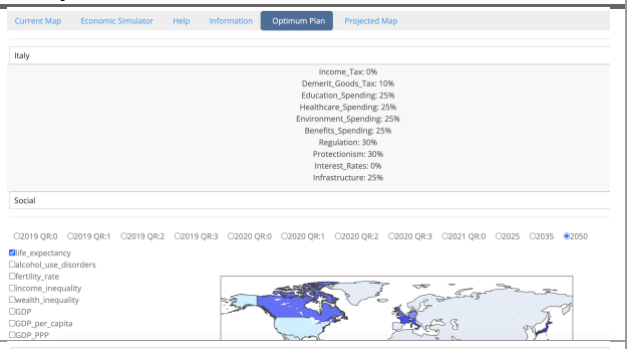
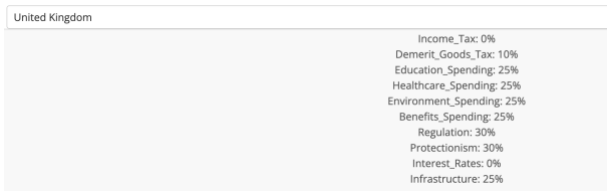
Input	Expected result	Output
Select 'Projected Map' in Menu	Projected map is displayed	
Select 'Social' in the map selection	Social map is displayed in blue colour	
Select 'Environmental' in the map selection	Environmental map is displayed in green colour	
Select 'Economic' in the map selection	"Invalid selection. Only Social and Environmental maps are available"	Users do not have availability to select maps other than Social or Environmental (drop-down menu)

A way to select your policies in the economic simulator

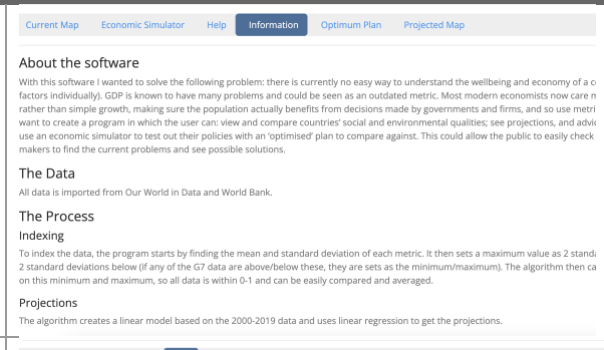
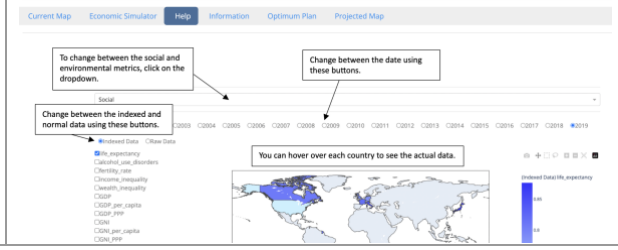
Input	Expected result	Output
Select 'Economic Simulator' in Menu	Economic Simulator is displayed	
Select 'Raise Interest Rates by 5%' in the Social map	Map changes accordingly, displaying the correct data	 *Interest rates are selcted, and correct data is displayed.*
Select all policies	Map changes accordingly, displaying the correct data	

Select no metrics	The map simply displays the projected data, as no changes have been selected.	
Select 'Quantitative Easing' in the policy selection	"Invalid selection. This policy is not available."	Users do not have availability to select policies that are not available (checklist)

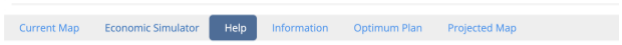
A way to view the 'optimised' policy plan

Input	Expected result	Output
Select 'Optimum Plan' in Menu	Optimum plan is displayed	
Select 'United Kingdom' in the country selection	Policies for the UK are displayed	
Select 'China' in the country selection	"Invalid selection. This country is not available."	Users do not have availability to select countries that are not available (drop-down menu)

A way to access tips and instructions of the economic simulator (and other sections)

Input	Expected result	Output
Select 'Information' in Menu	Information is displayed	
Select 'Help' in Menu	Help is displayed	

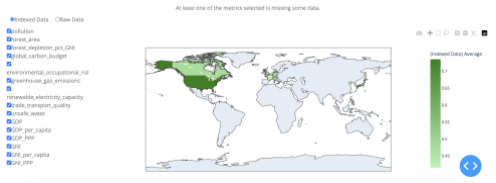
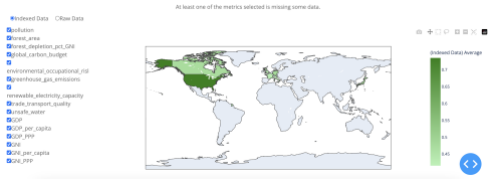
A menu to access all the features

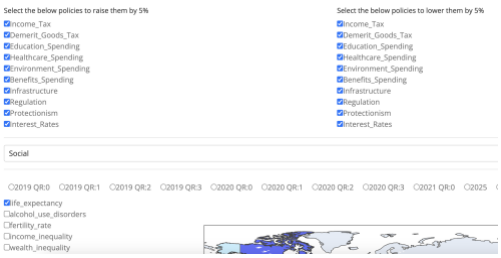
Input	Expected result	Output
Start program	Menu with all options is displayed	

Robustness

There are few ways in which the program can be pushed to its limit, these mainly being selecting all/none of […]. The following details how the program reacts to these extremes in each section of the program.

Some of these use the social metrics and some the environmental – this is done to showcase the testing more clearly, but the tests apply to both choices.

	Select all…	Select no…
Current	… metrics  The program displays the overall average without issues.	… metrics  The program displays the message 'At least one metric must be selected' and leaved the map empty, which is appropriate.
Projected	… metrics  The program displays the overall average without issues.	… metrics  The program displays the message 'At least one metric must be selected' and leaved the map empty, which is appropriate.

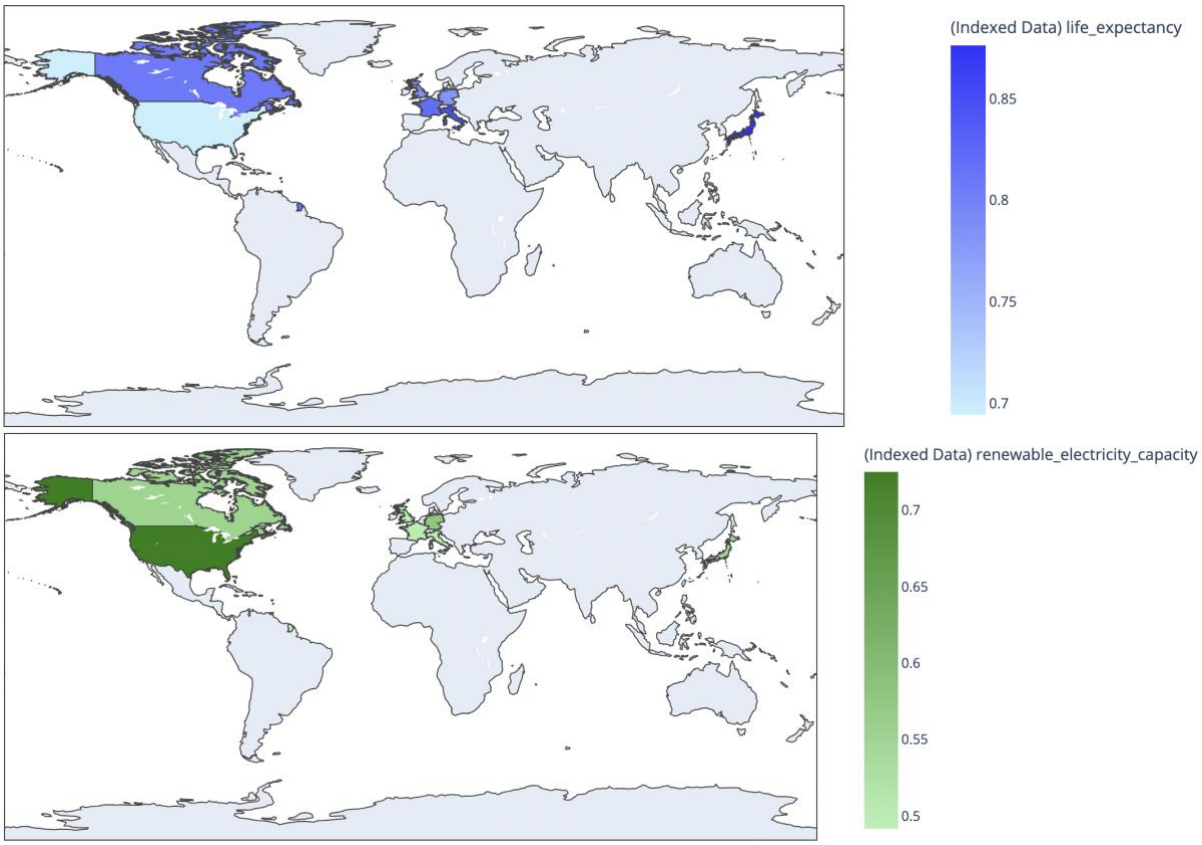
Economic Simulator	… policies  The program allows all policies to be selected without problem, but they directly offset each other, so the data shown will simply be the projected data. … metrics The economic simulator works in the same way as Current and Projected maps, there is no error.	… policies  The program has no problem with not selecting any policies, it simply displays the projected data. … metrics The economic simulator works in the same way as Current and Projected maps, there is no error.

Success Criteria

The following will show the **criteria** and *aim(/planned evidence)* of the planned success criteria, and the relevant evidence.

Program shows simple maps

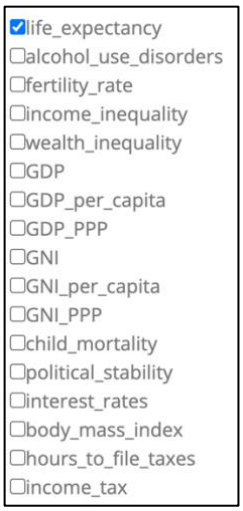
Screenshot of simple and uncluttered colour-coded maps of environmental and social data.



These two social and environmental maps are uncluttered and colour-coded correctly. The visual appeal and accessibility are confirmed by the stakeholders (*see final stakeholder feedback*).

A way to select different metrics for the maps

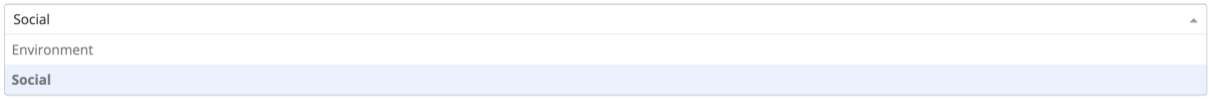
Screenshot of selection menu in the program as well as testing showing that it works.



Part of the metric selection, simply click the checkboxes to select.
Testing has been carried out across the development to test it works.

A way to view both environmental and social maps

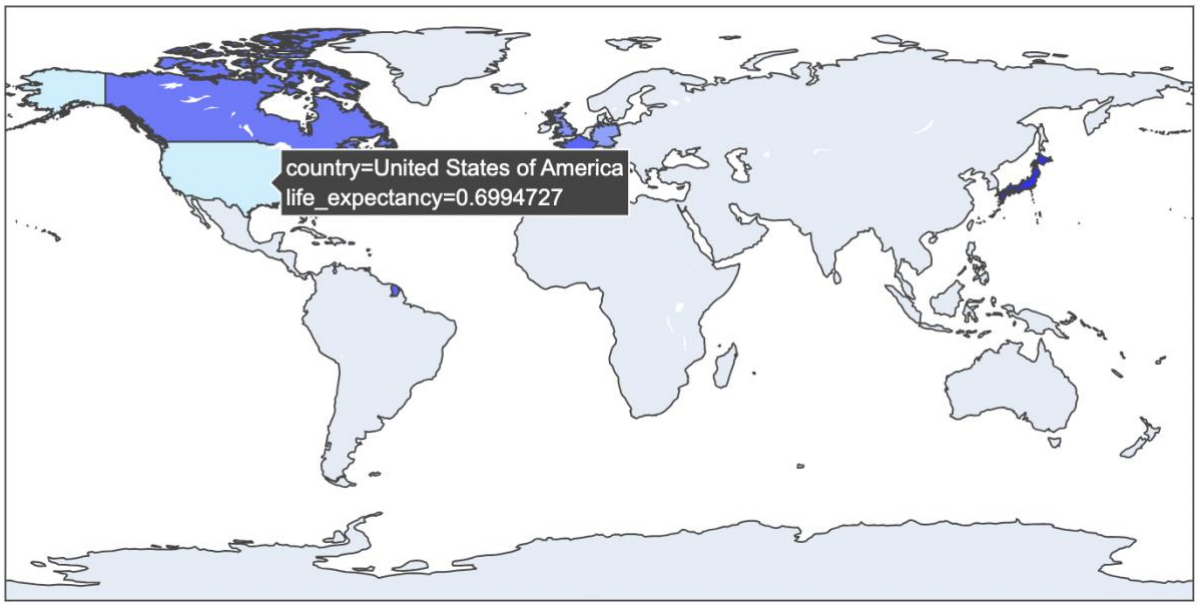
Screenshot of both maps in the program (and/or testing showing that it works if necessary).



Dropdown menu allows access to both the social and environmental map (in different pages, not side-to-side. Selecting on either brings you to the correct maps, confirmed by their colour.

A way to find more information of an area

Screenshot of the data being displayed as well as testing showing that it works.

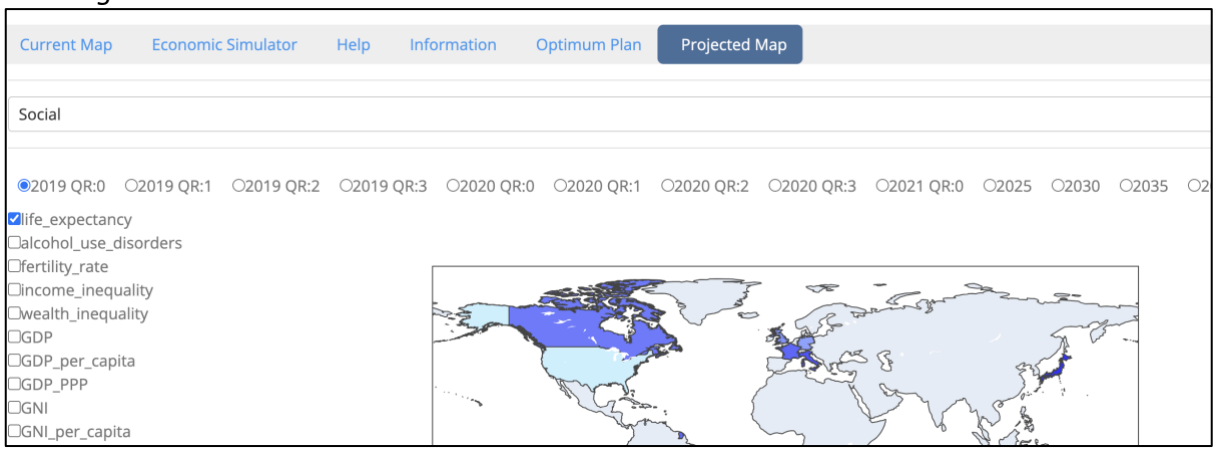


Users can hover over the country of their choice to see more information about the area regarding the chosen metrics and map.

A way to easily access the projections & A way to easily maneuverer projections

Screenshot of how the projections can be accessed, which should be simple and easy to find, and a test showing that it works.

