

Predicting HDI with Multilayer Perceptrons

Luke Williams, Candidate 2276
The Cherwell School, Centre 62309
OCR A-Level Computer Science NEA — H446/03

June 2025

Contents

Contents	1
1 Analysis	5
1.1 Introduction	5
1.1.1 The Problem	5
1.1.2 Amenable to Computational Approach	6
1.1.3 Computational Methods	6
1.2 Stakeholders	8
1.3 Research	9
1.3.1 Existing Solutions	9
1.3.2 Stakeholder Interview & Feedback	14
1.4 Essential Features	18
1.5 Limitations	19
1.6 Requirements	20
1.6.1 Hardware Requirements	20
1.6.2 Software Requirements	20
1.7 Success Criteria	21
2 Design of the 1st Prototype	23
2.1 Structure Diagram	23
2.2 Data Collection & Processing	24
2.2.1 Full List of Factors to Include	24
2.2.2 Finding the Average Distance between 2 objects $x, y, A(x, y)$	25
2.2.3 Finding the Density of some object $x, D(x)$	32
2.2.4 Overall Algorithm for Data Collection & Processing	39
2.2.5 List of Key Variables	40
2.3 Multilayer Perceptron	43
2.3.1 Structure	43
2.3.2 Feedforward Algorithm & Prediction	45
2.3.3 Backpropagation Algorithm & Training	51
2.3.4 Stochastic Gradient Descent	59
2.3.5 Class Diagram	61
2.4 User Interface	62
2.4.1 Interface Sketch	62
2.4.2 Input Validation	64

2.4.3	Usability Features	65
2.4.4	Finding an Optimal Place for a New Building	66
2.4.5	Overall Algorithm for User Predicting HDI	67
2.4.6	List of Key Variables	69
2.5	Data	71
2.5.1	Data Structures	71
2.5.2	Testing Data for Development	72
3	Developing the Coded Solution of the 1st Prototype	75
3.1	Milestone 1 – Data Collection & Processing	75
3.1.1	Success Criteria	75
3.1.2	Part 1 – Collecting HDI Data	76
3.1.3	Part 2 – Collecting OSM Data for a Given Region	80
3.1.4	Part 3 – Implementing $A(x, y)$	84
3.1.5	Part 4 – Implementing $D(x)$	88
3.1.6	Part 5 – Iterating over each Subnational Region	96
3.1.7	Review	115
3.2	Milestone 2 – Multilayer Perceptron	117
3.2.1	Success Criteria	117
3.2.2	Part 6 – Implementing the Neural Network Structure	117
3.2.3	Part 7 – Implementing the Feedforward Algorithm & Prediction	125
3.2.4	Part 8 – Implementing the Backpropogation Algorithm & Training	126
3.2.5	Part 9 – Testing the Neural Network	131
3.2.6	Part 10 – Training the Neural Network	140
3.2.7	Review	147
3.3	Milestone 3 – Initial User Interface	148
3.3.1	Success Criteria	148
3.3.2	Part 11 – Drawing on the Map	149
3.3.3	Part 12 – Predicting the HDI	156
3.3.4	Part 13 – Suggesting Improvements	171
3.3.5	Review	183
3.4	Feedback from Stakeholders	184
4	Design of the 2nd Prototype	186
4.1	Structure Diagram	186
4.2	Authentication System	187
4.2.1	Storing Data & Encryption	187
4.2.2	Account Management	188
4.2.3	List of Key Variables	191
4.3	User Interface	192
4.3.1	Interface Sketch – A Tabbed Interface	192
4.3.2	Input Validation	198
4.3.3	Usabilty Features	199
4.3.4	Analysis of how much Each Factor affects the Others	201
4.3.5	Comparison to Similar Regions	202

4.3.6	Improved Algorithm for Suggestions	203
4.3.7	List of Key Variables	206
4.4	Data	208
4.4.1	Storing Data in the MongoDB	208
4.4.2	Testing Data for Development	208
4.4.3	Testing Data for Post-Development	210
5	Developing the Coded Solution of the 2nd Prototype	217
5.1	Milestone 4 – Authentication System	217
5.1.1	Success Criteria	217
5.1.2	Part 14 – Authentication Frontend	218
5.1.3	Part 15 – Sign Up Backend	224
5.1.4	Part 16 – Log In Backend	236
5.1.5	Review	239
5.2	Milestone 5 – Final User Interface	240
5.2.1	Success Criteria	240
5.2.2	Part 17 – Comparing a Region to Training Data	241
5.2.3	Part 18 – Analysis of how much Each Factor affects the Others	250
5.2.4	Part 19 – Finding Similar Regions	257
5.2.5	Part 20 – Saving Regions	259
5.2.6	Part 21 – Improved Suggestions	282
5.2.7	Part 22 – Finishing Touches	292
5.2.8	Review	319
6	Evaluation	321
6.1	Testing to Inform Evaluation	321
6.1.1	Testing for Function & Robustness	321
6.1.2	Testing for Usability	326
6.2	Success Criteria	329
6.2.1	Cross Referencing with Test Data	329
6.2.2	Reasoning Behind Outcome of Success Criteria	331
6.2.3	Addressing Partially Met Success Criteria in Further Development	331
6.3	Usability Features	332
6.3.1	Comparison to Sketches & Effective Usability Features	333
6.3.2	Addressing Poor Usability Features in Further Development	342
6.4	Maintenance Issues	343
6.4.1	Maintainability	343
6.4.2	Documentation	343
6.4.3	Code Style Guidelines	343
6.5	Limitations of the Solution	343
6.5.1	Data Limitations	343
6.5.2	Usability Challenges	344
6.5.3	Scalability & Technical Constraints	344
6.6	Conclusion	344

A Final Code	345
A.1 Data Processing	345
A.1.1 data_processing / calc_factors.py	345
A.1.2 data_processing / compile_data.py	349
A.1.3 data_processing / get_osm_data.py	350
A.1.4 data_processing / pmcc.py	351
A.1.5 data_processing / process_hdi.py	352
A.1.6 data_processing / process_shapefile.py	353
A.2 Training the Neural Network	353
A.2.1 network_training / load_digits_data.py	353
A.2.2 network_training / load_hdi_data.py	354
A.2.3 network_training / test_digits.py	355
A.2.4 network_training / test_hdi.py	356
A.2.5 network_training / train_digits.py	357
A.2.6 network_training / train_hdi.py	357
A.3 The Main App	358
A.3.1 app.py	358
A.3.2 calc_factors.py	377
A.3.3 neural_network.py	380
Index	384
List of Figures	384
List of Code Snippets	389
List of Tables	394
Bibliography	399

1. Analysis

1.1 Introduction

1.1.1 The Problem

For my project, I have decided to train a neural network to predict the Human Development Index (HDI) of a given area. HDI is a number between 0 and 1, which represents how developed a certain country is. It is officially calculated each year by the United Nations, based on factors such as mean years of schooling or life expectancy, by finding the geometric mean of the Life Expectancy Index, the Education Index and the Income Index, as shown in the formula

$$\begin{aligned} \text{HDI} &= \sqrt[3]{\text{LEI} \times \text{EI} \times \text{II}} \\ &\approx \sqrt[3]{\frac{L - 20}{85 - 20} \times \frac{\frac{S_M}{15} + \frac{S_E}{18}}{2} \times \frac{\ln G - \ln 100}{\ln 75000 - \ln 100}} \\ &= \sqrt[3]{\frac{(L - 20)(18S_M + 15S_E)(\ln G - \ln 100)}{35100 \ln 750}} \end{aligned} \quad (1.1)$$

where L is the life expectancy at birth, S_M is the mean years of schooling, S_E is the expected years of schooling and G is the gross national income per capita.

However, these factors are very difficult to determine, for many reasons. For example, life expectancy at birth is defined as how long a newborn is expected to live if current death rates do not change. However, the actual death rate for any particular birth cohort cannot be known in advance, and therefore must be estimated on the basis of surveys and historical data, which makes it unreliable. Gross national income per capita also has limitations in its calculation, its accuracy depending on the availability and reliability of income and population data.

Motivated by this, I will instead be investigating how more geographical factors (which can be found by only looking at a map of the area) can affect HDI. The data for which these factors are based on will be pulled from OpenStreetMaps (OSM). OpenStreetMaps is a large data set that contains ~ 1.9 terabytes of information about the Earth. For example, it contains the location of every (registered) park bench, cafe, parking space, ATM machine, post office, school and many others. An API can be used to access this data called Overpass Turbo, in which you can specify an area and what you are looking for, and it will return a list of latitude and longitude coordinates correct to 7 decimal places, along with any extra information.

Therefore, the problem I am solving is that of trying to predict the HDI of a given area based only on physical features of that area (which may be natural or man-made). This is useful because once the HDI has been predicted, suggestions can be made to improve it, which will involve building tangible objects (such as schools), instead of vaguely suggesting “increase the mean years of schooling”.

1.1.2 Amenable to Computational Approach

This problem is amenable to a computational approach, as it involves inputs (various geographical factors), outputs (a predicted HDI) and calculations (calculating the predicted HDI from the set of geographical factors through a neural network, which involves many matrix multiplications).

This problem can also be solved in a finite, reasonable number of steps. Most algorithms that solve problems in a finite but unreasonable number of steps often involve a brute-force approach of trying all possibilities for a given problem, which I do not intend to implement.

This problem will also involve a vast amount of data, gathered by OpenStreetMaps, in the form of ordered latitude and longitude pairs. The storage requirements for this data however will not be so large, as I will only be dealing with (relatively) small amounts of data at any given time.

1.1.3 Computational Methods

Abstraction

I intend to use abstraction in my project to remove any unnecessary detail from it, in order to make problem solving easier. For example, I will abstract away specific details for what my neural network is going to do, to focus on writing the infrastructure for a general neural network, which can be tuned to do anything (including what I want it to do).

I will also be using abstraction to make models when solving particular problems, in order to make them simpler. For example, if the problem requires some geometry in the Earth, it helps to abstract the Earth into a perfectly smooth sphere, instead of considering its rough topography, or its slight eccentricity.

The data that will be fetched from OpenStreetMaps will also be abstracted, as all I will be considering is the object’s latitude and longitude coordinates, as well as what type of object it is. There is no need to consider any other attribute about the object, such as its size, its material, or its history.

I will also use abstraction to remove unnecessary details from the map the users will be selecting their areas from. Details which are given by satellite imagery are not required for the user to be able to recognise and select the area they want to analyse. Therefore I will only be using a digital map for users to select from.

Decomposition

Decomposition is the process of breaking down a problem into smaller subproblems, each of which are easier to solve. I will use decomposition to break down my project into 3 distinct

and independent modules: the collection of data, the neural network, and the user interface. These modules have almost nothing to do with each other, and can be considered completely separately. The only times they interact are when the neural network reads the gathered data, and when the user interface queries the neural network.

Furthermore, the purpose of hidden layers in a neural network (further described later in section 2.3.1) is inherently to represent decomposition. Calculating the HDI from a set of seemingly unrelated geographical factors is very difficult, and so the hidden layers attempt to calculate nebulous intermediate values in between the geographical factors and the HDI.

In other words, making use of a neural network with n hidden layers decomposes the problem of calculating HDI from a set of geographical factors into

- Calculating nebulous intermediate value 1 from set of geographical factors
- Calculating nebulous intermediate value 2 from nebulous intermediate value 1
- ⋮
- Calculating nebulous intermediate value n from nebulous intermediate value $n - 1$
- Calculating HDI from nebulous intermediate value n

with each nebulous intermediate value representing something closer and closer to HDI with each one. What the nebulous intermediate values actually represent are too abstract for us humans to understand.

Divide & Conquer

Once the HDI has been successfully predicted, I would like to make suggestions on how to improve it. The HDI will then be predicted again based on the improvements, and so to determine the best one, the different improvements must be sorted. An algorithm such as quicksort is suitable, as it makes use of the divide & conquer method. This is because it considers a problem smaller in scope but of the same format each iteration, which makes for an efficient and intuitive algorithm.

Data Mining

Data mining is a technique used to identify patterns and/or anomalies in large data sets (big data), by means of statistics or machine learning. This most certainly fits the description of my project, as the large data set I will be using is OpenStreetMaps (~ 1.9 TB), and I hope to recognise patterns within the locations of geographical objects, and how that may correlate with HDI. I plan to do this by making use of a neural network, which falls under machine learning.

Heuristics

It is impossible to *actually* calculate HDI from a set of geographical factors, as how you *actually* calculate it is with more social factors (as shown in equation 1.1). Therefore a

heuristic “good enough” approach with neural networks must be used. I don’t strictly plan on using heuristics, however in order to train a neural network, you must have a heuristic sense of how much each particular neuron will affect the output of the system (further described later in section 2.3.3).

Visualisation

For the user, I will make use of visualisation to show them various statistics. For example, I may implement a bar graph showing which suggestions will increase the HDI the most, or a map to show where new buildings should be placed in order to have the biggest effect.

For me, I will make use of visualisation in order to make solving problems easier. I plan on using diagrams to depict the various processes going on inside a neural network, and to further my understanding. Diagrams will also help in deriving different geometric equations, which will be useful throughout this project.

1.2 Stakeholders

My stakeholders described themselves as:

1. **Umanga Shrestha** – An A-Level Student interested in moving out of the UK
2. **Joel Tew** – An A-Level Student interested in Simulating Countries
3. **Victor Faustino** – A Local Council Member
4. **Kiefer Keyworth** – An A-Level student who has not studied Geography since Year 9

When asked how they would make use of my proposed solution, and why it is appropriate for them, they responded with:

No.	How they would make use of my Proposed Solution	Why it is appropriate to their needs
1	He will use it to understand the HDI within different areas. He will then use this information to decide where to move.	It is appropriate because he is very interested in HDI in different areas of the world and would love to provide help to increase it.
2	He will use it to estimate the HDI of a hypothetical country, Flabbistan (which the UN does not recognise). He believes that this will be the most developed “country” in the world.	It is appropriate because it will make his simulations involving Flabbistan more realistic, by providing an insight into how it may develop in the future.

3	As a local council member, he would love to use it to suggest changes to his local area that would raise the HDI.	It is appropriate because it would suggest changes to increase HDI while analysing the importance of each of the factors. This would help him to choose between different changes based on cost.
4	He will use the solution to gain a better understanding of what HDI is. He will be able to see the relevance of many factors (in particular, the number of park benches) to the development of a country.	It is appropriate because he is interested in geography but did not take it at GCSE or A-Level and is therefore in need of a program related to it. This solution will be very useful in showing how various factors correlates with development.

Table 1.1: Stakeholders

Each stakeholder's more specific needs and ideas will be discussed in the Stakeholder Interview (Section 1.3.2).

1.3 Research

1.3.1 Existing Solutions

sandeeponduru's human-development-index [1]

This solution makes use of a Jupyter Notebook (.ipynb) to predict the HDI (Figure 1.1), which allows for small python code snippets to be run independently, along with shell commands. They have used this to train the model as they go, and then use it for predictions and data visualisations in the same workflow.

```
In [2]:
import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

if os.environ.get('RUNTIME_ENV_LOCATION_TYPE') == 'external':
    endpoint_37c0ca8aa1a07d9f5e3b8c23077e5e = 'https://s3.us.cloud-object-storage.appdomain.cloud'
else:
    endpoint_37c0ca8aa1a07d9f5e3b8c23077e5e = 'https://s3.private.us.cloud-object-storage.appdomain.cloud'

client_37c0ca8aa1a07d9f5e3b8c23077e5e = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='JLla8t_xF04Gv1vqnoS8TDXsJr1wkvXbaab5yDH',
    ibm_auth_endpoint='https://iam.cloud.ibm.com/oidc/token',
    config=Config(signature_version='oauth'),
    endpoint_url=endpoint_37c0ca8aa1a07d9f5e3b8c23077e5e)

body = client_37c0ca8aa1a07d9f5e3b8c23077e5e.get_object(bucket='custommodeldeployment-donotdelete-pr-sdsnfdnm')

if not hasattr(body, '__iter__'): body.__iter__ = types.MethodType( __iter__, body )

Development = pd.read_csv(body)
Development.head()

Out[2]:
   Unnamed: 0   id  Country  HDI_Rank  HDI  expectancy  Life_mean_years_of_schooling  Gross_national_income_per_capita  GNI_per_capita_rank  Change_in_HDI_rank  ...  Coefficient_of_human_inequality  Inequality_in_life_expectancy_2010  Inequity_adj
0          0     1   Norway      1.0  0.949        81.7       12.7    67614.0         5.0      0.0  ...           5.4            3.3
1          1     2  Australia     2.0  0.939        82.5       13.2    42822.0         19.0      1.0  ...           8.0            4.3
2          2     3  Switzerland    2.0  0.939        83.1       13.4    56364.0         7.0      0.0  ...           8.4            3.8
3          3     4    Germany      4.0  0.926        81.1       13.2    45000.0         13.0      0.0  ...           7.0            3.7
4          4     5   Denmark      5.0  0.925        80.4       12.7    44519.0         13.0      2.0  ...           7.0            3.8

```

5 rows × 82 columns

Figure 1.1: Jupyter Notebook Format (Not Suitable)

However, for my project, my model needs to be pre-trained and then accessed via a GUI, so a Jupyter Notebook is not suitable.

This solution also has a scatter plot showing the HDI of various countries (Figure 1.2). It includes a select few countries, which have an HDI ≥ 0.90 , which are each represented by a coloured dot. (The specific colour seems to carry no meaning)

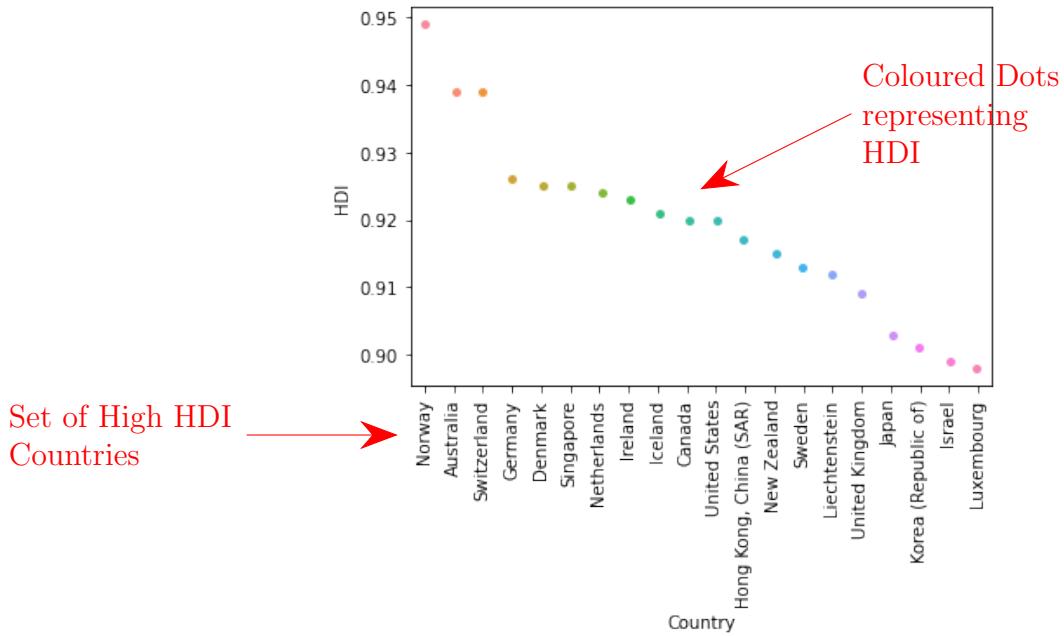


Figure 1.2: HDI Scatter Plot By Country

I believe that a bar graph would be more appropriate, as the x -axis is a discrete variable (which country) and the y -axis is continuous (HDI). This would be good to implement in my project as the HDI in the user's area could be compared to the HDI of other countries or

areas on a similar graph to this, so that if they don't fully understand the HDI scale, they can easily compare with what they know.

This solution also includes the use of a heatmap, describing the relationship between each factor they consider (Figure 1.3). Brighter cells represent strong positive correlation between the factors, and darker cells represent strong negative correlation.

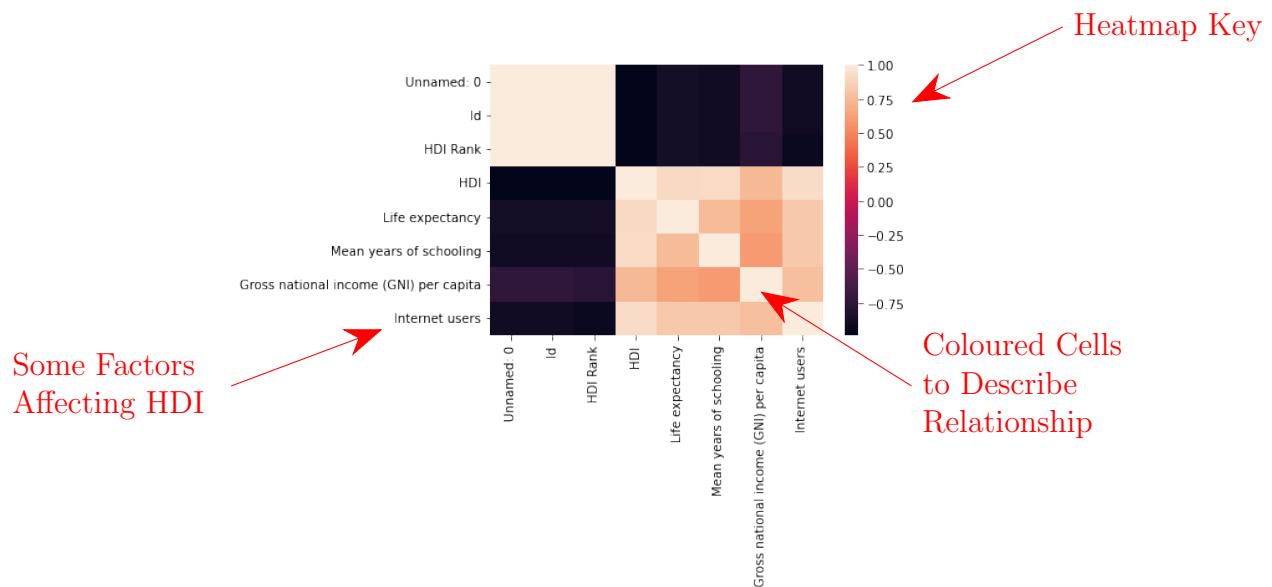


Figure 1.3: HDI Factor Heatmap

I think it would be useful to implement this into my solution, as it would inform which changes the user needs to make to their area, in order to not only increase the general HDI, but also perhaps will improve some other things, which will even further boost the HDI.

Both of the suggested features here (Comparison of predicted HDI to other countries & Analysis of how much each factor affects the others) will be discussed in the Stakeholder Interview (Section 1.3.2).

julieanneco's predictingHDI [2]

This solution also has a similar heatmap feature, but here each cell has a pie chart showing the correlation coefficient (Figure 1.4). The proportion of the pie that is filled carries the same meaning as the colour of the pie, with more filled in pies indicating stronger correlation. This chart also omits half of the possible combinations, as they have already been accounted for (for example, correlation between birth rate and life expectancy will be equal to the correlation between life expectancy and birth rate).

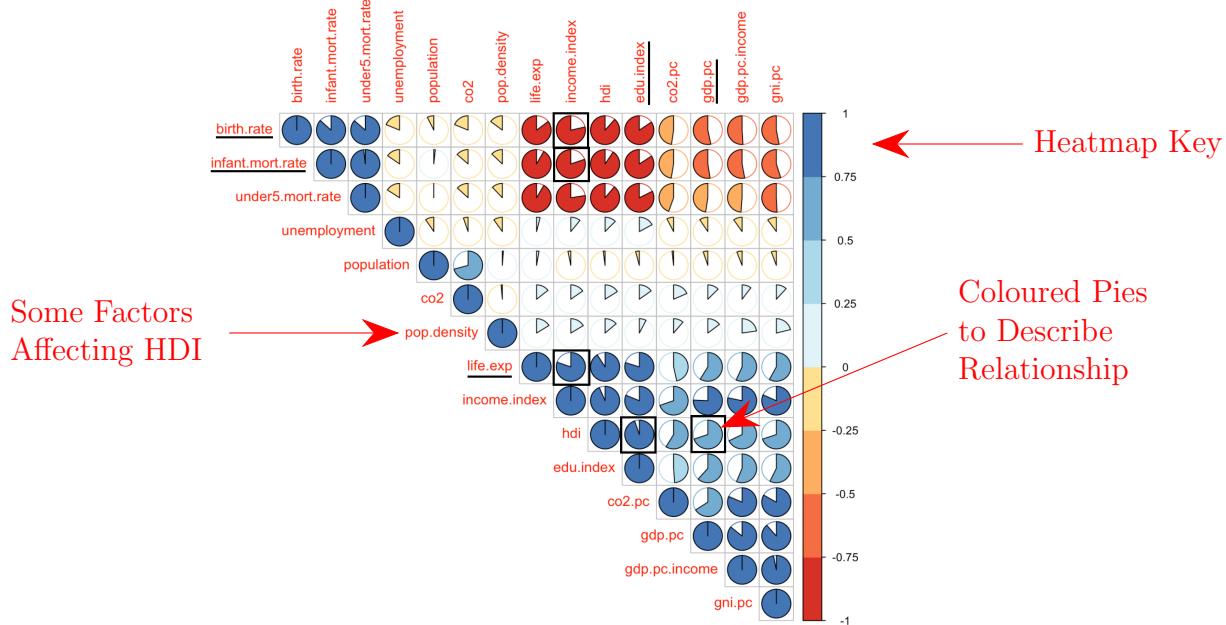


Figure 1.4: HDI Factor Pie Chart Heatmap

While omitting half of the combinations of factors is efficient, I believe the use of pie charts in each cell only makes this chart unnecessarily more complex and harder for someone to understand. If I were to add a feature similar to this, I would only include half of it (as shown in this solution), and fill the cells with solid colour (as shown in the previous solution).

The factors that this solution take into account are more social, economic and environmental factors (Figure 1.5). They have also taken into account multiple time periods (years).

Social, Economic & Environmental Factors

```
'data.frame': 6293 obs. of 25 variables:
 $ iso2c   : chr "AF" "AF" "AF" ...
 $ country  : chr "Afghanistan" "Afghanistan" "Afghanistan" ...
 $ population: num 37172386 36296400 35383128 34413603 33370794 ...
 $ year     : int 2018 2017 2016 2015 2014 2013 2012 2011 2010 2009 ...
 $ pop.density: num 56.9 55.6 54.2 52.7 51.1 ...
 $ gdp.pc    : num 568 572 571 574 584 ...
 $ gdp.pc.income: num 2242 2203 2129 2087 2069 ...
 $ co2       : num NA NA 8672 9035 8467 ...
 $ co2.pc    : num NA NA 0.245 0.263 0.254 ...
 $ birth.rate: num 32.5 33.2 34.3 34.8 35.7 ...
 $ life.exp  : num 64.5 64.1 63.8 63.4 63 ...
 $ infant.mort.rate: num 48 49.6 51.3 53.2 55.2 57.2 59.5 61.7 64.1 66.5 ...
 $ unemployment: num 11.1 11.2 11.3 11.4 11.4 ...
 $ under5.mort.rate: num 62.5 64.9 67.6 70.4 73.6 76.8 80.3 83.9 87.6 91.4 ...
 $ iso3c    : chr "AFG" "AFG" "AFG" "AFG" ...
 $ region   : chr "South Asia" "South Asia" "South Asia" ...
 $ capital  : chr "Kabul" "Kabul" "Kabul" ...
 $ longitude: chr "69.1761" "69.1761" "69.1761" ...
 $ latitude  : chr "34.5228" "34.5228" "34.5228" ...
 $ income   : chr "Low income" "Low income" "Low income" "Low income" ...
 $ lending  : chr "IDA" "IDA" "IDA" ...
 $ gni.pc   : num 1746 1761 1766 1783 1796 ...
 $ hdi      : num 0.496 0.493 0.491 0.49 0.488 0.485 0.479 0.465 0.464 0.447 ...
 $ edu.index: num 0.413 0.408 0.406 0.405 0.403 0.398 0.39 0.374 0.372 0.352 ...
 $ income.index: num 0.432 0.434 0.434 0.435 0.436 0.438 0.435 0.421 0.426 0.409 ...
```

Multiple Years

Figure 1.5: Social, Economic & Environmental Factors Across Years (Not Suitable)

While these are important factors in calculating HDI, this is not suitable for my project, as all of my factors are going to come from OpenStreetMaps, which only has geographical

data. However, some of this geographical data will somewhat represent these factors (for example, school density will be loosely proportional to the education index (`edu.index`)). The use of multiple years in this solution is also not suitable for my project, as I will only be accessing the most recent version of OpenStreetMaps. Any previous versions will just be more inaccurate, as new data is constantly being added.

This solution makes use of a Random Forest Classification Algorithm to determine the HDI Class of each country at a particular time, which can either be Low, Mid or High (Figure 1.6).

hdi.pred.rfc			
	Low	Mid	High
Low	406	3	0
Mid	2	403	4
High	0	5	112

Figure 1.6: Random Forest Classification (Not Suitable)

While this does seem to produce good results (the HDI Class has been correctly predicted most of the time), this is not suitable for my solution, as I would like the output of my model to be continuous (the actual HDI), so I will instead be using a Multilayer Perceptron model (described later, in Section 2.3).

Human Development Reports' Country Insights [3]

This solution does not predict HDI, it only visualises it in different graphs. One of those is this ranking chart, which shows the HDI Rank of a particular country compared with the rest (Figure 1.7).

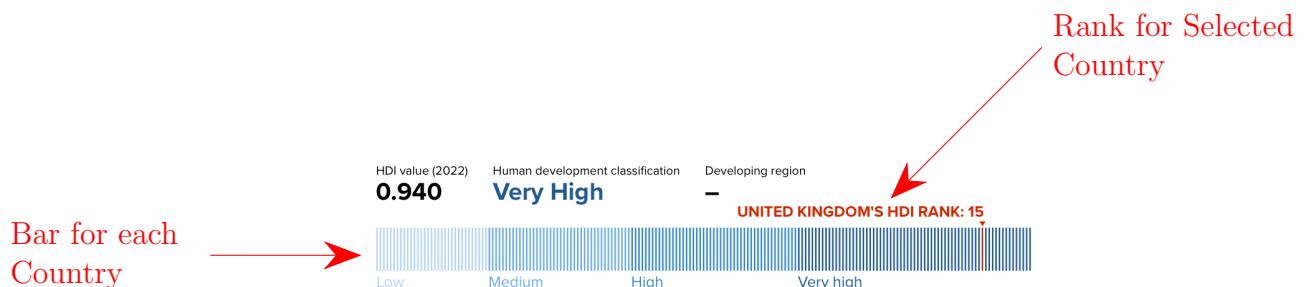


Figure 1.7: HDI Ranking Chart

You can also hover over this chart to reveal additional information for each of the other countries (Figure 1.8).



Figure 1.8: Hovering Over HDI Ranking Chart

I could implement a similar chart to this, except instead of a selected country, it would be whatever region the user is predicting the HDI of, so that they can compare it to the HDI of different countries.

This solution also contains other charts, such as this waterfall chart showing HDI of a particular country over time (Figure 1.9), however I believe none of these are suitable for my project, as I will only be dealing with present-day data.

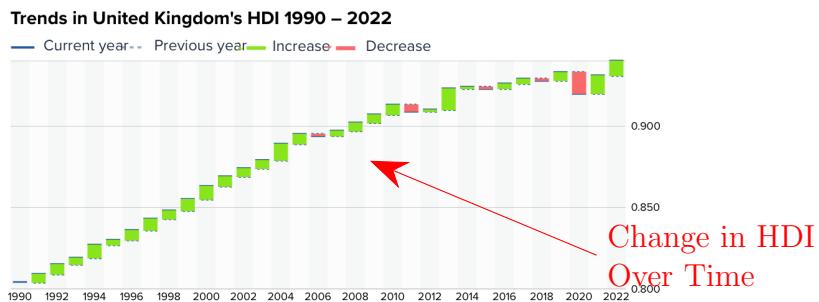


Figure 1.9: HDI Over Time Waterfall Chart (Not Suitable)

1.3.2 Stakeholder Interview & Feedback

For this interview, coloured quotes will represent each stakeholder, as shown in the Stakeholders Table (Table 1.1).