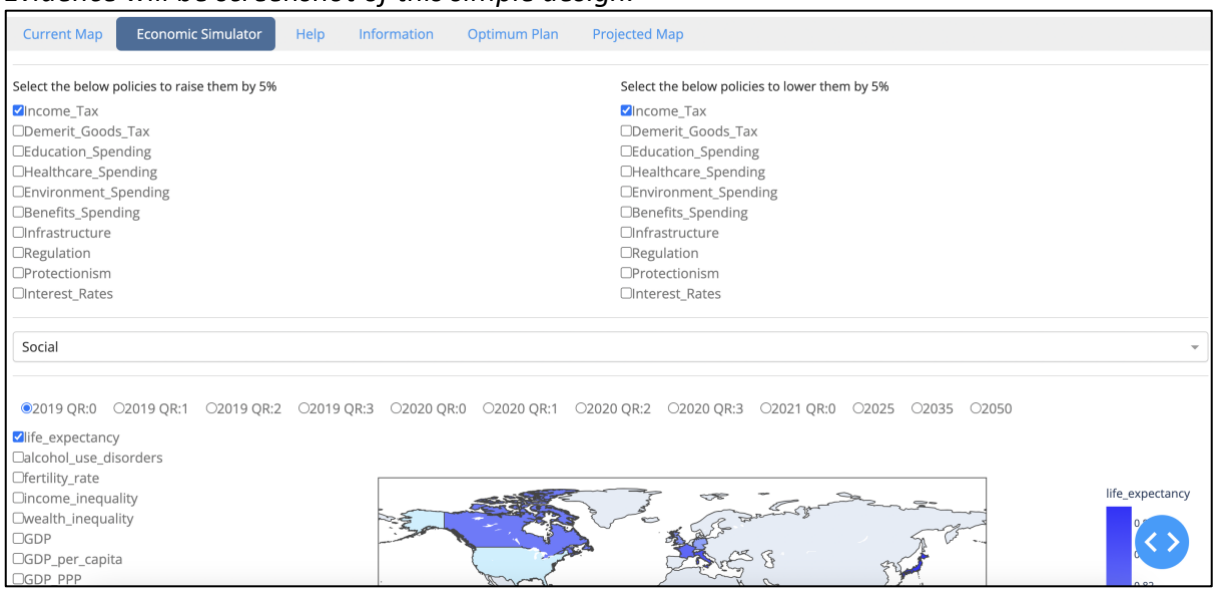
Screenshot of how the projections look, which should be simple and uncluttered, and a test showing that it works.



The projections can be accessed via the menu easily, and are easy to maneuverer and uncluttered, as confirmed by the stakeholders (*see final stakeholder feedback*).

Simple design for economic simulator and 'optimised' plan

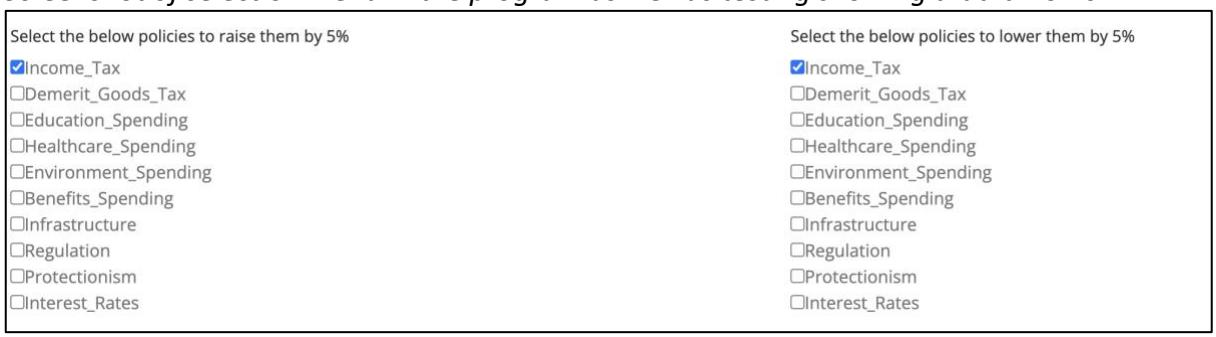
Evidence will be screenshot of this simple design.



The economic simulator and optimum plan hold the same accessible design, as shown above and confirmed by the stakeholders (*see final stakeholder feedback*).

A way to select policies for the economic simulator

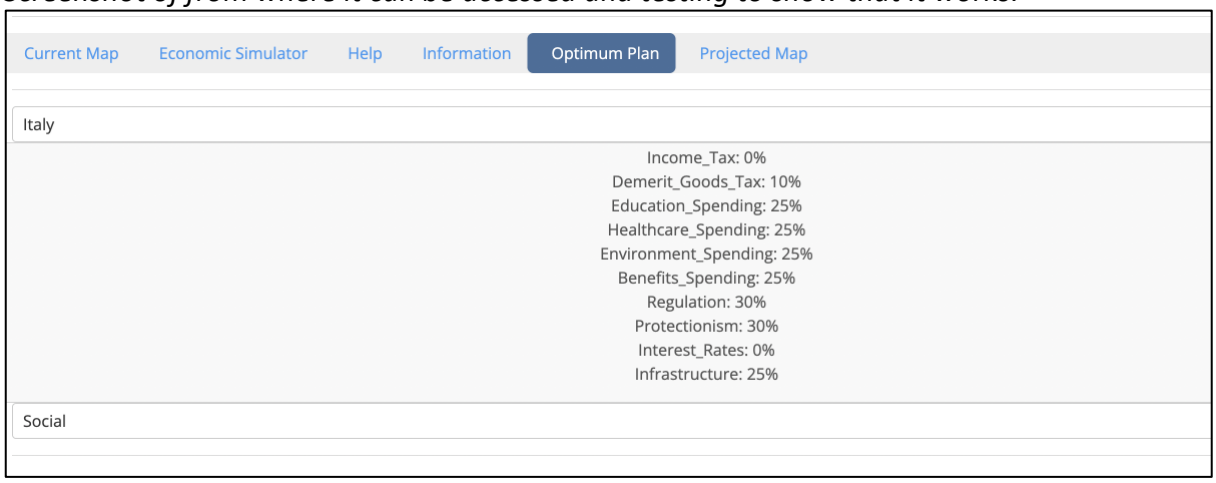
Screenshot of selection menu in the program as well as testing showing that it works.



The policies can be simply selected by clicking on their checkboxes.

A way to access the 'optimised' economic plan

Screenshot of from where it can be accessed and testing to show that it works.



The optimum plan can be accessed via the menu easily.

A somewhat accurate 'optimised' plan

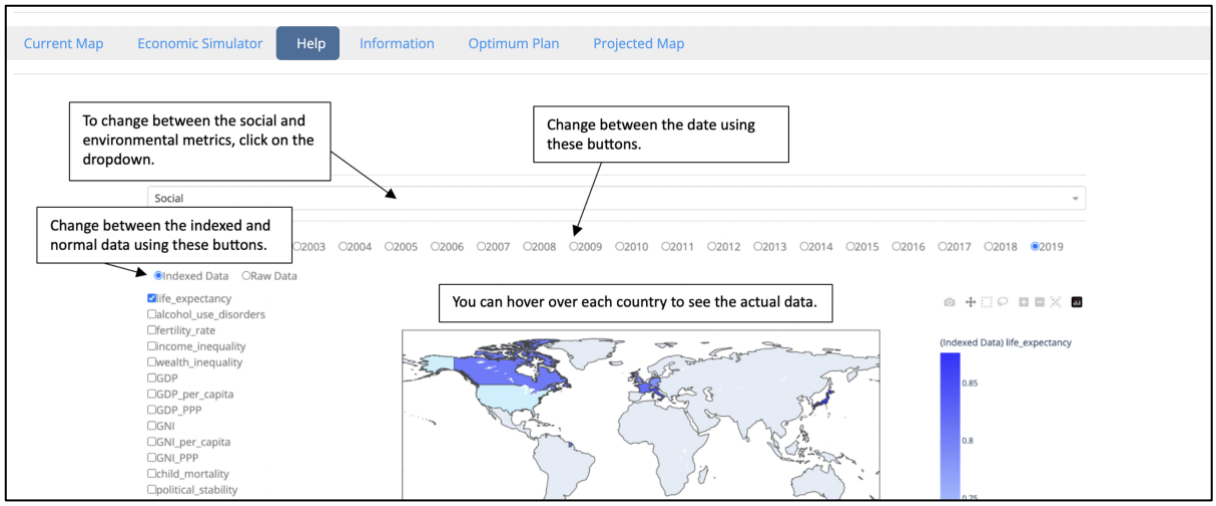
Testing to show how it works and an economist opinion on its accuracy.

Both of my stakeholders with economic knowledge agreed that the plans proposed were reasonable (*not in the stakeholder feedback, this was a conversation held along with it*).

Easily accessible instructions and tips on how to use the maps, projections, and economic simulator

Screenshot of where the information is placed in the program, which should be in an easily accessible place. Instructions should also be as clear as possible (may use a tester's opinion as evidence), and tips should be accurate and informative (may use an economist's opinion for evidence).

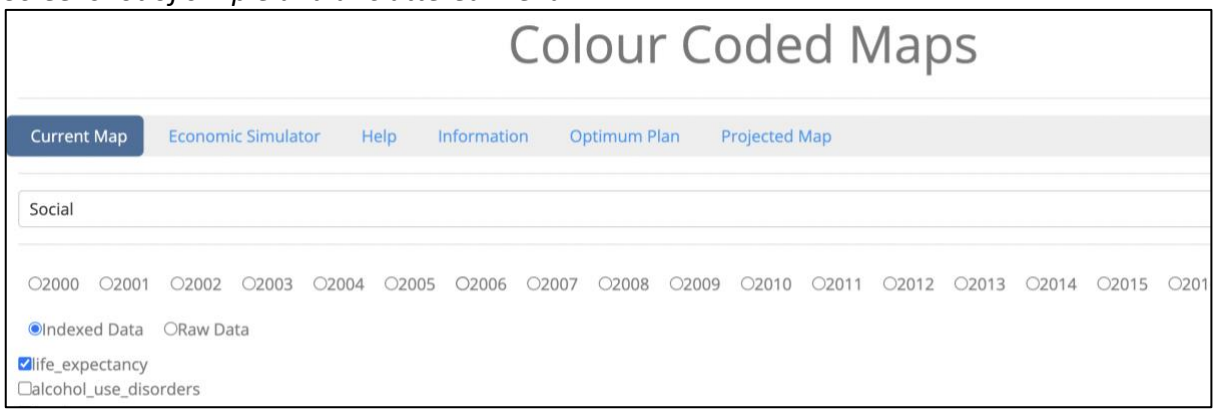
Menu displays an easily available section 'help' where instructions for the program can be accessed.



Accurate and clear instructions are available, confirmed by the stakeholders (*see final stakeholder feedback*).

A menu to access all features

Screenshot of simple and uncluttered menu.



All features can be accessed via the menu, seen in all pages, as is defined in app.py.

Consistent aesthetics throughout

Screenshot of all different features.

The consistent aesthetics can be seen across the previous evidence and were confirmed by the stakeholders (*see final stakeholder feedback*). Moreover, the style of the software is defined in app.py and carried across all the pages, ensuring consistency.

All of the features were thoroughly tested throughout development, and once more once the whole program was complete. This helped me ensure all of the key parts of the program were working and allowed me to detect errors that had arisen after the previous tests (mainly Our World in Data adding more metrics to their database and forcing me to make the metric requests more specific).

I also destructively tested every feature in the program to make sure it would function even if the user brought it to a breaking point. I tested selecting all/no metrics in all maps, as well as all the possible dates and policies, as the selection of data is the main feature that could fail. Everything worked as it should.

Success Criteria - Functionality

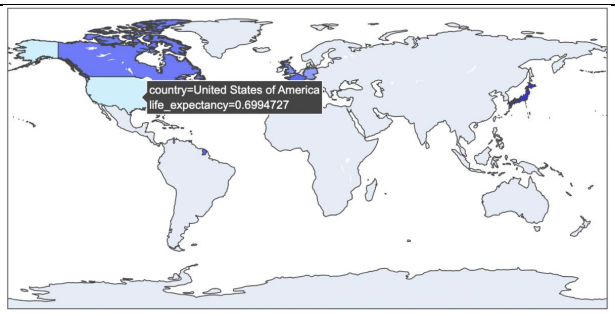
To see the explanation and justification of each of these, refer back to the analysis section.
(*In brackets how it would roughly look and work in the program*)

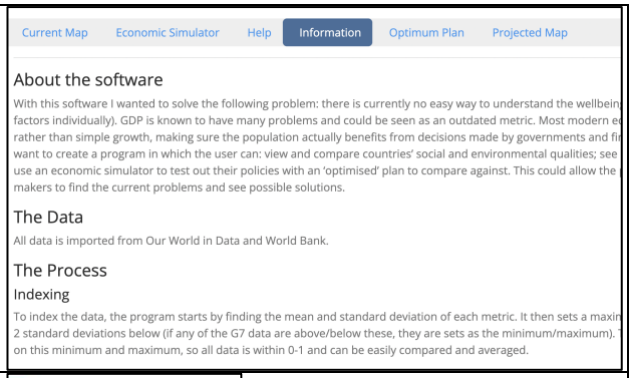
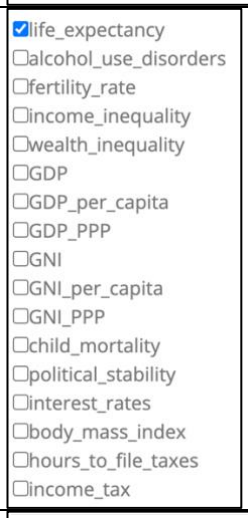
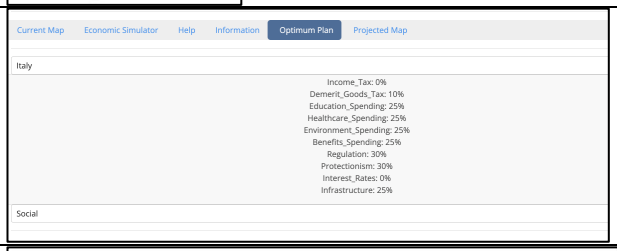
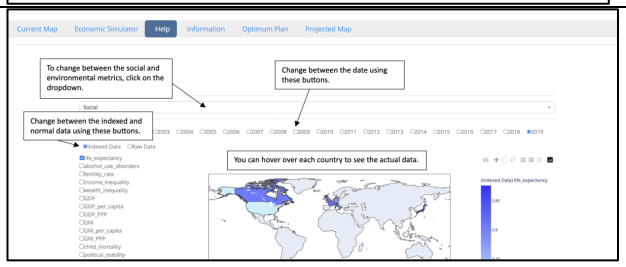
Requirements	Completed?
Ability to view both colour-coded environmental and social maps (either both visible in the same screen or a button to change from one to the other)	Fully
Ability to choose what metrics to include in the maps (in a tick-list)	Fully
A way to reset the maps (button to click on the screen)	Partially*
A way to click on area/'zoom in' for more information (once clicked, the screen will display the raw data of the metric in table/bar chart/line graph form for the user to choose between)	Partially**
A way to access and maneuverer the projection maps (displayed in the same colour-coded map manner)	Fully
A way to select your policies in the economic simulator (by selecting from a drop-down menu or tick-list the policies they want – e.g.: 'raise direct taxes by 2%' – and displaying the data in the same map form)	Fully
A way to view the 'optimised' policy plan (by clicking a button on screen to display the 'optimised' set of policies and the result of them in the same map form)	Fully
A way to access tips and instructions of the economic simulator (a button opens another screen with tips and instructions)	Fully
A menu to access all the features (button to open menu with: current data maps, projections, simulator and help)	Fully

*The maps can be simply reset by reloading the webpage.
**The raw data can still be accessed as shown in prototype 3, but not in a bar chart/line graph form, as that would have required a lot more coding to create different graphical displays to be shown. *This is elaborated on later in the documentation.*

Usability

The evidence section shows the tests run to prove the requirements have been met.
Again, to see the explanation and justification of each of these, refer back to the analysis section.

Requirements	Evidence	Completed?
Simple maps, line graphs and bar charts	 *Only maps, the other representations were not achieved.*	Fully

Obvious way to find more information (button on screen at all times and 'find out more' section in menu to access a screen with more information)		Fully
Obvious way to alter maps (tick-list menu with different metrics always on screen)		Fully
Obvious way to access projections and economic simulator (through menu to be accessed with a button)		Fully
Clear instructions on how the software can be used (button on screen at all times and 'help' section in menu to access a screen with more information)		Fully

Unachieved features

Some of the proposed features for the software were not achieved, mainly due to timing. Here I will explain in a bit more detail my reasoning behind deciding to not work on these features.

Select/unselect all

The aim of this feature is to facilitate the selection of a lot of metrics, as the number of metrics provided is quite large. It was a feature that my stakeholders had requested, and that was seen in some of the solutions I had analysed, so, even though not the most important, I wanted to include it.

However, towards the end of development (when I had planned to add the feature), I realised it required a much more complicated process than planned. A button would provide a simple solution (where pressing once selected all, and pressing again to unselect all, and so on), but would be unclear to the user, and I wanted to avoid unnecessary confusion. The other option was a checklist with complicated callbacks, and I did not want to make the already complex callback section in my code worse. Overall, I did not see that the time and effort required to add this feature would be worth the somewhat easier handling of the metric selection, and decided to not include it in the final program.

Multiple visual representations of data

The problem with this feature was once again time and complexity. One of my stakeholders requested the visualisation of the raw data in multiple formats, which I agreed to add if it was possible within the time frame. In the end, it would have required my understand of a lot more visual representations with Plotly, as well as making the callback more complex, and I did not have the time for it. It seems, however, that my other two stakeholders did not need or want it, so it does not seem to be a key concern.

Incomplete information and help section

The reason for this was time. Although I wanted to finish writing and creating all of the tutorials and metric descriptions that would make the program as easy to use as possible, I did not have the time for it. I found that completing the actual algorithms and programming of the software would be a lot more important, and that since I could simply help my stakeholders if they needed it, the help and information sections could be completed in a later version, for when any user would be able to access it.

Overall, these were not the key features of the program (and were not even in my success criteria), and the stakeholders were not too unhappy by the lack of them. They can simply be added in later versions of the program.

Limitations

There are not any major limitations to the program, but I will discuss the ones that have arisen from the design of the solution as well as the development, and some possible fixes for them.

Countries available

As planned, only the G7 economies are available to the user, but this could be expanded upon quite easily in future versions of the program, as the data for them is already being imported.

Metrics and dates available

The metrics and dates offered in the maps is largely limited by the data used. Future versions could import from more sources to reduce this limitation.

Policies available for economic simulator

The policies provided are quite limited by the amount and simplicity of them. In reality governments can attempt to do much more and in a range of other ways, which the user will not be able to experience. Perhaps future versions could expand the range of policies.

The 'Rules' table

The largest limitation to both the economic simulator and optimum plan is the 'Rules' table. In economics and in real life, the results of policies depends on a lot more than what is considered by the table. The starting point of the economies (e.g.: lowering interest rates could be problematic if inflation is already quite high), the quality and reactions of/to government intervention (e.g.: greater healthcare expenditure might not be effective), the stage of development of the economy (the G7 will not be affected in the same way as poor countries), etc. The table considers a basic starting point which is somewhat effective for the similar G7 economies, but would likely fail in other economies. In future versions it would be great to attempt a more complex algorithm that adapts the Rules table according to the economy it is considering.

Information held in the hover templates

While developing, I found that the limits of the information that could be held in the hover templates were very strict. I had attempted to add multiple lines of information when averaging data (so each value could be seen as well as the average), but found it impossible. In future versions it would be great to find a way to display this data in some other way, perhaps under the maps.

Incomplete help and information

As discussed before, the help and information sections were not able to be completed. In future versions I will complete them and add to them as new features are added. If possible, the most efficient way to keep the information section updated would be to automate the definitions of the metrics, as they are the most time-consuming (perhaps with data mining or an artificial intelligence such as ChatGPT).

Maintenance

The future maintenance of the program should be done easily, as the program is well documented, commented throughout, and stored in different files according to section. The software is heavily modular (both with functions and classes), so any adjustments the stakeholders wish to add would be added without difficulty.

New versions of the software could add the features suggested to solve the limitations above and to add the unachieved features.

The select/unselect all feature could be added to the pages without problem, as I ensured the callback section (the more complex part of this feature) was well documented, commented and as uncluttered as possible to ensure easy understanding. Developers could refer to this document or previous code to easily understand the callback process and what would be required.

For the multiple visual representations of the data, Plotly would be able to achieve all of these, and it is a very well documented library. Furthermore, my use of the Plotly library to create the maps will be very similar to the process of these new representations, so the process could be easily replicated by other developers.

The range of countries included could easily be adjusted for all metrics, as both types of inputs have a function where the countries are selected in some way (*selectCountries()* for Our World in Data and *formatWB()* for World Bank). This is also well documented for other parts of the code that set the countries, so other developers would be able to refer to this document if needed.

Expanding the range of policies or adjusting the 'Rules' table could be the hardest to maintain, as it has been done manually and requires knowledge of basic economic principles. However, there are resources online that could aid with this, and there is the possibility of automating this process in the future with some sort of algorithm or artificial intelligence.

Displaying more information in the page on something other than the hover can be easily done in any of the pages, as the structure of the page can be easily read in the code (thanks to the naming of variables, indenting and comments), and other developers will be able to add new components easily. Moreover, I ensured the callbacks where not too cluttered to ensure readability, and so could be added to without problem.

As with the 'Rules' table, completing the Help and Information sections has been currently done manually, but this probably has a higher chance of easily being automated (as all the information comes from the internet anyways).

The code has been annotated for maintenance (*e.g.: the purpose of selectCountries() is annotated above the function, which can be seen in the first page of the Final Code*), so anyone who wishes to adapt the code can easily understand what is going on at each stage and how the code is linked together, allowing easier maintenance.

Final Code

‘allImports.py’

```
from owid import catalog
import wbgapi as wb
import pandas as pd
import ssl
ssl._create_default_https_context = ssl._create_unverified_context


# the following further reduces the data (so only the G7 and the years I need are used) and places all of this on a new dataframe df1
# it does so by adding the selected columns (G7 countries) to an originally empty table, then returning said table
def selectCountries(df_in):
    countries = ["United Kingdom", "United States", "France", "Japan", "Germany", "Canada"]
    df_out = df_in.loc[("Italy",2000):("Italy",2023)]
    for country in countries:
        temp = df_in.loc[(country,2000):(country,2023)]
        df_out = pd.concat([df_out, temp])
    return df_out

# used to format the WB dataframes to the way the WB dataframes are
def formatWB(df_in):
    countries = ["Italy", "United Kingdom", "United States", "France", "Japan", "Germany", "Canada"]
    tuples = []
    for country in countries:
        for year in range(2000,2020):
            tuples.append((country,year))

    # creates multilevel index for each country with the years 2000-2022
    index = pd.MultiIndex.from_tuples(tuples, names=["country", "year"])

    # the following selects the data according to the country (ITA, GBR..) and the year (YR2000, YR2001..) so the data can be added
                to the correct index (Italy, UK.. and 2000, 2001)
    dataList = []
    for country in ['ITA','GBR', 'USA','FRA', 'JPN', 'DEU', 'CAN']:
        for year in range(2000,2020):
            year = 'YR'+str(year)
            dataList.append(df_in.loc[country, year])

    df_out = pd.Series(dataList, index=index)    # adds the data with the correct index
    return df_out

# the following indexes the OWID data by finding a minimum (mean -2stds) and a maximum (mean +2stds), then comparing the data values to these to find an index value between 0
and 1

def indexOWID(dfFull, dfCut):
    countries = ["Italy", "United Kingdom", "United States", "France", "Japan", "Germany", "Canada"]
    dfFull = dfFull.swaplevel()
    for year in range(2000,2020):
        dfInfo = dfFull.loc[year].describe()
        mean = dfInfo.loc['mean']    # finds mean for that year
        stdDev = dfInfo.loc['std']   # finds standard deviation for that year

        minval = mean - 2*stdDev                  # creates a minimum -2 stds away from mean
        maxval = mean + 2*stdDev                   # creates a maximum +2 stds away from mean
```

```
        dfCut = dfCut.swaplevel()                   # levels are swapped so the years can be searched for
        if minval > min(dfCut.loc[year]):
            minval = min(dfCut.loc[year])
        if maxval < max(dfCut.loc[year]):
            maxval = max(dfCut.loc[year])           # ensures the G7 do not surpass the maximum
        dfCut = dfCut.swaplevel()
        length = maxval - minval

        dfCut = dfCut.sort_index()     ####### Solving error: PerformanceWarning: indexing past lexsort depth may impact performance.
        for country in countries:                               # the error is easily fixed by sorting the index before iterating through it
            dfCut.loc[country,year] = (dfCut.loc[country,year]-minval)/length
    return dfCut


# the algorithm works in the same way as for OWID, but is adjusted for the difference in index terminology with WB

def indexWB(dfFull, dfCut):
    countries = ["Italy", "United Kingdom", "United States", "France", "Japan", "Germany", "Canada"]
    dfInfo = dfFull.describe()                      # gets all the summary statistics of the dataframe
    for year in range(2000,2020):
        mean = dfInfo["YR"+str(year)].loc['mean']    # finds mean for that year
        stdDev = dfInfo["YR"+str(year)].loc['std']   # finds standard deviation for that year
        minval = mean - 2*stdDev                    # creates a minimum -2 stds away from mean
        maxval = mean + 2*stdDev                    # creates a maximum +2 stds away from mean

        dfCut = dfCut.swaplevel()
        if minval > min(dfCut.loc[year]):
            minval = min(dfCut.loc[year])
        if maxval < max(dfCut.loc[year]):
            maxval = max(dfCut.loc[year])           # ensures the G7 do not surpass the maximum
        dfCut = dfCut.swaplevel()
        length = maxval - minval
        for country in countries:
            dfCut.loc[country,year] = (dfCut.loc[country,year]-minval)/length
    return dfCut


# used to reduce repeated code when importing data from OWID
def finiliseOWID(df_in, metricName, tableName):
    df = df_in[metricName]
    dfAll = df.sort_index()
    df = selectCountries(dfAll)
    df = indexOWID(dfAll, df)
    df = df.rename(tableName)
    return df
# I will only apply this function to the metrics from prototype 2 and onwards, to show the growth of the code

# same as finilisedWB, used to reduce repeated code
def finiliseWB(code, reName):
    regions = ['WLD', 'HIC', 'OED', 'AFE', 'AFW', 'ARB', 'CEB', 'EAP', 'EAR', 'EAS', 'ECA', 'ECS', 'EMU', 'EUU', 'IBT', 'IDA', 'IDB', 'IDX', 'LAC', 'LIC', 'LMC', 'LMY', 'LTE', 'MEA', 'MIC',
'MNA', 'NAC', 'PRE', 'PST', 'SAS', 'SSA', 'SSF', 'TEA', 'TEC', 'TLA', 'TMN', 'TSA', 'TSS', 'UMC', 'IBD']
    dfAll = wb.data.DataFrame(code)
    dfAll = dfAll.drop(index=regions)
    df = wb.data.DataFrame(code,['ITA','GBR', 'USA','FRA', 'JPN', 'DEU', 'CAN'])
    df_out = formatWB(df)
    df_out = df_out.rename(reName)
```

```
    df_out = indexWB(dfAll, df_out)
    df_out = df_out.sort_index()
    return df_out
# I will only apply this function to the metrics from prototype 3 and onwards, to show the growth of the code

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# print(catalog.find('life'))
# the above line is used to find the different metrics in the catalogue
# I used this line while developing to find the metrics I was looking for

df = pd.DataFrame(catalog.find('life_expectancy', version="2022-11-30").load())
# df holds all the data from OWID regarding life expectancy, with the specific version to ensure the correct table is selected

df = df["life_expectancy_0"]
df = df.rename("life_expectancy")
# here I am narrowing down the data to only hold the specific metric I need

countries = ["United Kingdom", "United States", "France", "Japan", "Germany", "Canada"]
dfLife = selectCountries(df)
dfLife = indexOWID(df, dfLife)
# the above shows the basic steps used for selecting, narrowing and indexing all of the metrics, which will be shortened later on


df = pd.DataFrame(catalog.find('neoplasms__particulate_matter_pollution__both_sexes__age').load())
df = df["dalys_from_neoplasms_attributed_to_particulate_matter_pollution_per_100_000_people_in__both_sexes_aged_age_standardized"]
df = df.sort_index()

dfPol = selectCountries(df)
dfPol = indexOWID(df, dfPol)

dfPol = dfPol.rename("pollution")
# renaming the table so the column is easier to identify (and much shorter)



# data in OWID got updated, so there are now multiple 'inequality' tables, which means I need to specify further
#df = pd.DataFrame(catalog.find('inequality').load())  <-- this was the previous line used
df = pd.DataFrame(catalog.find('inequality', version="2023-01-27").load())
df.to_csv('inequality.csv')

dfIncomeInqAll = df["p0p100_gini_pretax"]
dfWealthInqAll = df["p0p100_gini_wealth_extrapolated"]

dfIncomeInq = selectCountries(dfIncomeInqAll)
dfIncomeInq = indexOWID(dfIncomeInqAll, dfIncomeInq)
dfIncomeInq = dfIncomeInq.rename("income_inequality")

dfWealthInq = selectCountries(dfWealthInqAll)
dfWealthInq = indexOWID(dfWealthInqAll, dfWealthInq)
dfWealthInq = dfWealthInq.rename("wealth_inequality")

df = pd.DataFrame(catalog.find('global_carbon_budget', version='2023-09-28').load())
dfCarbon = finiliseOWID(df, 'consumption_emissions_per_capita', "global_carbon_budget")

df = pd.DataFrame(catalog.find('greenhouse_gas_emissions_by_sector', dataset='emissions_by_sector').load())
```

```
df.to_csv('greenhouse.csv')
dfGreenhouse = finiliseOWID(df, 'total_ghg_emissions_including_lucf_per_capita', "greenhouse_gas_emissions")

df = pd.DataFrame(catalog.find('renewable_electricity_capacity', version='2023-06-26').load())
dfRenewable = finiliseOWID(df, 'total_renewable_energy', "renewable_electricity_capacity")

df = pd.DataFrame(catalog.find('child_mortality', version='2020-12-19').load())
df = df.set_index(['country', 'year'])
dfChildMor = finiliseOWID(df, 'probability_of_death_under_5', "child_mortality")




# ~~~~~~~~~~~~~~~~~~~~~ PROTOTYPE 3 METRICS ~~~~~~~~~~~~~~~~~~~~~

df = pd.DataFrame(catalog.find('alcohol_use_disorders__both_sexes__all_ages', dataset='gbd_mental_health').load())
dfAlcohol = finiliseOWID(df, 'current_number_of_cases_of_alcohol_use_disorders_per_100_000_people__in_both_sexes_aged_all_ages', 'alcohol_use_disorders')

df = pd.DataFrame(catalog.find('all_causes__bullying_victimization__both_sexes__all_ages').load())
dfBullying = finiliseOWID(df, 'dalys_from_all_causes_attributed_to_bullying_victimization_per_100_000_people_in__both_sexes_aged_all_ages', "bullying_victimisation")

df = pd.DataFrame(catalog.find('all_causes__drug_use__both_sexes__all_ages').load())
dfDrugs = finiliseOWID(df, 'dalys_from_all_causes_attributed_to_drug_use_per_100_000_people_in__both_sexes_aged_all_ages', "drug_use")

df = pd.DataFrame(catalog.find('all_causes__environmental_occupational_risks__both_sexes__all_ages').load())
dfEnvOccRisk = finiliseOWID(df, 'dalys_from_all_causes_attributed_to_environmental_occupational_risks_per_100_000_people_in__both_sexes_aged_all_ages',
"environmental_occupational_risks")

df = pd.DataFrame(catalog.find('all_causes__high_body_mass_index__both_sexes__all_ages').load())
dfBMI = finiliseOWID(df, 'dalys_from_all_causes_attributed_to_high_body_mass_index_per_100_000_people_in__both_sexes_aged_all_ages', "body_mass_index")

df = pd.DataFrame(catalog.find('all_causes__unsafe_water_source__both_sexes__all_ages').load())
dfUnsafeWater = finiliseOWID(df, 'dalys_from_all_causes_attributed_to_unsafe_water_source_per_100_000_people_in__both_sexes_aged_all_ages', "unsafe_water")

df = pd.DataFrame(catalog.find('pedestrian_road_injuries__both_sexes__all_ages', dataset='gbd_cause').load())
dfRoadInj = finiliseOWID(df, 'dalys_from_pedestrian_road_injuries_per_100_000_people_in__both_sexes_aged_all_ages', "pedestrian_road_injuries")

df = pd.read_csv('mentalHealth.csv')
df = df.set_index(['country', 'year', 'sex', 'age'])
df = df.sort_index()

# vvvvvvvvvvvvvvvvvvvvv fixing mental health import
# for mental health, there were multiple data points I wanted to import and combine, so I opted to write out the process specifically for it. It also had very different indexing to other
metrics which meant I couldn't use my functions, and had to adjust the data tables before calling them
df = pd.read_csv('mentalHealth.csv')
df = df.set_index(['country', 'year', 'sex', 'age'])

df = df.xs('Both', level=2, drop_level=False)
df = df.xs('All ages', level=3, drop_level=False)
df = df.reset_index(level='sex')
df = df.reset_index(level='age')
df = df.drop(['sex', 'age'], axis=1)     # selecting 'Both' genders and 'All ages' and only keeping this data

df = df.sort_index()
dfCut = selectCountries(df)

# add together all of the metrics I want
```

```
df['sum'] = df[['share_anxiety_disorders','share_bipolar_disorders','share_depressive_disorders','share_eating_disorders','share_schizophrenia_disorders']].sum(axis=1)
dfCut['sum'] = dfCut[['share_anxiety_disorders','share_bipolar_disorders','share_depressive_disorders','share_eating_disorders','share_schizophrenia_disorders']].sum(axis=1)

df = indexOWID(df['sum'], dfCut['sum'])

dfMentalHealth = df.rename('mental_health')
# ^^^^^^^^ mental health import

df = pd.DataFrame(catalog.find('fertility_rate').load())
dfFertility = finiliseOWID(df, 'fertility_rate', "fertility_rate")

df = pd.DataFrame(catalog.find('education_lee_lee').load())
dfLiteracy = finiliseOWID(df, 'mf_primary_enrollment_rates_combined_wb', "literacy_rate")




# below line is used to find metrics in the WB catalog that include 'GDP'
#print(wb.series.info(q='GDP'))
dfAll = wb.data.DataFrame('NY.GDP.MKTP.CD')
dfAll.to_csv('gdphelp.csv')

# ensuring only countries are used for the indexing, not regions
regions = ['WLD', 'HIC', 'OED', 'AFE', 'AFW', 'ARB', 'CEB', 'EAP', 'EAR', 'EAS', 'ECA', 'ECS', 'EMU', 'EUU', 'IBT', 'IDA', 'IDB', 'IDX', 'LAC', 'LIC', 'LMC', 'LMY', 'LTE', 'MEA', 'MIC', 'MNA',
'NAC', 'PRE', 'PST', 'SAS', 'SSA', 'SSF', 'TEA', 'TEC', 'TLA', 'TMN', 'TSA', 'TSS', 'UMC', 'IBD']
dfAll = dfAll.drop(index=regions)
# from the WB data, import the table NY.GDP.MKTP.CN, including only the following countries
df = wb.data.DataFrame('NY.GDP.MKTP.CD',['ITA','GBR', 'USA','FRA', 'JPN', 'DEU', 'CAN'])
dfGDP = formatWB(df)
dfGDP = dfGDP.rename("GDP")
dfGDP = indexWB(dfAll, dfGDP)
dfGDP = dfGDP.sort_index()



# the following use the same method, but I only explained through the above

dfAll = wb.data.DataFrame('NY.GDP.PCAP.CD')
dfAll = dfAll.drop(index=regions)
df = wb.data.DataFrame('NY.GDP.PCAP.CD',['ITA','GBR', 'USA','FRA', 'JPN', 'DEU', 'CAN'])
dfGDPperC = formatWB(df)
dfGDPperC = dfGDPperC.rename("GDP_per_capita")
dfGDPperC = indexWB(dfAll, dfGDPperC)
dfGDPperC = dfGDPperC.sort_index()

dfAll = wb.data.DataFrame('NY.GDP.MKTP.PP.CD')
dfAll = dfAll.drop(index=regions)
df = wb.data.DataFrame('NY.GDP.MKTP.PP.CD',['ITA','GBR', 'USA','FRA', 'JPN', 'DEU', 'CAN'])
dfGDP_PPP = formatWB(df)
dfGDP_PPP = dfGDP_PPP.rename("GDP_PPP")
dfGDP_PPP = indexWB(dfAll, dfGDP_PPP)
dfGDP_PPP = dfGDP_PPP.sort_index()



# the below metrics were all added in Prototype 3

# the comments are used so I know what data is missing from them and if there are any other issues, so I can resolve it later on
```

```
dfGNI = finiliseWB('NY.GNP.MKTP.CD','GNI')  # great
dfGNIperC= finiliseWB('NY.GNP.PCAP.CD', 'GNI_per_capita') # great
dfGNI_PPP= finiliseWB('NY.GNP.MKTP.PP.CD', 'GNI_PPP')  # great
dfInflation= finiliseWB("FP.CPI.TOTL.ZG", 'inflation')   # great
dfCAperGDP= finiliseWB("BN.CAB.XOKA.GD.ZS", 'current_account_pct_GDP')     # have to adjust for negative values
dfRnDExpPerGDP= finiliseWB("GB.XPD.RSDV.GD.ZS", 'R&D_expenditure_pct_GDP')    # great
dfGovExpPerGDP= finiliseWB("NE.CON.GOVT.ZS", 'government_expenditure_pct_GDP')  # great
dfGovExpHelPerGDP= finiliseWB("SH.XPD.GHED.GD.ZS", 'health_pct_government_expenditure')   # beauty
dfRuralPerPop= finiliseWB("SP.RUR.TOTL.ZS", 'rural_living_pct_popoluation')     # pinacle of beauty
dfMigration= finiliseWB("SM.POP.NETM", 'migration')         # great
dfLabourPar= finiliseWB("SL.TLF.CACT.ZS", 'labour_participation')      # great
dfGovEffectiveness= finiliseWB("GE.EST", 'government_effectiveness')     # missing 2001 from all countries, rest is great
dfGovExpEduPerGDP= finiliseWB("SE.XPD.TOTL.GD.ZS", 'education_pct_government_expenditure')  # Canada 2003, 2004, 2006
dfIncomeTax= finiliseWB("GC.TAX.YPKG.CN", 'income_tax')      # no data for Japan
dfTimeForTaxes= finiliseWB("IC.TAX.DURS", 'hours_to_file_taxes')      # no data for 2000-2004,  for US n Japan no data for 2000-2013
dfInterestRate= finiliseWB("FR.INR.RINR",'interest_rates')    # VERY LACKING,,,
dfPoliticalStability= finiliseWB("PV.EST", 'political_stability')         # quite good, no 2001 data
dfTradeTransportQuality= finiliseWB("LP.LPI.INFR.XQ", 'trade_transport_quality')    # only data for 6 years, prob taken every 2 years
dfForestArea= finiliseWB("AG.LND.FRST.ZS", 'forest_area')     # amazing
dfForestDepleteion= finiliseWB("NY.ADJ.DFOR.GN.ZS", 'forest_depletion_pct_GNI')  # beatiful
dfUnemployment = finiliseWB('SL.UEM.TOTL.ZS', 'unemployment')



# track fo stuff that didnt work while joining the Current table :(
# df = df.drop_duplicates(keep='first')
# df.index = df.index.drop_duplicates(keep='first') (Length mismatch: Expected axis has 192 elements, new values have 168 elements)
# df = df.groupby(df.index).first()
'''
for df in [dfLife, dfPol, dfIncomeInq, dfWealthInq, dfGDP, dfGDPperC, dfGDP_PPP, dfCarbon, dfGreenhouse, dfChildMor, dfRenewable]:
    df['country', 'year'] = df.index
    df = df.drop_duplicates(keep='first')
    df.set_index = df['country', 'year']
    df = df.drop(['country', 'year'])
'''
# turns out it was an error within the country selection function




# the following joins all of the tables into the Current table
tableList = [dfLife, dfPol, dfIncomeInq, dfWealthInq, dfGDP, dfGDPperC, dfGDP_PPP, dfCarbon, dfGreenhouse, dfChildMor, dfRenewable,
dfGNI,dfGNIperC,dfGNI_PPP,dfInflation,dfCAperGDP,dfRnDExpPerGDP,dfGovExpPerGDP,dfGovExpEduPerGDP,dfGovExpHelPerGDP,dfGovEffectiveness,dfIncomeTax,dfTimeFo
rTaxes,dfRuralPerPop,dfInterestRate,dfMigration,dfLabourPar,dfPoliticalStability,dfTradeTransportQuality, dfForestArea, dfForestDepleteion, dfFertility, dfMentalHealth, dfRoadInj,
dfBMI, dfUnsafeWater, dfBullying, dfEnvOccRisk, dfAlcohol, dfDrugs, dfLiteracy, dfUnemployment]

current = pd.concat(tableList, axis=1, join='inner')
current.to_csv("Current.csv", header=True)
```