Question 1

1. “Are you familiar with the HDI scale for development?”

“Yes”
“Yes”
“Yes”
“Yes”

For each of the following questions, each stakeholder was given the options in a random order, and was made aware that they don't need to rank all of them.

Question 2

2. “Rank the following end-features by how important you believe they are.”

Analysis of how much each factor affects the others

Comparison of predicted HDI to other countries

Ranking of suggested changes to increase development

- “1. Ranking of suggested changes to increase development”
- 2. Analysis of how much each factor affects the others
- 3. Comparison of predicted HDI to other countries”
- “1. Analysis of how much each factor affects the others”
- 2. Comparison of predicted HDI to other countries
- 3. Ranking of suggested changes to increase development”
- “1. Ranking of suggested changes to increase development”
- 2. Analysis of how much each factor affects the others
- 3. Comparison of predicted HDI to other countries”
- “1. Ranking of suggested changes to increase development”
- 2. Analysis of how much each factor affects the others
- 3. Comparison of predicted HDI to other countries”

To decide which feature is the most wanted from this data, each option will be awarded $(4 - p)$ points for each ranking, where p is the place that it was ranked (for example, if it was ranked in first place, it would gain 3 points).

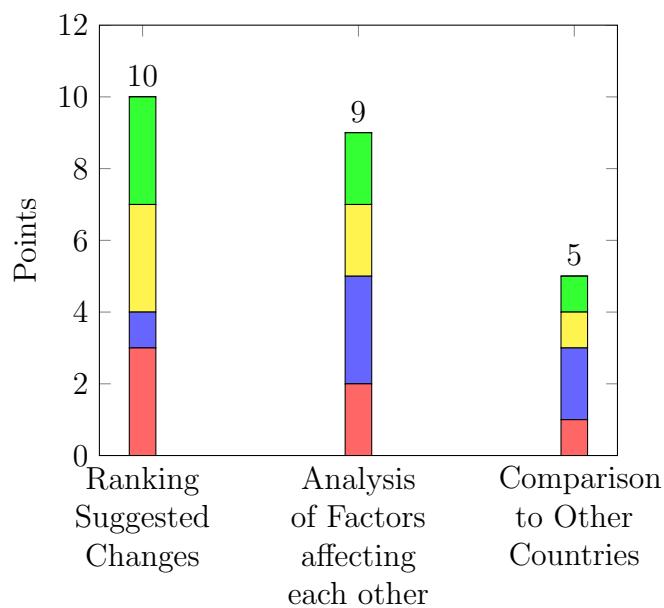


Figure 1.10: Score for each end-feature

Both the Ranking of suggested changes to increase development and the Analysis of how much each factor affects the others seem to be popular.

Notation Explanation

The following 2 questions are about various geographical factors which may have an impact on HDI.

As many of the following factors have a similar format, I will use the notation $D(x)$ to represent “ x Density”, for some object x (the number of x per unit area), and $A(x, y)$ to represent “Average Distance from x to nearest y ”, for some objects x, y (Note that $A(x, y)$ might not be the same as $A(y, x)$). The functions A and D will be implemented later, in Section 2.2.

There are many factors to consider which are directly linked to HDI, such as $D(\text{School})$ or $D(\text{Hospital})$, which I believe are essential factors to be considered. Therefore, I have decided to only ask about more ridiculous factors, such as $D(\text{Bench})$ or $D(\text{Post Box})$ which will have less of an effect on HDI. I have also included $D(\text{School})$ as a test to ensure that the stakeholders fully understand the meaning of HDI (it should always rank highly).

Question 3

3. *“Rank the following density factors affecting development in a particular area by how much you would like them to be implemented.”*

$D(\text{Bench})$	$D(\text{Fast-Food Place})$	$D(\text{Lake})$	$D(\text{Place of Worship})$
$D(\text{Pond})$	$D(\text{Post Box})$	$D(\text{Restaurant})$	$D(\text{Rock})$
$D(\text{School})$	$D(\text{Toilet})$	$D(\text{Tree})$	$D(\text{Vending Machine})$

“1. $D(\text{Toilet})$, 2. $D(\text{School})$, 3. $D(\text{Place of Worship})$, 4. $D(\text{Restaurant})$.”

“1. $D(\text{School})$, 2. $D(\text{Toilet})$, 3. $D(\text{Restaurant})$, 4. $D(\text{Vending Machine})$, 5. $D(\text{Bench})$, 6. $D(\text{Post Box})$, 7. $D(\text{Place of Worship})$, 8. $D(\text{Fast-Food Place})$, 9. $D(\text{Pond})$, 10. $D(\text{Lake})$, 11. $D(\text{Rock})$, 12. $D(\text{Tree})$.”

“1. $D(\text{School})$, 2. $D(\text{Restaurant})$, 3. $D(\text{Post Box})$, 4. $D(\text{Tree})$, 5. $D(\text{Place of Worship})$, 6. $D(\text{Vending Machine})$, 7. $D(\text{Toilet})$, 8. $D(\text{Fast-Food Place})$.”

“1. $D(\text{School})$, 2. $D(\text{Bench})$, 3. $D(\text{Post Box})$, 4. $D(\text{Toilet})$, 5. $D(\text{Vending Machine})$, 6. $D(\text{Fast-Food Place})$, 8. $D(\text{Restaurant})$, 9. $D(\text{Place of Worship})$, 10. $D(\text{Tree})$ ” (the absence of an option at number 7 is not a mistake, he did not seem to put anything there.)

These rankings will be scored similarly, with each option being awarded $(13 - p)$ points, where p is the place that it was ranked, as there are now 12 options instead of 3. If the option was not ranked, then it won’t be awarded any points. The points here are not comparable with the points from the previous question, as a different system is being used.

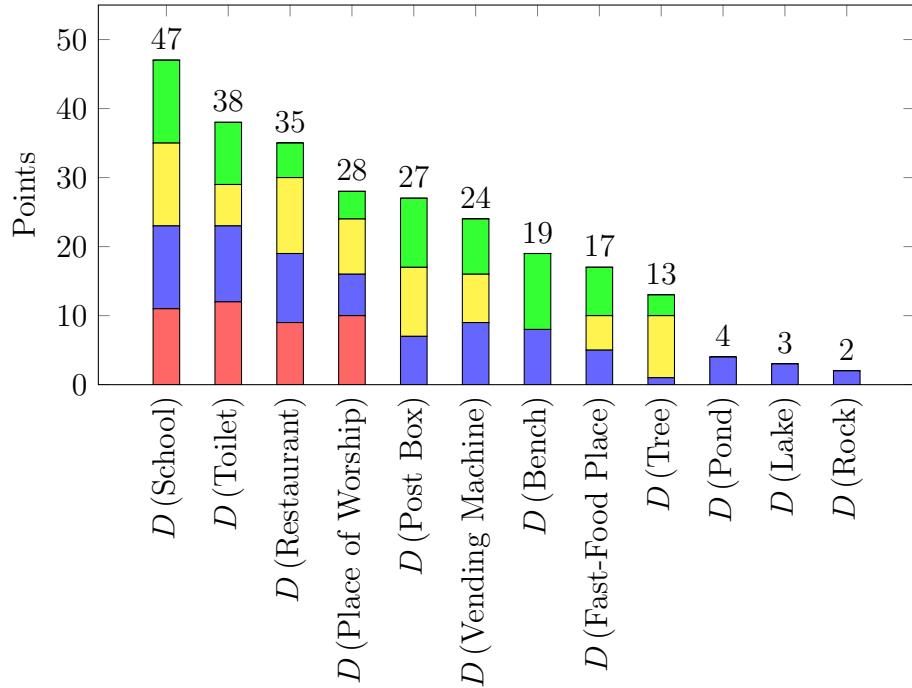


Figure 1.11: Score for each density factor

As Expected, the most popular answer was $D(\text{School})$. Among the other answers, $D(\text{Toilet})$, $D(\text{Restaurant})$, $D(\text{Place of Worship})$, $D(\text{Post Box})$ and $D(\text{Vending Machine})$ all proved to be very popular, with a score ≥ 20 .

Question 4

4. “Rank the following distance factors affecting development in a particular area by how much you would like them to be implemented.”

$A(\text{Bank, Slot Machine})$	$A(\text{Bench, Tree})$	$A(\text{Fast-Food Place, Toilet})$
$A(\text{House, Fast-Food Place})$	$A(\text{House, Place of Worship})$	$A(\text{House, School})$
$A(\text{House, Tree})$		

“1. $A(\text{House, School})$, 2. $A(\text{House, Place of Worship})$, 3. $A(\text{Fast-Food Place, Toilet})$.”

“1. $A(\text{House, School})$, 2. $A(\text{Bank, Slot Machine})$, 3. $A(\text{Fast-Food Place, Toilet})$, 4. $A(\text{House, Place of Worship})$, 5. $A(\text{House, Fast-Food Place})$, 6. $A(\text{Bench, Tree})$, 7. $A(\text{House, Tree})$ ”

“1. $A(\text{Bank, Slot Machine})$, 2. $A(\text{House, School})$, 3. $A(\text{Bench, Tree})$, 4. $A(\text{House, Tree})$, 5. $A(\text{House, Place of Worship})$, 6. $A(\text{House, Fast-Food Place})$, 7. $A(\text{Fast-Food Place, Toilet})$ ”

“1. $A(\text{House, School})$, 2. $A(\text{House, Tree})$, 3. $A(\text{House, Fast-Food Place})$, 4. $A(\text{Bench, Tree})$, 5. $A(\text{House, Place of Worship})$, 6. $A(\text{Fast-Food Place, Toilet})$, 7. $A(\text{Bank, Slot Machine})$ ”

The scoring again will work the same as the previous 2 questions, except each option will be awarded $(8 - p)$ points, where p is the place that it was ranked, as there are now 7 options instead of 3 or 12.

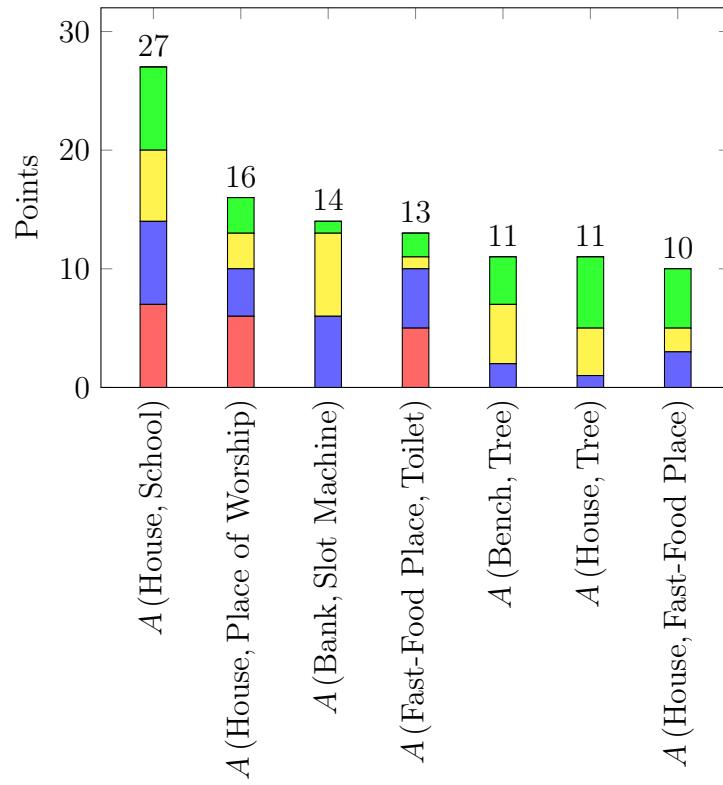


Figure 1.12: Score for each distance factor

Again, the serious factor $A(\text{House}, \text{School})$ was the most popular by far, and will be included among others such as $A(\text{House}, \text{Shop})$ or $A(\text{House}, \text{Pharmacy})$. Among the other answers, $A(\text{House}, \text{Place of Worship})$, $A(\text{Bank}, \text{Slot Machine})$ and $A(\text{Fast-Food Place}, \text{Toilet})$ proved to be very popular, with a score ≥ 12 .

I may ask further questions to my stakeholders as required, when I am developing my coded solution.

1.4 Essential Features

Feature	Explanation
Area Selection	The user must be able to select the area to predict the HDI of. They should be able to do this by either drawing on a map, or entering the data manually.
Multilayer Perceptron	The HDI must be predicted based on a range of scalar factors, which the user has just inputted.

Ranking of Suggested Changes to increase Development	This will allow for users to make informed decisions on what needs to be done to increase the development of their area. This feature proved popular in the Stakeholder Interview.
Analysis of how much each factor affects the others	Users can determine which factors may affect others, which will further increase the development. This feature was in many of the existing solutions and was also proved popular in the Stakeholder Interview.

Table 1.2: Essential Features

All of the above (except for the Multilayer Perceptron) will be implemented and accessed using a GUI.

1.5 Limitations

- **Only 1 Time Period** – As I am using OpenStreetMaps, I will only be able to access the data as it is now, so I won't be able to train the model on previous year's HDI. Older versions of OpenStreetMaps are only more inaccurate, so the training data will just be dirty and the final product will have a smaller chance of working.
- **Only Geographical Factors** – Factors such as life expectancy will not be accounted for, as I would like all the data to come from the same place (OpenStreetMaps), such that the user can circle an area on a map and the HDI will be estimated from that.



Figure 1.13: Map of all shelters from OpenStreetMaps

- **Skew Towards More Developed Countries** – OpenStreetMaps is not perfect. There will be a skew towards more developed countries because in those places, there are more people with computers who know about OpenStreetMaps, and are contributing to it. The map in Figure 1.13 shows this, as it suggests that there are many more shelters in Europe than in Africa, however in reality it is likely to be the other way around.

- **Potential Lack of Training Data** – HDI is only officially calculated for countries or sub-national regions (for example, in the United States, the sub-national regions are the 50 states). Therefore, I only have ~ 2000 pieces of data to train on (which is equal to the total number of sub-national regions), which may not be enough to get accurate results.

1.6 Requirements

1.6.1 Hardware Requirements

- **Standard Input Devices (Keyboard & Mouse)** – I need to be able to write code and the user needs to be able to enter their details.
- **Monitor** – The user and I need to be able to see the output of the program.
- **Main Memory** – The program will be loaded into here when booted up.
- **Secondary Storage** – The pre-trained model must be stored in secondary storage so that it can be called upon at any time.
- (For me only) **Graphical Processing Unit** – While no complex graphics need to be processed, a GPU would decrease the time taken for the model to be trained, as it would involve lots of the same type of calculation (matrix multiplication) to be carried out simultaneously.

I will be using a Mac Mini, M4 to develop this app.

1.6.2 Software Requirements

- (For me only) **IDE (Visual Studio Code)** – I will write the code for this project (and the documentation) in this IDE, as it provides many useful features such as syntax highlighting, automatic bracket completion and version control (git integration).
- (For me only) **Python 3.10** – The programming language I intend to write in.
- (For me only) **Python Libraries** Including but not limited to:
 - **Gradio** – A GUI Library, that will initialise a GUI on a local server, so that the app can be accessed from a web browser.
 - **Numpy** – A Maths Library, which contains many useful mathematical functions and data structures, such as matrices or multidimensional arrays, as well as the operations that come with them.
 - **Folium** – A Mapping Library, which will allow me to create various maps within the Gradio interface, for both input and output.

- **Web Browser** – The Gradio GUI must be accessed through a web browser. I need to be able to test the code, and the user needs to be able to use it. The user will only require a web browser as I plan to deploy this app once I have finished development.

I will be using macOS Sequoia 15.1 to develop this app.

1.7 Success Criteria

No.	Success Criterion	Justification
S_1	Find suitable HDI training data for the Neural Network	The neural network must train on pre-calculated HDI values for certain areas, along with their respective density and average distance factors.
S_2	Retrieve OSM data for a given area using Overpass Turbo	This will allow me to calculate useful geographical factors to predict the HDI from instead of more human ones (which are harder to determine).
S_3	Calculate average distance factors from a given area	The average distance from <i>some object</i> to <i>some other object</i> is also a good measure to consider. For example <i>A</i> (House, School) is a useful factor as school children shouldn't have to travel long distances for education.
S_4	Calculate density factors from a given area	The number of <i>some object</i> per unit area is a good way of measuring how many of that object are in a particular area while removing the bias of larger areas.
S_5	Successfully Implement the Feedforward Algorithm	This algorithm will allow us to make predictions on the HDI given a set of factors.
S_6	Successfully Implement the Backpropogation Algorithm	This algorithm will allow us to train the neural network base on a set of training examples.
S_7	Successfully train the Neural Network	In order to accurately predict the HDI of a given area, the neural network must train on existing examples to see what makes a high HDI.
S_8	User can select an area on the map for which they want to analyse	This will allow the user to choose which area they want to predict the HDI of by drawing on the map.
S_9	Neural Network accurately predicts HDI from those factors	Once the user has selected their area, the HDI must be predicted based on factors calculated from that area.

S_{10}	Accurately calculate where a new building would need to be in order to increase the HDI the most	The specific placement of new buildings (as suggestions to increase HDI) is important to consider, so a suitable place must be chosen in order to decrease the average distance to it as much as possible.
S_{11}	Changes are suggested to increase the HDI	This will allow the user to make an informed decision on what to do in order to improve their area.
S_{12}	User can create an account	Eventually, I would like the user to be able to create an account, which will allow them to save different areas they have already predicted the HDI of.
S_{13}	User can log into an account	Once the account has been created, the user must be able to log into it, in order to make use of the features.
S_{14}	User can compare a region to one in the training data	This will allow users to make a judgement on how certain regions have the HDI that they have, and how they can improve their own.
S_{15}	User can see how each factor can affect the others	This will allow the user to consider the secondary effects of their decisions, as the suggested changes may not only increase the HDI, but also some of the other factors.
S_{16}	Predicted HDI is ranked among other countries with a similar HDI	This will allow users who are not that familiar with HDI to get an idea of the HDI scale by comparing it to the gist of what they know.
S_{17}	User can save regions	This will allow users to save time and not re-download the buildings & re-calculate the factors. Users with an account should also be able to store regions there.
S_{18}	User can manually enter factors	If downloading & calculating the factors takes too long, the user may manually enter them instead.
S_{19}	App has a clean, simple and presentable GUI	This will help the user understand what they can do, and make it easier to use.

Table 1.3: Success Criteria

2. Design of the 1st Prototype

2.1 Structure Diagram



Figure 2.1: Structure Diagram

Figure 2.1 shows this project's structure, breaking down the larger problem of the entire app, into a series of smaller problems which are easier to consider. I will go into more detail for each of the smaller problems above (further decomposing them), in the approximate order I would be implementing them.

In this first prototype, I will be going through the algorithms involved in Data Collection & Processing (Section 2.2) and the Multilayer Perceptron (Section 2.3), with only the most essential features in the GUI. Various data visualisations such as the Factor Heatmap or Comparisons to Other Countries will be implemented in later prototypes.

2.2 Data Collection & Processing

2.2.1 Full List of Factors to Include

After considering the responses from the stakeholder interview (Section 1.3.2), as well as research on what contributes to a higher HDI, the full list of factors I would like to consider are (in no particular order):

- | | |
|---|----------------------------------|
| 1. $A(\text{House}, \text{School})$ | 13. $D(\text{School})$ |
| 2. $A(\text{House}, \text{Hospital})$ | 14. $D(\text{Hospital})$ |
| 3. $A(\text{House}, \text{Pharmacy})$ | 15. $D(\text{Pharmacy})$ |
| 4. $A(\text{House}, \text{Restaurant})$ | 16. $D(\text{Police})$ |
| 5. $A(\text{School}, \text{Hospital})$ | 17. $D(\text{Library})$ |
| 6. $A(\text{Police}, \text{Hospital})$ | 18. $D(\text{Toilet})$ |
| 7. $A(\text{House}, \text{Place of Worship})$ | 19. $D(\text{Restaurant})$ |
| 8. $A(\text{Bank}, \text{Slot Machine})$ | 20. $D(\text{Place of Worship})$ |
| 9. $A(\text{Fast-Food Place}, \text{Toilet})$ | 21. $D(\text{Post Box})$ |
| 10. $A(\text{House}, \text{Police})$ | 22. $D(\text{Vending Machine})$ |
| 11. $A(\text{University}, \text{Library})$ | 23. $D(\text{Bench})$ |
| 12. $A(\text{House}, \text{Library})$ | 24. $D(\text{Tree})$ |

Now we must figure out how to calculate $A(x, y)$, for some objects x, y , and $D(x)$, for some object x .

2.2.2 Finding the Average Distance between 2 objects $x, y, A(x, y)$

Finding the distance between 2 given points

This is a difficult problem to solve, so instead decompose it and consider a simple one, finding the distance between any 2 given points on the Earth's surface (latitude longitude pairs).

First, abstract the earth into a perfectly smooth sphere (which is fairly accurate anyway as $\frac{\text{height difference between Mt. Everest and the Mariana Trench}}{\text{Earth's average radius}} = 0.0031$), with centre O and radius r , and rotated such that the plane of the equator lies on the xy -plane and that the point with 0 latitude and 0 longitude lies on the positive x -axis. I am not considering the altitude at each point on the Earth's surface, nor the slight eccentricity in the shape of the Earth.

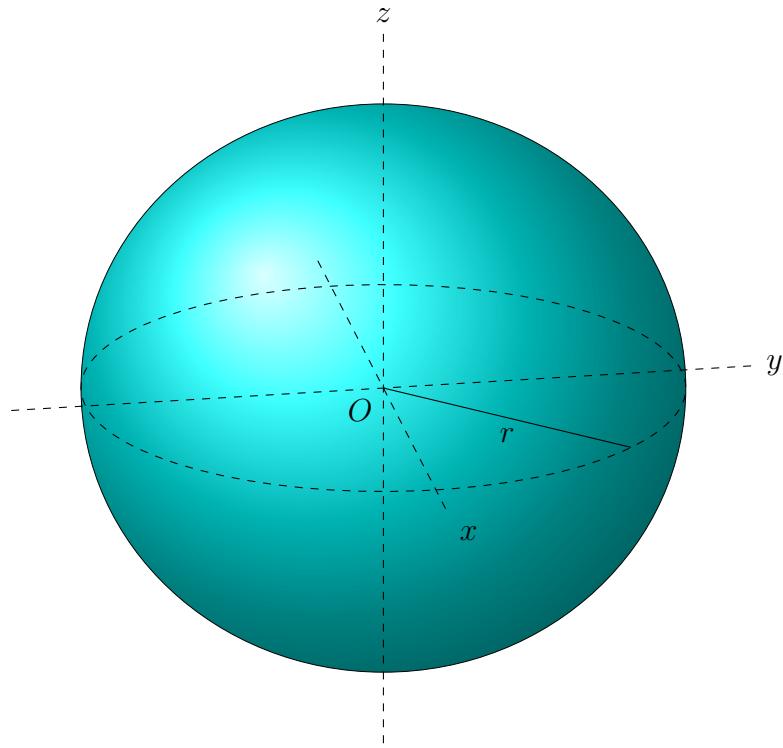
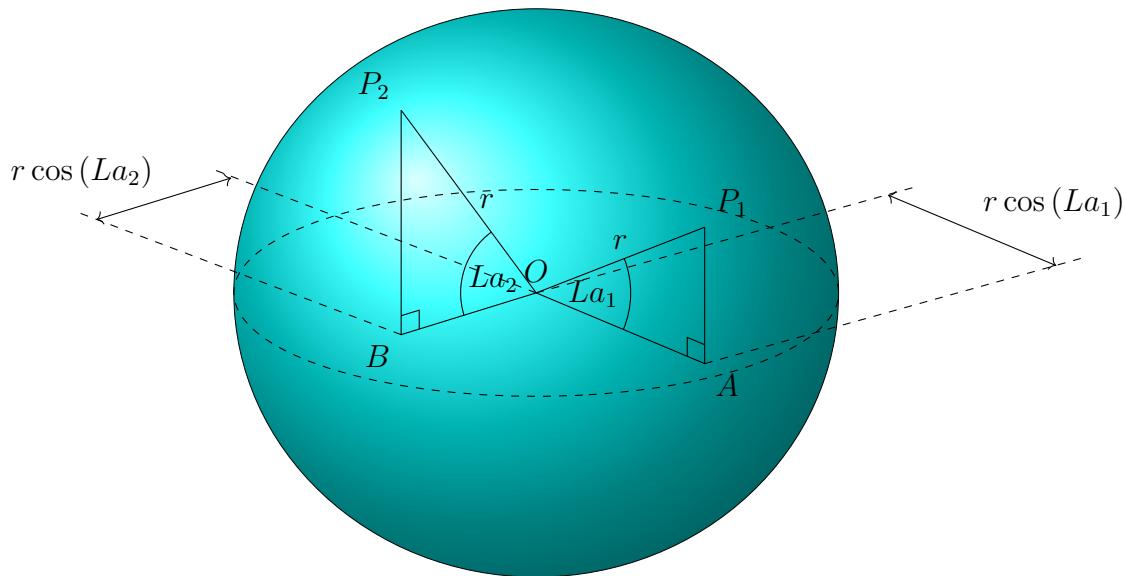


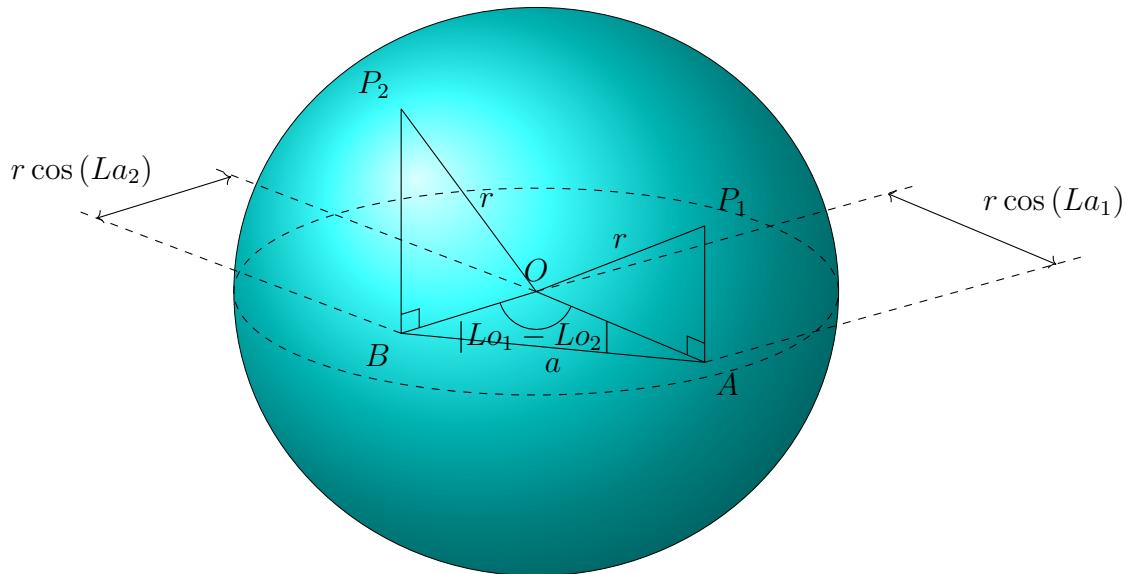
Figure 2.2: Abstracted Model of the Earth

The 2 points on its surface we are trying to find the distance between are P_1 and P_2 , each with a latitude and longitude (La_1, La_2, Lo_1, Lo_2) , given in radians.

Construct points A and B , such that they lie directly below P_1 and P_2 respectively, and sit on the plane of the equator. By definition of latitude, $\angle P_1OA = La_1$ and $\angle P_2OB = La_2$. From right-angled trigonometry, $OA = r \cos (La_1)$ and $OB = r \cos (La_2)$

Figure 2.3: P_1 and P_2

Construct the line AB , which lies on the plane of the equator. By definition of longitude, $\angle AOB = |Lo_1 - Lo_2|$ (the difference in longitudes). Define a to be the length of the line segment AB , which can be found using the cosine rule.

Figure 2.4: Line segment AB

$$\begin{aligned}
 a = AB &= \sqrt{OA^2 + OB^2 - 2 \times OA \times OB \times \cos(\angle AOB)} \\
 &= \sqrt{(r \cos(La_1))^2 + (r \cos(La_2))^2 + 2 \times r \cos(La_1) \times r \cos(La_2) \times \cos(|Lo_1 - Lo_2|)} \\
 &= \sqrt{r^2 \cos^2(La_1) + r^2 \cos^2(La_2) - 2r^2 \cos(La_1) \cos(La_2) \cos(Lo_1 - Lo_2)}
 \end{aligned} \tag{2.1}$$

From right-angled trigonometry, $P_1A = r \sin(La_1)$ and $P_2B = r \sin(La_2)$. Construct the line P_1P_2 , which will lie in the same vertical plane as AB (as P_1 is directly above A and P_2 is directly above B). If P_1 and P_2 are in the same hemisphere (north or south), then the shape P_1ABP_2 will be a right-angled trapezium. Define b to be the length of the line segment P_1P_2 .

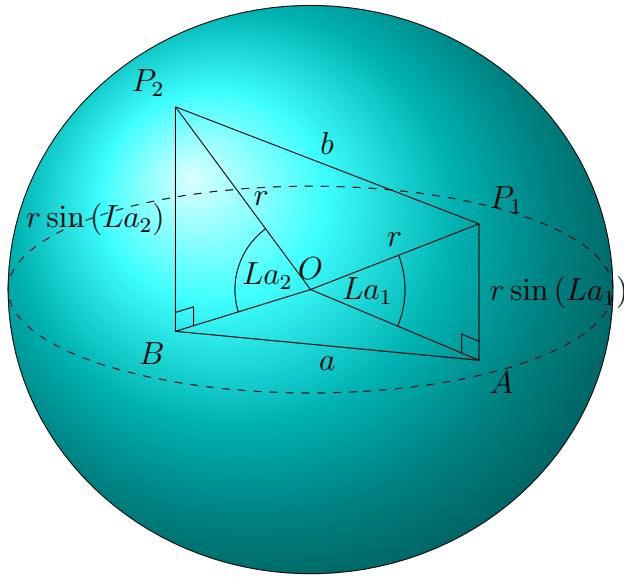
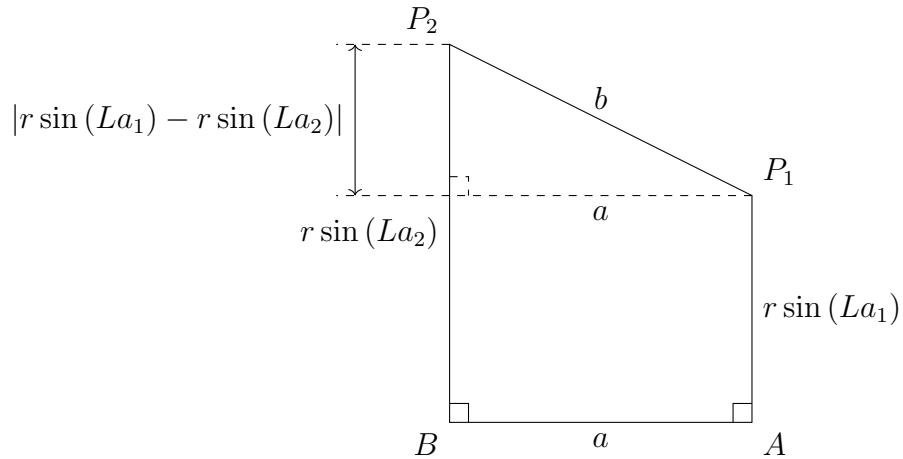


Figure 2.5: Line segment P_1P_2

b can be found using Pythagoras, by considering moving AB up until it meets either P_1 or P_2 . We don't know which one it will meet first, so the difference in heights is $|P_1A - P_2B|$. This will still be true even if P_1ABP_2 is not a trapezium (as a result of P_1 and P_2 being in opposite hemispheres), as the one in the Southern Hemisphere will have a negative height.