‘allImportsRaw.py’

This code is the exact same as the above, except without applying the indexing. For this reason I will not include it in this section.

‘projectedAlgorithm.py’

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression

'''
Testing LinearRegression
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])

print(x)

model = LinearRegression()
model.fit(x, y)

print('intercept =', model.intercept_)
print("slope =", model.coef_)

# must use 2d arrays for prediction, can resize array to fir this with .reshape((-1, 1))
print("for x= 65, y=", model.predict([[65]]))
'''

# below is the process that needs to be carried out for the projected plan (which has been adjusted with later developments)
# get values from .csv file
# turn them into array
# create models for the data
# predict the values for next 8 quarters in the future

df = pd.read_csv("Current.csv")
df = df.set_index(['country','year'])

data = {}

countries = ["United Kingdom", "United States", "France", "Japan", "Germany", "Canada", "Italy"]
metrics = ['life_expectancy', 'fertility_rate', 'GDP', 'GDP_per_capita', 'GDP_PPP', 'GNI','GNI_per_capita', 'GNI_PPP', 'political_stability', 'interest_rates', 'income_tax',
'education_pct_government_expenditure', 'government_effectiveness', 'labour_participation', 'migration', 'rural_living_pct_popoluation', 'health_pct_government_expenditure',
'government_expenditure_pct_GDP', 'R&D_expenditure_pct_GDP', 'current_account_pct_GDP', 'inflation', 'forest_area', 'global_carbon_budget', 'environmental_occupational_risks',
'greenhouse_gas_emissions', 'renewable_electricity_capacity',  'trade_transport_quality', 'literacy_rate', 'alcohol_use_disorders', 'income_inequality', 'wealth_inequality',
'child_mortality', 'body_mass_index', 'hours_to_file_taxes', 'drug_use', 'mental_health', 'pedestrian_road_injuries', 'pollution', 'bullying_victimisation', 'forest_depletion_pct_GNI',
'unsafe_water', 'unemployment']

for country in countries:
    countryData = {}
    for metric in metrics:
        dfCut = df.loc[country, metric]
        countryData[metric] = dfCut.tolist()
    data[country] = countryData
# the above transforms the .csv table into a dictionary so I can iterate through it and use the data to predict

projectedData = {}
slopesData = {}

for metric in metrics:
```

```
    metricList = []
    slopesMetricList = []
    for country in countries:
        x = np.array([i for i in range(2000, 2020)])
        y = np.array(data[country][metric])

        length = 0                      # this section is used to get rid of any NaN values in the data
        while length < len(y):              # the algorithm still works if there's data missing, but it cannot interpret NaN values
            if str(y[length]) == 'nan':
                y = np.delete(y, length)
                x = np.delete(x, length)
            else:
                length = length + 1



        if len(x) == 0:
            for i in range(15):
                metricList.append('NaN')            # if all of the data is NaN, all projections are also NaN
            slopesMetricList.append('NaN')
            continue                        # if this is the case, continue to the next metric (do not run the projection algorithm)

        x = x.reshape((-1, 1))          # reshaping is needed in order for LinearRegression to work

        model = LinearRegression()      # create a model based on the data and years
        model.fit(x, y)

        slope = model.coef_
        slopesMetricList.append(slope[0])       # store the slope value (to be used in the optimum plan)

        #                   times by 100 since range needs to be integers, will need to divide at the end
        for quarter in range(201900, 202125, 25):
            quarter = quarter/100
            projection = model.predict([[quarter]])
            metricList.append(projection[0])
        for year in range(2025, 2055, 5):
            projection = model.predict([[year]])
            metricList.append(projection[0])

    projectedData[metric] = metricList
    slopesData[metric] = slopesMetricList



projectedIndex = []
for country in countries:                       # creates an index of quarters and years (dates to project)
    for quarter in range(201900, 202125, 25):
        quarter =  quarter/100
        year = int(quarter//1)
        quarter = int((quarter%1)*4)
        projectedIndex.append((country, str(year)+" QR:"+str(quarter)))
    for year in range(2025, 2055, 5):
        projectedIndex.append((country, str(year)))



projectedIndex = pd.MultiIndex.from_tuples(projectedIndex, names=["country", "date"])     # creates MultiIndex
projectedDf = pd.DataFrame(data= projectedData, index=projectedIndex)       # stores projected data in table with projected index
projectedDf.to_csv('Projected.csv')
```

```
# this is for the economic simulator
slopesDf = pd.DataFrame(data= slopesData, index= countries)
slopesDf.index.name = 'country'
slopesDf.to_csv('Slopes.csv')
```

‘econSimOptimum.py’

```
import pandas as pd

dfRules = pd.read_csv("Rules.csv")
dfRules = dfRules.set_index(['policy','date'])

dfSlopes = pd.read_csv("Slopes.csv")
dfSlopes = dfSlopes.set_index('country')

def optPlan(dfR, dfS, c):
    dfS = dfS.loc[c]
    # dont need to check for NaN values, since
    # 1. Slopes are only NaN if the WHOLE metric for the country is NaN
    # 2. and so it'd be NaN * NaN (no change)
    # (The slopes will never override data)


    policies = ['Income Tax', 'Demerit Goods Tax', 'Education Spending', 'Healthcare Spending', 'Environment Spending', 'Benefits Spending', 'Interest Rates', 'Regulation',
'Protectionism', 'Infrastructure']

    data = {}

    for policy in policies:              # creates a data list based on the overall change of each policy (sum of metric changes)
        temp = dfR.loc[policy].sum(axis='index')
        temp = temp * (1 - (5*dfS))
        s = temp.sum()
        data[policy] = s


    # WARNING,, THE KEYS ARE NOT UPDATED TO HAVE _
    # initialising all nodes for Dijkstra's algorithm according to planned graph design
    Start = {'Income_Tax': data['Income Tax'], 'Demerit_Goods_Tax': data['Demerit Goods Tax']}
    Income_Tax = {'Demerit_Goods_Tax': data['Demerit Goods Tax'], 'Education_Spending': data['Education Spending'], 'Healthcare_Spending': data['Healthcare Spending'],
'Environment_Spending': data['Environment Spending'], 'Benefits_Spending': data['Benefits Spending'], 'Infrastructure': data['Infrastructure']}
    Demerit_Goods_Tax = {'Income_Tax': data['Income Tax'], 'Education_Spending': data['Education Spending'], 'Healthcare_Spending': data['Healthcare Spending'],
'Environment_Spending': data['Environment Spending'], 'Benefits_Spending': data['Benefits Spending'], 'Infrastructure': data['Infrastructure']}
    Education_Spending = {'Regulation': data['Regulation'], 'Protectionism': data['Protectionism'], 'Healthcare_Spending': data['Healthcare Spending'], 'Environment_Spending':
data['Environment Spending'], 'Benefits_Spending': data['Benefits Spending'], 'Infrastructure': data['Infrastructure']}
    Healthcare_Spending = {'Regulation': data['Regulation'], 'Protectionism': data['Protectionism'], 'Education_Spending': data['Education Spending'], 'Environment_Spending':
data['Environment Spending'], 'Benefits_Spending': data['Benefits Spending'], 'Infrastructure': data['Infrastructure']}
    Benefits_Spending = {'Regulation': data['Regulation'], 'Protectionism': data['Protectionism'], 'Education_Spending': data['Education Spending'], 'Environment_Spending':
data['Environment Spending'], 'Healthcare_Spending': data['Healthcare Spending'], 'Infrastructure': data['Infrastructure']}
    Environment_Spending = {'Regulation': data['Regulation'], 'Protectionism': data['Protectionism'], 'Education_Spending': data['Education Spending'], 'Benefits_Spending':
data['Benefits Spending'], 'Healthcare_Spending': data['Healthcare Spending'], 'Infrastructure': data['Infrastructure']}
    Infrastructure = {'Regulation': data['Regulation'], 'Protectionism': data['Protectionism'], 'Education_Spending': data['Education Spending'], 'Benefits_Spending': data['Benefits
Spending'], 'Healthcare_Spending': data['Healthcare Spending'], 'Environment_Spending': data['Environment Spending']}
    Regulation = {'Protectionism': data['Protectionism'], 'Interest_Rates': data['Interest Rates']}
```

```
Protectionism = {'Regulation': data['Regulation'], 'Interest_Rates': data['Interest Rates']}
Interest_Rates = {'End': 0}
End = {}


nodes = {'Start': Start, 'Income_Tax': Income_Tax, 'Demerit_Goods_Tax': Demerit_Goods_Tax, 'Education_Spending': Education_Spending,
'Healthcare_Spending':Healthcare_Spending, 'Benefits_Spending': Benefits_Spending, 'Environment_Spending': Environment_Spending,
'Infrastructure': Infrastructure, 'Regulation': Regulation, 'Protectionism': Protectionism, 'Interest_Rates': Interest_Rates, 'End': End}

start = 'Start'
distance = {}

for node in nodes:          # initialises distances for Dijkstra's algorithm (maximises distance, so initially set very low)
    if node == start:
        distance[node] = 0
    else:
        distance[node] = -100000

# initialises priority queue (Start must be first, the rest of the order is not important)

priorityQueue = ['Start', 'Income_Tax', 'Demerit_Goods_Tax', 'Education_Spending', 'Healthcare_Spending', 'Benefits_Spending', 'Environment_Spending', 'Infrastructure',
'Regulation', 'Protectionism', 'Interest_Rates', 'End']
previous = {}

while len(priorityQueue) > 0:
    currentNode = priorityQueue.pop(0)
    for neighbour in nodes[currentNode]:
        newDist = distance[currentNode] + nodes[currentNode][neighbour]
        if newDist > distance[neighbour]:
            distance[neighbour] = newDist
            sortedDist = dict(sorted(distance.items(), key=lambda item: item[1], reverse=True))
            temp = []
            for node in sortedDist:
                if node in priorityQueue:
                    temp.append(node)
            priorityQueue = temp
            if neighbour not in previous:
                previous[neighbour] = 1
            else:
                previous[neighbour] += 1



        newDist = distance[currentNode] - nodes[currentNode][neighbour]
        if newDist > distance[neighbour]:
            distance[neighbour] = newDist
            sortedDist = dict(sorted(distance.items(), key=lambda item: item[1], reverse=True))
            temp = []
            for node in sortedDist:
                if node in priorityQueue:
                    temp.append(node)
            priorityQueue = temp
            if ('Negative' + neighbour) not in previous:
                previous["Negative " + neighbour] = 1
            else:
                previous["Negative " + neighbour] += 1
```

```
    # EXPLANATION OF DIJSTRA'S ALGORITHM (above)
    #    For every node in the priority list, and for every neighbour of that node, the 'positive' and 'negative' distance of that policy is calculated to find the best path from the current node to the neighbour nodes.
    #    If the new distance is greater than the previous one, the distance is updated, and the priority queue is reordered in descending order by distance. The neighbour is then added to the 'previous' list.
    #    The same process is repeated for negative paths

    for node in distance:
        distance[node] = round(distance[node], 1)     # fixing sum error (padas/dictionary error requires rounding)

    return previous, distance


econSimData = []

countries = ["Italy", "United Kingdom", "United States", "France", "Japan", "Germany", "Canada"]
for country in countries:
    route, distances = optPlan(dfRules, dfSlopes, country)  # calculates the optimum plan based on each country

    countryData = {}
    for path in route:       # the following figures out the overall policies for each country
        if 'Negative' in path:
            temp = path.partition(' ')[2]               # this is used to substract 'Negative' changes from the positive changes
            countryData[temp] = countryData[temp] - route[path]      # eg: Raise Income tax by 10% and Lower Income tax by 5% results
        elif path not in econSimData and path != 'End':              #     in an overall change of 5%
            countryData[path] = route[path]

    for policy in countryData:
        econSimData.append(countryData[policy])


temp = {}
temp['amount'] = econSimData       # just to have the column named 'amount' in the dataframe
econSimData = temp


policies = ['Income_Tax', 'Demerit_Goods_Tax', 'Education_Spending', 'Healthcare_Spending', 'Environment_Spending', 'Benefits_Spending', 'Infrastructure', 'Regulation', 'Protectionism', 'Interest_Rates']

econSimIndex = []
for country in countries:        # adds all the data to a list, which will be turned into a table
    for policy in policies:
        econSimIndex.append((country, policy))


econSimIndex = pd.MultiIndex.from_tuples(econSimIndex, names=["country", "policy"])

projectedDf = pd.DataFrame(data= econSimData, index=econSimIndex)
projectedDf.to_csv('EconSimOptPlan.csv')
```

'mapClasses.py'

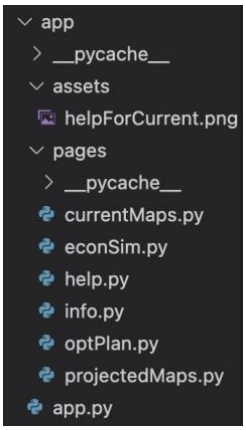
```
import pandas as pd
import json
```

```
import plotly.express as px



class CCMap:
    def __init__(self, mapColour, mapDataframeName):
        # initialises the metrics of maps according to the selected colour (as well as name, index name, and map date)
        if mapColour == 'blue':
            self.mapColour = ["rgb(200, 240, 255)", "rgb(50, 50, 255)"]
            self.mapMetrics = ['life_expectancy', 'alcohol_use_disorders', 'fertility_rate', 'income_inequality', 'wealth_inequality', 'GDP', 'GDP_per_capita', 'GDP_PPP',
'GNI','GNI_per_capita', 'GNI_PPP','child_mortality', 'political_stability', 'interest_rates', 'body_mass_index','hours_to_file_taxes', 'income_tax', 'education_pct_government_expenditure',
'government_effectiveness', 'labour_participation', 'migration', 'rural_living_pct_popoluation', 'health_pct_government_expenditure', 'government_expenditure_pct_GDP',
'R&D_expenditure_pct_GDP', 'current_account_pct_GDP', 'inflation', 'bullying_victimisation', 'drug_use', 'mental_health', 'pedestrian_road_injuries', 'literacy_rate', 'unemployment']
        elif mapColour == 'green':
            self.mapColour = ["rgb(179, 242, 180)", "rgb(38, 128, 13)"]
            self.mapMetrics = ['pollution', 'forest_area', 'forest_depletion_pct_GNI', 'global_carbon_budget', 'environmental_occupational_risks', 'greenhouse_gas_emissions',
'renewable_electricity_capacity',  'trade_transport_quality',  'unsafe_water','GDP', 'GDP_per_capita', 'GDP_PPP', 'GNI','GNI_per_capita', 'GNI_PPP']
        else:
            return 'invalid colour'
        self.mapDataframeName = mapDataframeName
        self.indexName = 'year'
        self.mapDate = 2019

    def getMetrics(self):          # get method to access metrics
        return self.mapMetrics



    def getDataframe(self):
        df = pd.read_csv(self.mapDataframeName)
        df = df.set_index(self.indexName)
        df = df.replace("United States", "United States of America")  # the geojson file has a different name, so changing it is
                                                                                                                                                    necessary
        self.mapDataframe = df
        return self.mapDataframe

    def createMap(self):
        countriesMap = json.load(open("countries.geojson", 'r'))

        df = pd.read_csv(self.mapDataframeName)
        df = df.set_index(self.indexName)
        df = df.loc[self.mapDate]
        df = df.replace("United States", "United States of America")  # the geojson file has a different name, so changing it is
                                                                                                                                                    necessary

        metrics = df.columns.values.tolist()
        for metric in metrics:                      # drops all of the metrics that are not included in the specific map
            if metric not in self.mapMetrics and metric != 'country':
                df = df.drop(metric, axis=1)

        self.mapDataframe = df

        for country in countriesMap['features']:
            country['id'] = country['properties']['ADMIN']
            # adding a new property 'id' to the main list of properties of every country
            # the id is simply the name of the country, as they are unique
            # this is used so the id (country name) can be accessed from the first level of the geojson file
```

```
        colorscale = self.mapColour

        ccmap = px.choropleth(df, locations='country', geojson=countriesMap, color='GDP', color_continuous_scale=colorscale, height=500)

        userOptions = df.columns.values.tolist()
        del userOptions[0]    # removing 'country' from the list of options, as we only want user to choose between metrics

        return ccmap, userOptions


class ProjectedCCMap(CCMap):
    # initialises the Projected map with the slight changes from Current maps (map dates and index name)
    def __init__(self, mapColour, mapDataframe):
        CCMap.__init__(self, mapColour, mapDataframe)
        self.indexName = 'date'
        self.mapDate = "2019 QR:0"


class EconSimCCMap(ProjectedCCMap):
    # initialises the 'normal' and 'opposite' metrics so they list can be accessed and consistent across programs
    def __init__(self, mapColour, mapDataframe):
        CCMap.__init__(self, mapColour, mapDataframe)
        self.indexName = 'date'
        self.mapDate = "2019 QR:0"
        self.normalMetrics = ['life_expectancy', 'fertility_rate', 'GDP', 'GDP_per_capita', 'GDP_PPP', 'GNI','GNI_per_capita', 'GNI_PPP', 'political_stability', 'interest_rates', 'income_tax',
'education_pct_government_expenditure', 'government_effectiveness', 'labour_participation', 'migration', 'rural_living_pct_popoluation', 'health_pct_government_expenditure',
'government_expenditure_pct_GDP', 'R&D_expenditure_pct_GDP', 'current_account_pct_GDP', 'inflation', 'forest_area', 'global_carbon_budget', 'environmental_occupational_risks',
'greenhouse_gas_emissions', 'renewable_electricity_capacity',  'trade_transport_quality', 'literacy_rate']
        self.oppositeMetrics = ['alcohol_use_disorders', 'income_inequality', 'wealth_inequality', 'child_mortality', 'body_mass_index', 'hours_to_file_taxes', 'drug_use', 'mental_health',
'pedestrian_road_injuries', 'pollution', 'bullying_victimisation', 'forest_depletion_pct_GNI', 'unsafe_water', 'unemployment']
        self.policies = ['Income_Tax', 'Demerit_Goods_Tax', 'Education_Spending', 'Healthcare_Spending', 'Environment_Spending', 'Benefits_Spending', 'Infrastructure', 'Regulation',
'Protectionism', 'Interest_Rates']

        dates = []              # initialises dates to the specific ones needed for the economic simulator
        for quarter in range(201900, 202125, 25):
            quarter =  quarter/100
            year = int(quarter//1)
            quarter = int((quarter%1)*4)
            dates.append(str(year)+" QR:"+str(quarter))
        dates.append("2025")
        dates.append("2035")
        dates.append("2050")
        self.dates = dates


    def getRules(self):
        policies = ['Income_Tax', 'Demerit_Goods_Tax', 'Education_Spending', 'Healthcare_Spending', 'Environment_Spending', 'Benefits_Spending', 'Regulation', 'Protectionism',
'Interest_Rates', 'Infrastructure']
        # different from self.policies due to the ordering of the Rules table
        df = pd.read_csv('Rules.csv')
        df = df.set_index(['policy', 'date'])
        for metric in self.oppositeMetrics:    # adjusts the data for 'opposite' metrics so they cause reverse their effect
            data = []                          # ie: a drop in pollution should be considered positive
            for policy in policies:
                for date in self.dates:
                    temp = -1 * df.loc[(policy, date)][metric]
```

```
                if temp == 0:
                    temp = 0        # adjusts for -0
                data.append(temp)
            df[metric] = data

        return df

    def getEmpty(self):        # used for economic simulator while calculating policy/metric changes
        df = pd.read_csv('RulesEmpty.csv')
        df = df.set_index(['total', 'date'])
        df = df.replace("United States", "United States of America")
        return df

    def getOptEmpty(self):    # used for optimum plan while calculating metric changes
        df = pd.read_csv('RulesOptEmpty.csv')
        df = df.set_index(['country', 'date'])
        return df

    def getOptPlan(self):
        df = pd.read_csv('EconSimOptPlan.csv')
        df = df.set_index(['country', 'policy'])

        optPlan = {}
        for i in df.index.tolist():
            optPlan[i] = df.loc[i]['amount']

        return optPlan

    def getOptPlanText(self, country):        # returns formatted optimum plan (to be displayed more easily)
        df = pd.read_csv('EconSimOptPlan.csv')
        df = df.set_index(['country', 'policy'])

        policies = ['Income_Tax', 'Demerit_Goods_Tax', 'Education_Spending', 'Healthcare_Spending', 'Environment_Spending', 'Benefits_Spending', 'Regulation', 'Protectionism',
'Interest_Rates', 'Infrastructure']

        optPlanText = []

        for policy in policies:
            temp = policy + ': ' + str((df.loc[(country, policy)]['amount'] * 5)) + '%       \n'
            optPlanText.append(temp)
        optPlanText = ''.join(optPlanText)

        return optPlanText
```

‘app.py’



The application files as stored as follows:

```
import dash
from dash import html, dcc
import dash_bootstrap_components as dbc

# initialising a multipage app with the stylesheet theme SPACELAB
app = dash.Dash(__name__, use_pages=True, external_stylesheets=[dbc.themes.SPACELAB])

menu = dbc.Nav(
        # creating a menu to access every page
        [
            dbc.NavLink(                            # navlink with the name and path asigned in each file
                [
                    html.Div(page["name"], className="ms-2"),
                ],
                href=page["path"],
                active="exact",    # highlights active page in menu
            )

            for page in dash.page_registry.values()         # this is done for every page in the registry
        ],
        vertical=False,
        pills=True,
        className="bg-light",
)


app.layout = dbc.Container([
    # title of the page
    dbc.Row([
        dbc.Col(html.Div("Colour Coded Maps",
                         style={'fontSize':50, 'textAlign':'center'}))
    ]),

    html.Hr(),  # line break
    dbc.Row([
        menu
    ]),
    html.Hr(),

    dbc.Row([
        dash.page_container
    ])
```

```
], fluid=True)



if __name__ == "__main__":
    app.run(debug=True)
```

‘currentMaps.py’

```
import pandas as pd
import json
import plotly.express as px
import dash
from dash import html, dcc, callback, Output, Input, Patch
import dash_bootstrap_components as dbc

import sys
sys.path.append('/Users/irenepeleteiropaniagua/Documents/shhs/computing/projectPrograms')
from mapClasses import *

# registering this page as the homepage (path='/') to the app registry
dash.register_page(__name__, path='/', name="Current Map")

# MAP CLASSES
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

SCurrentData = CCMap('blue', 'Current.csv')
ECurrentData = CCMap('green', 'Current.csv')

SCurrentRawData = CCMap('blue', 'CurrentRaw.csv')
ECurrentRawData = CCMap('green', 'CurrentRaw.csv')

currentMaps = {'SCurrentData': SCurrentData, 'ECurrentData': ECurrentData}
currentRawMaps = {'SCurrentRawData': SCurrentRawData, 'ECurrentRawData': ECurrentRawData}

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


ccmap, userOptions = currentMaps['SCurrentData'].createMap()
df = currentMaps['SCurrentData'].getDataframe()
df = df.loc[currentMaps['SCurrentData'].mapDate]
years = []
for i in range(2000,2020):
    years.append(i)

layout = html.Div(
    [
        # dropdown selection of environmental/social map
        dbc.Row(
            dcc.Dropdown(options=['Environment', 'Social'],
                        id='map-choiceC',
                        value='Social',
                        clearable=False
```