Figure 2.6: Finding b

$$\begin{aligned}
 b &= P_1P_2 = \sqrt{|P_1A - P_2B|^2 + AB^2} \\
 &= \sqrt{(|r \sin(La_1) - r \sin(La_2)|)^2 + a^2} \\
 &= \sqrt{(r \sin(La_1) - r \sin(La_2))^2 + a^2}
 \end{aligned} \tag{2.2}$$

This is not quite what we are after, as this is the straight line distance from \$P_1\$ to \$P_2\$, but we want the distance along the surface of the Earth, which will be curved. To find this, we must find the angle \$\theta\$ between \$P_1\$ and \$P_2\$, which we can do by using the cosine rule in reverse.

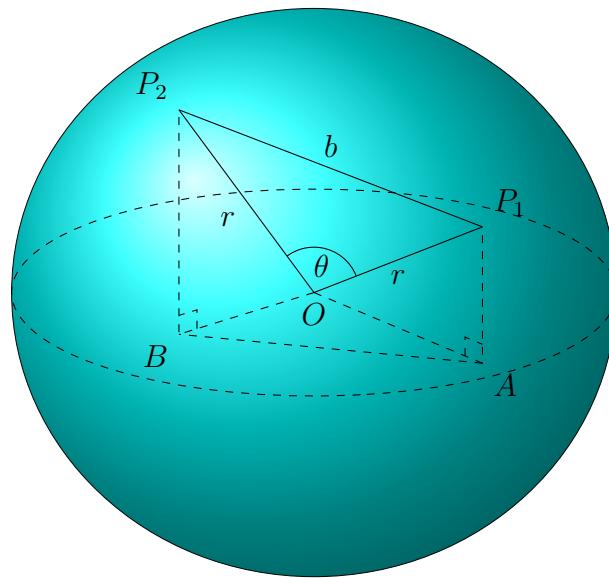


Figure 2.7: Angle \$\theta\$ between \$P_1\$ and \$P_2\$

$$\begin{aligned}
 \theta &= \angle P_1 O P_2 = \arccos \left(\frac{(OP_1)^2 + (OP_2)^2 - (P_1 P_2)^2}{2 \times OP_1 \times OP_2} \right) \\
 &= \arccos \left(\frac{r^2 + r^2 - b^2}{2 \times r \times r} \right) \\
 &= \arccos \left(\frac{2r^2 - b^2}{2r^2} \right) \\
 &= \arccos \left(1 - \frac{b^2}{2r^2} \right)
 \end{aligned} \tag{2.3}$$

The arclength between 2 points on a circle (in this case, the great circle on the surface of the Earth containing P_1 and P_2) separated by an angle of θ is given by arclength = $r\theta$.

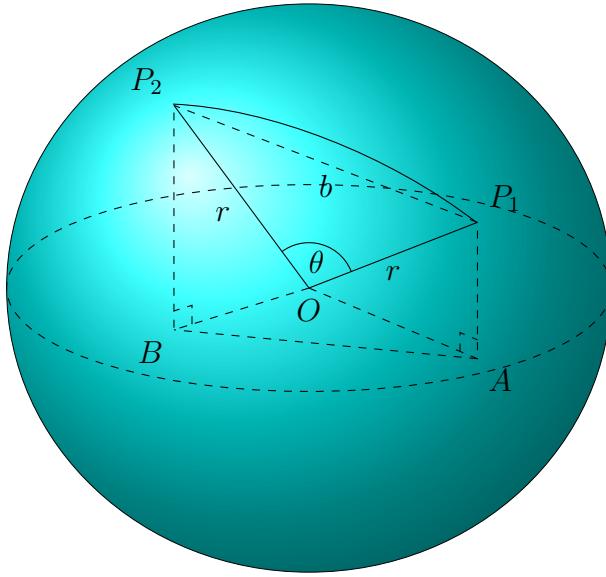


Figure 2.8: Arc between P_1 and P_2

Substituting θ for equation 2.3 gives

$$\text{arclength} = r \arccos \left(1 - \frac{b^2}{2r^2} \right) \tag{2.4}$$

Substituting b for equation 2.2 gives

$$\begin{aligned}
 \text{arclength} &= r \arccos \left(1 - \frac{(r \sin(La_1) - r \sin(La_2))^2 + a^2}{2r^2} \right) \\
 &= r \arccos \left(1 - \frac{r^2 \sin^2(La_1) - 2r^2 \sin(La_1) \sin(La_2) + r^2 \sin^2(La_2) + a^2}{2r^2} \right)
 \end{aligned} \tag{2.5}$$

Substituting a for equation 2.1 gives

$$\begin{aligned}
\text{arclength} &= r \arccos \left(1 - \frac{r^2 \left(\sin^2(La_1) - 2 \sin(La_1) \sin(La_2) + \sin^2(La_2) + \cos^2(La_1) \right. \right. \\
&\quad \left. \left. + \cos^2(La_2) - 2 \cos(La_1) \cos(La_2) \cos(Lo_1 - Lo_2) \right)}{2r^2} \right) \\
&= r \arccos \left(1 - \frac{\sin^2(La_1) + \cos^2(La_1) + \sin^2(La_2) + \cos^2(La_2) \right. \\
&\quad \left. - 2 \sin(La_1) \sin(La_2) - 2 \cos(La_1) \cos(La_2) \cos(Lo_1 - Lo_2)}{2} \right) \\
&= r \arccos \left(1 - \frac{2 - 2 \sin(La_1) \sin(La_2) - 2 \cos(La_1) \cos(La_2) \cos(Lo_1 - Lo_2)}{2} \right) \\
&= r \arccos(1 - (1 - \sin(La_1) \sin(La_2) - \cos(La_1) \cos(La_2) \cos(Lo_1 - Lo_2))) \\
&= r \arccos(\sin(La_1) \sin(La_2) + \cos(La_1) \cos(La_2) \cos(Lo_1 - Lo_2))
\end{aligned} \tag{2.6}$$

Therefore, the distance in km between 2 latitude and longitude coordinates along the surface of the Earth is given by:

$$\text{distance} = r \arccos(\sin(La_1) \sin(La_2) + \cos(La_1) \cos(La_2) \cos(Lo_1 - Lo_2)) \tag{2.7}$$

where r is the radius of the Earth in km and La_1 , La_2 , Lo_1 and Lo_2 are the 2 points' latitudes and longitudes in radians.

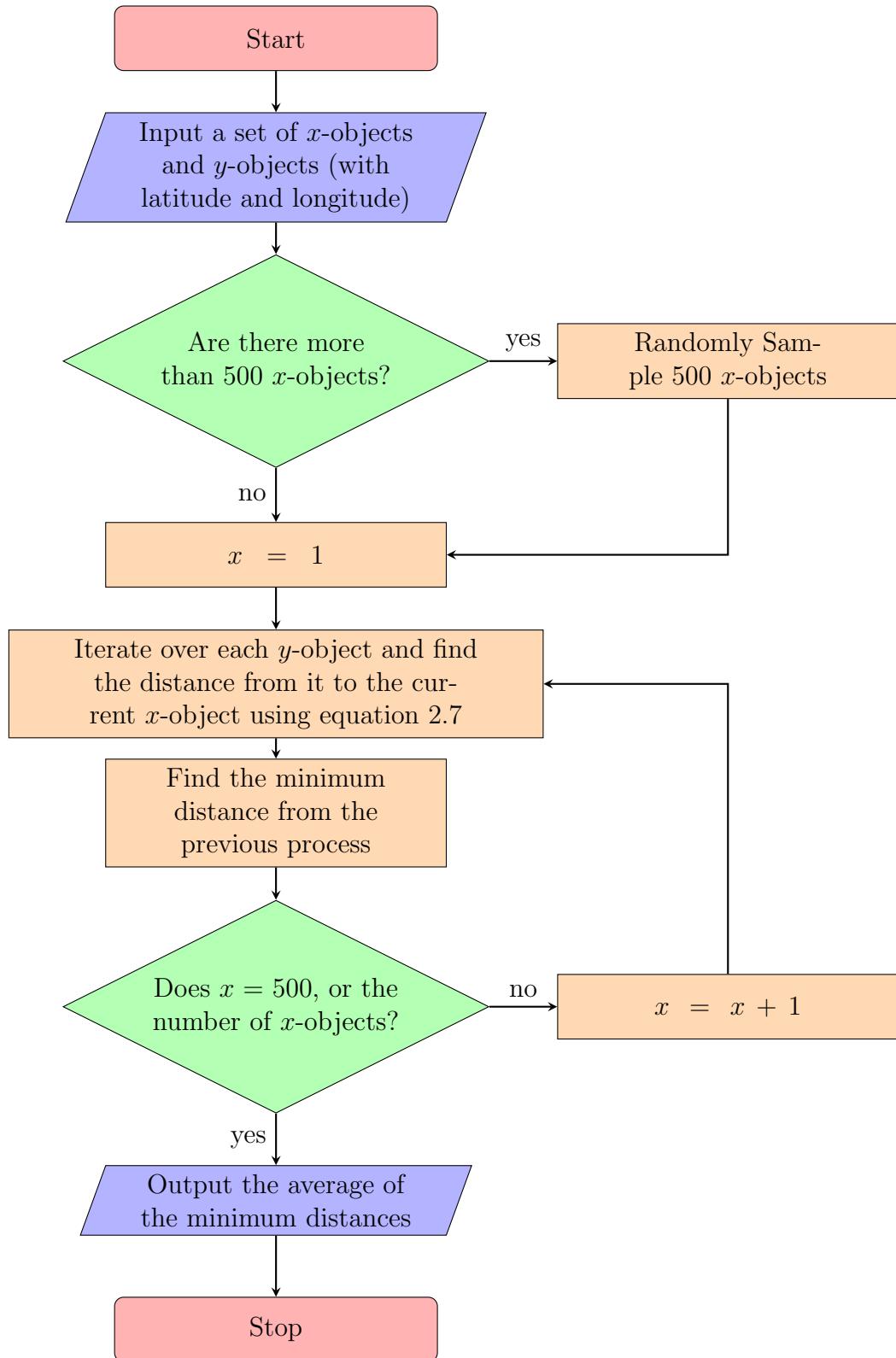
However, the latitude and longitude coordinates returned by OpenStreetMaps are not measured in radians, but instead measured in degrees, so we must use the conversion factor (angle in radians) = $\frac{\pi}{180} \times$ (angle in degrees) on each one before they are passed into the formula.

Finding the Average Distance between 2 objects $x, y, A(x, y)$

Now that we have the means to find the distance between any 2 points along the Earth's surface, we can consider the average distance between 2 particular types of object. Recall that $A(x, y)$ is specifically defined as "Average Distance from x to nearest y " (for example, x might be House, and y might be School), so the input for this function will be 2 sets of lists of latitude and longitude coordinates, corresponding to all the x -objects and y -objects in a particular area.

As there are likely to be hundreds of thousands of each object (depending on how large of an area the user selects), it will take too long to calculate the average distance over all of these data points. This is important to consider, as this function will be run thousands of times, both in preparing the data (for me only) and when users are using the app.

Therefore, instead of considering every single data point, randomly sample an arbitrary 500 of the x -objects (and if there are less than 500, consider all of them), and then find out which of the y -objects is closest to each of them. We must consider every y -object for each considered x -object as we don't know which one will be closest to that x . Considering all the y -objects should be fine, as for all the factors in the form $A(x, y)$, there will be many more x -objects than y -objects (for example, there are many more houses than schools).

Figure 2.9: $A(x, y)$ Flowchart

For validation, if the entered list of x -objects is empty (in other words, there does not

exist any x -objects in the region), the average distance returned should be infinity.

2.2.3 Finding the Density of some object x , $D(x)$

Definition of $D(x)$

The density of some object is defined as “The number of that object per unit area”. Therefore, $D(x)$ is

$$\frac{\text{Total number of } x\text{-objects in a given region}}{\text{Area of that region}} \quad (2.8)$$

Finding the total number of x -objects in a region is easy. As they will come in a list, we just need to find the length of that list. However, finding the area of that region is much harder, as the only information we have about it is a list (of unknown size) of its bounding latitude and longitude coordinates.

To do this, I will be using the same abstraction for the Earth, as shown in Figure 2.2.

Area of a Polygon on the Surface of a Sphere

First, assume that any polygon on the surface of a sphere (any region that the user may draw) can be cut into a series of internal spherical triangles (in fact, a spherical polygon with n sides can always be broken up into $n - 2$ spherical triangles).

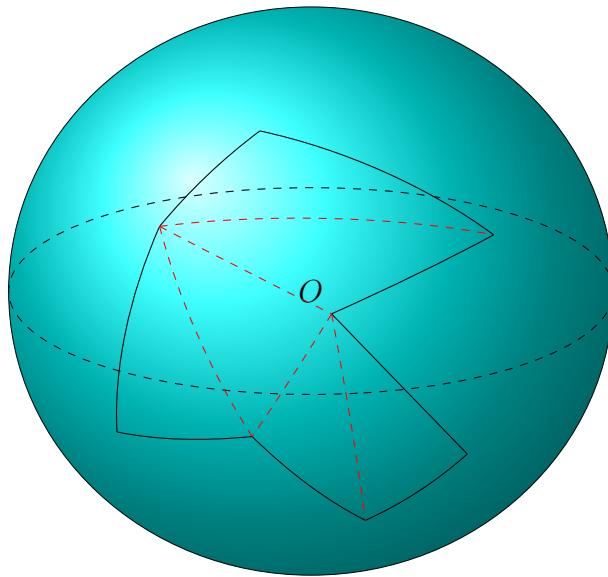


Figure 2.10: Cutting into Triangles

To find the area of the polygon, we can individually find the area of each triangle and then add them up.

To find the area of a triangle on the surface of the sphere, consider extending the sides of the triangle to make great circles around the sphere.

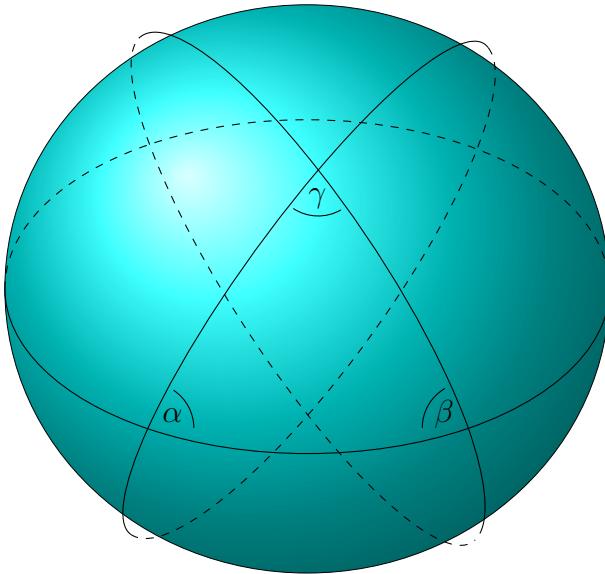


Figure 2.11: Triangle Edges Extended

As you can see from Figure 2.11, the total surface area of the sphere can be given by the sum of these 6 “spherical lune” shapes, with an excess of 4 of the triangles we want (at either end of the sphere, the triangle is formed by the intersection of 3 spherical lunes, and so therefore we must subtract 2 for each side to get the full area of the sphere with no excess, which is why we must subtract 4 triangles overall).

$$A_{\text{Sphere}} = \sum A_{\text{Lune}} - 4A_{\text{Triangle}} \quad (2.9)$$

The surface area of a sphere is $4\pi r^2$, and the area of a spherical lune of an angle θ can be given as a fraction of the full area of the sphere.

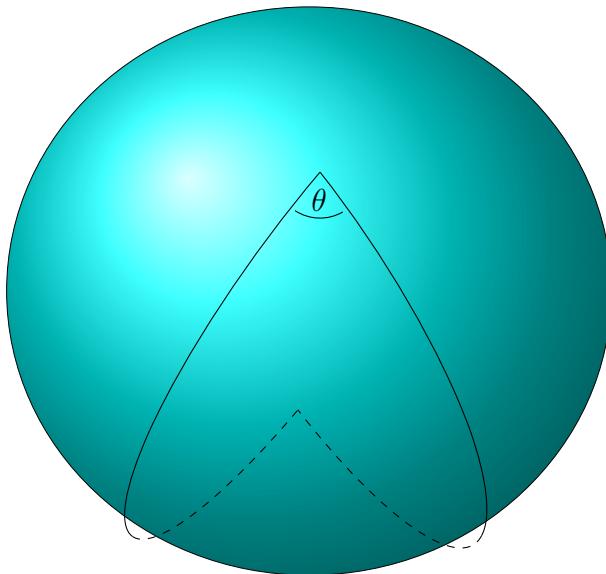


Figure 2.12: Area of a Spherical Lune

$$\begin{aligned} A_{\text{Lune}} &= \frac{\theta}{2\pi} \times 4\pi r^2 \\ &= 2\theta r^2 \end{aligned} \tag{2.10}$$

The 6 lunes shown in Figure 2.11 can be broken down into 3 sets of 2 identical lunes, with angles α , β and γ . Therefore, from equation 2.9

$$\begin{aligned} A_{\text{Triangle}} &= \frac{\sum A_{\text{Lune}} - A_{\text{Sphere}}}{4} \\ &= \frac{2 \times (2\alpha r^2 + 2\beta r^2 + 2\gamma r^2) - 4\pi r^2}{4} \\ &= \frac{4r^2 (\alpha + \beta + \gamma - \pi)}{4} \\ &= r^2 \left(\sum \theta - \pi \right) \end{aligned} \tag{2.11}$$

where $\sum \theta$ represents the sum of the angles in the spherical triangle.

As our original region (an n -sided spherical polygon) was split up into $n - 2$ spherical triangles, we can see that the area of the region R is given by

$$\begin{aligned} R &= \sum_{i=1}^{n-2} (A_{\text{Triangle}})_i \\ &= \sum_{i=1}^{n-2} r^2 \left(\left(\sum \theta \right)_i - \pi \right) \\ &= r^2 \left(\sum_{i=1}^{n-2} \left(\sum \theta \right)_i - (n-2)\pi \right) \end{aligned} \tag{2.12}$$

where $\left(\sum \theta \right)_i$ represents the sum of the angles in spherical triangle i .

As the sum of the angles in a polygon equals the sum of all the sums of angles in the triangles it was broken up into, this simplifies to

$$R = r^2 \left(\sum \theta - (n-2)\pi \right) \tag{2.13}$$

where $\sum \theta$ represents the sum of the interior angles in the spherical polygon, and n is the number of sides of the spherical polygon.

Finding the Interior Angles

To find the interior angles of our polygon, we could use the cosine rule in reverse, as before. However, as $\cos \theta \equiv \cos (360 - \theta)$, this will essentially result in both the interior angle and exterior angle for all the vertices, with no indication of which ones which. Therefore, we must introduce some directionality, by considering vectors.

The vectors we want to consider are those in the plane tangent to the point in which we want to find the interior angle of.

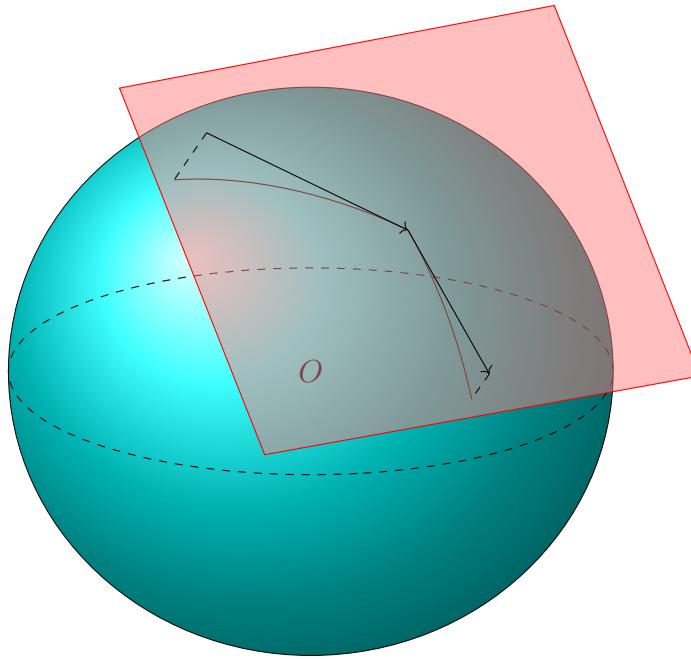


Figure 2.13: Considering Vectors

To find these vectors, we first must convert each of the coordinates from their latitude-longitude forms into their 3D position vector forms (xyz coordinates).

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} r \cos(La) \cos(Lo) \\ r \cos(La) \sin(Lo) \\ r \sin(La) \end{pmatrix} \quad (2.14)$$

where La is the latitude of the point, Lo is the longitude of the point, and r is the radius of the Earth. As we are only using this to find angles, the radius of the Earth is actually irrelevant, and so to simplify things, we can let $r = 1$.

Finding these vectors on a plane which is rotated strangely is difficult, so we can instead rotate the whole system such that this plane lies on the xy -plane. This can be done by rotating it about the z -axis anticlockwise by an angle of $-Lo$, by using the matrix

$$\begin{pmatrix} \cos(-Lo) & -\sin(-Lo) & 0 \\ \sin(-Lo) & \cos(-Lo) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (2.15)$$

followed by rotating it about the y -axis anticlockwise by an angle of $La - 90^\circ$, by using the matrix

$$\begin{pmatrix} \cos(La - 90^\circ) & 0 & \sin(La - 90^\circ) \\ 0 & 1 & 0 \\ -\sin(La - 90^\circ) & 0 & \cos(La - 90^\circ) \end{pmatrix} \quad (2.16)$$

finally followed by translating it down by 1 unit (as, for now, $r = 1$), by adding the vector

$$\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \quad (2.17)$$

This will have the effect of moving the point we want to find the angle at, to $(0, 0, 0)$, while transforming the other points in the same way.

As the plane with the vectors we want to find is now the xy -plane, all we need to do to project the other points onto that plane is set their z -coordinates to 0.

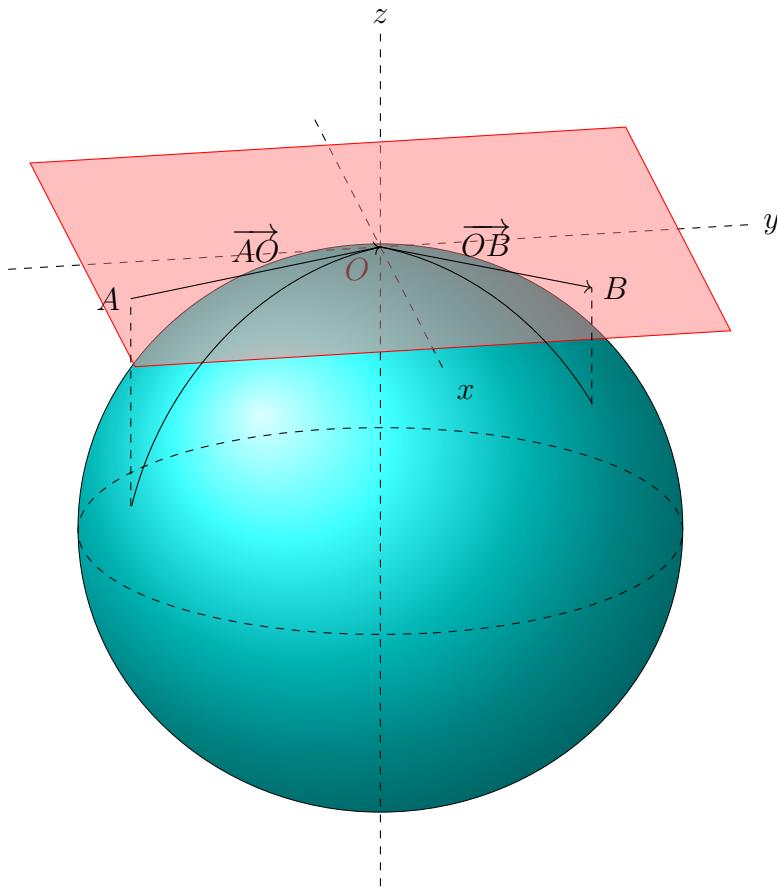


Figure 2.14: Projecting the Other Points onto the xy -plane

As the points will be given in a list, in the other the user has drawn them in the polygon, \overrightarrow{AO} (the vector from the projection of the previous point to the current point) will be given by the coordinates of A times -1 , and \overrightarrow{OB} (the vector from the current point to the projection of the next point) will be given by the coordinates of B .

Imagine travelling along the line segment AO . When you get to O , you must turn some angle θ anticlockwise (between 180° and -180°) to be facing in the right direction to travel along OB . Note that this angle is not the angle between AO and OB , but the angle between \overrightarrow{AO} and \overrightarrow{OB} . The *anticlockwise* angle θ between \overrightarrow{AO} and \overrightarrow{OB} is given by

$$\theta = \arctan 2 ((a_2 b_3 - a_3 b_2) - (a_3 b_1 - a_1 b_3) + (a_1 b_2 - a_2 b_1), a_1 b_1 + a_2 b_2 + a_3 b_3) \quad (2.18)$$

where

$$\overrightarrow{AO} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}, \overrightarrow{OB} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \quad (2.19)$$

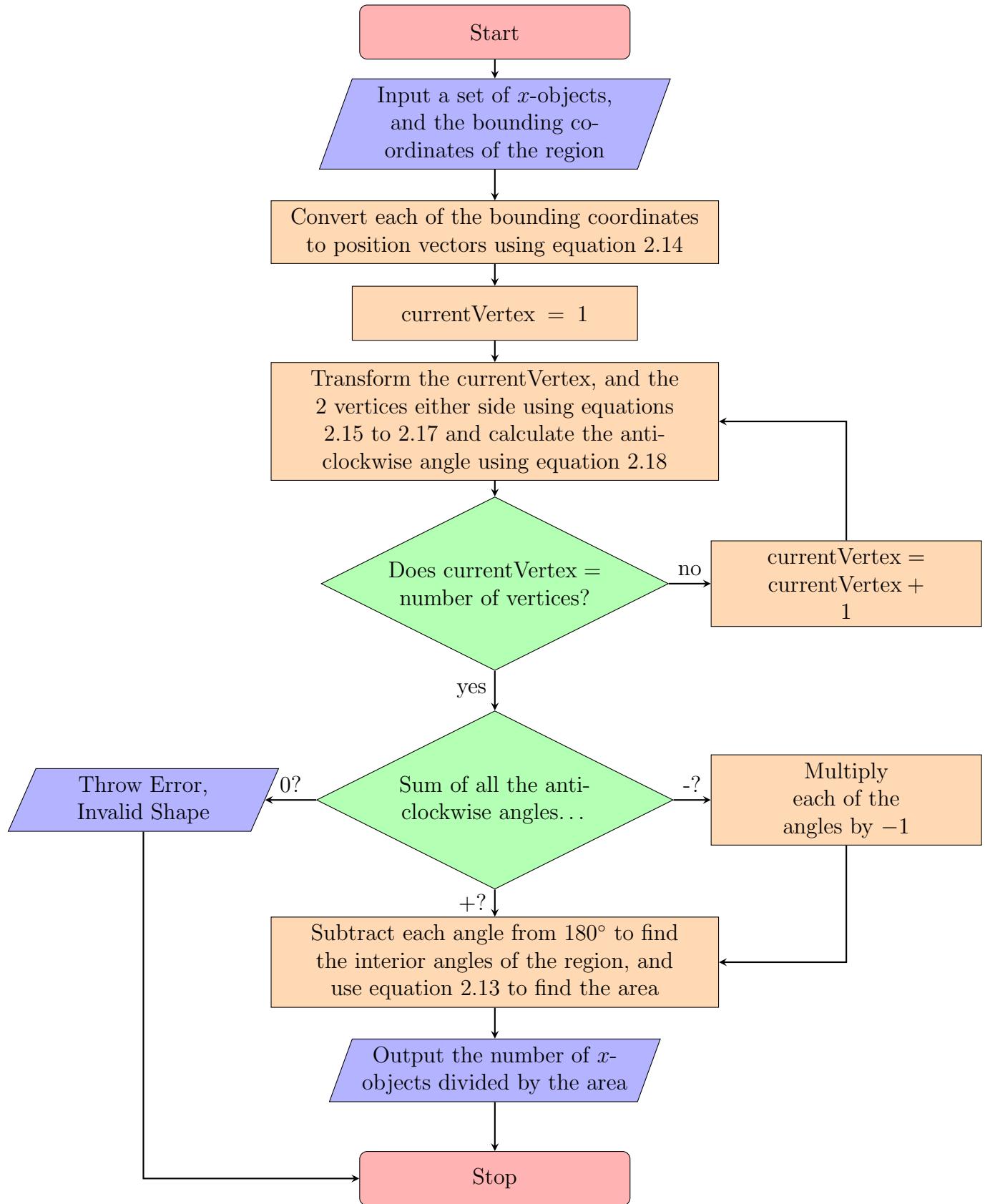
Therefore, if the user has drawn the region in an anticlockwise-winding order, then the interior angle of this particular point will be given by $180^\circ - \theta$.

However, the user may decide to draw their region in a clockwise-winding order, which would currently result in the wrong area being calculated. To detect if the user has drawn their shape in a clockwise or anticlockwise winding order, we can find the sum of the “anticlockwise” angles. If it is positive, they have drawn it anticlockwise, and if it is negative, they must have drawn it clockwise. If it is 0, that means that the shape must self-intersect at some point, which should not be allowed.

If the shape was drawn clockwise, then instead of calculating the anticlockwise angles, this process will have calculated the clockwise angles between the vectors (as the vectors are pointing the other way), which is why I put “anticlockwise” in quotation marks. In order to turn them into the correct anticlockwise angles, we must multiply each of them by -1 , before calculating the interior angles.

Final Algorithm

Now that we have all the interior angles, all we need to do is plug them into the formula for the area of the region (equation 2.13), and then divide by the number for x -objects to find the density.

Figure 2.15: $D(x)$ Flowchart

2.2.4 Overall Algorithm for Data Collection & Processing

In order to train a neural network, I need both input and output data. The input data will be 24 factors listed above, in section 2.2.1, and the output data will be the true HDI of the region.

For this, I will be using the subnational region data from the Global Data Lab [4], in order to maximise the number of training examples. In order to calculate the 24 factors for each of these regions, I will need the bounding coordinates of all the regions. I will also get this data from the Global Data Lab [5], which contains shapefile data on all of the subnational regions.

Therefore, all we need to do is for each subnational region, calculate each factor and store it in an external file (e.g a .csv file) along with the correct HDI, ready for the neural network to train on.

In order to calculate each factor however, we must first query Overpass Turbo, in order to get the data about the locations objects.

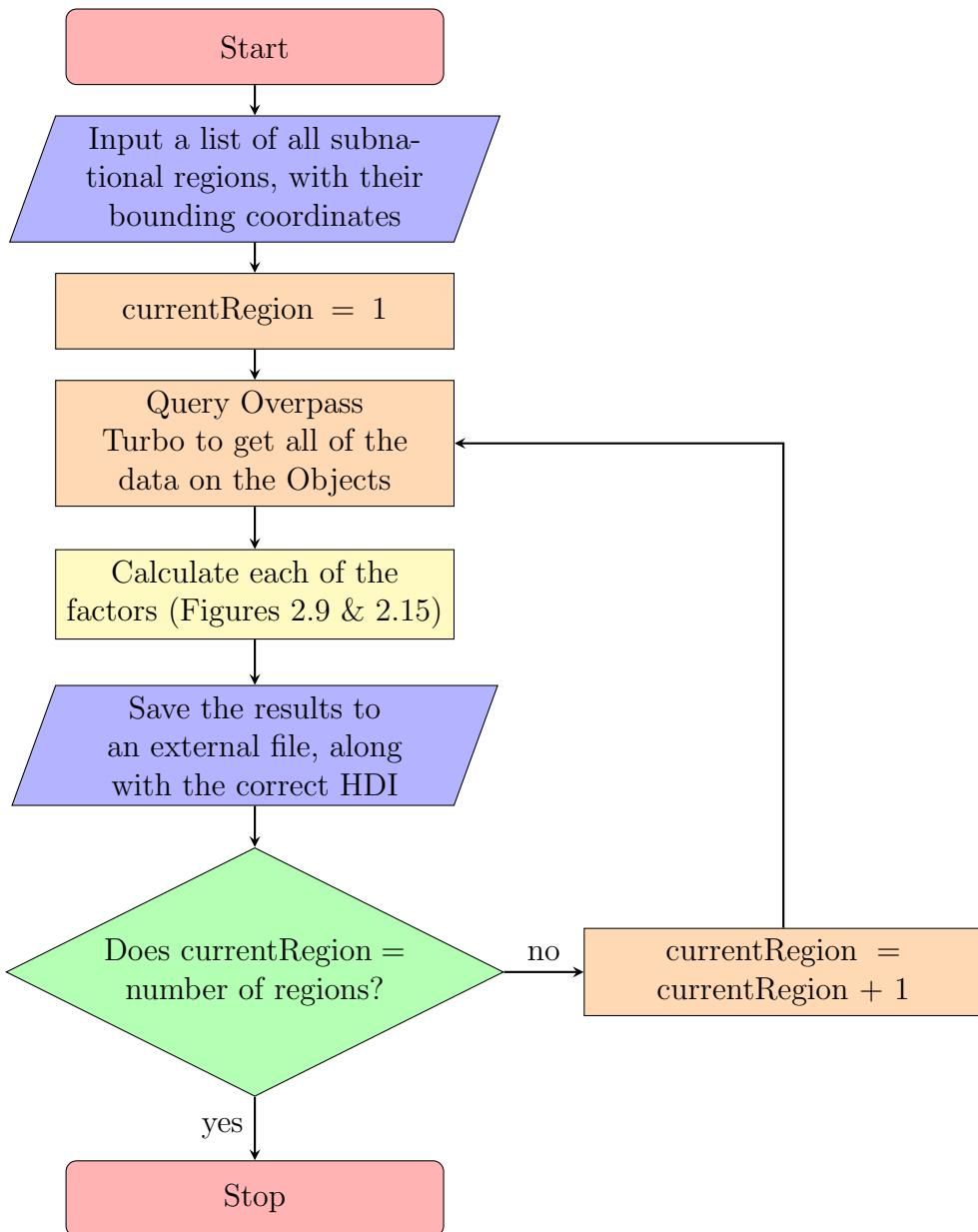


Figure 2.16: Overall Data Collection & Processing Algorithm Flowchart

2.2.5 List of Key Variables

Variables in $A(x, y)$

Identifier	Data Type	Explanation	Justification
x_objects	<code>list[tuple[float]]</code>	Parameter of the function, holds a list of the coordinates of all the x-objects.	Each coordinate is a tuple as the latitudes and longitudes will never change.

y_objects	<code>list[tuple[float]]</code>	Parameter of the function, holds a list of the coordinates of all the y -objects.	Each coordinate is a tuple as the latitudes and longitudes will never change.
max_x_objects	<code>int</code>	Parameter of the function, holds the maximum number of x -object to consider. Was 500 in the explanation above.	Considering every x -object would take a long time, and so some random sampling must be done.
x_object	<code>tuple[float]</code>	Iterator variable for when looping through <code>x_objects</code> .	Is the singular of <code>x_objects</code> .
y_object	<code>tuple[float]</code>	Iterator variable for when looping through <code>y_objects</code> .	Is the singular of <code>y_objects</code> .
dists_to_ys	<code>list[float]</code>	Holds the distances to each y -object for a particular x -object.	We must use this to find the minimum distance out of the distances stored.
min_dists	<code>list[float]</code>	Holds the minimum distances from each x -object to the nearest y -object.	$A(x, y)$ is given by the average of this list.

Table 2.1: List of Key Variables in $A(x, y)$

Variables in $D(x)$

Identifier	Data Type	Explanation	Justification
x_objects	<code>list[tuple[float]]</code>	Parameter of the function, holds a list of the coordinates of all the x -objects.	Each coordinate is a tuple as the latitudes and longitudes will never change.
bounding_coords	<code>list[tuple[float]]</code>	Parameter of the function, holds a list of the bounding coordinates of the region.	Each coordinate is a tuple as the latitudes and longitudes will never change.

<code>bounding_」 vectors</code>	<code>list[column_」 vector]</code>	Holds the <code>bounding_」 coords</code> once converted into column vectors.	This is so that we can use matrix multiplication to rotate them.
<code>current_」 vertex</code>	<code>int</code>	Iterator variable to keep track of which vertex I am dealing with.	Use it to get the vertex from the <code>bounding_」 vectors</code> as well as the previous and next one.
<code>transformed_」 vertices</code>	<code>list[column_」 vector]</code>	Holds the transformed <code>current_vertex</code> as well as the previous and next one.	We must compute the vectors and anticlockwise angles from this.
<code>anticlockwise_」 angles</code>	<code>list[float]</code>	Holds all of the anticlockwise angles at each vertex.	We need to sum this to determine if they are actually anticlockwise angles or not.
<code>interior_」 angles</code>	<code>list[float]</code>	Holds the interior angles of the region the user has drawn.	This is one of the terms in the formula to calculate the area of the region.
<code>area</code>	<code>float</code>	Holds the area of the region the user has drawn.	$D(x)$ is given by the length of <code>x_objects</code> divided by this.

Table 2.2: List of Key Variables in $D(x)$

The data structure `column_vector` does not come with python, but can be implemented with the `numpy` module.

Other Variables

Identifier	Data Type	Explanation	Justification
<code>current_」 region</code>	<code>int</code>	Iterator variable for iterating through the regions.	We must calculate all of the factors for each region.

dataset_path	<code>str</code>	Stores the file path to the external file for which the neural network will read the data from.	The neural network may need to be trained more than once, due to mistakes, so storing this in an external file would be beneficial.
--------------	------------------	---	---

Table 2.3: List of Other Key Variables in Data Collection & Processing

As the user never interacts with this part of the program, there is no validation that needs to be done (e.g. as stuff is automatically in the right type).

2.3 Multilayer Perceptron

All the information in this section, I learned from 3blue1brown's videos [6] and Micheal Nielsen's e-book [7] on the topic.

2.3.1 Structure

Layers & Neurons

The structure of a multilayer perceptron is similar to that of a graph, in the sense that it has a series of nodes (neurons) and edges (weights). The neurons are arranged in layers, the first of which being the input layer, the last being the output layer, and any in between being hidden layers.

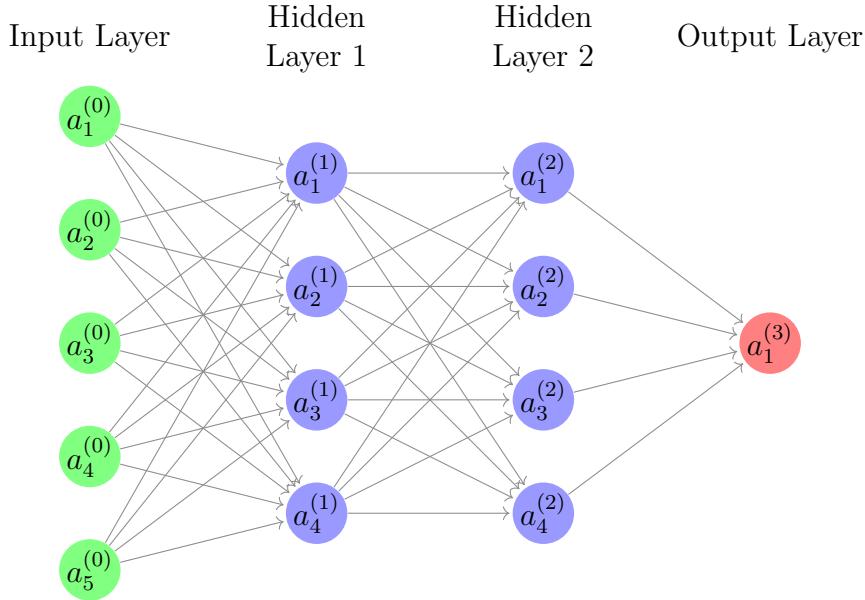


Figure 2.17: Neural Network Structure

Figure 2.17 shows an example neural network, with 5 neurons in the input layer (shown in green), 1 neuron in the output layer (shown in red), and 2 hidden layers each with 4 neurons (shown in blue). “Multilayer Perceptron” is just the fancy official name for this type of machine learning model.

Each neuron will hold a number, which will represent something. In the input layer, the neurons represent the various inputs to the system. In my case, they will be the various factors that may affect the HDI, so my neural network will have 24 neurons in the input layer. In the output layer, the neuron(s) will represent whatever output the system should have. For example, I only want 1 output, the HDI, so there is 1 neuron in the output layer which will hold the predicted HDI.

In the hidden layers, the hope is that the neurons will represent something intermediate in between the input and the output. For example, if the input neurons represent geographical factors, and the output neuron represents the HDI, the neurons in hidden layer 1 might represent the *gist* of how good the schooling is, or how good the healthcare is, and the neurons in hidden layer 2 might represent the *gist* of how these things work together to result in a higher HDI. This means that the original problem of calculating HDI from a set of factors has been decomposed into a series of smaller tasks.

As shown in Figure 2.17, I will use the notation $a_i^{(l)}$ to denote a particular neuron, where l is the index of the layer that it is in, and i is its index within the layer. In general, a network will have L layers, not including the input layer (as the layers are 0-indexed).

Weights

The value of each neuron (that isn't in the input layer) should be based on all the neurons of the previous layer, as shown by the arrows in Figure 2.17.

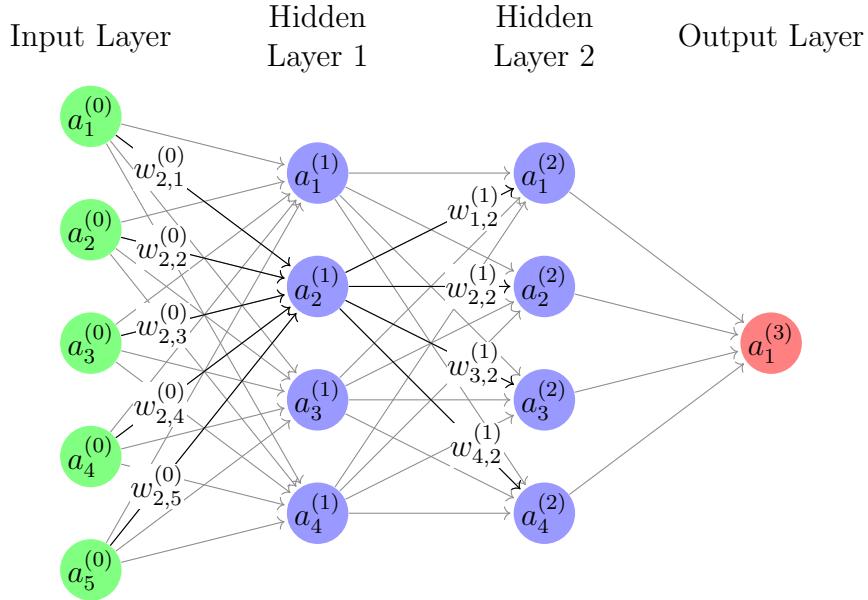


Figure 2.18: Neural Network Weights

I will use the notation $w_{i,j}^{(l)}$ to denote a particular weight, where l is the index of the layer it is pointing from, i is the neuron's index within its layer that the weight is pointing to, and j is the neuron's index within its layer that the weight is pointing from, as shown in Figure 2.18.

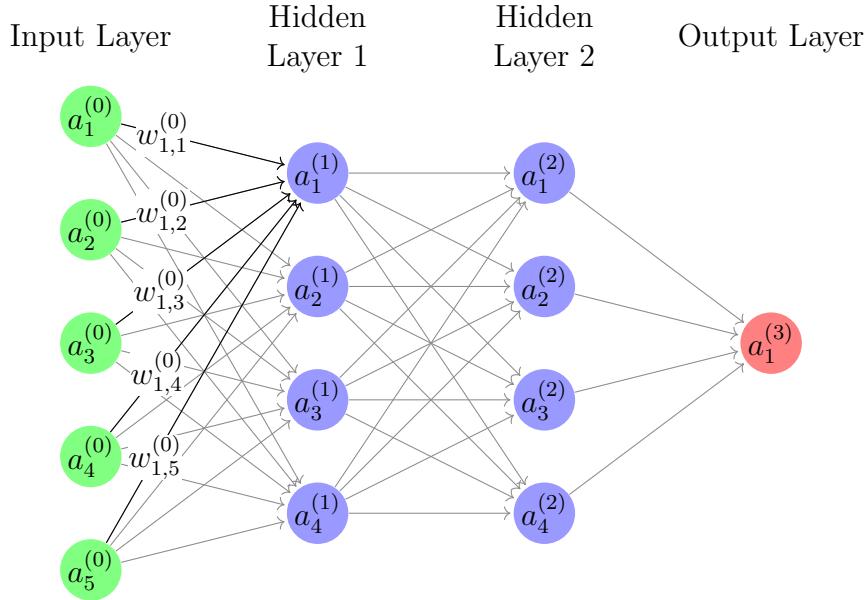
Each weight will also hold a value (any real number), which will not be calculated during prediction (like neurons' values are), and so the weights are parameters of the system. Initially, they will be set to random values, which will change during training.

2.3.2 Feedforward Algorithm & Prediction

Weighted Sum

Again, the value of each neuron (that isn't in the input layer) should be based on all the neurons of the previous layer. This can be done using a weighted sum. For example, the equation to calculate the value of $a_1^{(1)}$ in this example neural network would be:

$$a_1^{(1)} = w_{1,1}^{(0)}a_1^{(0)} + w_{1,2}^{(0)}a_2^{(0)} + w_{1,3}^{(0)}a_3^{(0)} + w_{1,4}^{(0)}a_4^{(0)} + w_{1,5}^{(0)}a_5^{(0)} \quad (2.20)$$

Figure 2.19: Calculating $a_1^{(1)}$ Example

In general, the equation to calculate the value of some neuron $a_i^{(l)}$ in any given network would be:

$$a_i^{(l)} = w_{i,1}^{(l-1)}a_1^{(l-1)} + w_{i,2}^{(l-1)}a_2^{(l-1)} + w_{i,3}^{(l-1)}a_3^{(l-1)} + \dots + w_{i,n}^{(l-1)}a_n^{(l-1)} \quad (2.21)$$

where n is the number of neurons in layer $l - 1$ (the previous layer).

Activation Function

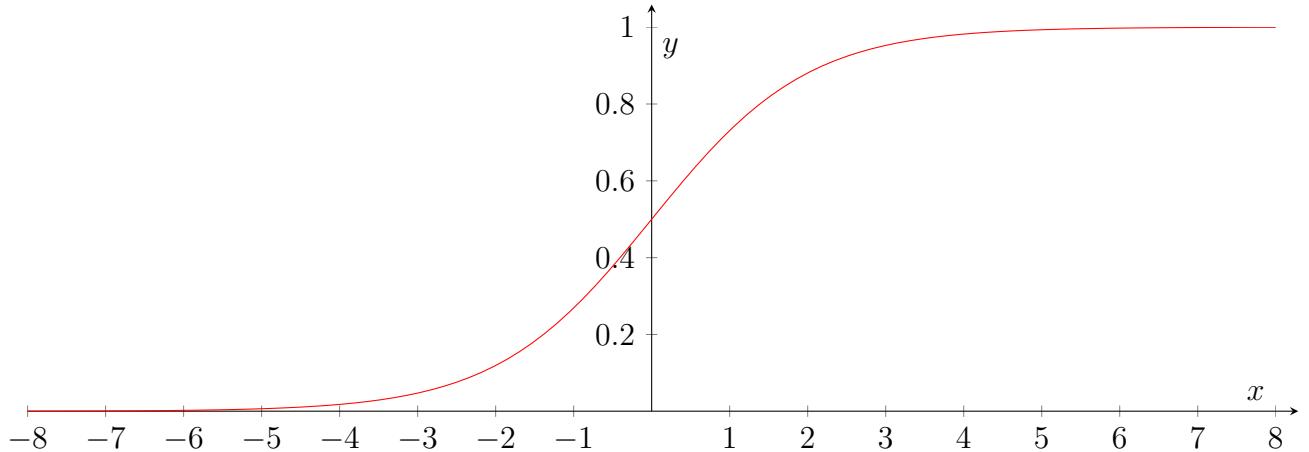
The value of each neuron should also represent how “activated” that neuron is, ideally on a continuous scale from 0 to 1. This is to match what happens in biological neural networks (brains), as this is effectively a model for that. The 0 to 1 range is also suitable in this particular instance, as HDI is also measured on a continuous scale from 0 to 1.

However, the result of this weighted sum could be any real number (as the weights can be any real number), so in order to squish it into the 0 to 1 range, it must be passed through some function, the activation function.

There are many activation functions that could be used, but the one I will use is $\sigma(x)$ (“sigmoid of x ”), which is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2.22)$$

Why this is suitable is best shown in the graph of $y = \sigma(x)$:

Figure 2.20: Graph of $y = \sigma(x)$

As shown in Figure 2.20, the domain of $\sigma(x)$ is \mathbb{R} (all real numbers), which is what the evaluation of the weighted sum could be, and the range is $0 < \sigma(x) < 1$, which is what we want it to be. Therefore, if the weighted sum is negative, then the activation of the neuron (its value) will be between 0 and 0.5, and if the weighted sum is positive, its activation will be between 0.5 and 1.

As the graph has asymptotes at $y = 0$ and $y = 1$ (which means that the function approaches it but never reaches it), very negative weighted sums will result in activations ≈ 0 , and very positive weighted sums will result in activations ≈ 1 . The smooth curve in the middle will also mean that the change from 0 to 1 is gradual, and there are no sudden jumps. (The function is also differentiable on its entire domain, which will be helpful in the Backpropogation Algorithm).

Therefore, the equation to calculate the activation of some neuron $a_i^{(l)}$ in any given network would now be:

$$a_i^{(l)} = \sigma \left(w_{i,1}^{(l-1)} a_1^{(l-1)} + w_{i,2}^{(l-1)} a_2^{(l-1)} + w_{i,3}^{(l-1)} a_3^{(l-1)} + \dots + w_{i,n}^{(l-1)} a_n^{(l-1)} \right) \quad (2.23)$$

where n is the number of neurons in layer $l - 1$.

Bias

A neuron may be considered “activated” if its value is ≥ 0.5 . For now, this is where the weighted sum is ≥ 0 , as $\sigma(0) = 0.5$. But what if a particular neuron should be activated when the weighted sum is (for example) ≥ 5 , or ≥ -87.2 ?

To achieve this, each neuron (that isn’t in the input layer) has a bias, which is another real number which is added onto the weighted sum before the activation function. I will use the notation $b_i^{(l)}$ to denote the bias of neuron $a_i^{(l)}$. The biases (like the weights) are also parameters of the system.

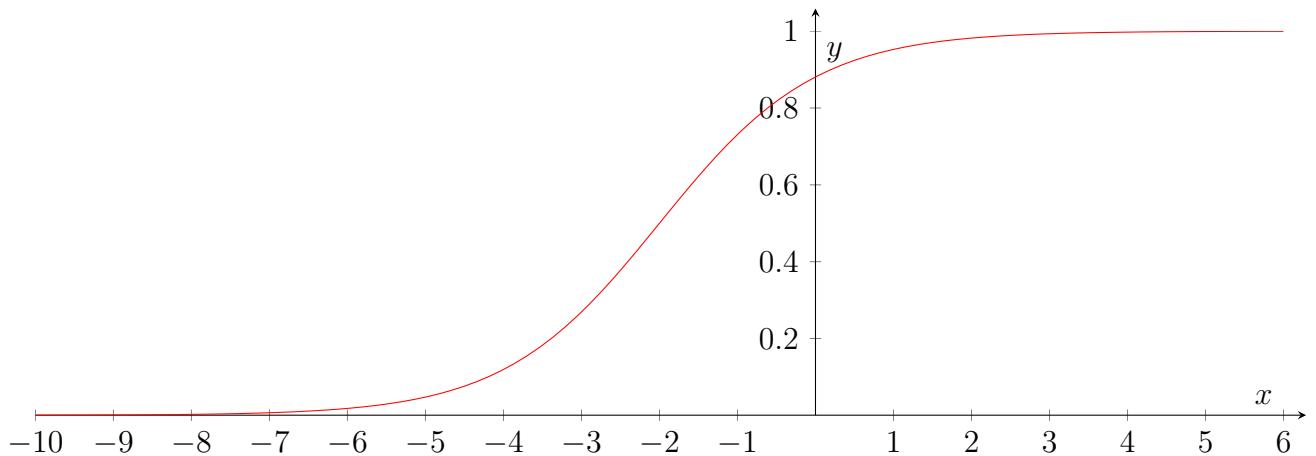
Figure 2.21: Graph of $y = \sigma(x + 2)$

Figure 2.21 shows the graph of $y = \sigma(x + 2)$, which shows how a particular neuron would be activated if its weighted sum is ≥ -2 .

Therefore, if a particular neuron $a_i^{(l)}$ should be activated if its weighted sum is $\geq p$, then $b_i^{(l)} = -p$. However, we have no way of knowing what this value of p should be for any of the neurons in the network, as what the neurons represent are too abstract for us to understand. Like the weights, they will be initially set to random values, and will change during training.

The final equation to calculate the activation of some neuron $a_i^{(l)}$ in any given network is:

$$a_i^{(l)} = \sigma \left(w_{i,1}^{(l-1)} a_1^{(l-1)} + w_{i,2}^{(l-1)} a_2^{(l-1)} + w_{i,3}^{(l-1)} a_3^{(l-1)} + \dots + w_{i,n}^{(l-1)} a_n^{(l-1)} + b_i^{(l)} \right) \quad (2.24)$$

where n is the number of neurons in layer $l - 1$.

Rearranging into Matrices & Column Vectors

Instead of calculating the activation of each neuron 1 by 1, consider calculating an entire layer of neurons at once.

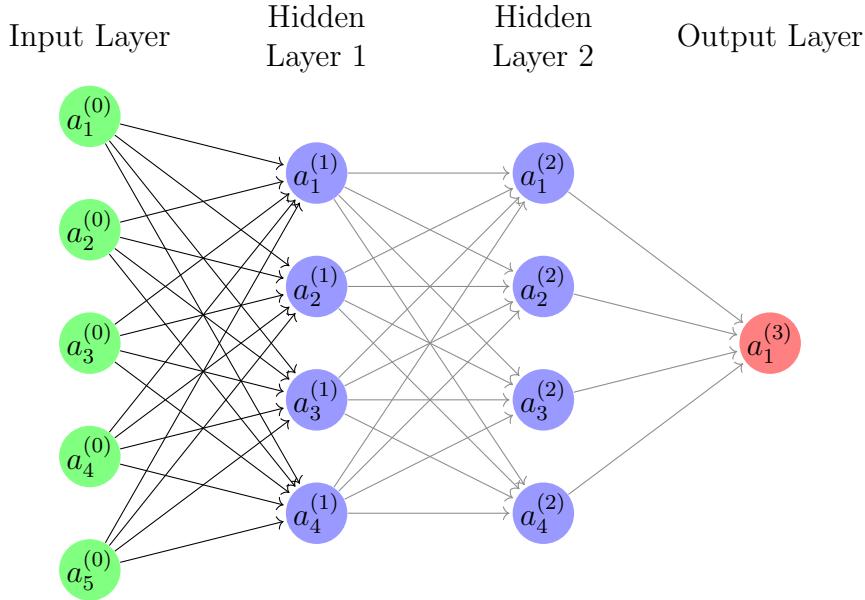


Figure 2.22: Calculating an entire layer of neurons

To do this, we can arrange each layer of neurons and their biases into column vectors (transposed arrays), $\mathbf{a}^{(l)}$ and $\mathbf{b}^{(l)}$ respectively, where l is the index of their layer:

$$\mathbf{a}^{(l)} = \begin{pmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_n^{(l)} \end{pmatrix}, \mathbf{b}^{(l)} = \begin{pmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_n^{(l)} \end{pmatrix} \quad (2.25)$$

where n is the number of neurons in layer l .

Arrange the weights in between each layer into a matrix (2D array), $\mathbf{W}^{(l)}$, where l is the layer the weights are pointing from:

$$\mathbf{W}^{(l)} = \begin{pmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \cdots & w_{1,n}^{(l)} \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \cdots & w_{2,n}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1}^{(l)} & w_{m,2}^{(l)} & \cdots & w_{m,n}^{(l)} \end{pmatrix} \quad (2.26)$$

where n is the number of neurons in layer l and m is the number of neurons in layer $l + 1$.

Consider the matrix multiplication, $\mathbf{W}^{(l)}\mathbf{a}^{(l)}$:

$$\begin{pmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \cdots & w_{1,n}^{(l)} \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \cdots & w_{2,n}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1}^{(l)} & w_{m,2}^{(l)} & \cdots & w_{m,n}^{(l)} \end{pmatrix} \begin{pmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_n^{(l)} \end{pmatrix} = \begin{pmatrix} w_{1,1}^{(l)}a_1^{(l)} + w_{1,2}^{(l)}a_2^{(l)} + \cdots + w_{1,n}^{(l)}a_n^{(l)} \\ w_{2,1}^{(l)}a_1^{(l)} + w_{2,2}^{(l)}a_2^{(l)} + \cdots + w_{2,n}^{(l)}a_n^{(l)} \\ \vdots \\ w_{m,1}^{(l)}a_1^{(l)} + w_{m,2}^{(l)}a_2^{(l)} + \cdots + w_{m,n}^{(l)}a_n^{(l)} \end{pmatrix} \quad (2.27)$$

The result is a column vector, where each item is a weighted sum of all the neurons in that layer, which is almost the correct column vector for the activations of the neurons in the next layer! To make it correct, we just need to add the bias column vector for the next layer, and pass it through the activation function.

Therefore, the equation to calculate the activations of all the neurons in the next layer, $l + 1$ is:

$$\begin{pmatrix} a_1^{(l+1)} \\ a_2^{(l+1)} \\ \vdots \\ a_m^{(l+1)} \end{pmatrix} = \sigma \left(\begin{pmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \cdots & w_{1,n}^{(l)} \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \cdots & w_{2,n}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1}^{(l)} & w_{m,2}^{(l)} & \cdots & w_{m,n}^{(l)} \end{pmatrix} \begin{pmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_n^{(l)} \end{pmatrix} + \begin{pmatrix} b_1^{(l+1)} \\ b_2^{(l+1)} \\ \vdots \\ b_m^{(l+1)} \end{pmatrix} \right) \quad (2.28)$$

where n is the number of neurons in layer l , m is the number of neurons in layer $l + 1$ and σ of a column vector = σ of all the items in the column vector.

This can be simply written as:

$$\mathbf{a}^{(l+1)} = \sigma(\mathbf{W}^{(l)}\mathbf{a}^{(l)} + \mathbf{b}^{(l+1)}) \quad (2.29)$$

where σ of a column vector = σ of all the items in the column vector, which is the formula that I will be using in my program.

Final Algorithm

In order to fully make a prediction, we just need to iterate this process over all the layers:

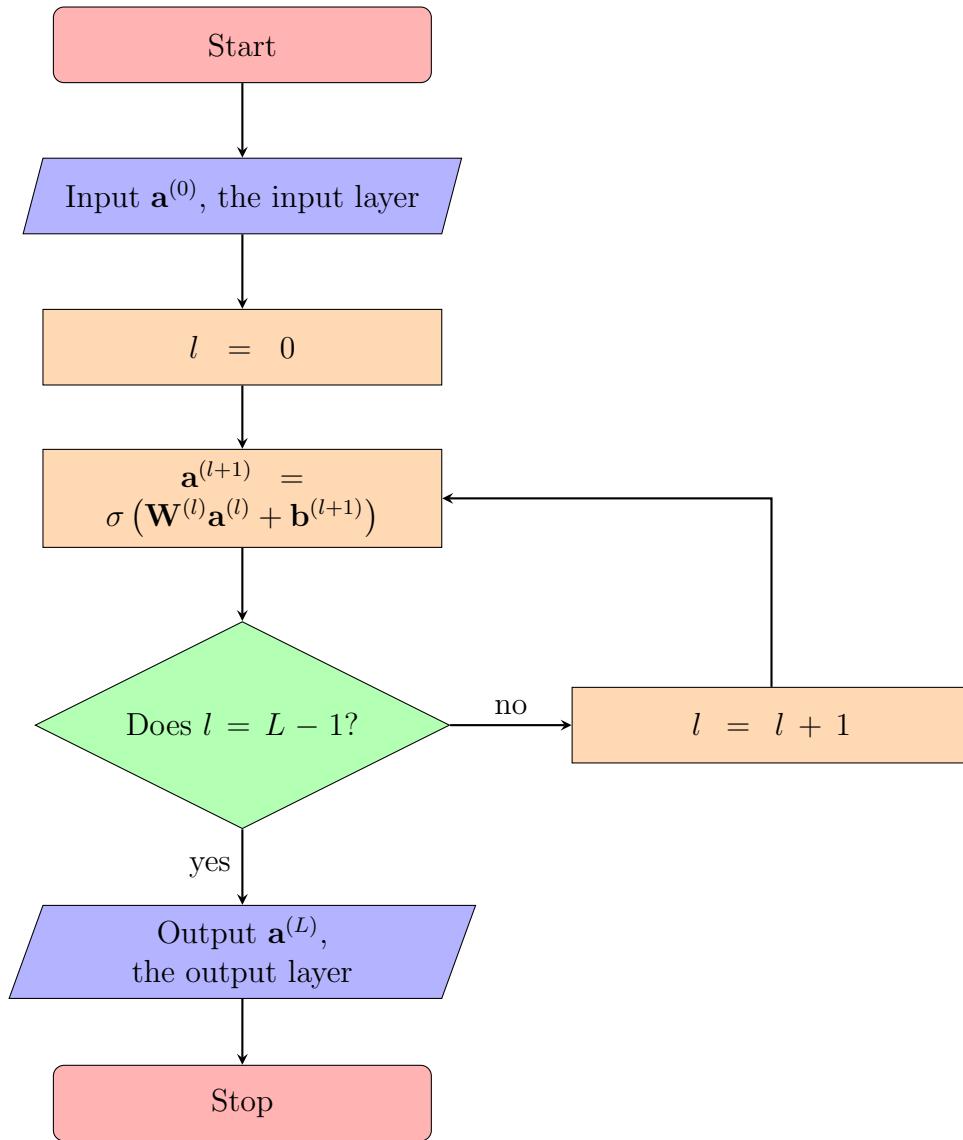


Figure 2.23: Feedforward Algorithm Flowchart

2.3.3 Backpropagation Algorithm & Training

The Cost Function & Gradient Descent

This feedforward algorithm to make predictions works, but if you were to test it right now it would return absolute nonsense. This is because all the weights and biases of the network are initially set to random values, so in reality the network isn't doing anything in particular. In order for it to recognise patterns within factors which may contribute to HDI, it must be trained on a set of examples. The network should also know after a prediction how far off it was from the correct answer.

Therefore, define the “cost” of a single example, c to be:

$$c = \sum_{i=1}^n \frac{1}{2} (a_i^{(L)} - y_i)^2 \quad (2.30)$$

where $a_i^{(L)}$ is the activation of the i^{th} neuron in the output layer, y_i is the “correct” activation, as given by the training example, and n is the number of neurons in layer L . As I will only have 1 neuron in the output layer, this simplifies to:

$$c = \frac{1}{2} (a_1^{(L)} - y)^2 \quad (2.31)$$

where $a_1^{(L)}$ is the predicted HDI and y is the true HDI. The difference between the true value and the predicted value is squared to always make it positive, as some predictions may be less than the true value and others may be greater than the true value.