```
        )
    ),

    html.Hr(),
    # radio button selection of dates
    dbc.Row(
        dcc.RadioItems(options=years,
                       id='date-choiceC',
                       value=2019,
                       inline=True,
                       labelStyle={'padding':'10px'}
        )
    ),
    # warning may be displayed according to user choices, initially empty
    html.Div(id='warningC', style= {'textAlign':'center'}),

    # radio button selection between indexed and raw data
    dbc.Row(
        dcc.RadioItems(options=['Indexed Data', 'Raw Data'],
                       id='data-choiceC',
                       value='Indexed Data',
                       inline = True,
                       labelStyle={'padding':'10px'}
        )
    ),



    dbc.Row(
        [
            dbc.Col(
                [

                                    # checklist used for selection of metrics
                    dcc.Checklist(options=currentMaps['SCurrentData'].getMetrics(),
                                  id='metric-choiceC')
                ], xs=4, sm=3, md=2, lg=2, xl=2, xxl=2      # ensures site is divided in a sensible way according to size
            ),

            dbc.Col(
                [
                                    # displays map according to user selections
                    dcc.Graph(id='currentMap',
                              figure= ccmap)
                ], xs=8, sm=9, md=10, lg=10, xl=10, xxl=10
            )
        ]
    ),

    html.Hr(),


  ]
)
# changes the metrics displayed according to the choice of map
@callback(
   Output(component_id='metric-choiceC', component_property='options'),
```

```
    Input(component_id='map-choiceC', component_property='value')
)
def setMetricChoices(mapChoice):
    if mapChoice == 'Environment':
        return currentMaps['ECurrentData'].getMetrics()
    elif mapChoice == 'Social':
        return currentMaps['SCurrentData'].getMetrics()
    else:
        return []

# sets the initial value of the metric choices (eg: life_expectancy for Social maps)
@callback(
    Output(component_id='metric-choiceC', component_property='value'),
    Input(component_id='metric-choiceC', component_property='options')
)
def setInitialValue(metricOptions):
    value = metricOptions[0]
    return [value]

# sets the warnings according to user selection (missing data or no metric selected warnings)
@callback(
    Output(component_id='warningC', component_property='children'),
    Input(component_id='metric-choiceC', component_property='value')
)
def changeWaring(metricChoice):
    if len(metricChoice) > 0:
        for metric in metricChoice:
            if metric in
['income_tax','hours_to_file_taxes','governement_effectiveness','education_pct_government_expenditure','interest_rates','trade_transport_quality','political_stability','fertility_rate','literac
y_rate']:
                newWarning = 'At least one of the metrics selected is missing some data.'
                break
            else:
                newWarning = ''
        return newWarning
    else:
        return 'At least one metric must be selected.'

# main callback to change the map according to user choices

@callback(
    Output(component_id='currentMap', component_property='figure'),
    [Input(component_id='metric-choiceC', component_property='value'),
     Input(component_id='date-choiceC', component_property='value'),
     Input(component_id='map-choiceC', component_property='value'),
     Input(component_id='data-choiceC', component_property='value')]
)
def update_graph(metricChoice, dateChoice, mapChoice, dataChoice):
    patchedFig = Patch()

    if dataChoice == 'Indexed Data':          # selects indexed or raw data
        df = currentMaps['SCurrentData'].getDataframe()
    elif dataChoice == 'Raw Data':
        df = currentRawMaps['SCurrentRawData'].getDataframe()
```

```
df = df.loc[dateChoice]



if mapChoice == 'Environment':      # sections the data selected according to map choice, and sets correct colour
    userOptions = currentMaps['ECurrentData'].getMetrics()    # list of metrics is same for raw and indexed
    metrics = df.columns.values.tolist()
    for metric in metrics:
        if metric not in userOptions and metric != 'country':
            df = df.drop(metric, axis=1)
    patchedFig['layout']['coloraxis']['colorscale'] = ((0.0, "rgb(179, 242, 180)"), (1.0, "rgb(38, 128, 13)"))

elif mapChoice == 'Social':
    userOptions = currentMaps['SCurrentData'].getMetrics()
    metrics = df.columns.values.tolist()
    for metric in metrics:
        if metric not in userOptions and metric != 'country':
            df = df.drop(metric, axis=1)
    patchedFig['layout']['coloraxis']['colorscale'] = ((0.0, "rgb(200, 240, 255)"), (1.0,  "rgb(50, 50, 255)"))



if len(metricChoice) == 0:     # if no metrics are selected, no data
    mean = []
elif len(metricChoice) == 1:      # if one metric is selected, the mean is simply the data values
    mean = df[metricChoice].values.tolist()
    fix = []
    for i in range(len(mean)):     # fix is used to adapt data into list
        fix.append(mean[i][0])
    mean = fix
elif len(metricChoice) > 1:     # if more than one metric is selected, the mean for each country is calculated
    dataList = df[metricChoice[0]]
    for i in range(len(metricChoice)-1):
        i += 1
        dataList = pd.concat([dataList, df[metricChoice[i]]], axis=1)
    mean = dataList.mean('columns')
    mean = mean.tolist()

patchedFig['data'][0]['z'] = mean     # sets the map data to the calculated values (according to user selection)

# adapts the hover values according to user choices (name of metric if 1 selected, average if >1)
if len(metricChoice) == 0:
    pass
elif len(metricChoice) == 1:
    patchedFig['layout']['coloraxis']['colorbar']['title']['text'] = '('+dataChoice+') ' + metricChoice[0]
    patchedFig['data'][0]['hovertemplate'] = 'country=%{location}<br>'+ metricChoice[0] +'=%{z}<extra></extra>'
else:
    patchedFig['layout']['coloraxis']['colorbar']['title']['text'] = '('+dataChoice+') ' + 'Average'
    patchedFig['data'][0]['hovertemplate'] = 'country=%{location}<br>Average=%{z}<extra></extra>'

    # inside the geojson file, the hovertemplate is the following: 'country=%{location}<br>GDP=%{z}<extra></extra>'
    # to change the metric, I simply need to change the name of the metric (from GDP to whatever 'value' is being chosen)

return patchedFig
```

‘projectedMaps.py’

```
import pandas as pd
import json
import plotly.express as px
import dash
from dash import html, dcc, callback, Output, Input, Patch
import dash_bootstrap_components as dbc

import sys
sys.path.append('/Users/irenepeleteiropaniagua/Documents/shhs/computing/projectPrograms')
from mapClasses import *

# registering this page as the homepage (path='/') to the app registry
dash.register_page(__name__, name="Projected Map")

# MAP CLASSES
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

SProjectedData = ProjectedCCMap('blue', 'Projected.csv')
EProjectedData = ProjectedCCMap('green', 'Projected.csv')

projectedMaps = {'SProjectedData': SProjectedData, 'EProjectedData': EProjectedData}


# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ccmap, userOptions = projectedMaps['SProjectedData'].createMap()
df = projectedMaps['SProjectedData'].getDataframe()
df = df.loc[projectedMaps['SProjectedData'].mapDate]

quarters = []
for quarter in range(201900, 202125, 25):        # creates list of quarters and years
    quarter =  quarter/100
    year = int(quarter//1)
    quarter = int((quarter%1)*4)
    quarters.append(str(year)+" QR:"+str(quarter))
for year in range(2025, 2050, 5):
    quarters.append(str(year))


layout = html.Div(
    [

        # dropdown option between social and environment
        dbc.Row(
            dcc.Dropdown(options=['Environment', 'Social'],
                        id='map-choiceP',
                        value='Social',
                        clearable=False
            )
        ),

        html.Hr(),
```

```
        # radio button selection of dates
        dbc.Row(
            dcc.RadioItems(options=quarters,
                           id='date-choiceP',
                           value="2019 QR:0",
                           inline=True,
                           labelStyle={'padding':'10px'}
            )
        ),
        # warning may be displayed according to user choices, initially empty
        html.Div(id='warningC', style= {'textAlign':'center'}),

        # radio button selection between indexed and raw data
        dbc.Row(
            [
                dbc.Col(
                    [
                        # checklist used for selection of metrics
                        dcc.Checklist(options=projectedMaps['EProjectedData'].getMetrics(),
                                      id='metric-choiceP')
                    ], xs=4, sm=3, md=2, lg=2, xl=2, xxl=2
                ),

                dbc.Col(
                    [
                        # displays map according to user selections
                        dcc.Graph(id='projectedMap',
                                  figure= ccmap)
                    ], xs=8, sm=9, md=10, lg=10, xl=10, xxl=10
                )
            ]
        ),

        html.Hr(),
    ]
)

# changes the metrics displayed according to the choice of map
@callback(
    Output(component_id='metric-choiceP', component_property='options'),
    Input(component_id='map-choiceP', component_property='value')
)
def setMetricChoices(mapChoice):
    if mapChoice == 'Environment':
        return projectedMaps['EProjectedData'].getMetrics()
    elif mapChoice == 'Social':
        return projectedMaps['SProjectedData'].getMetrics()
    else:
        return []

# sets the initial value of the metric choices (eg: life_expectancy for Social maps)
@callback(
    Output(component_id='metric-choiceP', component_property='value'),
    Input(component_id='metric-choiceP', component_property='options')
)
```

```
def setInitialValue(metricOptions):
    value = metricOptions[0]
    return [value]

# sets the warnings according to user selection (missing data or no metric selected warnings)
@callback(
    Output(component_id='warningP', component_property='children'),
    Input(component_id='metric-choiceP', component_property='value')
)
def changeWaring(metricChoice):
    if len(metricChoice) > 0:
        for metric in metricChoice:
            if metric in
['income_tax','hours_to_file_taxes','governement_effectiveness','education_pct_government_expenditure','interest_rates','trade_transport_quality','political_stability','fertility_rate','literac
y_rate']:
                newWarning = 'At least one of the metrics selected is missing some data.'
                break
            else:
                newWarning = ''
        return newWarning
    else:
        return 'At least one metric must be selected.'


# main callback to change the map according to user choices

@callback(
    Output(component_id='projectedMap', component_property='figure'),
    [Input(component_id='metric-choiceP', component_property='value'),
     Input(component_id='date-choiceP', component_property='value'),
     Input(component_id='map-choiceP', component_property='value')]
)
def update_graph(metricChoice, dateChoice, mapChoice):
    patchedFig = Patch()

    print(metricChoice, dateChoice, mapChoice)

    df = projectedMaps['SProjectedData'].getDataframe()
    df = df.loc[dateChoice]


    if mapChoice == 'Environment':          # sections the data selected according to map choice, and sets correct colour
        userOptions = projectedMaps['EProjectedData'].getMetrics()
        metrics = df.columns.values.tolist()
        for metric in metrics:
            if metric not in userOptions and metric != 'country':
                df = df.drop(metric, axis=1)
        patchedFig['layout']['coloraxis']['colorscale'] = ((0.0, "rgb(179, 242, 180)"), (1.0, "rgb(38, 128, 13)"))

    elif mapChoice == 'Social':
        userOptions = projectedMaps['SProjectedData'].getMetrics()
        metrics = df.columns.values.tolist()
        for metric in metrics:
            if metric not in userOptions and metric != 'country':
                df = df.drop(metric, axis=1)
        patchedFig['layout']['coloraxis']['colorscale'] = ((0.0, "rgb(200, 240, 255)"), (1.0,  "rgb(50, 50, 255)"))
```

```
    if len(metricChoice) == 0:        # if no metrics are selected, no data
        mean = []
    elif len(metricChoice) == 1:       # if one metric is selected, the mean is simply the data values
        mean = df[metricChoice].values.tolist()
        fix = []
        for i in range(len(mean)):
            fix.append(mean[i][0])      # fix is used to adapt data into list
        mean = fix
    elif len(metricChoice) > 1:        # if more than one metric is selected, the mean for each country is calculated
        dataList = df[metricChoice[0]]
        for i in range(len(metricChoice)-1):
            i += 1
            dataList = pd.concat([dataList, df[metricChoice[i]]], axis=1)
        mean = dataList.mean('columns')
        mean = mean.tolist()

    patchedFig['data'][0]['z'] = mean      # sets the map data to the calculated values (according to user selection)

    # adapts the hover values according to user choices (name of metric if 1 selected, average if >1)
    if len(metricChoice) == 0:
        pass
    elif len(metricChoice) == 1:
        patchedFig['layout']['coloraxis']['colorbar']['title']['text'] = metricChoice[0]
        patchedFig['data'][0]['hovertemplate'] = 'country=%{location}<br>'+ metricChoice[0] +'=%{z}<extra></extra>'
    else:
        patchedFig['layout']['coloraxis']['colorbar']['title']['text'] = 'Average'
        patchedFig['data'][0]['hovertemplate'] = 'country=%{location}<br>Average=%{z}<extra></extra>'

        # inside the geojson file, the hovertemplate is the following: 'country=%{location}<br>GDP=%{z}<extra></extra>'
        # to change the metric, I simply need to change the name of the metric (from GDP to whatever 'value' is being chosen)

    return patchedFig
```

'econSim.py'

```
import pandas as pd
import json
import plotly.express as px
import dash
from dash import html, dcc, callback, Output, Input, Patch
import dash_bootstrap_components as dbc

import sys
sys.path.append('/Users/irenepeleteiropaniagua/Documents/shhs/computing/projectPrograms')
from mapClasses import *

# registering this page as the homepage (path='/') to the app registry
dash.register_page(__name__, name="Economic Simulator")

# MAP CLASSES
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
SEconSim = EconSimCCMap('blue', 'Projected.csv')
EEconSim = EconSimCCMap('green', 'Projected.csv')

econSimMaps = {'SEconSim': SEconSim, 'EEconSim': EEconSim}

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


ccmap, userOptions = econSimMaps['SEconSim'].createMap()
df = econSimMaps['SEconSim'].getDataframe()
df = df.loc[econSimMaps['SEconSim'].mapDate]

dates = econSimMaps['SEconSim'].dates

layout = html.Div(
    [
        dbc.Row(
            [
            dbc.Col(
                    [
                        dbc.Row(
                            html.H1("Select the below policies to raise them by 5%", style={'fontSize':15})
                        ),

                        dbc.Row(     # allows user selection of positive policies
                            dcc.Checklist(options=['Income_Tax', 'Demerit_Goods_Tax', 'Education_Spending', 'Healthcare_Spending', 'Environment_Spending', 'Benefits_Spending',
'Infrastructure', 'Regulation', 'Protectionism', 'Interest_Rates'],
                            id='policy-choiceE',
                            value=['Income_Tax']))
                        ]
                    ),

                dbc.Col(
                    [
                        dbc.Row(
                            html.H1("Select the below policies to lower them by 5%", style={'fontSize':15})
                        ),

                        dbc.Row(      # allows user selection of negative policies
                            dcc.Checklist(options=['Income_Tax', 'Demerit_Goods_Tax', 'Education_Spending', 'Healthcare_Spending', 'Environment_Spending', 'Benefits_Spending',
'Infrastructure', 'Regulation', 'Protectionism', 'Interest_Rates'],
                            id='neg-policy-choiceE',
                            value=['Income_Tax']))
                    ]
                )
            ]
        ),

        html.Hr(),

        # dropdown option between social and environment
        dbc.Row(
            dcc.Dropdown(options=['Environment', 'Social'],
                            id='map-choiceE',
```

```
                    value='Social',
                    clearable=False
                )
            ),

            html.Hr(),
            # radio button selection of dates
            dbc.Row(
                dcc.RadioItems(options=dates,
                    id='date-choiceE',
                    value='2019 QR:0',
                    inline=True,
                    labelStyle={'padding':'10px'}
                )
            ),
            # warning may be displayed according to user choices, initially empty
            html.Div(id='warningE', style= {'textAlign':'center'}),