Similarly to the activation function, there are many ways of defining the cost, c . I have chosen this one as it is simple to understand how it works, it is very quick to compute (as all you have to do is subtract 2 values, square and half) and it is differentiable on its entire domain, which will be helpful later.

Now define the cost function, C , to be the average cost over many pieces of training data, in terms of all the weights and biases:

$$C(w_{1,1}^{(0)}, \dots, w_{i,j}^{(L-1)}, b_1^{(1)}, \dots, b_i^{(L)}) = ??? \text{ (depends on what the network does)} \quad (2.32)$$

The goal is to find a minimum to this function, as when this function returns a small value, that means that all the predictions that the network gave were close to the true values.

We can do this by using gradient descent. This is where you find the gradient of the function, and then take a small step in the direction such that the gradient will decrease the most. This will work as the gradient at minimums of functions = 0. In other words, we must find out how much each of the weights and biases affects the cost, and then using this information, change the weights and biases such that the cost will decrease.

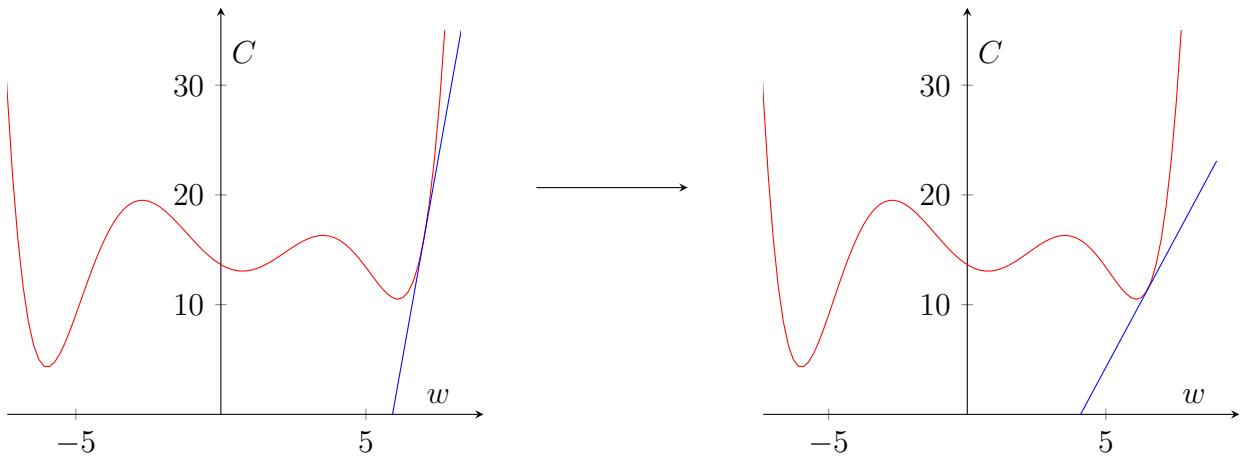


Figure 2.24: Gradient Descent Example

Figure 2.24 shows how C may vary with some weight w , and how gradient descent will be used to find a minimum of C with respect w . Note that it may not converge on the absolute minimum, but it will converge onto some local minimum.

Error of a Neuron, $\delta_i^{(l)}$

To do this, it will help to find out how much each neuron affects the output of the network, or the cost. This will be called the error of a neuron.

First, define $z_i^{(l)}$ to be neuron $a_i^{(l)}$'s activation before it was passed into the activation function:

$$z_i^{(l)} = w_{i,1}^{(l-1)}a_1^{(l-1)} + w_{i,2}^{(l-1)}a_2^{(l-1)} + w_{i,3}^{(l-1)}a_3^{(l-1)} + \dots + w_{i,n}^{(l-1)}a_n^{(l-1)} + b_i^{(l)} \quad (2.33)$$

Now we can define the error of a neuron $a_i^{(l)}$, $\delta_i^{(l)}$ to be:

$$\delta_i^{(l)} = \frac{dC}{dz_i^{(l)}} \quad (2.34)$$

which represents the quantity answered by the question: “*If I make a tiny change to $z_i^{(l)}$, $\Delta z_i^{(l)}$ what is the resulting change in C ?*”

If $\delta_i^{(l)}$ is close to 0, there isn't much we can do to $z_i^{(l)}$ to have a big effect on C . On the other hand, if $\delta_i^{(l)}$ is large, then we only have to change $z_i^{(l)}$ a little bit to have a big effect on C . Therefore, there is a heuristic sense of how $\delta_i^{(l)}$ measures the sensitivity, or error, of a neuron.

The plan is to find the error of the output layer, then backpropagate that to find the error of previous layers, and finally find the change in the cost function C with respect to the weights and biases in terms of these errors.

Error of the Output Layer, $\delta^{(L)}$

The error of the neurons in the output layer will therefore be given by:

$$\delta_i^{(L)} = \frac{dC}{dz_i^{(L)}} \quad (2.35)$$

as L is the number of layers in the network (not including the input layer), and therefore represents the output layer. However, this is not easily computable, and should be rearranged into something that is.

We can break up this expression using the chain rule for differentiation:

$$\delta_i^{(L)} = \frac{dC}{da_i^{(L)}} \frac{da_i^{(L)}}{dz_i^{(L)}} \quad (2.36)$$

As $a_i^{(l)} = \sigma(z_i^{(l)})$ by definition (from equations 2.24 and 2.33),

$$\frac{da_i^{(L)}}{dz_i^{(L)}} = \sigma'(z_i^{(L)}) \quad (2.37)$$

where $\sigma'(x)$ is the derivative of the sigmoid function.

As the output of C is given by $\sum_{i=1}^n \frac{1}{2} \left(a_i^{(L)} - y_i \right)^2$ by definition (from equation 2.30),

$$\frac{dC}{da_i^{(L)}} = a_i^{(L)} - y_i \quad (2.38)$$

Therefore, from equations 2.36, 2.37 and 2.38,

$$\delta_i^{(L)} = \left(a_i^{(L)} - y_i \right) \left(\sigma' \left(z_i^{(L)} \right) \right) \quad (2.39)$$

which is all easily computable.

As done before with $\mathbf{a}^{(l)}$ and $\mathbf{b}^{(l)}$, arrange each of the z -values and errors into column vectors, $\mathbf{z}^{(l)}$ and $\boldsymbol{\delta}^{(l)}$, where l is the index of their layer:

$$\mathbf{z}^{(l)} = \begin{pmatrix} z_1^{(l)} \\ z_2^{(l)} \\ \vdots \\ z_n^{(l)} \end{pmatrix}, \boldsymbol{\delta}^{(l)} = \begin{pmatrix} \delta_1^{(l)} \\ \delta_2^{(l)} \\ \vdots \\ \delta_n^{(l)} \end{pmatrix} \quad (2.40)$$

where n is the number of neurons in layer l .

$\boldsymbol{\delta}^{(L)}$ must therefore be given by:

$$\boldsymbol{\delta}^{(L)} = (\mathbf{a}^{(L)} - \mathbf{y}) \odot (\sigma'(\mathbf{z}^{(L)})) \quad (2.41)$$

where \mathbf{y} is the column vector of the expected outputs and \odot represents the operation to compute the element-wise product of 2 column vectors with the same dimension. For example,

$$\begin{pmatrix} 2 \\ 6 \\ 3 \end{pmatrix} \odot \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix} = \begin{pmatrix} 6 \\ 24 \\ 15 \end{pmatrix} \quad (2.42)$$

Error of the Previous Layer in terms of the current one

Now that we have the error of the output layer, $\boldsymbol{\delta}^{(L)}$, we can attempt to backpropagate by finding the error of each layer before it. In other words, we want $\boldsymbol{\delta}^{(l-1)}$ in terms of $\boldsymbol{\delta}^{(l)}$ and other easily computable terms. Consider $\delta_i^{(l-1)}$:

$$\delta_i^{(l-1)} = \frac{dC}{dz_i^{(l-1)}} \quad (2.43)$$

We can again break this up using the chain rule:

$$\delta_i^{(l-1)} = \sum_{j=1}^n \frac{dC}{dz_j^{(l)}} \frac{dz_j^{(l)}}{dz_i^{(l-1)}} \quad (2.44)$$

except this time we must sum over each j from 1 to n , where n is the number of neurons in layer l . This is because the activation of all the neurons in layer l is based on $z_i^{(l-1)}$, so we must add up all the small changes to C from layer l to calculate the overall effect from $z_i^{(l-1)}$.

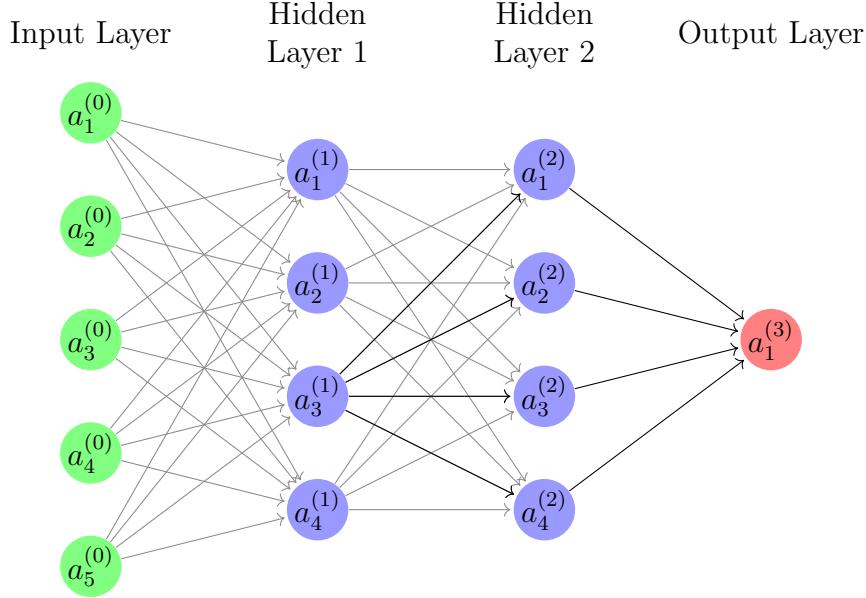


Figure 2.25: $z_i^{(l-1)}$ Affects all neurons in layer l , which all affect C

By definition, $\frac{dC}{dz_j^{(l)}} = \delta_j^{(l)}$, so equation 2.44 simplifies to:

$$\delta_i^{(l-1)} = \sum_{j=1}^n \frac{d z_j^{(l)}}{d z_i^{(l-1)}} \delta_j^{(l)} \quad (2.45)$$

Using equation 2.33, we can write $z_j^{(l)}$ as:

$$z_j^{(l)} = w_{j,1}^{(l-1)} a_1^{(l-1)} + w_{j,2}^{(l-1)} a_2^{(l-1)} + \cdots + w_{j,i}^{(l-1)} a_i^{(l-1)} + \cdots + w_{j,n}^{(l-1)} a_n^{(l-1)} + b_j^{(l)} \quad (2.46)$$

where n is the number of neurons in layer $l - 1$ and i is some index of a particular neuron in the previous layer which we care about.

As $a_i^{(l-1)} = \sigma(z_i^{(l-1)})$ by definition (from equations 2.24 and 2.33),

$$z_j^{(l)} = w_{j,1}^{(l-1)} a_1^{(l-1)} + w_{j,2}^{(l-1)} a_2^{(l-1)} + \cdots + w_{j,i}^{(l-1)} \sigma(z_i^{(l-1)}) + \cdots + w_{j,n}^{(l-1)} a_n^{(l-1)} + b_j^{(l)} \quad (2.47)$$

Therefore,

$$\frac{d z_j^{(l)}}{d z_i^{(l-1)}} = w_{j,i}^{(l-1)} \sigma'(z_i^{(l-1)}) \quad (2.48)$$

Substituting that into equation 2.45 gives:

$$\delta_i^{(l-1)} = \sum_{j=1}^n w_{j,i}^{(l-1)} \sigma' \left(z_i^{(l-1)} \right) \delta_j^{(l)} \quad (2.49)$$

Consider the matrix multiplication, $(\mathbf{W}^{(l-1)})^T \boldsymbol{\delta}^{(l)}$, where \mathbf{M}^T represents the transposed matrix of matrix \mathbf{M} :

$$\begin{pmatrix} w_{1,1}^{(l-1)} & w_{2,1}^{(l-1)} & \cdots & w_{n,1}^{(l-1)} \\ w_{1,2}^{(l-1)} & w_{2,2}^{(l-1)} & \cdots & w_{n,2}^{(l-1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,m}^{(l-1)} & w_{2,m}^{(l-1)} & \cdots & w_{n,m}^{(l-1)} \end{pmatrix} \begin{pmatrix} \delta_1^{(l)} \\ \delta_2^{(l)} \\ \vdots \\ \delta_n^{(l)} \end{pmatrix} = \begin{pmatrix} w_{1,1}^{(l-1)} \delta_1^{(l)} + w_{2,1}^{(l-1)} \delta_2^{(l)} + \cdots + w_{n,1}^{(l-1)} \delta_n^{(l)} \\ w_{1,2}^{(l-1)} \delta_1^{(l)} + w_{2,2}^{(l-1)} \delta_2^{(l)} + \cdots + w_{n,2}^{(l-1)} \delta_n^{(l)} \\ \vdots \\ w_{1,m}^{(l-1)} \delta_1^{(l)} + w_{2,m}^{(l-1)} \delta_2^{(l)} + \cdots + w_{n,m}^{(l-1)} \delta_n^{(l)} \end{pmatrix} \quad (2.50)$$

Again, this is almost the result that we want as we just need to multiply each term by $\sigma' \left(z_i^{(l-1)} \right)$ to get a column vector of elements in the form of equation 2.49. Therefore, the equation to backpropagate the error from one layer to the previous is given by:

$$\boldsymbol{\delta}^{(l-1)} = \left((\mathbf{W}^{(l-1)})^T \boldsymbol{\delta}^{(l)} \right) \odot \left(\sigma' \left(\mathbf{z}^{(l-1)} \right) \right) \quad (2.51)$$

which again, is all easily computable.

Gradient of the Cost Function

Now that we have $\delta_i^{(l)}$ for any given l and i in the network in terms of easily computable terms, we can try and find the change in the cost function C with respect to all the weights and biases, which is the gradient of the cost function, which is what we need to train the model. In other words, we are trying to find $\frac{dC}{dw_{i,j}^{(l)}}$ and $\frac{dC}{db_i^{(l)}}$ for any l, i, j in the network, in terms of errors, in order to take a small step in the opposite direction (as to decrease the gradient and find a minimum).

We can again break up these expressions using the chain rule into:

$$\frac{dC}{dz_i^{(l+1)}} \frac{dz_i^{(l+1)}}{dw_{i,j}^{(l)}} \text{ and } \frac{dC}{dz_i^{(l)}} \frac{dz_i^{(l)}}{db_i^{(l)}} \text{ respectively} \quad (2.52)$$

By definition, $\frac{dC}{dz_i^{(l)}} = \delta_i^{(l)}$ so this simplifies to:

$$\frac{dz_i^{(l+1)}}{dw_{i,j}^{(l)}} \delta_i^{(l+1)} \text{ and } \frac{dz_i^{(l)}}{db_i^{(l)}} \delta_i^{(l)} \text{ respectively} \quad (2.53)$$

From equation 2.33,

$$\frac{dz_i^{(l+1)}}{dw_{i,j}^{(l)}} = a_j^{(l)} \text{ and } \frac{dz_i^{(l)}}{db_i^{(l)}} = 1 \quad (2.54)$$

Therefore, from equations 2.53 and 2.54,

$$\frac{dC}{dw_{i,j}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)} \text{ and } \frac{dC}{db_i^{(l)}} = \delta_i^{(l)} \quad (2.55)$$

which is what we need.

Therefore, for a given training example index x , we must change $\mathbf{b}^{(l)}$ by an amount $-\eta \boldsymbol{\delta}_x^{(l)}$ where $\boldsymbol{\delta}_x^{(l)}$ the error for the neurons in layer l for the specific training example x (this notation also follows for \mathbf{a} and \mathbf{z}) and η is some constant, the learning rate. The negative sign is involved as to carry out gradient descent we need to take a step in the direction opposite to the gradient.

Similarly, we must change $\mathbf{W}^{(l)}$ by an amount $-\eta \boldsymbol{\delta}_x^{(l+1)} (\mathbf{a}_x^{(l)})^T$. The activations column vector has been transposed so that when the error column vector is multiplied by it the result is a matrix with the same dimensions of the weight matrix.

Final Algorithm

Now all we need to do is iterate this process over multiple training examples and then change the gradient by an average over all the pieces of training data.

Therefore, the equations to update the weights and biases are:

$$\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \frac{\eta}{m} \sum_{x=1}^m \boldsymbol{\delta}_x^{(l+1)} (\mathbf{a}_x^{(l)})^T \quad (2.56)$$

and

$$\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \frac{\eta}{m} \sum_{x=1}^m \boldsymbol{\delta}_x^{(l)} \quad (2.57)$$

where m is the number of training examples.

Each training example will be given in the format (\mathbf{x}, \mathbf{y}) , where \mathbf{x} is a column vector of activations for the input layer and \mathbf{y} is a column vector for the expected outputs for the output layer.

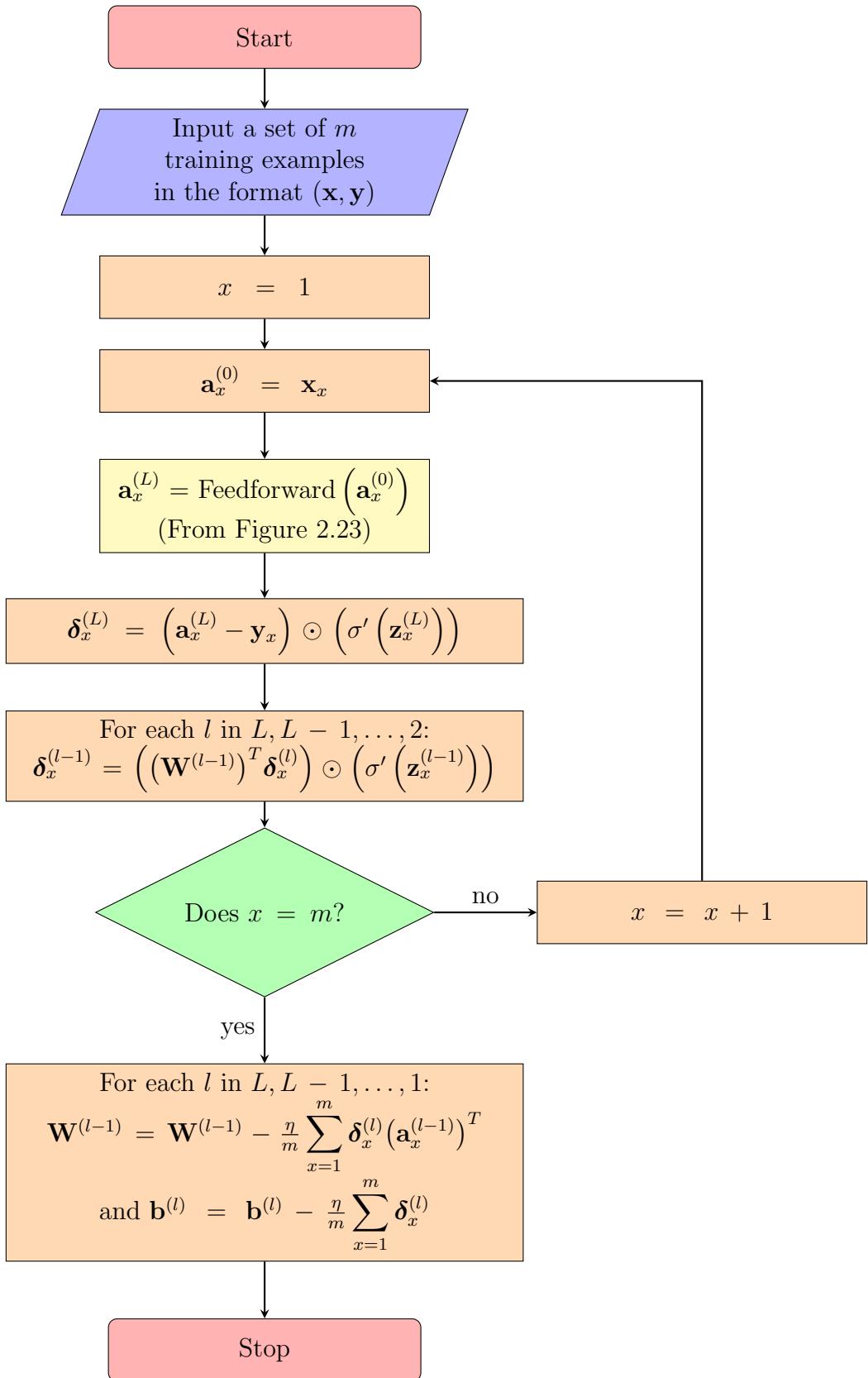


Figure 2.26: Backpropagation Algorithm Flowchart

2.3.4 Stochastic Gradient Descent

In practice, averaging over all the training examples is not very efficient, as you have to create a copy of the column vectors \mathbf{a} , \mathbf{z} and $\boldsymbol{\delta}$ for each layer and training example, which can get very large very fast.

To combat this, instead break up the training data into mini batches which are easier to handle, while still having enough to represent a range of things that the network may want to look for. The training examples are also randomly shuffled before being sorted into mini batches to ensure that the contents of each one are different every time.

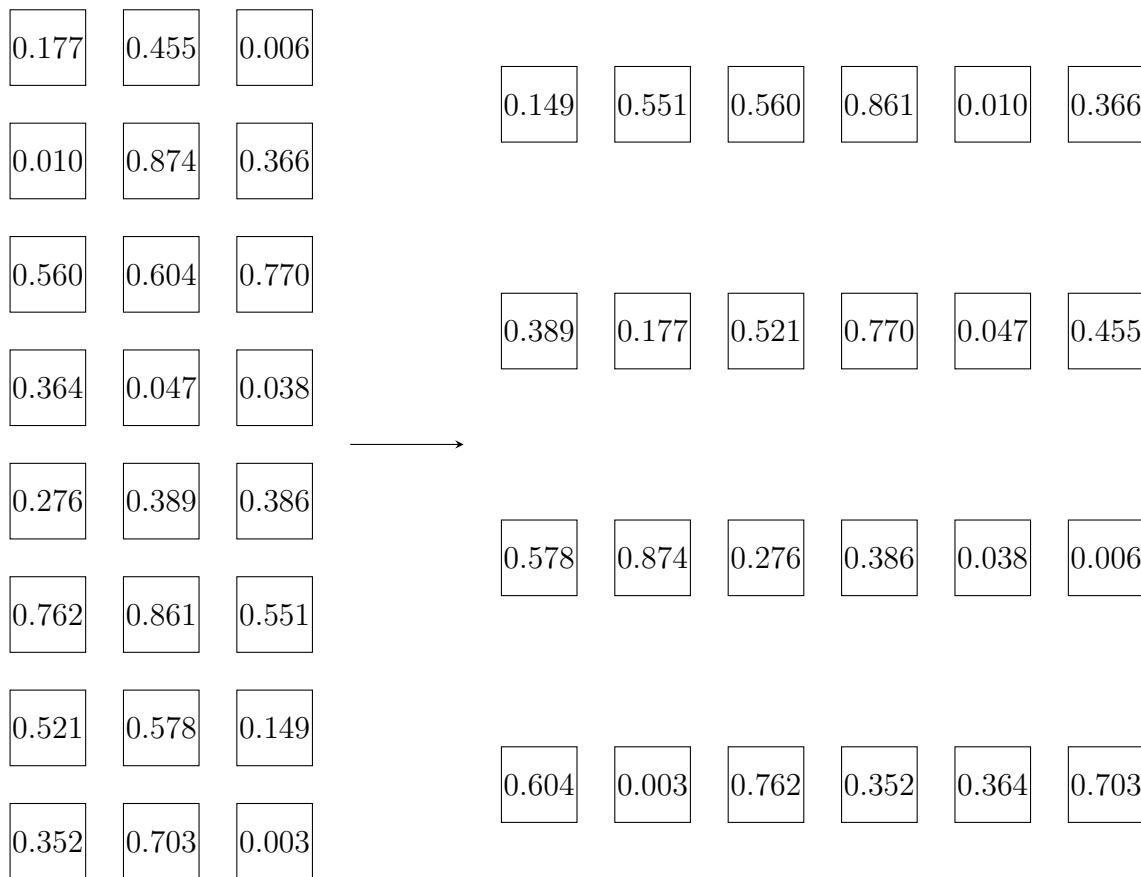


Figure 2.27: Shuffling into Mini Batches

When carrying out the gradient descent, this will have the effect of taking a step in some direction close to that which decreases the gradient by the most, instead of going directly in that direction. This is called Stochastic Gradient Descent.

Finally, this entire process should be repeated multiple times, over a number of “epochs”, in order to ensure that it has enough time to learn.

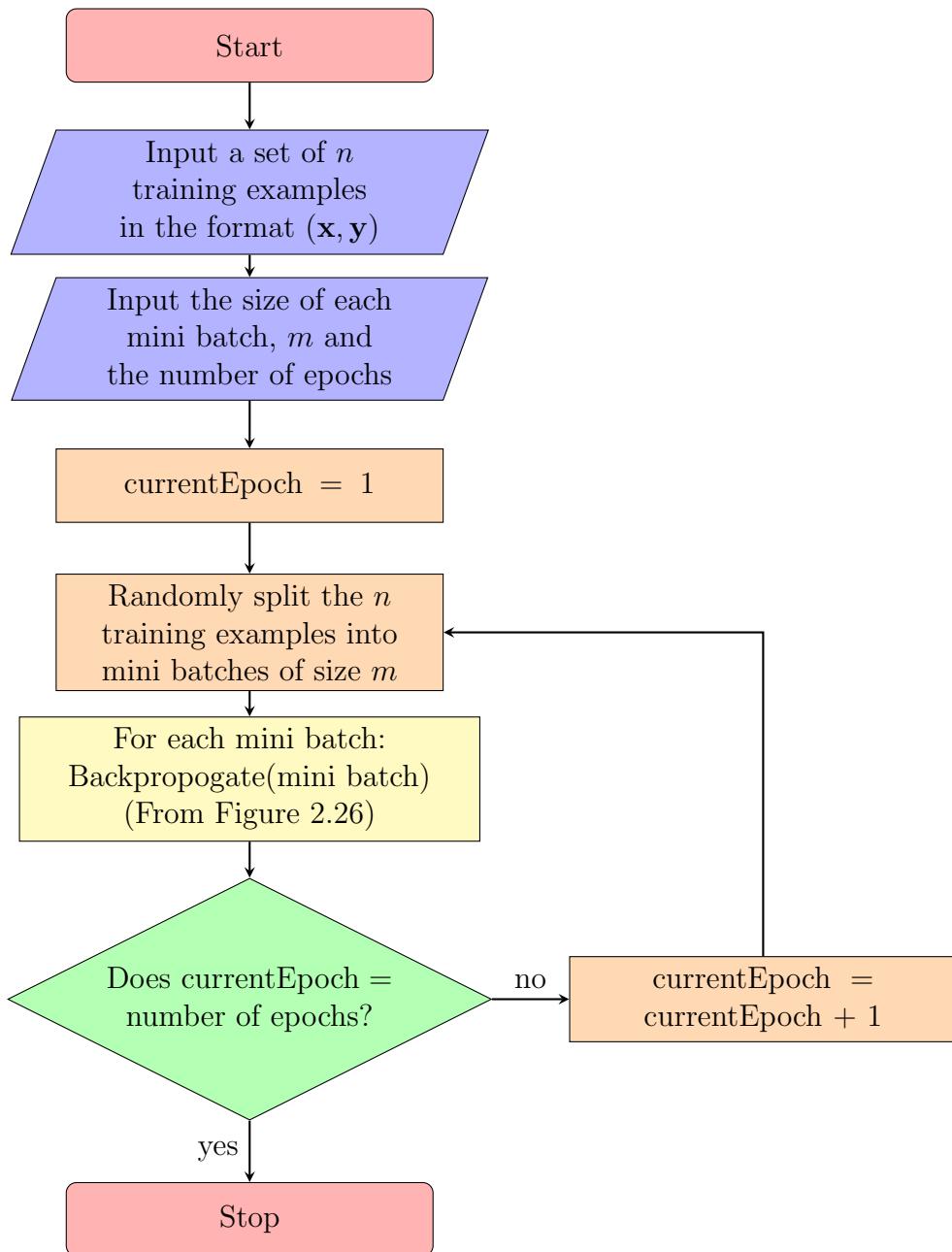


Figure 2.28: Full Training Algorithm Flowchart

2.3.5 Class Diagram

```

class MultilayerPerceptron(object)
layer_sizes: list[int]
L: int
activations: list[column_vector]
weights: list[matrix]
biases: list[column_vector]
z: list[column_vector]
errors: list[column_vector]
training_activations: list[list[column_vector]]
training_z: list[list[column_vector]]
training_errors: list[list[column_vector]]
def __init__(self, layer_sizes)
def predict(self, input_data)
def train(self, examples, mini_batch_size, num_epochs, learning_rate)
def feedforward(self, a0)
def backpropogate(self, examples)
def sigmoid(self, x)
def sigmoid_prime(self, x)
def save_model(self, filename)
def load_model(self, file_path)

```

Figure 2.29: Multilayer Perceptron Class Diagram

The attribute `layer_sizes` is a list of integers, which represent the number of neurons in each layer. The attributes `L` and `z` represent the respective variables in the mathematical explanation above, `activations` represents the \mathbf{a} vectors, `weights` represents the \mathbf{W} matrices, `biases` represents the \mathbf{b} vectors, and `errors` represents the δ vectors. The data structures `column_vector` and `matrix` are not built into python, but come with the library `numpy`. The attributes `training_activations`, `training_z` and `training_errors` will hold the values for each of the training examples in a mini batch, as all of them need to be computed for each example before the gradient is calculated (which is why they are in an extra `list`).

The constructor method (`def __init__`) only takes in the layer sizes, as everything else will either be defined during training or prediction, or be initially set to random values. The other methods are as you would expect, with functions for prediction and training that are explained above. There are also the methods `def save_model` and `def load_model`, as the model will be trained beforehand and then the GUI will query that model. Therefore, we need a way of saving the weights and biases to an external file and then loading them from it at a later time.

All the methods will be public functions, as python does not allow for anything else. However, in theory, they would all be procedures except for `def predict`, `def sigmoid` and `def sigmoid_prime`, which will all return `floats`, while the rest will return `None`.

There is not a “List of key variables” for this section, as all the results of calculations will

be stored in the attributes in the `class MultilayerPerceptron`, and have therefore been described above.

2.4 User Interface

There are many features I would like to add eventually, which I have included in the Structure Diagram (Figure 2.1). I will not be including these in this section, and will be saving them for the next prototype. I will only be planning for the user entering their region on a map, the HDI being predicted, and the suggestions for improvements being made, as I believe these are the most essential feautures.

2.4.1 Interface Sketch

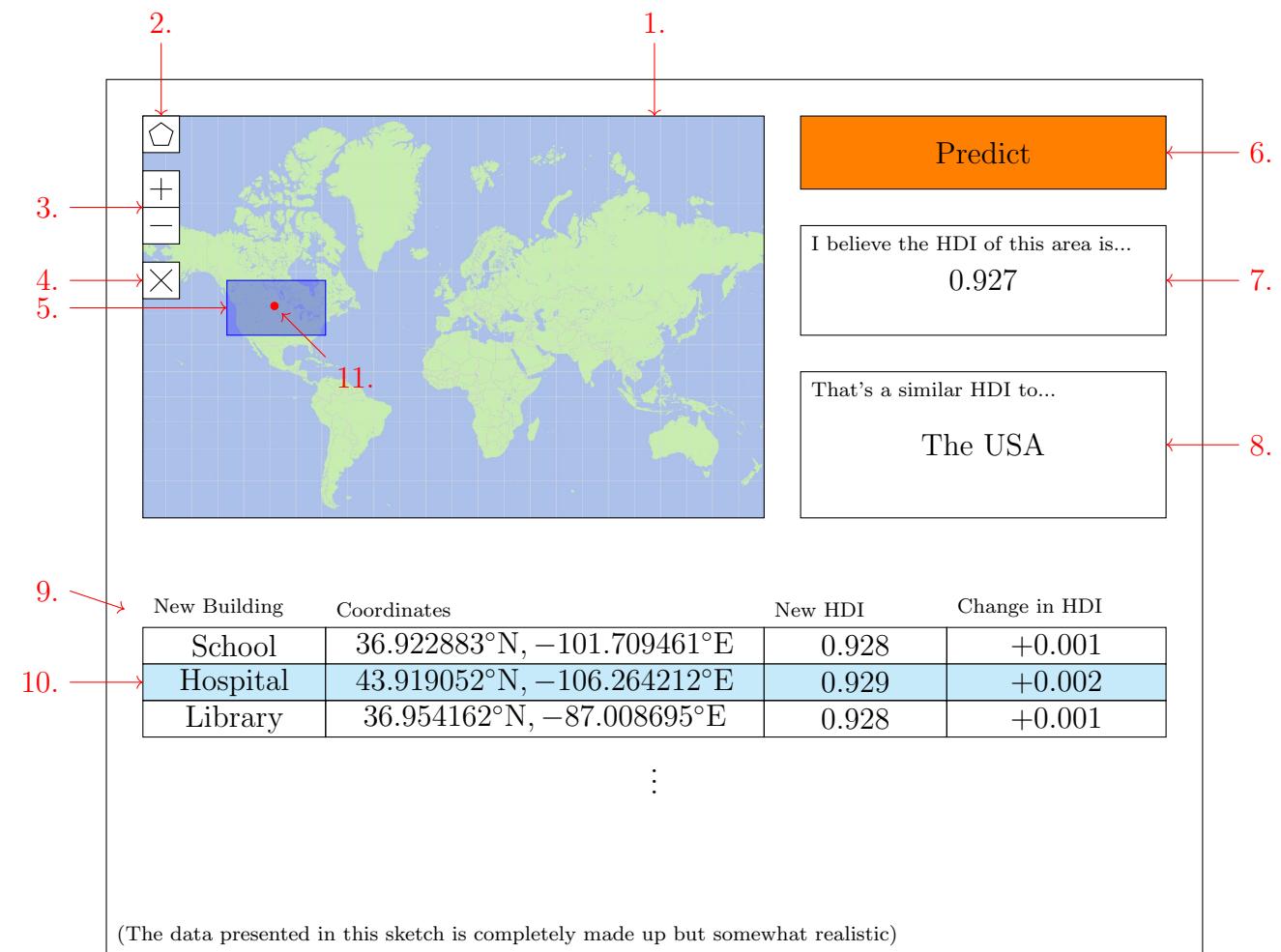
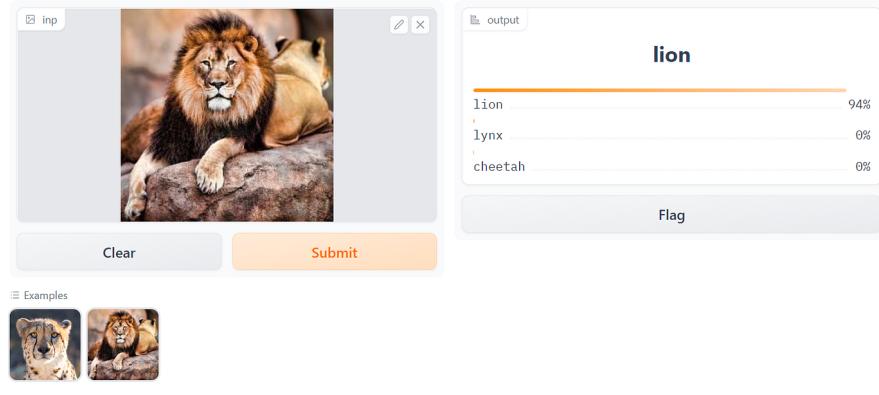


Figure 2.30: Technical Interface Sketch

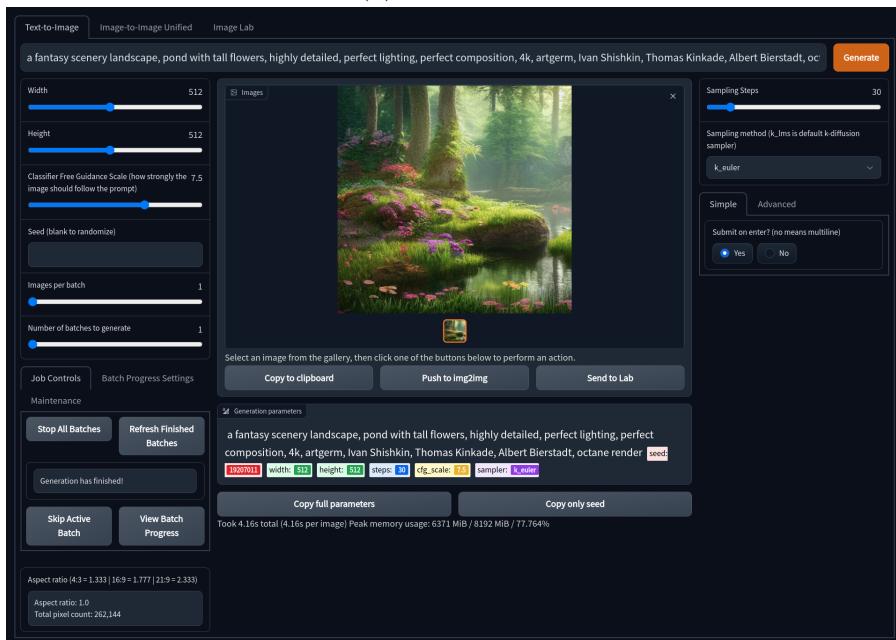
The labelled items shown in Figure 2.30 are:

1. Map for the user to draw on
2. Button to toggle between using the mouse for polygon drawing and moving
3. Zoom in and out buttons
4. Cancel Button
5. Example area the user may draw
6. Predict Button
7. Predicted HDI
8. A comparison to a country with a similar HDI
9. A list of suggestions to improve the HDI
10. A selected suggestion to view in more detail (that the user has selected)
11. The location on the map for the selected suggestion

Figure 2.30 shows the general structure of the user interface, while the style will be provided by the `gradio` python module. Examples of the style can be seen in Figure 2.31. It will automatically adapt to light mode or dark mode based on the device's or browser's settings.



(a) Light Mode



(b) Dark Mode

Figure 2.31: Interface Style Examples

Additional features that will be added in later prototypes (such as an analysis of how much each factor affects the others or entering factors manually) will be accessed from different tabs at the top of the interface.

2.4.2 Input Validation

As the only input the user is providing is that of drawing on the map to select their region (for now), only that needs to be validated. I will ensure that the user can only draw polygons on the map, the polygons must be completed, and the polygons cannot self intersect. The mapping module I will be using, `folium` already accounts for this, except for the self

intersection. However, this can be detected by using part of the $D(x)$ algorithm, as shown in Figure 2.15.

The user should also only be able to interact with the map and any data visualisations. The GUI module I will be using, `gradio` also already accounts for this.

2.4.3 Usability Features

Map & Map Buttons

The map should be the biggest thing on the screen, as it is the main source of input the user will be providing. It should also be as big as possible to accommodate those with smaller screens, so that they can still accurately pinpoint locations.

The button to draw a polygon on the map will be displayed as a regular pentagon. This is because there are often distinct tools to draw rectangles or triangles, which would be represented by regular 4-sided or 3-sided polygons respectively. Therefore, if a regular 5-sided polygon is used (a pentagon) there is less likely to be confusion about what the button does, while still having a small number of sides (a large number of sides will look too similar to a circle). It also suggests that the user will not be able to draw freely (as this is often represented by a pencil), which they will not be able to do.

The buttons to zoom in and out will be represented by a plus and minus sign. This is also the standard design for this type of button. The zoom in button and zoom out buttons will be right next to each other, to minimise the consequence in accidentally zooming in/out one too many times.

The button to cancel will be represented by an X, as this is also a common symbol used for this type of button. It will be in the same approximate location as the other 3 buttons, as they all have similar functions.

The region the user draws will be shown in blue, and filled with a slightly transparent blue. This is so that they can see the region that they are drawing in real time, while also still being able to see the map beneath it.

Prediction of HDI

The button to make the prediction (and submit the region they have drawn) will be the largest button on the screen, and in the primary colour. By default, the primary colour of `gradio` interfaces is orange, as it complements the (default, dark mode) dark blue background. This will make it very easy for the user to see what they need to do.

The 2 pieces of information below the predict button (the predicted HDI and the country comparison) will have the largest text size out of all the elements in the interface. This is because this is most likely the main information the user is trying to find, and so should be easy to find. Placing these elements next to the predict button is for the same reason.

Suggestions to Improve HDI

The suggestions to improve the HDI will be given in a table, as they will all follow a similar format: *“Build a new (building) at (these coordinates) to improve the HDI by (this much),*

which increases it to (this value). ”. This information can be efficiently organised into a table, in order to retrieve the most important information quickly.

Each row of the table (each suggestion) may be selected to show the location of that new building on the map. This is because reading coordinates such as $36.922883^{\circ}\text{N}, -101.709461^{\circ}\text{E}$ does not give an intuitive sense of location, especially within a small area.

2.4.4 Finding an Optimal Place for a New Building

When suggestions are being made to the user to improve the HDI, they will be in the format: “Build a new *some building* at *some coordinates*”. Which building will be determined from a predefined list of possible suggestions, which will match the factors I am considering. The coordinates at which to place it are harder to determine.

In theory, increasing the Density factor involving this building, and decreasing the Average Distance factors involving this building will increase the HDI. No matter where we place the new building in the region, the Density factor will increase by the same amount, as the number of that building will always increase by 1, and the area of the region will never change.

Therefore, we only need to consider how the position of a new building will affect the Average Distance factors. Any place we choose will decrease it by some amount, but we need to find a place that will decrease it by a significant amount. Finding the *exact* optimal place is very difficult, so instead I will be using a heuristic, “good enough” approach.

One way that I would consider is “good enough”, is finding the midpoint of all the positions of the other objects, to find where to place this new building. For example, if we wish to decrease $A(\text{House}, \text{Hospital})$, we could place a hospital in the midpoint of all the positions of the houses. This will involve taking a mean of all the latitude and longitude coordinates of all the houses.

However, there may be some other factors involving hospitals, such as $A(\text{School}, \text{Hospital})$ for which we would need to follow the same procedure. This will result in a different optimal place to what was calculated from the houses.

To make a compromise between all the factors that involve the building we are trying to place, we can average the previously calculated optimal places for each different factor to get an overall optimal place to build the building.

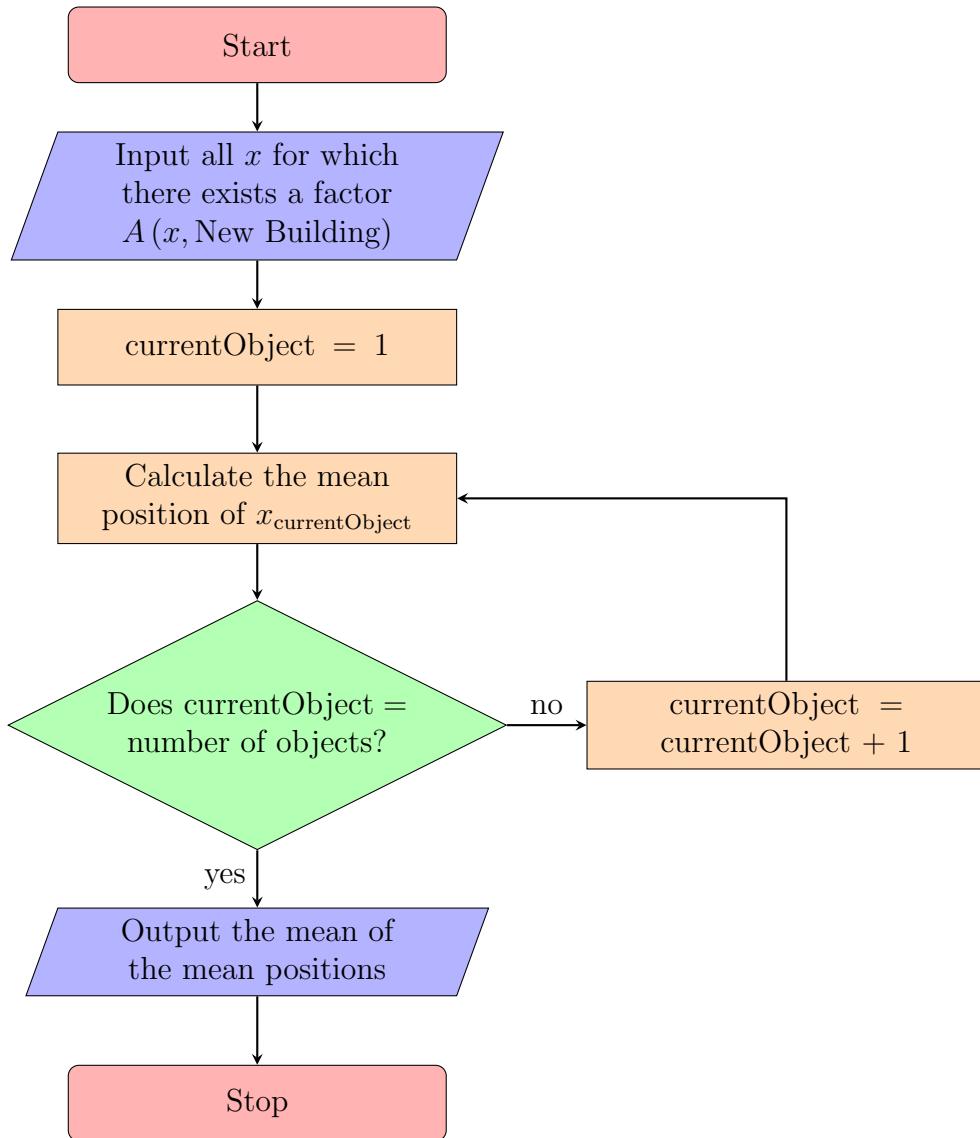


Figure 2.32: Finding an Optimal Place for New Building Flowchart

For validation, this function should only be called when there exists at least 1 factor in the form $A(x, \text{New Building})$.

I intend to implement a better algorithm to find an optimal place for a new building in future prototypes, however this will do for now.

2.4.5 Overall Algorithm for User Predicting HDI

The user will draw their region on the map, from that the relevant data will be retrieved from OSM using Overpass Turbo, and then the Density and Average Distance factors will be calculated. This will be fed into the pre-trained neural network, which will predict the HDI.

On top of this, a series of suggestions will be made to improve the HDI. The full list of suggestions I would like to make are (in no particular order):

1. Building a new School
2. Building a new Hospital
3. Building a new Place of Worship
4. Building a new Police Station
5. Building a new Restaurant
6. Building a new Slot Machine
7. Building a new Library
8. Building a new Pharmacy

For each of the suggested buildings, the position at which to build it must be calculated, and then the factors must be calculated again before being fed into the pre-trained neural network.

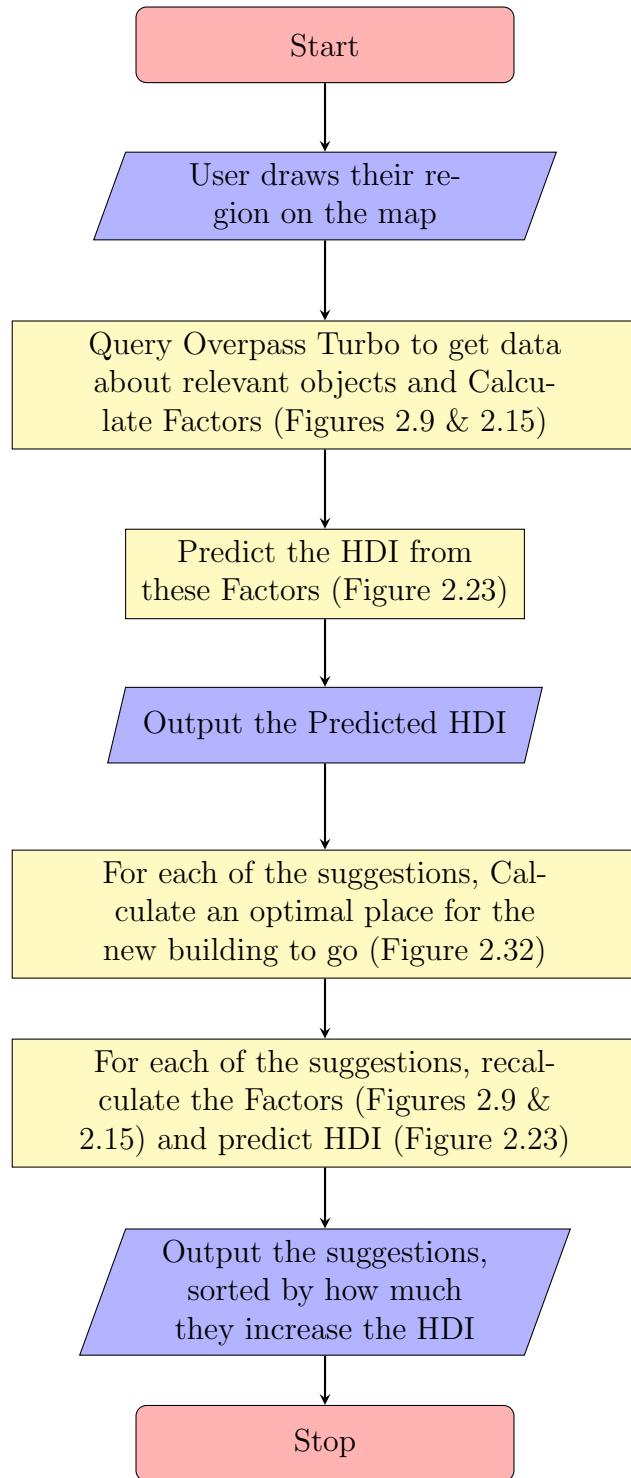


Figure 2.33: Overall User Predicting HDI Algorithm Flowchart

2.4.6 List of Key Variables

Variables in Finding Optimal Place for New Building

Identifier	Data Type	Explanation	Justification
new_ building_type	str	Holds the type of new building to be placed.	This is used to determine if there are any other factors which contain this type of building.
other_objects	list[str]	Holds each of the other types of object to consider.	We must iterate over this to find an optimal place for a new building.
current_ object	str	Iterator variable for other_objects	This will be used to keep track of which object we are currently considering
mean_ positions	list[tuple[float]]	Holds the mean positions of all the other objects we are considering	This is a list of latitude and longitude coordinates, so a tuple is used as before.

Table 2.4: List of Key Variables in Finding Optimal Place for New Building

Other Variables

Identifier	Data Type	Explanation	Justification
app	object	Contains the <code>gradio</code> app, which will be ran on the local web server.	The app (highest level “interface block”) must be stored in a variable to pass to the web server.
neural_ network	object	Stores the pre-trained model for predicting HDI.	This must be called upon when the user draws their region on the map.
suggestions_ types	list[str]	List containing the different types of suggestion to be made.	Each str will be the name of the new building type to place.
suggestions	dict[float]	A dictionary, containing each predicted HDI, each with a key.	The key will be a string containing the type of new building.

Table 2.5: List of Other Key Variables in User Interface

2.5 Data

2.5.1 Data Structures

Coordinates

Any time I am dealing with coordinates on the surface of the Earth, I will be using a `tuple[float]`. The latitude and longitude are stored as `floats`, as they are real numbers that may not be integers. They are stored in a `tuple`, as the length of the data structure will never change (as that would correspond to adding a new perpendicular dimension), and the value of the coordinate never needs to be changed, and so `tuples` being immutable and static is suitable.

Coordinates in 3D space can also be stored as `tuple[float]`, with 3 elements instead of 2 (corresponding to the xyz coordinate space).

Column Vectors & Matrices

A matrix is essentially a 2D array of numbers, which have some special operations. I will be using the `numpy` (as `np`) module's `np.matrix` for this, as it takes in a python 2D `list`, and turns it into a matrix, validating the fact that all of its entries are numbers (`int` or `float`, type check), and that all sublists are the same length (length check). This, along with the `np.matmul()` function (for multiplying matrices) will give all the required functionality of matrices.

A column vector is just a matrix with 1 column, and so `np.matrix` can still be used, except I will need to pass in a python `list` of `lists` of length 1, containing each element of the column vector.

Column vectors and matrices are used in the neural network and for calculating the area of a region.

Layers of Neurons & Weights

The neural network is mainly where column vectors & matrices will be used, as each layer of activations, biases, errors and z -values will be stored in their own column vectors, with weights being stored in matrices (as each of the neurons in 1 layer is connected to all of the neurons in the next, we need some form of 2D data structure).

Lists of Things

Any time I need to store multiple of a similar thing, I will be using a `list`. This is because lists are both dynamic and mutable, so I can add, remove or change elements as I like.

For example, the column vectors corresponding to the layers of neurons will be stored in their own lists, so that they can be easily retrieved from the index of their layer. I will also be using a `list` to store a series of coordinates (`tuple[float]`) which define a particular region, or just contain the positions of objects we are interested in.

Neural Network & File Structure

Once the neural network has been fully trained to predict HDI, I would like to be able to store it in an external file, so that predictions can be made without having to go through the training process again.

Only the weights and biases need to be saved, as the particular activations and errors are dependant on what the input layer's activations are, while the weights and biases define how the network can make predictions.

Therefore, I will first store the network's `weights` and `biases` attributes in a python `dict`, such as `{"weights": ... , "biases": ...}`, in order to store them in 1 place. I will then use the `pickle` module in order to save this variable (containing a single `dict`) to a `.pkl` file, which is a binary file that can store variables. When the model needs to be loaded (e.g. in the GUI), this file can be read to get the `dict` back, and then the weights and biases can be retrieved using the keys `"weights"` and `"biases"`.

2.5.2 Testing Data for Development

I plan to have done all the tests under each milestone by the time I reach it. Once all the tests have gone successfully, across all the milestones (for this prototype), I will ask for my stakeholders' opinions, to see what needs improving.

I will still be testing code in small increments as I am developing, to ensure that I do not make any silly mistakes.

Milestone 1 – Data Collection & Processing

No.	Test	Inputs	Expected Output/Justification
T_1	Overpass Turbo Query	Closed Polygon, Open Polygon, Valid object type, Invalid object type	The query should be able to handle both open and closed polygons as regions, and shouldn't break when an invalid object type is passed in.
T_2	Formula for distance between 2 points	2 Points in the same hemisphere, 2 Points in opposite hemispheres, 2 Points either side of the international date line, 2 Points whose shortest distance crosses the North Pole	The formula should be resiliant to all of the listed edge cases, and will be checked using an online calculator.

T_3	$A(x, y)$	Less than 500 x -objects, 500 x -objects, More than 500 x -objects, 0 x -objects	If more than 500 x -objects, random sampling should happen to find 500 random x -objects. If there are 0 x -objects, None should be returned, as there is no distance to find.
T_4	$D(x)$	Region drawn Clockwise, Region drawn Anticlockwise, Region drawn in multiple loops of the Earth, Region that self-intersects	This function should be able to handle any region that does not self-intersect, no matter how it is drawn and how many times it wraps around the Earth. The correct area will be found on the internet (as I will be testing subnational regions)

Table 2.6: Testing Data for Milestone 1

Milestone 2 – Multilayer Perceptron

Testing to see if the neural network is actually working is incredibly difficult. Due to the many limitations as a result of a (relatively) small data set, wrong predictions may be made, while the neural network is working perfectly. Therefore, in order to truly test if it is working, it must be trained on a different data set, which certainly has enough data. This data set will have nothing to do with HDI (as if it did, I would be using it for training anyway), but instead a data set on handwritten digits.

Testing each individual algorithm in the neural network is also very difficult, as there are so many small calculations that go into it, doing it by hand to check would take days. Therefore, I will only be testing the neural network once I have fully implemented it. However, I will still be doing the small, incremental tests to ensure no silly mistakes.

No.	Test	Inputs	Expected Output/Justification
T_5	Neural Network works	Using a <i>different</i> data set, e.g. handwritten digits, which certainly has enough training data	This is to ensure that the neural network works, and if it doesn't work for predicting HDI, it is as a result of a lack of training data.

Table 2.7: Testing Data for Milestone 2

Milestone 3 – Initial User Interface

No.	Test	Inputs	Expected Output/Justification
T_6	Overall Flow	User Draws Region and then Predicts, User tries to predict before selecting Region, User draws region and doesn't predict	The only valid sequence of steps the user should be able to take is drawing the region and then making a prediction.
T_7	Optimal Place for New Building	Building for which there exists 1 Factor, Building for which there exists more than 1 Factor, Building for which there exists 0 factors	This function should take an average over all of the factors the building is involved in, while throwing an error if it has 0 factors.
T_8	Suggestions fed back into Neural Network	All 8 suggestions for where to place new buildings	A new predicted HDI for each suggestion

Table 2.8: Testing Data for Milestone 3

3. Developing the Coded Solution of the 1st Prototype

3.1 Milestone 1 – Data Collection & Processing

3.1.1 Success Criteria

By the end of this milestone, I hope to have all the training data for the neural network to train on.

No.	Success Criterion	Justification
S_1	Find suitable HDI training data for the Neural Network	The neural network must train on pre-calculated HDI values for certain areas, along with their respective density and average distance factors.
S_2	Retrieve OSM data for a given area using Overpass Turbo	This will allow me to calculate useful geographical factors to predict the HDI from instead of more human ones (which are harder to determine).
S_3	Calculate average distance factors from a given area	The average distance from <i>some object</i> to <i>some other object</i> is also a good measure to consider. For example <i>A</i> (House, School) is a useful factor as school children shouldn't have to travel long distances for education.
S_4	Calculate density factors from a given area	The number of <i>some object</i> per unit area is a good way of measuring how many of that object are in a particular area while removing the bias of larger areas.

Table 3.1: Success Criteria for Milestone 1

To determine if I have met this success criteria, I will be using the following tests:

No.	Test	Inputs	Expected Output/Justification
T_1	Overpass Turbo Query	Closed Polygon, Open Polygon, Valid object type, Invalid object type	The query should be able to handle both open and closed polygons as regions, and shouldn't break when an invalid object type is passed in.
T_2	Formula for distance between 2 points	2 Points in the same hemisphere, 2 Points in opposite hemispheres, 2 Points either side of the international date line, 2 Points whose shortest distance crosses the North Pole	The formula should be resilient to all of the listed edge cases, and will be checked using an online calculator.
T_3	$A(x, y)$	Less than 500 x -objects, 500 x -objects, More than 500 x -objects, 0 x -objects	If more than 500 x -objects, random sampling should happen to find 500 random x -objects. If there are 0 x -objects, None should be returned, as there is no distance to find.
T_4	$D(x)$	Region drawn Clockwise, Region drawn Anticlockwise, Region drawn in multiple loops of the Earth, Region that self-intersects	This function should be able to handle any region that does not self-intersect, no matter how it is drawn and how many times it wraps around the Earth. The correct area will be found on the internet (as I will be testing subnational regions)

Table 3.2: Testing Data for Milestone 1

3.1.2 Part 1 – Collecting HDI Data

The first thing I need is a list of all subnational regions, with their corresponding true HDI. As stated before, I will download this from the Global Data Lab [4].