            # radio button selection between indexed and raw data
            dbc.Row(
                [
                    dbc.Col(
                        [
                                    # checklist used for selection of metrics
                            dcc.Checklist(options=econSimMaps['SEconSim'].getMetrics(),
                                    id='metric-choiceE')
                        ], xs=4, sm=3, md=2, lg=2, xl=2, xxl=2   # ensures site is divided in a sensible way according to size
                    ),

                    dbc.Col(
                        [
                                    # displays map according to user selections
                            dcc.Graph(id='econSimMap',
                                    figure= ccmap)
                        ], xs=8, sm=9, md=10, lg=10, xl=10, xxl=10
                    )
                ]
            ),

            html.Hr(),

        ]
)
# changes the metrics displayed according to the choice of map
@callback(
    Output(component_id='metric-choiceE', component_property='options'),
    Input(component_id='map-choiceE', component_property='value')
)
def setMetricChoices(mapChoice):
    if mapChoice == 'Environment':
        return econSimMaps['EEconSim'].getMetrics()
    elif mapChoice == 'Social':
        return econSimMaps['SEconSim'].getMetrics()
    else:
        return []
```

```
# sets the initial value of the metric choices (eg: life_expectancy for Social maps)
@callback(
    Output(component_id='metric-choiceE', component_property='value'),
    Input(component_id='metric-choiceE', component_property='options')
)
def setInitialValue(metricOptions):
    value = metricOptions[0]
    return [value]

# sets the warnings according to user selection (missing data or no metric selected warnings)
@callback(
    Output(component_id='warningE', component_property='children'),
    Input(component_id='metric-choiceE', component_property='value')
)
def changeWaring(metricChoice):
    if len(metricChoice) > 0:
        for metric in metricChoice:
            if metric in
['income_tax','hours_to_file_taxes','governement_effectiveness','education_pct_government_expenditure','interest_rates','trade_transport_quality','political_stability','fertility_rate','literac
y_rate']:
                newWarning = 'At least one of the metrics selected is missing some data.'
                break
            else:
                newWarning = ''
        return newWarning
    else:
        return 'At least one metric must be selected.'
# main callback to change the map according to user choices

@callback(
    Output(component_id='econSimMap', component_property='figure'),
    [Input(component_id='metric-choiceE', component_property='value'),
     Input(component_id='date-choiceE', component_property='value'),
     Input(component_id='map-choiceE', component_property='value'),
     Input(component_id='policy-choiceE', component_property='value'),
     Input(component_id='neg-policy-choiceE', component_property='value')]
)
def update_graph(metricChoice, dateChoice, mapChoice, policyChoice, negPolicyChoice):
    patchedFig = Patch()

    df = econSimMaps['SEconSim'].getDataframe()
    df = df.loc[dateChoice]

    dfRules = econSimMaps['SEconSim'].getRules()

    totalRules = econSimMaps['SEconSim'].getEmpty()      # gets an empty rules table and adds the effects of the user's policies
    for policy in policyChoice:
        totalRules = totalRules + dfRules.loc[policy]
    for policy in negPolicyChoice:
        totalRules = totalRules - dfRules.loc[policy]

    df['date'] = df.index
    df = df.reset_index(drop=True)
    df = df.set_index(['country', 'date'])        # setting a different index so the data can be accessed and modified
    for country in ["Italy", "United Kingdom", "United States of America", "France", "Japan", "Germany", "Canada"]:
```

```
    df.loc[(country, dateChoice)] = df.loc[(country, dateChoice)]*(1 + totalRules.loc[('Total', dateChoice)]/100)

tempC = []                      # resetting the index to be only 'date', not ['country', 'date']
tempD = []
for i in df.index:
    tempC.append(i[0])
    tempD.append(i[1])
df['date'] = tempD
df['country'] = tempC
df = df.reset_index(drop=True)
df = df.set_index('date')



if mapChoice == 'Environment':          # sections the data selected according to map choice, and sets correct colour
    userOptions = econSimMaps['SEconSim'].getMetrics()    # list of metrics is same for raw and indexed
    metrics = df.columns.values.tolist()
    for metric in metrics:
        if metric not in userOptions and metric != 'country':
            df = df.drop(metric, axis=1)
    patchedFig['layout']['coloraxis']['colorscale'] = ((0.0, "rgb(179, 242, 180)"), (1.0, "rgb(38, 128, 13)"))

elif mapChoice == 'Social':
    userOptions = econSimMaps['SEconSim'].getMetrics()
    metrics = df.columns.values.tolist()
    for metric in metrics:
        if metric not in userOptions and metric != 'country':
            df = df.drop(metric, axis=1)
    patchedFig['layout']['coloraxis']['colorscale'] = ((0.0, "rgb(200, 240, 255)"), (1.0,  "rgb(50, 50, 255)"))

if len(metricChoice) == 0:      # if no metrics are selected, no data
    mean = []
elif len(metricChoice) == 1:      # if one metric is selected, the mean is simply the data values
    mean = df[metricChoice].values.tolist()
    fix = []
    for i in range(len(mean)):
        fix.append(mean[i][0])      # fix is used to adapt data into list
    mean = fix
elif len(metricChoice) > 1:       # if more than one metric is selected, the mean for each country is calculated
    dataList = df[metricChoice[0]]
    for i in range(len(metricChoice)-1):
        i += 1
        dataList = pd.concat([dataList, df[metricChoice[i]]], axis=1)
    mean = dataList.mean('columns')
    mean = mean.tolist()

patchedFig['data'][0]['z'] = mean    # sets the map data to the calculated values (according to user selection)

# adapts the hover values according to user choices (name of metric if 1 selected, average if >1)
if len(metricChoice) == 0:
    pass
elif len(metricChoice) == 1:
    patchedFig['layout']['coloraxis']['colorbar']['title']['text'] = metricChoice[0]
    patchedFig['data'][0]['hovertemplate'] = 'country=%{location}<br>'+ metricChoice[0] +'=%{z}<extra></extra>'
else:
    patchedFig['layout']['coloraxis']['colorbar']['title']['text'] = 'Average'
    patchedFig['data'][0]['hovertemplate'] = 'country=%{location}<br>Average=%{z}<extra></extra>'
```

```
        # inside the geojson file, the hovertemplate is the following: 'country=%{location}<br>GDP=%{z}<extra></extra>'
        # to change the metric, I simply need to change the name of the metric (from GDP to whatever 'value' is being chosen)

    return patchedFig
```

'optPlan.py'

```
import pandas as pd
import json
import plotly.express as px
import dash
from dash import html, dcc, callback, Output, Input, Patch
import dash_bootstrap_components as dbc

import sys
sys.path.append('/Users/irenepeleteiropaniagua/Documents/shhs/computing/projectPrograms')
from mapClasses import *

# registering this page as the homepage (path='/') to the app registry
dash.register_page(__name__, name="Optimum Plan")

# MAP CLASSES
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

SEconSim = EconSimCCMap('blue', 'Projected.csv')
EEconSim = EconSimCCMap('green', 'Projected.csv')

econSimMaps = {'SEconSim': SEconSim, 'EEconSim': EEconSim}

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


ccmap, userOptions = econSimMaps['SEconSim'].createMap()
df = econSimMaps['SEconSim'].getDataframe()
df = df.loc[econSimMaps['SEconSim'].mapDate]

dates = econSimMaps['SEconSim'].dates

layout = html.Div(
    [
        dbc.Row(    # allows user to choose country with a dropdown menu
            dcc.Dropdown(options=["Italy", "United Kingdom", "United States of America", "France", "Japan", "Germany", "Canada"],
                        id='country-choice',
                        value='Italy',
                        clearable=False
            )
        ),

        dbc.Row(      # displays optimum plan (according to chosen country)
            dcc.Textarea(id='optPlanText', value= '', disabled=True, style= {'textAlign':'center', 'height': 260})
        ),
```

```
        # dropdown option between social and environment
        dbc.Row(
            dcc.Dropdown(options=['Environment', 'Social'],
                         id='map-choiceO',
                         value='Social',
                         clearable=False
            )
        ),

        html.Hr(),
        # radio button selection of dates
        dbc.Row(
            dcc.RadioItems(options=dates,
                         id='date-choiceO',
                         value='2019 QR:0',
                         inline=True,
                         labelStyle={'padding':'10px'}
            )
        ),
        # warning may be displayed according to user choices, initially empty
        html.Div(id='warningO', style= {'textAlign':'center'}),

        # radio button selection between indexed and raw data
        dbc.Row(
            [
                dbc.Col(
                    [
                        # checklist used for selection of metrics
                        dcc.Checklist(options=econSimMaps['SEconSim'].getMetrics(),
                                      id='metric-choiceO')
                    ], xs=4, sm=3, md=2, lg=2, xl=2, xxl=2        # ensures site is divided in a sensible way according to size
                ),

                dbc.Col(
                    [
                        # displays map according to user selections
                        dcc.Graph(id='optPlanMap',
                                  figure= ccmap)
                    ], xs=8, sm=9, md=10, lg=10, xl=10, xxl=10
                )
            ]
        ),

        html.Hr(),

    ]
)
# changes the metrics displayed according to the choice of map
@callback(
    Output(component_id='metric-choiceO', component_property='options'),
    Input(component_id='map-choiceO', component_property='value')
)
def setMetricChoices(mapChoice):
    if mapChoice == 'Environment':
        return econSimMaps['EEconSim'].getMetrics()
```

```
    elif mapChoice == 'Social':
        return econSimMaps['SEconSim'].getMetrics()
    else:
        return []

# sets the initial value of the metric choices (eg: life_expectancy for Social maps)
@callback(
    Output(component_id='metric-choiceO', component_property='value'),
    Input(component_id='metric-choiceO', component_property='options')
)
def setInitialValue(metricOptions):
    value = metricOptions[0]
    return [value]

# sets the warnings according to user selection (missing data or no metric selected warnings)
@callback(
    Output(component_id='warningO', component_property='children'),
    Input(component_id='metric-choiceO', component_property='value')
)
def changeWaring(metricChoice):
    if len(metricChoice) > 0:
        for metric in metricChoice:
            if metric in
['income_tax','hours_to_file_taxes','governement_effectiveness','education_pct_government_expenditure','interest_rates','trade_transport_quality','political_stability','fertility_rate','literac
y_rate']:
                newWarning = 'At least one of the metrics selected is missing some data.'
                break
            else:
                newWarning = ''
        return newWarning
    else:
        return 'At least one metric must be selected.'




# displays the correct optimum plan according to country selected
@callback(
    Output(component_id='optPlanText', component_property='value'),
    Input(component_id='country-choice', component_property='value')
)
def changeWaring(countryChoice):
    optPlanText = econSimMaps['SEconSim'].getOptPlanText(countryChoice)
    return optPlanText

# main callback to change the map according to user choices

@callback(
    Output(component_id='optPlanMap', component_property='figure'),
    [Input(component_id='metric-choiceO', component_property='value'),
     Input(component_id='date-choiceO', component_property='value'),
     Input(component_id='map-choiceO', component_property='value')]
)
def update_graph(metricChoice, dateChoice, mapChoice):
    patchedFig = Patch()

    df = econSimMaps['SEconSim'].getDataframe()
```

```
df = df.loc[dateChoice]

dfRules = econSimMaps['SEconSim'].getRules()



countries = ["Italy", "United Kingdom", "United States of America", "France", "Japan", "Germany", "Canada"]
totalRules = econSimMaps['SEconSim'].getOptEmpty()
optPlan =  econSimMaps['SEconSim'].getOptPlan()
for country in countries:                           # finds the total rule changes according to optimum plan
    for policy in econSimMaps['SEconSim'].policies:
        sumOfRules = dfRules.loc[policy] * optPlan[(country, policy)]
        total = totalRules.add(sumOfRules, fill_value=0)
        totalRules.loc[country] = total



df['date'] = df.index
df = df.reset_index(drop=True)
df = df.set_index(['country', 'date'])        # setting a different axis so the data can be accessed and modified
for country in countries:
    df.loc[(country, dateChoice)] = df.loc[(country, dateChoice)]*(1 + totalRules.loc[(country, dateChoice)]/100)



tempC = []                      # resetting the index to be only 'data'
tempD = []
for i in df.index:
    tempC.append(i[0])
    tempD.append(i[1])
df['date'] = tempD
df['country'] = tempC
df = df.reset_index(drop=True)
df = df.set_index('date')




if mapChoice == 'Environment': # sections the data selected according to map choice, and sets correct colour
    userOptions = econSimMaps['SEconSim'].getMetrics()    # list of metrics is same for raw and indexed
    metrics = df.columns.values.tolist()
    for metric in metrics:
        if metric not in userOptions and metric != 'country':
            df = df.drop(metric, axis=1)
    patchedFig['layout']['coloraxis']['colorscale'] = ((0.0, "rgb(179, 242, 180)"), (1.0, "rgb(38, 128, 13)"))

elif mapChoice == 'Social':
    userOptions = econSimMaps['SEconSim'].getMetrics()
    metrics = df.columns.values.tolist()
    for metric in metrics:
        if metric not in userOptions and metric != 'country':
            df = df.drop(metric, axis=1)
    patchedFig['layout']['coloraxis']['colorscale'] = ((0.0, "rgb(200, 240, 255)"), (1.0,  "rgb(50, 50, 255)"))



if len(metricChoice) == 0:     # if no metrics are selected, no data
    mean = []
elif len(metricChoice) == 1:      # if one metric is selected, the mean is simply the data values
    mean = df[metricChoice].values.tolist()
```

```
        fix = []
        for i in range(len(mean)):        # fix is used to adapt data into list
            fix.append(mean[i][0])
        mean = fix
    elif len(metricChoice) > 1:         # if more than one metric is selected, the mean for each country is calculated
        dataList = df[metricChoice[0]]
        for i in range(len(metricChoice)-1):
            i += 1
            dataList = pd.concat([dataList, df[metricChoice[i]]], axis=1)
        mean = dataList.mean('columns')
        mean = mean.tolist()

    patchedFig['data'][0]['z'] = mean      # sets the map data to the calculated values (according to user selection)

    # adapts the hover values according to user choices (name of metric if 1 selected, average if >1)
    if len(metricChoice) == 0:
        pass
    elif len(metricChoice) == 1:
        patchedFig['layout']['coloraxis']['colorbar']['title']['text'] = metricChoice[0]
        patchedFig['data'][0]['hovertemplate'] = 'country=%{location}<br>'+ metricChoice[0] +'=%{z}<extra></extra>'
    else:
        patchedFig['layout']['coloraxis']['colorbar']['title']['text'] = 'Average'
        patchedFig['data'][0]['hovertemplate'] = 'country=%{location}<br>Average=%{z}<extra></extra>'

        # inside the geojson file, the hovertemplate is the following: 'country=%{location}<br>GDP=%{z}<extra></extra>'
        # to change the metric, I simply need to change the name of the metric (from GDP to whatever 'value' is being chosen)

    return patchedFig
```

'info.py'

```
import pandas as pd
import json
import plotly.express as px
import dash
from dash import html, dcc, callback, Output, Input, Patch, dash_table
import dash_bootstrap_components as dbc


import sys
sys.path.append('/Users/irenepeleteiropaniagua/Documents/shhs/computing/projectPrograms')
from mapClasses import *

# registering this page as the homepage (path='/') to the app registry
dash.register_page(__name__, name="Information")

df = pd.read_csv('Info.csv')

layout = html.Div(
    [
        dbc.Row([
            html.H4('About the software'),
            html.P('With this software I wanted to solve the following problem: there is currently no easy way to understand the wellbeing and economy of a country without simply looking at its GDP (or other factors individually). GDP is known to have many problems and could be seen as an outdated metric. Most modern economists now care more about the
```