The full dataset contains lots of unnecessary data for my project, as this gives the HDI (and many other indicators) over time, whereas I just want the most recent HDI for each subnational region.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	iso_code	country	year	GDLCODE	level	region	continent	sgdi	shdi	shdf	shdim	healthindex	healthindexref	healthindexincdex	
2	AFG	Afghanistan	1990	AFGr101	Subnat	Central (Kabi Asia/Pacific		0.343				0.415		0.5	
3	AFG	Afghanistan	1990	AFGr102	Subnat	Central High Asia/Pacific		0.296				0.375		0.4	
4	AFG	Afghanistan	1990	AFGr103	Subnat	East (Nangar Asia/Pacific		0.298				0.453		0.4	
5	AFG	Afghanistan	1990	AFGr104	Subnat	North (Sama Asia/Pacific		0.272				0.375		0.4	
6	AFG	Afghanistan	1990	AFGr105	Subnat	North East (E Asia/Pacific		0.279				0.404		0.5	
7	AFG	Afghanistan	1990	AFGr106	Subnat	South (Urugz Asia/Pacific		0.221				0.444		0.5	
8	AFG	Afghanistan	1990	AFGr107	Subnat	South East (C Asia/Pacific		0.285				0.372		0.5	
9	AFG	Afghanistan	1990	AFGr108	Subnat	West (Ghor I Asia/Pacific		0.265				0.369		0.5	
10	AFG	Afghanistan	1990	AFGr109	National	Total Asia/Pacific		0.284				0.399		0.5	
11	AFG	Afghanistan	1991	AFGr101	Subnat	Central (Kabi Asia/Pacific		0.352				0.426		0.5	
12	AFG	Afghanistan	1991	AFGr102	Subnat	Central High Asia/Pacific		0.305				0.386		0.4	
13	AFG	Afghanistan	1991	AFGr103	Subnat	East (Nangar Asia/Pacific		0.306				0.464		0.4	
14	AFG	Afghanistan	1991	AFGr104	Subnat	North (Sama Asia/Pacific		0.28				0.386		0.4	
15	AFG	Afghanistan	1991	AFGr105	Subnat	North East (E Asia/Pacific		0.286				0.415		0.5	
16	AFG	Afghanistan	1991	AFGr106	Subnat	South (Urugz Asia/Pacific		0.227				0.455		0.4	
17	AFG	Afghanistan	1991	AFGr107	Subnat	South East (C Asia/Pacific		0.293				0.383		0.5	
18	AFG	Afghanistan	1991	AFGr108	Subnat	West (Ghor I Asia/Pacific		0.273				0.379		0.5	
19	AFG	Afghanistan	1991	AFGr109	National	Total Asia/Pacific		0.292				0.41		0.5	
20	AFG	Afghanistan	1992	AFGr101	Subnat	Central (Kabi Asia/Pacific		0.361				0.441		0.5	
21	AFG	Afghanistan	1992	AFGr102	Subnat	Central High Asia/Pacific		0.313				0.4		0.4	
22	AFG	Afghanistan	1992	AFGr103	Subnat	East (Nangar Asia/Pacific		0.313				0.48		0.	
23	AFG	Afghanistan	1992	AFGr104	Subnat	North (Sama Asia/Pacific		0.287				0.399		0.4	
24	AFG	Afghanistan	1992	AFGr105	Subnat	North East (E Asia/Pacific		0.293				0.429		0.4	
25	AFG	Afghanistan	1992	AFGr106	Subnat	South (Urugz Asia/Pacific		0.232				0.47		0.4	
26	AFG	Afghanistan	1992	AFGr107	Subnat	South East (C Asia/Pacific		0.3				0.397		0.5	
27	AFG	Afghanistan	1992	AFGr108	Subnat	West (Ghor I Asia/Pacific		0.28				0.393		0.4	

Figure 3.1: Full dataset containing unnecessary data

Therefore, I must write an algorithm to generate a suitable .csv file, containing only useful information.

data_processing / process_hdi.py

```
1 import csv
2
3 #retrieve data
4 with open('data/subnationalHDI.csv', 'r') as file:
5     reader = csv.DictReader(file)
```

Code Snippet 3.1: Initialising csv reader

The csv module contains many useful methods for handling csv files. I am opening 'data/subnationalHDI.csv' in read mode, and saving it to the variable file. The with statement automatically closes the file once I am done with it. The csv.DictReader is a class that can read the csv file, and convert each record into a python dict.

data_processing / process_hdi.py

```

6     newcsv = []
7     for record in reader:
8         if record['year'] == '2022': # the most recent year
9             newRecord = {
10                 "country": record["country"],
11                 "region": record["region"],
12                 "code": record["GDLCODE"],
13                 "hdi": record["shdi"]
14             } # only the most useful information
15             newcsv.append(newRecord)
16             print(newRecord)

```

Code Snippet 3.2: Refining the HDI Data

Here, I define a `list`, `newcsv` to hold all the new (necessary) records. For each of the records in the full dataset, we must determine if it is from the most recent year (2022). If it is, we make a new record of only the most important information (the region's name, its code (which will be used to uniquely identify it), and its HDI), and append this to `newcsv`.

```
{
'country': 'Afghanistan', 'region': 'Central (Kabul Wardak Kapisa Logar Parwan Panjsher)', 'code': 'AFGr101', 'hdi': '0.531'}
{'country': 'Afghanistan', 'region': 'Central Highlands (Bamyan Daikundi)', 'code': 'AFGr102', 'hdi': '0.459'}
{'country': 'Afghanistan', 'region': 'East (Nangarhar Kunar Laghman Nooristan)', 'code': 'AFGr103', 'hdi': '0.442'}
{'country': 'Afghanistan', 'region': 'North (Samangan Sar-e-Pul Balkh Jawzian Faryab)', 'code': 'AFGr104', 'hdi': '0.481'}
{'country': 'Afghanistan', 'region': 'North East (Baghlan Takhar Badakhshan Kunduz)', 'code': 'AFGr105', 'hdi': '0.429'}
{'country': 'Afghanistan', 'region': 'South (Uruzgan Helmand Zabol Nimroz Kandahar)', 'code': 'AFGr106', 'hdi': '0.393'}
{'country': 'Afghanistan', 'region': 'South East (Ghazni Paktika Khost)', 'code': 'AFGr107', 'hdi': '0.461'}
{'country': 'Afghanistan', 'region': 'West (Ghor Herat Badghis Farah)', 'code': 'AFGr108', 'hdi': '0.433'}
{'country': 'Afghanistan', 'region': 'Total', 'code': 'AFGr109', 'hdi': '0.462'}
{'country': 'Angola', 'region': 'Cabinda', 'code': 'AGOr201', 'hdi': '0.688'}
{'country': 'Angola', 'region': 'Zaire', 'code': 'AGOr202', 'hdi': '0.622'}
{'country': 'Angola', 'region': 'Uiige', 'code': 'AGOr203', 'hdi': '0.545'}
{'country': 'Angola', 'region': 'Luanda', 'code': 'AGOr204', 'hdi': '0.71'}
{'country': 'Angola', 'region': 'Kuanza Norte', 'code': 'AGOr205', 'hdi': '0.546'}
{'country': 'Angola', 'region': 'Kuanza Sul', 'code': 'AGOr206', 'hdi': '0.454'}
{'country': 'Angola', 'region': 'Malange', 'code': 'AGOr207', 'hdi': '0.571'}
{'country': 'Angola', 'region': 'Lunda Norte', 'code': 'AGOr208', 'hdi': '0.526'}
{'country': 'Angola', 'region': 'Benguela', 'code': 'AGOr209', 'hdi': '0.534'}
{'country': 'Angola', 'region': 'Huambo', 'code': 'AGOr210', 'hdi': '0.539'}
{'country': 'Angola', 'region': 'Bie', 'code': 'AGOr211', 'hdi': '0.564'}
{'country': 'Angola', 'region': 'Moxico', 'code': 'AGOr212', 'hdi': '0.508'}
{'country': 'Angola', 'region': 'Kuando Kubango', 'code': 'AGOr213', 'hdi': '0.511'}
{'country': 'Angola', 'region': 'Namibe', 'code': 'AGOr214', 'hdi': '0.587'}
{'country': 'Angola', 'region': 'Huila', 'code': 'AGOr215', 'hdi': '0.521'}
{'country': 'Angola', 'region': 'Cunene', 'code': 'AGOr216', 'hdi': '0.522'}
{'country': 'Angola', 'region': 'Lunda Sul', 'code': 'AGOr217', 'hdi': '0.586'}
{'country': 'Angola', 'region': 'Bengo', 'code': 'AGOr218', 'hdi': '0.584'}
{'country': 'Angola', 'region': 'Total', 'code': 'AGOr219', 'hdi': '0.591'}

```

Figure 3.2: The new records

Figure 3.2 shows each `newRecord`, which are correct. Therefore, I can now add these to a new `csv` file.

No.	Test	Input	Type	Expectation	Output
t_1	Refined HDI Data	—	—	Only region and HDI of the most recent year	As Expected

Table 3.3: Testing Table for Code Snippets 3.1 & 3.2

data_processing / process_hdi.py

```

16 #write to new csv
17 with open('data/hdi.csv', 'w') as file:
18     field_names = ['country', 'region', 'code', 'hdi']
19     writer = csv.DictWriter(file, fieldnames=field_names)
20     writer.writeheader()
21     for record in newcsv:
22         writer.writerow(record)

```

Code Snippet 3.3: Writing back to csv

Here, I must open '`data/hdi.csv`' in write mode, to ensure I can make changes to the empty file. The `csv.DictWriter` does the same thing as the `csv.DictReader` but in reverse, so we must specify the field names. After this, we iterate over the `newcsv` list and write all of the new records.

A	B	C	D
1 country	region		
2 Afghanistan	Central (Kabul Wardak Kapisa Logar Parwan Panjsher)	AFGr101	0.531
3 Afghanistan	Central Highlands (Bamyan Daikundi)	AFGr102	0.459
4 Afghanistan	East (Nangarhar Kunar Laghman Nooristan)	AFGr103	0.442
5 Afghanistan	North (Samangan Sar-e-Pul Balkh Jawzjan Faryab)	AFGr104	0.481
6 Afghanistan	North East (Baghan Takhar Badakhshan Kunduz)	AFGr105	0.429
7 Afghanistan	South (Uruzgan Helmand Zabol Nimroz Kandahar)	AFGr106	0.393
8 Afghanistan	South East (Ghazni Paktika Paktika Khost)	AFGr107	0.461
9 Afghanistan	West (Ghor Herat Badghis Farah)	AFGr108	0.433
10 Afghanistan		AFGr109	0.442
11 Angola	Cabinda	AGOr201	0.688
12 Angola	Zaire	AGOr202	0.622
13 Angola	Uige	AGOr203	0.545
14 Angola	Luanda	AGOr204	0.71
15 Angola	Kuanza Norte	AGOr205	0.546
16 Angola	Kuanza Sul	AGOr206	0.454
17 Angola	Malange	AGOr207	0.571
18 Angola	Lunda Norte	AGOr208	0.526
19 Angola	Benguela	AGOr209	0.534
20 Angola	Huambo	AGOr210	0.539
21 Angola	Bie	AGOr211	0.504
22 Angola	Moxico	AGOr212	0.508
23 Angola	Kuando Kubango	AGOr213	0.511
24 Angola	Namibe	AGOr214	0.587
25 Angola	Hulla	AGOr215	0.52
26 Angola	Cunene	AGOr216	0.522
27 Angola	Lunda Sul	AGOr217	0.586
28 Angola	Bengo	AGOr218	0.584
29 Angola	Total	AGOr	0.591

Figure 3.3: Dataset of only useful HDI information

Figure 3.3 shows the csv file with only the useful information required. Therefore, I can now delete the original `subnationalHDI.csv` file, as I won't be needing it anymore.

No.	Test	Input	Type	Expectation	Output
t_2	Written HDI Data to csv	—	—	csv (with correct fields) written correctly	As Expected

Table 3.4: Testing Table for Code Snippet 3.3

3.1.3 Part 2 – Collecting OSM Data for a Given Region

Next, I need to be able to retrieve the locations of all the objects from OpenStreetMaps, using Overpass Turbo.

data_processing / get_osm_data.py

```

1 import requests
2
3 # makes a request to overpass turbo to get osm data
4 def get_osm_data(object_type, bounding_coords):
5     overpass_url = 'http://overpass-api.de/api/interpreter'
6     overpass_query = [
7         '(overpass turbo query)'
8     ] # placeholder query
9     overpass_query = '\n'.join(overpass_query) # concatenates list items
10    ↵ into a multi-line string
11
12     response = requests.get(overpass_url, params={'data': overpass_query})
13     print(response) # log message, shows the HTTP response code
14     data = response.json()

```

Code Snippet 3.4: Overpass Turbo GET Request

Here, I have written a function, `def get_osm_data`, which takes in 2 parameters (the type of object we are looking for, and the region we are looking in). We then must make a query to Overpass Turbo, which often comes in multiple lines. Therefore, I have decided to store each line of the query as a `str`, in a `list`, `overpass_query`, and then join the list with newline characters (`'\n'`).

We then make a GET request to the URL of Overpass Turbo, passing in the query to ensure we get the correct data. This can be easily done with the `requests` module's `get` function. Once we get a response (hopefully a 200: OK response, as otherwise something is wrong at Overpass Turbo), we must get the data from it, which will be in a `json` format.

```

data_processing / get_osm_data.py / def get_osm_data

6     key_type = ('building' if object_type == 'house' else 'natural' if
7         ↵ object_type == 'tree' else 'gambling' if object_type ==
8         ↵ 'slot_machines' else 'amenity') # get the right key type
9     region_poly = ' '.join([' '.join([str(latlong) for latlong in coord])
10        ↵ for coord in bounding_coords]) # converts list of tuples to
11        ↵ correct format for overpass turbo
12     overpass_query = [
13         ' [out:json];',
14         '(',
```

Code Snippet 3.5: Overpass Turbo Query

OSM can store data in 3 distinct ways: nodes, ways and relations. A node is a single coordinate, a way is a collection of nodes (which, for example, might represent the outline of a building) and a relation is a collection of nodes, ways or other relations.

The first line of the query '`[out:json];`' specifies that I want the output to be in a **json** format. Using the '`nwr`' command specifies that I want all **nodes**, **ways** and **relations**, as what we are looking for could be represented as any of them.

We then must specify that we want only the ways, nodes, and relations of a particular type. Each node/way/relation on OSM has a series of keys, each of which has a value (for example, "`amenity`"="`restaurant`" would specify that a particular node represents the amenity of a restaurant). The majority of the objects I am looking for are amenities, with a few exceptions. Therefore, I must first determine the correct key before defining the query (as shown on line 6).

We then also need to specify which region we want the ways, nodes and relations to come from. I didn't know how to do this, so I carried out some research on the Overpass Turbo documentation, and found that you can specify a polygon by separating each coordinate with a space ("`La1 Lo1 La2 Lo2 ...`"). This is a different format to how it is passed into the `def get_osm_data` function, so I must first join each of the coordinate pairs with spaces, (`[' '.join(str(latlong) for latlong in coord)]`), and then join each of those with spaces, again before the query is defined (as shown on line 7).

Finally, including '`out center;`' specifies that for each of the ways and relations, their centers should be outputted, instead of their bounding coordinates.

```

  'type': 'relation',
  {'center': {'lat': 51.775109, 'lon': -1.2563298},
   'id': 3479283,
   'members': [{ref: 257782234, role: 'outer', type: 'way'},
              {ref: 97683888, role: 'outer', type: 'way'}],
   'tags': {addr:city: 'Oxford',
            addr:postcode: 'OX2 7EE',
            addr:street: 'Marston Ferry Road',
            amenity: 'school',
            capacity: '1850',
            isced:level: '2;3',
            max_age: '18',
            min_age: '11',
            name: 'The Cherwell School',
            phone: '+44 1865 558719',
            ref:GB:uprn: '100121296851',
            ref:edubase: '137970',
            ref:edubase:group: '2644',
            religion: 'none',
            school:boarding: 'no',
            school:gender: 'mixed',
            school:selective: 'no',
            school:trust: 'yes',
            school:trust:name: 'River Learning Trust',
            school:trust:type: 'multi_academy',
            school:type: 'academy',
            short_name: 'Cherwell School',
            type: 'multipolygon',
            website: 'https://www.cherwell.oxon.sch.uk/',
            wikidata: 'Q5092663',
            wikipedia: 'en:Cherwell School'},
            'type': 'relation',
            {'center': {'lat': 51.7632368, 'lon': -1.2665437},
             'id': 6660278,
             'members': [{ref: 79710677, role: 'outer', type: 'way'},
                         {ref: 448835379, role: 'outer', type: 'way'},
                         {ref: 448835378, role: 'outer', type: 'way'},
                         {ref: 448835411, role: 'outer', type: 'way'},
                         {ref: 448835408, role: 'outer', type: 'way'},
                         {ref: 448835405, role: 'outer', type: 'way'},
                         {ref: 448835399, role: 'outer', type: 'way'}],
             'tags': {addr:postcode: 'OX2 6HX'}}]
```

Figure 3.4: Output of Example Overpass Turbo Query

Executing this function, passing in '`school`' for `object_type` and a list of coordinates (in the `list[tuple[float]]` form) of a ring around my local area for `bounding_coords` and printing the what is stored in the variable `data` gives a list of schools in my local area (one of which I go to (and a bit of the next one) is shown in Figure 3.4).

No.	Test	Input	Type	Expectation	Output
$T_{1.1}$	Overpass Turbo Query	Closed Polygon,Valid object type	Valid	Returns the appropriate data	As Expected
$T_{1.2}$	Overpass Turbo Query	Open Polygon,Valid object type	Edge Case	Returns the appropriate data, as if there was a line joining the first and last vertices	As Expected
$T_{1.3}$	Overpass Turbo Query	Invalid object type	Invalid	Returns nothing	As Expected

Table 3.5: T_1 Testing Table (for Code Snippets 3.4 & 3.5)

However, again, there is lots of unnecessary data here (for example names, phone numbers etc.) so we must refine it.

```
data_processing / get_osm_data.py / def get_osm_data
```

```
21     coords = []
22     for nwr in data['elements']:
23         if nwr['type'] == 'node':
24             coords.append((nwr['lat'], nwr['lon']))
25         else: # it is a way or relation
26             coords.append((nwr['center']['lat'], nwr['center']['lon']))
27
28     return coords
```

Code Snippet 3.6: Getting a List of Latitude/Longitude Pairs

The objects we are receiving come in a list, which has the key '`elements`', so I am iterating over `data['elements']`. For each one, we must append only the latitude longitude coordinates (in a `tuple`) to a `list`. If a particular one is stored as a node, then the latitude and longitude coordinates are stored directly, but if it is a way or relation, then they are stored in an additional `dict` with the key '`center`'.

Once we have iterated over each node/way/relation, we can return the `list` of coordinates.

```
[(51.7546965, -1.2594876), (51.738484, -1.2265718), (51.7241049, -1.2095659), (51.7368965, -1.2012643), (51.7586564, -1.2345702), (51.7485855, -1.2349221), (51.7365529, -1.214302), (51.7874249, -1.2616194), (51.7538688, -1.2985268), (51.7265657, -1.2324789), (51.7428616, -1.218293), (51.7450632, -1.2605576), (51.7372123, -1.2311163), (51.7380926, -1.2273243), (51.7641502, -1.2278057), (51.7589064, -1.2251667), (51.7461635, -1.2862479), (51.7635249, -1.1926637), (51.7630379, -1.1898475), (51.732761, -1.2351862), (51.7534768, -1.2244231), (51.784562, -1.280275), (51.7403821, -1.2421973), (51.7510535, -1.2757784), (51.7603838, -1.2075961), (51.7524678, -1.2030074), (51.7593023, -1.266659), (51.7678747, -1.2724486), (51.7336174, -1.2115186), (51.7580632, -1.2057114), (51.7382056, -1.2557097), (51.7744163, -1.2591828), (51.7716276, -1.2258946), (51.7574877, -1.217863), (51.7369958, -1.2110372), (51.7571114, -1.2238353), (51.7484696, -1.2347665), (51.7669542, -1.2604464), (51.7783094, -1.259319), (51.7781822, -1.2686885), (51.7565292, -1.2532005), (51.7217629, -1.2149064), (51.7718743, -1.2394002), (51.7719641, -1.242301), (51.7495517, -1.2575039), (51.7529223, -1.2600046), (51.7811525, -1.2715384), (51.7391512, -1.2080504), (51.7673448, -1.2662258), (51.7677108, -1.2600939), (51.7823832, -1.2681758), (51.7693443, -1.262655), (51.7707973, -1.242397), (51.7279666, -1.2146291), (51.775109, -1.2563298), (51.7632368, -1.2665437), (51.7494109, -1.2461454), (51.7726646, -1.2651714), (51.7538272, -1.2590957), (51.7784973, -1.2726615), (51.7676965, -1.2589464), (51.7678128, -1.2552128), (51.7703703, -1.2087175), (51.7694505, -1.2286333)]
```

Figure 3.5: Output of a `list` of Latitude/Longitude pairs

Figure 3.5 shows the new output of the `def get_osm_data` function with the same input data from Figure 3.4, which is correct.

No.	Test	Input	Type	Expectation	Output
t_3	Refined OSM Data	-	-	Only Latitude/Longitude pairs	As Expected

Table 3.6: Testing Table for Code Snippet 3.6

3.1.4 Part 3 – Implementing $A(x, y)$

Now, it's time to start calculating the factors from this data. Firstly, the average distance factors.

data_processing / calc_factors.py

```

1 import numpy as np
2
3 EARTH_RADIUS = 6371 # (in km)
4
5 # converts angle in degrees to angle in radians
6 def deg2rad(angle):
7     return (np.pi/180)*angle
8
9 # finds the distance (in km) along the surface of the earth, between 2
10    ↵ coordinates
11 def distBetween2Points(p1, p2):
12     La1 = deg2rad(p1[0])
13     Lo1 = deg2rad(p1[1])
14     La2 = deg2rad(p2[0])
15     Lo2 = deg2rad(p2[1])
16     return EARTH_RADIUS * np.arccos(np.sin(La1)*np.sin(La2) +
17         ↵ np.cos(La1)*np.cos(La2)*np.cos(Lo1-Lo2))

```

Code Snippet 3.7: Finding the Distance between any 2 points

This code snippet implements equation 2.7, which finds the distance along the surface of the Earth, between 2 points (p_1 and p_2), whose coordinates are given in degrees.

The first thing we need to do, is convert each of these angles into radians, by using the conversion factor (angle in radians) = $\frac{\pi}{180} \times$ (angle in degrees), and implemented with the function `def deg2rad`.

No.	Test	Input	Type	Expectation	Output
t_4	Conversion from degrees to radians	angle in degrees	–	$\frac{\pi}{180} \times$ (angle)	As Expected

Table 3.7: Testing Table for Code Snippet 3.7

We then must calculate the distance with equation 2.7, and return it. I am using the module `numpy` for the sin, cos and arccos functions, as well as a precise value for π .

For each of the following tests in T_2 , the 2 points I selected to test were 30° away from each other, along a great circle. Therefore, the output for all of them should be $\frac{1}{12} \times 2\pi r$, which ≈ 3335.8 km.

No.	Test	Input	Type	Expectation	Output
$T_{2.1}$	Distance between 2 points	2 Points in the same hemisphere: $(30^\circ, 0^\circ)$, $(60^\circ, 0^\circ)$	Valid	3335.8 km	As Expected
$T_{2.2}$	Distance between 2 points	2 Points in opposite hemispheres: $(15^\circ, 0^\circ)$, $(-15^\circ, 0^\circ)$	Edge Case	3335.8 km	As Expected
$T_{2.3}$	Distance between 2 points	2 Points either side of the international date-line: $(0^\circ, 165^\circ)$, $(0^\circ, -165^\circ)$	Edge Case	3335.8 km	As Expected
$T_{2.4}$	Distance between 2 points	2 Points whose shortest distance crosses the North Pole: $(75^\circ, 90^\circ)$, $(75^\circ, -90^\circ)$	Edge Case	3335.8 km	As Expected

Table 3.8: T_2 Testing Table (for Code Snippet 3.7)

```
data_processing / calc_factors.py
```

```
2 import random
```

```
data_processing / calc_factors.py
```

```
18 # finds the average distance between 2 types of objects, A(x,y)
19 def averageDistance(x_objects, y_objects, max_x_objects=500):
20     if len(x_objects) == 0 or len(y_objects) == 0: # the average distance
21         → is undefined
22         return None
23     if len(x_objects) > max_x_objects: # random sample
24         x_objects = random.sample(x_objects, max_x_objects)
```

Code Snippet 3.8: $A(x, y)$ Validation

Here, I am defining the `def averageDistance` function, to calculate $A(x, y)$. It takes in `x_objects` (x), `y_objects` (y) and `max_x_objects` as parameters. By default (if nothing is passed in for `max_x_objects`), `max_x_objects` is set to 500, as it was in the flowchart in Figure 2.9.

If there are either no x -objects, or no y -objects, then the average distance between them is undefined, so we must return `None`. When this gets fed into the neural network, it will be replaced by an arbitrarily large number, as (in theory) larger values of $A(x, y)$ for any x, y are “bad”, and should be decreased.

If there are instead more than (`max_x_objects`) x -objects, then we must take a random sample of (`max_x_objects`), as otherwise the function will take too long to execute. To do this, we must first `import random`, and then use its `random.sample()` method.

No.	Test	Input	Type	Expectation	Output
$T_{3.1}$	$A(x, y)$	Less than 500 x -objects	Valid	Uses all of the x -objects	As Expected
$T_{3.2}$	$A(x, y)$	500 x -objects	Boundary	Uses all 500 of the x -objects	As Expected
$T_{3.3}$	$A(x, y)$	More than 500 x -objects	Invalid	Uses a random 500 of the x -objects	As Expected
$T_{3.4}$	$A(x, y)$	0 x -objects	Invalid	Returns <code>None</code>	As Expected

Table 3.9: T_3 Testing Table (for Code Snippet 3.8)

data_processing / calc_factors.py / `def averageDistance`

```

24     # iterate over each x_object, and find the closest y_object
25     min_dists = []
26     for x_object in x_objects:
27         dist_to_ys = []
28         for y_object in y_objects:
29             dist_to_ys.append(distBetween2Points(x_object, y_object))
30         min_dists.append(min(dist_to_ys)) # the closest y_object
31     return np.mean(min_dists)

```

Code Snippet 3.9: Calculating $A(x, y)$

This code snippet implements the rest of the algorithm to find $A(x, y)$, as shown in Figure 2.9. (We must iterate over each x -object, find the closest y -object, and return the mean of those distances).

Running this function, passing in the list of schools shown in Figure 3.5, and a list of houses in the same area, returns 0.37311016450362056 km, which sounds about right. Running the function again, with the same inputs, but with no random sampling however returns 0.3586665816532994 km, which is the true value, but takes longer to calculate. Therefore, in the first run, the random sampling must have (randomly) selected particular houses that happened to be slightly further away from schools than usual (approximately 15 more metres).

No.	Test	Input	Type	Expectation	Output
t_5	$A(x, y)$	List of houses and schools in my local area	-	~ 0.36 km	As Expected

Table 3.10: Testing Table for Code Snippet 3.9

To see the full effect of the random sampling, we can run the function on the same data multiple times, to see the range of values it produces:

```
average distance from house to nearest school in oxford is: 0.3471860081253106 km
average distance from house to nearest school in oxford is: 0.34935882509741467 km
average distance from house to nearest school in oxford is: 0.3557128766643569 km
average distance from house to nearest school in oxford is: 0.35344981434591427 km
average distance from house to nearest school in oxford is: 0.35875781665549433 km
average distance from house to nearest school in oxford is: 0.3613735418982461 km
average distance from house to nearest school in oxford is: 0.3557637814311824 km
average distance from house to nearest school in oxford is: 0.35691715365803295 km
average distance from house to nearest school in oxford is: 0.34808368746306134 km
average distance from house to nearest school in oxford is: 0.3524024184457041 km
average distance from house to nearest school in oxford is: 0.364156322089742 km
average distance from house to nearest school in oxford is: 0.3528441770837829 km
average distance from house to nearest school in oxford is: 0.3636918565334129 km
average distance from house to nearest school in oxford is: 0.369403201575993 km
average distance from house to nearest school in oxford is: 0.362327943106565 km
average distance from house to nearest school in oxford is: 0.364705313921878 km
average distance from house to nearest school in oxford is: 0.3549901458177277 km
average distance from house to nearest school in oxford is: 0.3480712819732799 km
average distance from house to nearest school in oxford is: 0.3609290654924819 km
average distance from house to nearest school in oxford is: 0.3589591342932028 km
that took 11.004106998443604 seconds
```

Figure 3.6: $A(x, y)$ 20 times, with `max_x_objects` set to 500

Figure 3.6 shows executing $A(x, y)$ 20 times, with the same input data as before, and `max_x_objects` set to 500. The value it returns is often correct to 1 significant figure (0.4 km), and correct to 2 significant figures about half the time (0.36 km). In total, the whole thing took around 11 seconds and so each one took approximately 0.55 seconds.

This is not very accurate, and so I believe we should increase the `max_x_objects`. This will also increase the time for which it takes, so we shouldn't increase it by too much.

```

average distance from house to nearest school in oxford is: 0.35699701939607803 km
average distance from house to nearest school in oxford is: 0.36155987000484535 km
average distance from house to nearest school in oxford is: 0.3568930106373999 km
average distance from house to nearest school in oxford is: 0.3535205857963649 km
average distance from house to nearest school in oxford is: 0.36394807974342963 km
average distance from house to nearest school in oxford is: 0.358652190929027 km
average distance from house to nearest school in oxford is: 0.3555692513092449 km
average distance from house to nearest school in oxford is: 0.35734571898276885 km
average distance from house to nearest school in oxford is: 0.35994759185886527 km
average distance from house to nearest school in oxford is: 0.36577563831020554 km
average distance from house to nearest school in oxford is: 0.36094046155354886 km
average distance from house to nearest school in oxford is: 0.35652504272471075 km
average distance from house to nearest school in oxford is: 0.3591947485562978 km
average distance from house to nearest school in oxford is: 0.36079439696851606 km
average distance from house to nearest school in oxford is: 0.36447506968526566 km
average distance from house to nearest school in oxford is: 0.35276955289898515 km
average distance from house to nearest school in oxford is: 0.3631400347535967 km
average distance from house to nearest school in oxford is: 0.3554118911780664 km
average distance from house to nearest school in oxford is: 0.35665813539156627 km
average distance from house to nearest school in oxford is: 0.3567446332467618 km
that took 38.9518940448761 seconds

```

Figure 3.7: $A(x, y)$ 20 times, with `max_x_objects` set to 2000

Figure 3.7 shows executing $A(x, y)$ 20 times, with the same input data as before, and `max_x_objects` set to 2000. The value it now returns is always correct to 1 significant figure (0.4 km), and often correct to 2 significant figures (0.36 km). In total, the whole thing took around 39 seconds and so each one took approximately 1.95 seconds. This makes sense, as $A(x, y)$ should scale linearly with `max_x_objects`, and $0.55 \times \frac{2000}{500} \approx 1.95$.

This is a better result, in still a reasonable amount of time, and so I have decided to change the default value for `max_x_value` from 500 to 2000. The value can of course still be changed at any time, by passing in a different number. When the user comes to predict the HDI of their area in a later prototype, they may be able to change `max_x_objects` with a slider, to make a tradeoff between accuracy and time.

3.1.5 Part 4 – Implementing $D(x)$

Now its time for the harder factors to calculate, the density factors.

First, I decided to make the part of $D(x)$ that calculates the area of the region a seperate function, as the area will be the same for all density factors of a given area, and so it would be more efficient to calculate it once, beforehand.

data_processing / calc_factors.py

```

33 # calculates the area (in km^2) of a given region bound by a set of
34   ↵ latitude/longitude coordinates
35 def calcArea(bounding_coords):
36     if bounding_coords[0] == bounding_coords[-1]: # if the first and last
37       ↵ coordinates are the same, then delete the last one
38     bounding_coords.pop(-1)
39     # convert lat/lon pairs to 3d position vectors
40     bounding_vectors = []
41     for coord in bounding_coords:
42       La = coord[0]
43       Lo = coord[1]
44       bounding_vectors.append(np.matrix([
45         [np.cos(La)*np.cos(Lo)],
46         [np.cos(La)*np.sin(Lo)],
47         [np.sin(La)]
48       ]))

```

Code Snippet 3.10: Converting list of bounding coordinates into Column Vectors

Here, I am defining the `def calcArea` function, it takes in the `bounding_coords` as a parameter. The first thing we need to check is if the first and last elements of that list are the same. If they are, then remove the last one, as we don't want to be double-counting a particular vertex.

Next, we must find the absolute position of each coordinate in 3D space, by using equation 2.14 (remembering that r can be set to 1, as the radius of the Earth is irrelevant). As stated before, I am using the `np.matrix` data structure to represent column vectors and matrices.

No.	Test	Input	Type	Expectation	Output
t_6	Converting lat/lon to column vectors	A lat/lon coordinate, for example, $(45^\circ, 45^\circ)$	—	In this example, $\left(\frac{1}{2}, \frac{1}{2}, \frac{1}{\sqrt{2}}\right)$	In this example, $(0.276, 0.445, 0.851)$, to 3 significant figures

Table 3.11: Testing Table for Code Snippet 3.10

This is clearly the wrong output. I first thought that maybe equation 2.14 was wrong, but then I checked my working and couldn't find any mistakes. I then realised I hadn't converted from degrees to radians before calculating the vectors.

```
data_processing / calc_factors.py / def calcArea
```

```
40     La = deg2rad(coord[0])
41     Lo = deg2rad(coord[1])
```

Code Snippet 3.11: Converting from Degrees to Radians

No.	Test	Input	Type	Expectation	Output
t_6	Converting lat/lon to column vectors	A lat/lon coordinate, for example, $(45^\circ, 45^\circ)$	-	In this example, $\left(\frac{1}{2}, \frac{1}{2}, \frac{1}{\sqrt{2}}\right)$	In this example, $(0.5, 0.5, 0.707)$, to 3 significant figures, As Expected

Table 3.12: Testing Table for Code Snippet 3.11

This now produces correct vectors. We now must iterate over each vertex, and find its anticlockwise angle.

```

data_processing / calc_factors.py / def calcArea

47     # iterate over each vertex and find its anticlockwise angle
48     for vertex_index, vertex in enumerate(bounding_vectors):
49         prev_vertex = bounding_vectors[vertex_index-1]
50         next_vertex = bounding_vectors[0 if vertex_index ==
51             len(bounding_vectors)-1 else vertex_index+1]
52         La = deg2rad(bounding_coords[vertex_index][0])
53         Lo = deg2rad(bounding_coords[vertex_index][1])
54         # define the rotation and translation matrices
55         rotation_matrix_z = np.matrix([
56             [np.cos(-Lo), -np.sin(-Lo), 0],
57             [np.sin(-Lo), np.cos(-Lo), 0],
58             [0, 0, 1]
59         ])
59         rotation_matrix_y = np.matrix([
60             [np.cos(La-np.pi/2), 0, np.sin(La-np.pi/2)],
61             [0, 1, 0],
62             [-np.sin(La-np.pi/2), 0, np.cos(La-np.pi/2)]
63         ])
63         translation_vector = np.matrix([
64             [0],
65             [0],
66             [-1]
67         ])
68

```

Code Snippet 3.12: Defining Key Variables in calculating area

Firstly, we need to get the positions 2 vertices adjacent to the one we are considering. To get the previous one, we can just subtract 1 from the index of the current one, (as in the case that we are considering the first vertex, the index -1 refers to the last element in a list, which is the previous one in the shape). To get the next vertex, the index will usually be 1 more than the current index, except for when we are considering the final vertex, as the next one will be the first vertex.

We then must find the latitude and longitude of the current vertex, so that we can rotate the plane tangent to it to be the xy -plane. Luckily, `bounding_coords` and `bounding_vectors` have the same indices, so we just need to pass the `bounding_coord` at the index `vertex_index` through the `def deg2rad` function to get what we need.