```
environmental and social issues rather than simple growth, making sure the population actually benefits from decisions made by governments and firms, and so use metrics relating directly to these issues. For this reason, I want to create a program in which the user can: view and compare countries' social and environmental qualities; see projections, and advice on what should be done to improve them; and use an economic simulator to test out their policies with an 'optimised' plan to compare against. This could allow the public to easily check the current economic picture, and allow policy makers to find the current problems and see possible solutions.')
        ]),

        dbc.Row([
            html.H4('The Data'),
            html.P('All data is imported from Our World in Data and World Bank.')
        ]),

        dbc.Row([
            html.H4('The Process'),
            html.H5('Indexing'),
            html.P('To index the data, the program starts by finding the mean and standard deviation of each metric. It then sets a maximum value as 2 standard deviations above the mean, and a minimum as 2 standard deviations below (if any of the G7 data are above/below these, they are sets as the minimum/maximum). The algorithm then calculates the index value of the G7 countries based on this minimum and maximum, so all data is within 0-1 and can be easily compared and averaged.'),
            html.H5('Projections'),
            html.P('The algorithm creates a linear model based on the 2000-2019 data and uses linear regression to get the projections.'),
            html.H5('Optimum Plan'),
            html.P("The optimum plan is calculated based on an adjusted version of Dijkstra's algorithm to find the 'optimum' route to take with the policies.")
        ]),

        dbc.Row([
            html.H4('The Metrics'),
            html.H6('Below are descriptions of all of the metrics included in the application:'),

            dash_table.DataTable(           # creates a table out of the metric information
                data=df.to_dict('records'),
                columns=[{'id': c, 'name': c} for c in df.columns],
                style_cell={'textAlign': 'left'},
                style_data={
                    'whiteSpace': 'normal',
                    'height': 'auto',
                }
            )
        ])

    ]
)
```

'help.py'

```
import pandas as pd
import json
import plotly.express as px
import dash
from dash import html, dcc, callback, Output, Input, Patch
import dash_bootstrap_components as dbc

import sys
sys.path.append('/Users/irenepeleteiropaniagua/Documents/shhs/computing/projectPrograms')
from mapClasses import *
```

```
# registering this page as the homepage (path='/') to the app registry
dash.register_page(__name__, name="Help")

image_path = 'assets/helpForCurrent.png'


layout = html.Div(
  [
    dbc.Row(
      html.Img(src=image_path)
    )
  ]
)
```

'Current.csv'

Section of 'Current' table.

country	year	life_expectancy	pollution	income_inequality	wealth_inequality	GDP	GDP_per_capita	GDP_PPP
Canada	2000	0.7921673300435590	0.3048174378476880 0	0.4517644538118680	0.3270325505305140	0.1905030278035160	0.7375067884026190	0.20910319721374500
Canada	2001	0.7916942037484110	0.3046225448782810	0.4072580257630570	0.3226715907174860	0.1850639514507 1200	0.7561021909824970	0.21022172961323300
Canada	2002	0.7898139069953370	0.3044501149144870	0.3809531524524140	0.3211181847041170	0.1828369519356 5500	0.7572973491274680	0.21072335280173200
Canada	2003	0.7874481055756610	0.3077061667 2908400	0.3920943043090190	0.3193355633186850	0.1912989805147990	0.7866813196764960	0.21169750204835400
Canada	2004	0.7884171807820300	0.3106923686 13938	0.4172766279465740	0.3261760436006210 0	0.1972043157460060	0.7865694659200660	0.21186168745110000
Canada	2005	0.7868015492256270	0.3082421757641440	0.4138441192361820 0	0.3310762001196790	0.2015960069987230	0.8124114237812440	0.2138918281672080
Canada	2006	0.7925658310875040	0.3039660626 9792700	0.4261553485813030	0.3281824786399150 0	0.2057658343621 1900	0.8092202167941800	0.2159185419721550
Canada	2007	0.7893123072356 79	0.2926395154 9807900	0.4266628244188730 0	0.3247393819177590	0.2122429905000 990	0.7891968365947000	0.2189234603121770
Canada	2008	0.7887299632123560	0.2773258292075770	0.4223137803441720	0.3279187008061800 0	0.2177236834183 6900	0.7795723954270930	0.2223940711688840
Canada	2009	0.7965546712378820	0.2638139349 4095700	0.3956177016439350 0	0.316772684502201	0.2091941265377 6900	0.7777217928381410	0.2255640580827030 0
Canada	2010	0.7965749604592700	0.2485064849928770	0.4094171515169440	0.3188603262926710	0.2224743225656 4200	0.8345863269063500	0.2292406963834460 0
Canada	2011	0.7994413263574150	0.2402466980 81377	0.4274119861778080 0	0.3146479278438430 0	0.2320159389375 3600	0.8356070196767620	0.2336812485008310
Canada	2012	0.8041819840043250	0.2331491179668520	0.4120432410272080	0.315814632310402	0.2303402206709940	0.8510777136347 95	0.2351749722980690 0
Canada	2013	0.8050692091741 74	0.2263783649 2308700	0.437214943161109	0.3113549415552110	0.2275290239499 1600	0.8241761832132720	0.2381399030609930 0
Canada	2014	0.8027144678254300	0.2239990364 3904200	0.4371168636224290 0	0.3151359978719160 0	0.2218385536437 9100	0.7985968840039150	0.2388995539674280 0
Canada	2015	0.8042462198776930	0.2204255957 1244000	0.4317171967008200 0	0.3199125149739210 0	0.2051031576813 8900	0.7753617131966130	0.2344267791694300 0
Canada	2016	0.8034384547480850	0.2245202413 4090200	0.3915206484732140 0	0.3059737720463870 0	0.2015942508703 6000	0.7555887179775810	0.2378791349008960
Canada	2017	0.8019001743853 66	0.2286937401 7248000	0.453095657104202	0.3171463318546230 0	0.2055446054583 0600	0.7664169911961070	0.2398499453072720 0

Canada	2018	0.8013893015327600	0.2286460522915310	0.36480503815921000	0.32166405354422500	0.20644855118085700	0.7536723227833670	0.24144249139727900
Canada	2019	0.8076268230955080	0.2237028132299160	0.36480503815921000	0.31940465380396700	0.20352345850103500	0.7565830611935700	0.24107482782819800
France	2000	0.7869289240185330	0.5072085060371240	0.26117292861546200	0.30955735061304700	0.24337280639505900	0.7048311207973270	0.26734688631378800
France	2001	0.7838418045700760	0.5114808266458540	0.24704402227586700	0.28723626316852000	0.23794239066380700	0.7309508617126500	0.27157980589233000
France	2002	0.7845282748754530	0.5116342239603170	0.19546458081304800	0.25210452999059900	0.24236627337388800	0.7578789458591440	0.27349089152518300
France	2003	0.7794294764000940	0.5091998242534960	0.21793383201437800	0.23617209408347100	0.26396897251868400	0.8074875036714510	0.26685730259955400
France	2004	0.7964794118069180	0.4954222056284180	0.23119618600779400	0.22201100027642200	0.27561096835315200	0.8094388309289440	0.26414912931106200
France	2005	0.7922576389939650	0.4956248902219700	0.19612975852961900	0.20398747987947700	0.27048434473743200	0.7913926915657130	0.26393848576804800

‘CurrentRaw.csv’

Section of ‘CurrentRaw’ table.

country	year	life_expectancy	pollution	income_inequality	wealth_inequality	GDP	GDP_per_capita	GDP_PPP
Canada	2000	79.2	53.965614	0.52538216	0.734283	744773415931.587	24271.0020563821	900996986801.064
Canada	2001	79.5	52.703915	0.516006	0.7332109	738962729044.525	23821.4455867378	937786776176.929
Canada	2002	79.6	51.677162	0.51672053	0.73279476	760649334098.005	24255.3385818322	971003788233.131
Canada	2003	79.7	51.423298	0.5182721	0.732944	895540646634.787	28300.4630963791	1023682523002.92
Canada	2004	80.0	51.145504	0.5228989	0.73451495	1026690238278.25	32143.6814078562	1083609294356.44
Canada	2005	80.1	49.748238	0.52874994	0.7357195	1173108598778.68	36382.5079164537	1171326187910.13
Canada	2006	80.5	48.32063	0.5332416	0.73701334	1319264809590.97	40504.0607253203	1241594039979.59
Canada	2007	80.5	45.41875	0.5346661	0.7375096	1468820407783.26	44659.8951408034	1301593351698.17
Canada	2008	80.7	41.99527	0.5274539	0.73656046	1552989690721.65	46710.5055759013	1342395839240.43
Canada	2009	81.1	38.50683	0.51634884	0.73227	1374625142157.29	40876.3101540295	1307000908301.48
Canada	2010	81.3	34.68942	0.52287894	0.73418975	1617343367486.26	47562.0834253057	1363577286343.32
Canada	2011	81.5	32.429077	0.5246289	0.734845	1793326630174.52	52223.696112356	1430806739876.64
Canada	2012	81.7	30.598839	0.5204705	0.7337875	1828366481521.6	52669.0899632316	1468097961846.59
Canada	2013	81.8	28.792646	0.5290777	0.7352768	1846597421834.98	52635.1749580433	1554125000000.0
Canada	2014	81.9	27.592302	0.5307158	0.7362165	1805749878439.94	50955.9983232404	1621395561291.92
Canada	2015	81.9	26.251558	0.5294287	0.736898	1556508816217.14	43596.1355365546	1594851775821.65
Canada	2016	82.0	25.941084	0.5124083	0.73321056	1527994741907.43	42315.6037056806	1678092366067.54
Canada	2017	82.0	25.633984	0.53608376	0.73561496	1649265644244.09	45129.4292980922	1765762548007.76
Canada	2018	82.1	25.637527	0.52486557	0.73588467	1725297938435.76	46547.7951820036	1852985933143.48
Canada	2019	82.4	24.75273	0.52486557	0.7355051	1743725183672.52	46374.1527517191	1873759736036.69
France	2000	79.0	99.806175	0.4502523	0.72919446	1365639660792.16	22416.425417915	1589551596985.56
France	2001	79.2	99.207664	0.4499299	0.72305197	1377657339291.34	22449.3396931175	1687042717869.47

‘Projected.csv’

Section of the ‘Projected’ table.

country	date	life_expectancy	fertility_rate	GDP	GDP_per_capita	GDP_PPP

United Kingdom	2019 QR:0	0.7808907192741230	0.3343349292573590	0.26347357031673200	0.7145444233565850	0.2963225615060600
United Kingdom	2019 QR:1	0.7812500881702860	0.3349383741246010	0.2631505773507060	0.7124718459454800	0.2967511731680960
United Kingdom	2019 QR:2	0.7816094570664490	0.3355418189918430	0.262827584384679	0.7103992685343740	0.2971797848301320
United Kingdom	2019 QR:3	0.7819688259626130	0.33614526385908500	0.26250459141865300	0.708326691123272	0.2976083964921680
United Kingdom	2020 QR:0	0.7823281948587760	0.33674870872632700	0.2621815984526260	0.7062541137121660	0.2980370081542040
United Kingdom	2020 QR:1	0.7826875637549390	0.33735215359356900	0.2618586054865990	0.7041815363010610	0.2984656198162400
United Kingdom	2020 QR:2	0.7830469326511020	0.3379555984608120	0.26153561252057200	0.702108958889955	0.2988942314782760
United Kingdom	2020 QR:3	0.7834063015472660	0.3385590433280530	0.26121261955454500	0.7000363814788530	0.2993228431403120
United Kingdom	2021 QR:0	0.7837656704434290	0.3391624881952960	0.260889626588519	0.6979638040677470	0.2997514548023480
United Kingdom	2025	0.7895155727820410	0.34881760607117000	0.2557217391320910	0.6648025654900740	0.3066092413949240
United Kingdom	2030	0.7967029507053050	0.3608865034160120	0.24926187981155700	0.6233510172679820	0.31518147463564400
United Kingdom	2035	0.8038903286285710	0.372955400760854	0.24280202049102300	0.5818994690458940	0.3237537078763640
United Kingdom	2040	0.8110777065518350	0.38502429810569600	0.236342161170489	0.5404479208238020	0.3323259411170840
United Kingdom	2045	0.8182650844751	0.39709319545053700	0.22988230184995500	0.4989963726017100	0.34089817435780300
United Kingdom	2050	0.8254524623983650	0.40916209279538000	0.22342244252942000	0.45754482437962100	0.34947040759852400
United States	2019 QR:0	0.699472674770369	0.3258729975995720	1.0	0.8546810714162650	1.0
United States	2019 QR:1	0.6990732673390490	0.3256192719885230	1.0	0.8537308082452480	1.0
United States	2019 QR:2	0.6986738599077290	0.3253655463774740	1.0	0.8527805450742320	1.0
United States	2019 QR:3	0.6982744524764090	0.32511182076642400	1.0	0.851830281903216	1.0
United States	2020 QR:0	0.6978750450450900	0.32485809515537500	1.0	0.850880018732199	1.0
United States	2020 QR:1	0.6974756376137700	0.32460436954432500	1.0	0.8499297555611830	1.0
United States	2020 QR:2	0.69707623018245	0.32435064393327600	1.0	0.8489794923901670	1.0
United States	2020 QR:3	0.69667682275113	0.32409691832222700	1.0	0.8480292292191510	1.0
United States	2021 QR:0	0.6962774153198100	0.3238431927111770	1.0	0.8470789660481340	1.0
United States	2025	0.6898868964186920	0.31978358293438700	1.0	0.8318747553118730	1.0
United States	2030	0.6818987477922940	0.31470907071339900	1.0	0.8128694918915450	1.0
United States	2035	0.6739105991658960	0.3096345584924110	1.0	0.7938642284712180	1.0
United States	2040	0.6659224505394990	0.30456004627142400	1.0	0.7748589650508910	1.0

‘Rules.csv’

Section of ‘Rules’ table.

policy	date	life_expectancy	pollution	income_inequality	wealth_inequality	GDP	GDP_per_capita	GDP_PPP
Income_Tax	2019 QR:0	0	0	0	0	0	0	0
Income_Tax	2019 QR:1	0	0	0	0	0	0	0
Income_Tax	2019 QR:2	0	0	0	0	-0.1	-0.1	-0.1
Income_Tax	2019 QR:3	0	0	0	0	-0.3	-0.3	-0.3
Income_Tax	2020 QR:0	0	0	0.5	0	-0.1	-0.1	-0.1
Income_Tax	2020 QR:1	0	0	0	0	-0.1	-0.1	-0.1
Income_Tax	2020 QR:2	0	0	0	0	-0.1	-0.1	-0.1
Income_Tax	2020 QR:3	0	0	0	0	-0.1	-0.1	-0.1
Income_Tax	2021 QR:0	0	0	0.3	0	-0.1	-0.1	-0.1
Income_Tax	2025	0	0	0.1	0	-2	-2	-2
Income_Tax	2035	0	0	1	0	-5	-5	-5
Income_Tax	2050	0	0	7	0	-7	-7	-7
Demerit_Goods_Tax	2019 QR:0	0	0	0	0	0	0	0
Demerit_Goods_Tax	2019 QR:1	0	0	0	0	0	0	0
Demerit_Goods_Tax	2019 QR:2	0	0	0	0	0	0	0
Demerit_Goods_Tax	2019 QR:3	0	0	0	0	0	0	0
Demerit_Goods_Tax	2020 QR:0	0	0	0	0	0	0	0
Demerit_Goods_Tax	2020 QR:1	0	0	0	0	0	0	0
Demerit_Goods_Tax	2020 QR:2	0	0	0	0	0	0	0
Demerit_Goods_Tax	2020 QR:3	0	0	0	0	0	0	0
Demerit_Goods_Tax	2021 QR:0	0	0	0	0	0	0	0
Demerit_Goods_Tax	2025	1	0	0	0	0	0	0
Demerit_Goods_Tax	2035	2	0	0	0	0	0	0
Demerit_Goods_Tax	2050	0	0	0	0	0	0	0
Education_Spending	2019 QR:0	0	0	0	0	0	0	0
Education_Spending	2019 QR:1	0	0	0	0	0	0	0
Education_Spending	2019 QR:2	0	0	0	0	0	0	0
Education_Spending	2019 QR:3	0	0	0	0	0	0	0

Education_Spending	2020 QR:0	0	0	0	0	0	0	0
Education_Spending	2020 QR:1	0	0	0	0	0	0	0
Education_Spending	2020 QR:2	0	0	0	0	0	0	0
Education_Spending	2020 QR:3	0	0	0	0	0	0	0
Education_Spending	2021 QR:0	0	0	0	0	0	0	0
Education_Spending	2025	0	0	0.1	0.1	0.3	0.7	0.3
Education_Spending	2035	0	0	0.5	0.5	5	7	5
Education_Spending	2050	0	0	1	1	7	9	7