Finally, we must define the 3 matrices used to transform the plane into the xy -plane, as shown in equations 2.15 to 2.17.

```
data_processing / calc_factors.py / def calcArea
```

```

69      # rotate and translate the plane tangent to the vertex such that
    ↵ it is the xy-plane
70      prev_vertex = np.matmul(rotation_matrix_y,
    ↵ np.matmul(rotation_matrix_z, prev_vertex)) +
    ↵ translation_vector
71      vertex = np.matmul(rotation_matrix_y, np.matmul(rotation_matrix_z,
    ↵ vertex)) + translation_vector
72      next_vertex = np.matmul(rotation_matrix_y,
    ↵ np.matmul(rotation_matrix_z, next_vertex)) +
    ↵ translation_vector
73      # project onto the xy-plane
74      prev_vertex[2][0] = 0
75      next_vertex[2][0] = 0

```

Code Snippet 3.13: Rotating the plane tangent to each vertex

Now we must transform each of these vertices, using the matrices we just defined. As stated before, the first is to perform a rotation about the z -axis, such that it is parallel to the y -axis, the second is to perform a rotation about the y -axis such that it is parallel to the x -axis (and hence a translation of the xy -plane), and the third is to translate it onto the xy -plane.

Then, to project the previous and next vertices onto the xy -plane, we can just set their z -coordinates to 0. This is at index $[2][0]$, as z is the third axis (second with 0-indexing) and the column vectors are stored as a matrix, so each item in the column vector is actually an array (hence the $[0]$ index).

No.	Test	Input	Type	Expectation	Output
t_7	Getting the previous and next vertices	—	—	Works in the edge cases concerning the first and last vertices	As Expected
t_8	Transformation of vertices	—	—	Current vertex should be transformed to $(0, 0, 0)$	As Expected
t_9	Projection of other vertices	—	—	z -coordinate set to 0	As Expected

Table 3.13: Testing Table for Code Snippets 3.12 & 3.13

```
data_processing / calc_factors.py / def calcArea
```

```
48     anticlockwise_angles = []
```

```
data_processing / calc_factors.py / def calcArea
```

```
77     # calculate anticlockwise angle between the vectors
78     prev_to_current = vertex - prev_vertex
79     current_to_next = next_vertex - vertex
80     anticlockwise_angle =
81         ← np.arctan2((prev_to_current.item(1,0)*current_to_next.item(2,0))
81         ← - prev_to_current.item(2,0)*current_to_next.item(1,0)) -
81         ← (prev_to_current.item(2,0)*current_to_next.item(0,0) -
81         ← prev_to_current.item(0,0)*current_to_next.item(2,0)) +
81         ← (prev_to_current.item(0,0)*current_to_next.item(1,0) -
81         ← prev_to_current.item(1,0)*current_to_next.item(0,0)),
81         ← prev_to_current.item(0,0)*current_to_next.item(0,0) +
81         ← prev_to_current.item(1,0)*current_to_next.item(1,0) +
81         ← prev_to_current.item(2,0)*current_to_next.item(2,0))
81     anticlockwise_angles.append(anticlockwise_angle)
```

Code Snippet 3.14: Finding the anticlockwise angles

Now we must find the anticlockwise angle between the vertices. First, we find the vector to get from the previous vertex to the current vertex, and the vector to get from the current vertex to the next vertex, and then use equation 2.18 to find the anticlockwise angle.

`np.arctan2` is the 2-argument form of the arctan function, giving a range of $-\pi$ to π , instead of the regular $-\frac{\pi}{2}$ to $\frac{\pi}{2}$.

As we will want to find the sum of these anticlockwise angles, we must first declare a `list` before the `for` loop (started in Code Snippet 3.12), and then append each anticlockwise angle to it.

No.	Test	Input	Type	Expectation	Output
t_{10}	Calculating the anticlockwise angle	-	-	Angle that you need to turn anticlockwise to be facing in the new direction	As Expected

Table 3.14: Testing Table for Code Snippet 3.14

```
data_processing / calc_factors.py / def calcArea
```

```

82     # find the interior angles from the anticlockwise angles
83     sum_anticlockwise_angles = sum(anticlockwise_angles)
84     if sum_anticlockwise_angles == 0:
85         return None
86     if sum_anticlockwise_angles < 0:
87         anticlockwise_angles = [-1*angle for angle in
88             ↵ anticlockwise_angles]
89     interior_angles = [np.pi-angle for angle in anticlockwise_angles]
90     # return the area
91     num_edges = len(bounding_coords)
92     return (EARTH_RADIUS ** 2) * (sum(interior_angles) -
93         ↵ np.pi*(num_edges-2))

```

Code Snippet 3.15: Calculating the Area

This code snippet implements the bottom half of the flowchart for $D(x)$, in Figure 2.15. We find the sum of the anticlockwise angles, if it is 0, then the shape must be invalid and so we `return None`.

If the sum is negative, then the shape must have been drawn the wrong way around, and so we must multiply each of the anticlockwise angles by -1 , before continuing, which can be done with a list comprehension.

Then, to calculate the interior angles, we subtract each angle from π , again using a list comprehension. Finally, to calculate the area of the region we use equation 2.13. To find the number of edges of the shape, we just count the number of vertices, as they will always be the same thing.

No.	Test	Input	Type	Expectation	Output
$T_{4.1}$	$D(x)$	Region drawn Clockwise	Valid	Correct area calculated	As Expected
$T_{4.2}$	$D(x)$	Region drawn Anticlockwise	Valid	Correct area calculated, as if it were drawn clockwise	As Expected
$T_{4.3}$	$D(x)$	Region drawn in multiple loops of the Earth	Valid	Correct area calculated, as if all coordinates were in the regular range	As Expected
$T_{4.4}$	$D(x)$	Region that self-intersects	Invalid	<code>None</code>	Some large area

Table 3.15: T_4 Testing Table (for Code Snippet 3.15)

For the areas drawn clockwise and anticlockwise, I used the same area that I used to get the data about houses and schools in the $A(x, y)$ section, which was a polygon around Oxford. The areas they returned were about right, after researching the true area of Oxford. My values turned out greater as I was drawing *around* Oxford. The discrepancy between the clockwise and anticlockwise values is most likely due to floating point arithmetic and calculating things in different orders.

For the area drawn in multiple loops around the Earth, I again used the same area of Oxford, except some of the coordinates had multiples of 360 added to them. Again, the discrepancy between values can be explained by floating point binary.

```
area of oxford but it was drawn clockwise: 54.51510926217421 km^2
area of oxford but it was drawn anticlockwise: 54.51510919007252 km^2
area of oxford but in multiple loops around the earth: 54.515115535020925 km^2
some random polygon that self-intersects: 255032235.95489413
```

Figure 3.8: Area of Oxford, with an Invalid Self-Intersection

The area that self-intersects must also have not worked as a result of floating point binary. When inspecting the sum of the anticlockwise angles, I found that it was very close to 0, but not quite. Therefore, I must round the sum of the anticlockwise angles before seeing if it is 0.

```
data_processing / calc_factors.py / def calcArea
```

```
83     sum_anticlockwise_angles = round(sum(anticlockwise_angles), 5)
```

Code Snippet 3.16: Rounding the sum of anticlockwise angles

5 decimal places should be enough to determine if it is supposed to be 0, while not affecting anything that shouldn't be 0.

No.	Test	Input	Type	Expectation	Output
$T_{4.4}$	$D(x)$	Region that self-intersects	Invalid	None	As Expected

Table 3.16: Testing Table for Code Snippet 3.16

```
area of oxford but it was drawn clockwise: 54.51510926217421 km^2
area of oxford but it was drawn anticlockwise: 54.51510919007252 km^2
area of oxford but in multiple loops around the earth: 54.515115535020925 km^2
some random polygon that self-intersects: None
```

Figure 3.9: Area of Oxford, with a Valid Self-Intersection

Finally, all that is left to do is calculate $D(x)$.

`data_processing / calc_factors.py`

```
93 # finds the density of a type of object, D(x)
94 def density(x_objects, area):
95     return len(x_objects)/area
```

Code Snippet 3.17: Calculating $D(x)$

$D(x)$ is given by the number of x -objects per unit area, so we can just find the length of the `x_objects` list and divide by the area to get our answer.

No.	Test	Input	Type	Expectation	Output
t_{11}	$D(x)$	List of schools in Oxford, Area of Oxford	–	~ 1 School per km^2	As Expected

Table 3.17: Testing Table for Code Snippet 3.17

D(School) for oxford: 1.192329995843938 schools/km²

Figure 3.10: $D(\text{School})$ for Oxford

3.1.6 Part 5 – Iterating over each Subnational Region

Now it is time to finally compile all of our training data. We first must be able to find all of the required factors for any given area.

data_processing / calc_factors.py

```
3   from get_osm_data import get_osm_data
```

data_processing / calc_factors.py

```
98 # finds all the factors for a given region
99 def getAllFactors(region):
100     # retrieve all relevant osm data
101     object_types = ['house', 'school', 'hospital', 'pharmacy',
102                     'restaurant', 'place_of_worship', 'bank', 'slot_machines',
103                     'fast_food', 'toilets', 'police', 'university', 'library',
104                     'post_box', 'vending_machine', 'bench', 'tree']
105     all_objects = {}
106     for object_type in object_types:
107         print(f'getting {object_type} data...') # log message
108         all_objects[object_type] = get_osm_data(object_type, region)
```

Code Snippet 3.18: Retrieving All OSM Data

To do this, I have defined a function, `def getAllFactors`, which takes in a region (in the `list[tuple[float]]` format), and returns a `dict` of all the factors. We must first retrieve all of the relevant data from OSM (using the `def get_osm_data` function implemented in Section 3.1.3).

Here, I define a list of all the object types needed for all of the factors, and an empty dictionary. For each of the object types, a key is created in the dictionary, with the value being the returned `list` of latitude and longitude coordinates.

I am also printing a log message, as some of these fetches may take a long time, and so it is useful to keep track of where we are in the process.

No.	Test	Input	Type	Expectation	Output
t_{12}	Collecting all objects	—	—	Each list saved in a <code>dict</code>	As Expected

Table 3.18: Testing Table for Code Snippet 3.18

```
getting house data...
<Response [200]>
getting school data...
<Response [200]>
getting hospital data...
<Response [200]>
getting pharmacy data...
<Response [200]>
getting restaurant data...
<Response [200]>
getting place_of_worship data...
<Response [200]>
getting bank data...
<Response [200]>
getting slot_machines data...
<Response [200]>
getting fast_food data...
<Response [200]>
getting toilets data...
<Response [200]>
getting police data...
<Response [200]>
getting university data...
<Response [200]>
getting library data...
<Response [200]>
getting post_box data...
<Response [200]>
getting vending_machine data...
<Response [200]>
getting bench data...
<Response [200]>
getting tree data...
<Response [200]>
```

Figure 3.11: Log messages for Collecting data for all objects

Now we must calculate each factor with this information.

```

data_processing / calc_factors.py / def getAllFactors

106     # return a dictionary of all the factors
107     print('calculating area...') # log message
108     area = calcArea(region)
109     print('calculating factors...') # log message
110     return {
111         "A(House School)": averageDistance(all_objects['house'],
112             ↳ all_objects['school']),
113         "A(House Hospital)": averageDistance(all_objects['house'],
114             ↳ all_objects['hospital']),
115         "A(House Pharmacy)": averageDistance(all_objects['house'],
116             ↳ all_objects['pharmacy']),
117         "A(House Restaurant)": averageDistance(all_objects['house'],
118             ↳ all_objects['restaurant']),
119         "A(School Hospital)": averageDistance(all_objects['school'],
120             ↳ all_objects['hospital']),
121         "A(Police Hospital)": averageDistance(all_objects['police'],
122             ↳ all_objects['hospital']),
123         "A(House Place of Worship)": averageDistance(all_objects['house'],
124             ↳ all_objects['place_of_worship']),
125         "A(Bank Slot Machine)": averageDistance(all_objects['bank'],
126             ↳ all_objects['slot_machines']),
127         "A(Fast-Food Place Toilet)": averageDistance(all_objects['fast_food'],
128             ↳ all_objects['toilets']),
129         "A(House Police)": averageDistance(all_objects['house'],
130             ↳ all_objects['police']),
131         "A(University Library)": averageDistance(all_objects['university'],
132             ↳ all_objects['library']),
133         "A(House Library)": averageDistance(all_objects['house'],
134             ↳ all_objects['library']),

```

Code Snippet 3.19: Calculating All Average Distance Factors

Here, I am returning a `dict` with a key for each factor. For the average distance factors, we can just pass the relevant data into the `def averageDistance` function (implemented in Section 3.1.4).

The factor names don't have a comma between the 2 parameters of the `A` function, as that would mess up the `csv` headers.

```
data_processing / calc_factors.py / def getAllFactors
```

```

123     "D(School)": density(all_objects['school'], area),
124     "D(Hospital)": density(all_objects['hospital'], area),
125     "D(Pharmacy)": density(all_objects['pharmacy'], area),
126     "D(Police)": density(all_objects['police'], area),
127     "D(Library)": density(all_objects['library'], area),
128     "D(Toilet)": density(all_objects['toilets'], area),
129     "D(Restaurant)": density(all_objects['restaurant'], area),
130     "D(Place of Worship)": density(all_objects['place_of_worship'],
131                                     ↪ area),
131     "D(Post Box)": density(all_objects['post_box'], area),
132     "D(Vending Machine)": density(all_objects['vending_machine'],
133                                    ↪ area),
133     "D(Bench)": density(all_objects['bench'], area),
134     "D(Tree)": density(all_objects['tree'], area),
135 }
```

Code Snippet 3.20: Calculating All Density Factors

For the density factors, we can just pass the relevant data into the `def density` function (implemented in Section 3.1.5), along with the area of the region. The area of the region must therefore be calculated before, by using the `def calcArea` function (also implemented in Section 3.1.5).

No.	Test	Input	Type	Expectation	Output
t_{13}	Each factor is calculated	—	—	We have a <code>dict</code> of all the factors of a given region	As Expected

Table 3.19: Testing Table for Code Snippets 3.19 & 3.20

```

{'A(Bank Slot Machine)': None,
 'A(Fast-Food Place Toilet)': 0.32378007261588504,
 'A(House Hospital)': 1.8255475111340795,
 'A(House Library)': 0.7865995162066026,
 'A(House Pharmacy)': 0.48865710003324125,
 'A(House Place of Worship)': 0.32835926047180214,
 'A(House Police)': 1.2440715688997028,
 'A(House Restaurant)': 0.5761586539617497,
 'A(House School)': 0.3831354808318887,
 'A(Police Hospital)': 1.5752519946611343,
 'A(School Hospital)': 2.296870639121618,
 'A(University Library)': 0.3198281217956042,
 'D(Bench)': 11.428024421704206,
 'D(Hospital)': 0.09171769198799523,
 'D(Library)': 1.3023912262295323,
 'D(Pharmacy)': 0.45858845993997616,
 'D(Place of Worship)': 2.6231259908566638,
 'D(Police)': 0.09171769198799523,
 'D(Post Box)': 3.7604253715078046,
 'D(Restaurant)': 2.8799355284230503,
 'D(School)': 1.192329995843938,
 'D(Toilet)': 0.9538639966751504,
 'D(Tree)': 62.56980947421035,
 'D(Vending Machine)': 0.4402449215423771}

```

Figure 3.12: Example all factors `dict` for Oxford

Figure 3.12 shows the output of the `def getAllFactors` function for the same region I have been testing in previously (a rough ring around Oxford).

I now need a list of all of the subnational regions (that is, a `list` of bounding coordinates for each one). As stated before, I will also download this from the Global Data Lab [5].

The full dataset is a shapefile `.shp`, which I didn't know how to read. After researching the problem, I found a StackOverflow solution [8] on how to convert a shapefile into a `.json` file, which I do know how to read.

(This code has been adapted from this StackOverflow solution: [8])

`data_processing / process_shapefile.py`

```

1 import shapefile
2 import json
3
4 # read records of shapefile
5 reader = shapefile.Reader("data/shapefiles/GDL Shapefiles V6.3 large.shp")
6 fields = reader.fields[1:]
7 field_names = [field[0] for field in fields]
8 data = []
9 for shape_record in reader.shapeRecords():
10     print(shape_record.record) # log message
11     data.append({
12         "type": "Feature",
13         "geometry": shape_record.shape.__geo_interface__,
14         "properties": dict(zip(field_names, shape_record.record))
15     })
16
17 # write data to file
18 print('writing to file...') # log message
19 with open("data/region_coords.json", "w") as file:
20     json.dump(data, file)
21 print('done!') # log message

```

Code Snippet 3.21: Processing the Shapefile

Code Snippet 3.21 shows the code from the StackOverflow solution [8], with some slight modifications.

First, we must `import shapefile` and `import json`, to be able to handle the 2 different types of files. It then looks like we define a `reader`, similar to what we did with the `.csv` file.

It appears that shapefiles are similarly structured to databases, as we have a set of fields, and records containing values for each field. To turn this into a `.json` file, we can iterate over each record, create a dictionary out of it, and append it to a `list`. Once we have a full list, we can write it to a `.json` file.

```

Record #0: ['AFGr101', 'Asia/Pacific', 'AFG']
Record #1: ['AFGr102', 'Asia/Pacific', 'AFG']
Record #2: ['AFGr103', 'Asia/Pacific', 'AFG']
Record #3: ['AFGr104', 'Asia/Pacific', 'AFG']
Record #4: ['AFGr105', 'Asia/Pacific', 'AFG']
Record #5: ['AFGr106', 'Asia/Pacific', 'AFG']
Record #6: ['AFGr107', 'Asia/Pacific', 'AFG']
Record #7: ['AFGr108', 'Asia/Pacific', 'AFG']
Record #8: ['AGOr201', 'Africa', 'AGO']
Record #9: ['AGOr202', 'Africa', 'AGO']
Record #10: ['AGOr203', 'Africa', 'AGO']
Record #11: ['AGOr204', 'Africa', 'AGO']
Record #12: ['AGOr205', 'Africa', 'AGO']
Record #13: ['AGOr206', 'Africa', 'AGO']
Record #14: ['AGOr207', 'Africa', 'AGO']
Record #15: ['AGOr208', 'Africa', 'AGO']
Record #16: ['AGOr209', 'Africa', 'AGO']
Record #17: ['AGOr210', 'Africa', 'AGO']
Record #18: ['AGOr211', 'Africa', 'AGO']
Record #19: ['AGOr212', 'Africa', 'AGO']
Record #20: ['AGOr213', 'Africa', 'AGO']
Record #21: ['AGOr214', 'Africa', 'AGO']
Record #22: ['AGOr215', 'Africa', 'AGO']
Record #23: ['AGOr216', 'Africa', 'AGO']
Record #24: ['AGOr217', 'Africa', 'AGO']
Record #25: ['AGOr218', 'Africa', 'AGO']
Record #26: ['ALBr201', 'Europe', 'ALB']
Record #27: ['ALBr202', 'Europe', 'ALB']
Record #28: ['ALBr203', 'Europe', 'ALB']

```

Figure 3.13: Each record of the shapefile

Figure 3.13 shows each record being read from the shapefile. The list of coordinates does not appear to be here, so that must be what is stored in the `shape_record.shape.__geo_interface__` attribute.

No.	Test	Input	Type	Expectation	Output
t_{14}	Process Shapefile	—	—	Readable .json file	As Expected

Table 3.20: Testing Table for Code Snippet 3.21

```
[{"type": "Feature", "geometry": {"type": "Polygon", "coordinates": [[[68.76605987600016, 33.66236495900017], [68.75682830900007, 33.67773437400001], [68.74545288100018, 33.68974685700016], [68.72645568800016, 33.705661773000145], [68.70303344799999, 33.730712892000156], [68.69406128000014, 33.737762452000084], [68.68125152500016, 33.751533508000136], [68.67073059200015, 33.74743271000011], [68.63925170900012, 33.717758180000146], [68.6115188600001, 33.69800567700014], [68.59790801999998, 33.690837861000034], [68.56291198700012, 33.678665161000026], [68.54441833600004, 33.67549896200018], [68.53060913100012, 33.67411041200006], [68.52435302800018, 33.675918579000154], [68.51708984300006, 33.679290772], [68.532043456, 33.700199127000076], [68.51139068600008, 33.70270538400007], [68.49730682300014, 33.70201492400014], [68.48451995900012, 33.70460510300012], [68.46796417300004, 33.71432876600005], [68.46389770500002, 33.744544983000026], [68.4640884390002, 33.74955368000013], [68.46519470300007, 33.764682771000025], [68.47048187299998, 33.783905029000096], [68.50379180900006, 33.849910737000016], [68.51702880900018, 33.882026672000165], [68.52551269500015, 33.91154480000017], [68.52380371200019, 33.92324447600009], [68.51586914000006, 33.93159485000018], [68.49897003100006, 33.94415283300146], [68.48198700000006, 33.95152282600003]]}}
```

Figure 3.14: Sample of the resulting .json file

Now it is time to execute the `def getAllFactors` function on each of these regions, to make our training data.

`data_processing / compile_data.py`

```

1 import json
2 import csv
3
4 print('reading files...') # log message
5 # read json file
6 with open('data/region_coords.json', 'r') as file:
7     regionData = json.load(file)
8
9 # read csv file
10 with open('data/hdi.csv', 'r') as file:
11     reader = csv.DictReader(file)
12     hdiData = []
13     for record in reader:
14         hdiData.append(record)

```

Code Snippet 3.22: Reading the Region & HDI files

Here, I am reading the 2 files I need, to get the region data and the HDI data. We must first `import json` and `import csv`, as those are the 2 types of file we are dealing with. Reading these files is done in the same way as before, by defining a `csv.DictReader` of the file.

No.	Test	Input	Type	Expectation	Output
t_{15}	Read <code>csv</code> and <code>json</code> file	—	—	Data stored to variables	As Expected

Table 3.21: Testing Table for Code Snippet 3.22

data_processing / compile_data.py

```

16 SKIP_THE_FIRST = 0
17
18 # iterate over each region
19 for region_number, region in enumerate(regionData):
20     if region_number < SKIP_THE_FIRST:
21         continue
22     print(f'Processing {region["properties"]["gdlcode"]}...') # log
→     message

```

Code Snippet 3.23: Iterating over each region

This code will iterate over each `region`, contained within the `region_coords.json` file, by making use of a `for` loop. Carrying this out is likely to take an extremely long time, and so I might want to pause the process, or if it crashes for some reason, I won't want to start from the beginning. Therefore, I have defined a variable, `SKIP_THE_FIRST` n , which will tell the program to not calculate the first n regions. We can implement this by using the `enumerate` function, which will allow for us to keep count, and the `continue` key word, which will skip the remaining code of the `for` loop, and continue to the next iteration.

No.	Test	Input	Type	Expectation	Output
t_{16}	Skipping the first n regions	n (3, 5, 0 etc)	—	Starts on region $n + 1$	As Expected

Table 3.22: Testing Table for Code Snippet 3.23

I then noticed that from Figure 3.14, the regions seem to be in a different format to what I expected, as they were in `list[list[coordinate]]`, where `coordinate` is a `list[float]`, where I had thought it would have been just a `list[coordinate]`. This must be due to the fact that a region may contain different polygons, which, for example, may represent different islands or exclaves. To store each of these, there must be an extra `list`.

If the region contains multiple polygons (exclaves), it is often the fact that there is 1 main one, and a number of tiny ones that can be considered insignificant. Therefore, I must find which of these `list[coordinate]`s is the main one, in order to use that as my region.

data_processing / compile_data.py

```

23  # find the main bit of the region (if it has multiple)
24  shapes = region['geometry']['coordinates']
25  lengths = [len(polygon) for polygon in shapes]
26  mainShape = shapes[lengths.index(max(lengths))]
27  mainShape = [coordinate[::-1] for coordinate in mainShape]
```

Code Snippet 3.24: Finding the Main Area

Here, I am getting the lengths of each region, finding the maximum of those lengths, and using the shape at that index (of the maximum length).

After further inspection on Figure 3.14, I also noticed that in each coordinate, the longitude was at index 0, and the latitude was at index 1, which is the opposite to how everything else in this program works. Therefore, we must reverse each coordinate, to swap the indices of the latitude and longitude, such that the latitude is first.

No.	Test	Input	Type	Expectation	Output
t_{17}	Finding the main shape	First 5 regions	–	Shape with most vertices is selected	As Expected
t_{18}	Reversing Coordinates	–	–	Latitude is first	As Expected

Table 3.23: Testing Table for Code Snippet 3.24

data_processing / compile_data.py

```
1 from calc_factors import getAllFactors
```

data_processing / compile_data.py

```
17 field_names = ['country', 'region', 'code', 'hdi', 'A(House School)',  
    ↵ 'A(House Hospital)', 'A(House Pharmacy)', 'A(House Restaurant)',  
    ↵ 'A(School Hospital)', 'A(Police Hospital)', 'A(House Place of  
    ↵ Worship)', 'A(Bank Slot Machine)', 'A(Fast-Food Place Toilet)',  
    ↵ 'A(House Police)', 'A(University Library)', 'A(House Library)',  
    ↵ 'D(School)', 'D(Hospital)', 'D(Pharmacy)', 'D(Police)', 'D(Library)',  
    ↵ 'D(Toilet)', 'D(Restaurant)', 'D(Place of Worship)', 'D(Post Box)',  
    ↵ 'D(Vending Machine)', 'D(Bench)', 'D(Tree)']
```

data_processing / compile_data.py

```
30 # find all the factors  
31 allFactors = getAllFactors(mainShape)  
32 # match it with the hdi  
33 matchingHDI = [record for record in hdiData if record['code'] ==  
    ↵ region['properties']['gdlcode']]  
34 if len(matchingHDI) != 0:  
    newRecord = matchingHDI[0]  
    newRecord.update(allFactors)  
    print('writing to file...') # log message  
    with open('data/training_data.csv', 'a') as file:  
        writer = csv.DictWriter(file, fieldnames=field_names)  
        writer.writerow(newRecord)
```

Code Snippet 3.25: Writing training data to csv

Now we can calculate each of the factors for this region, by passing it into the `def getAllFactors` function. Before we do this however, we must `import` it `from calc_factors`, as it is in a different file (`calc_factors.py`).

We then need to find the “true” value for the HDI for this region, which will be in the .csv file made in Section 3.1.2. For each of the records in that .csv file, we must check if the `code` is the same as the `gdlcode` stored in this region. The number of records in this file and the number of regions are not the same, and so there may be some double-counting, or missing codes. Therefore, we want to get the first instance of the matching HDI (at index 0), if there is at least 1 record for which the codes match.

`newRecord` is then a dictionary, representing the record containing the matching HDI. We then call the `update` method, passing in `allFactors`, which will merge the `newRecord` and `allFactors` dictionaries, which will represent a new record in the training data .csv

file.

Finally, we write to the .csv file, in the same way as before, by defining a `csv.DictWriter`, specifying the field names with a `list` of field names, and writing a row. We must `open` the file in '`a`' mode (append), rather than '`w`' mode, as we don't want to erase all previous records each time we write a new one.

No.	Test	Input	Type	Expectation	Output
t_{19}	Writing Full Record	The AFGr101 region	—	HDI and all factors written to .csv	HTTP 414: URI Too Long error

Table 3.24: Testing Table for Code Snippet 3.25

```
Processing AFGr101...
getting house data...
<Response [414]>
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/requests/models.py", line 974, in json
    return complexjson.loads(self.text, **kwargs)
  File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/json/_init_.py", line 346, in loads
    return _default_decoder.decode(s)
  File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/json/decoder.py", line 337, in decode
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
  File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/json/decoder.py", line 355, in raw_decode
    raise JSONDecodeError("Expecting value", s, err.value) from None
json.decoder.JSONDecodeError: Expecting value: line 1 column 1 (char 0)
```

Figure 3.15: HTTP 414 Error

Unfortunately, this did not work and threw a HTTP 414: URI Too Long error, when trying to get the house data. As a result of this, the GET request returned an empty string, and so the rest of the error message is a `JSONDecodeError`, trying to decode an empty string into `json`.

After researching the problem, I found that GET requests have a maximum length, and going over it is considered “abusing GET”. I then went to the Overpass Turbo frontend, to test if my query works there, which it does. This lead me to believe that they aren’t using a GET request.

To find out what type of request they were using, I used the Safari Developer Tools, to see all of the incomming and outgoing traffic in the browser, including all HTTP requests.

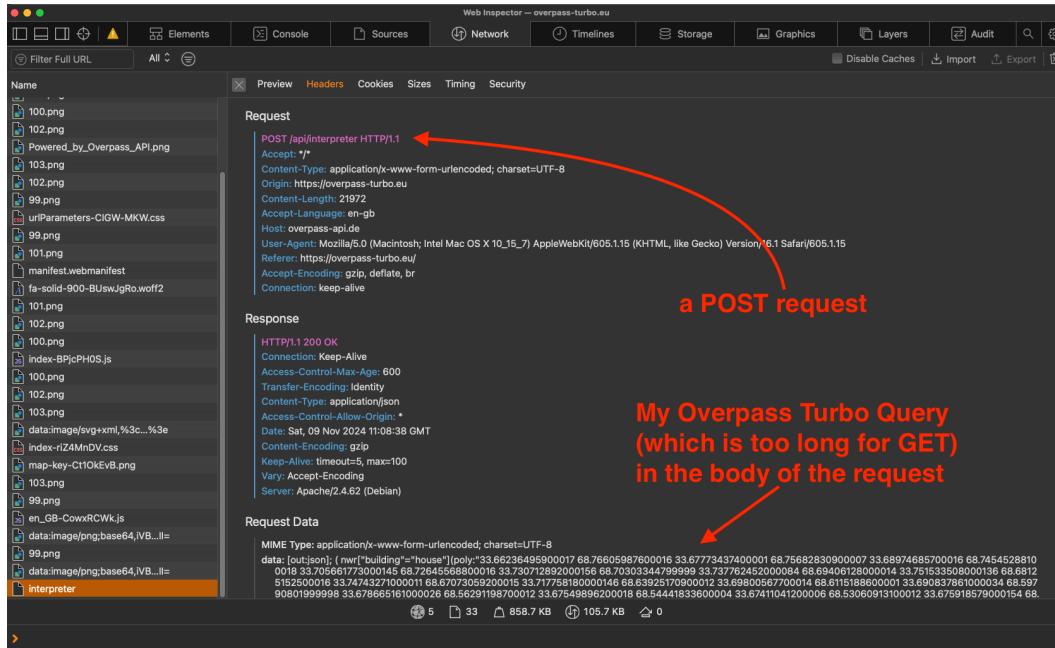


Figure 3.16: Overpass Turbo's POST request

It turns out that they are using a POST request, and sending the Overpass Turbo query in the body of the request, instead of using a GET request, and sending the query in the header.

```
data_processing / get_osm_data.py / def get_osm_data
```

```
    response = requests.post(overpass_url, data=overpass_query)
```

Code Snippet 3.26: Using a POST request instead of a GET request

This code snippet replaces the original request, and implements the same thing that Overpass Turbo are doing on their frontend (using a POST request and sending the query in the body of the request).

No.	Test	Input	Type	Expectation	Output
t_{19}	Writing Record Full	The AFGr101 region	—	HDI and all factors written to .csv	As Expected

Table 3.25: Testing Table for Code Snippet 3.26

```
Processing AFGr101...
getting house data...
<Response [200]>
getting school data...
<Response [200]>
getting hospital data...
<Response [200]>
getting pharmacy data...
<Response [200]>
getting restaurant data...
<Response [200]>
getting place_of_worship data...
<Response [200]>
getting bank data...
```

Figure 3.17: All HTTP Responses now 200: OK

Unfortunately, by the time it reached the 8th region (AFGr108), it came up with another error. This time, a HTTP 400: Bad Request error, which suggest that the format of my query is incorrect.

After looking into the problem, I found that the line to convert the `list[coordinate]` (where `coordinate` is a `list[float]` or `tuple[float]`) into a `str` (which Overpass Turbo can read as a polygon) had not worked correctly, as each coordinate was still in a `list`, rather than 2 numbers separated by a space. I then found that this region was stored as a `list[list[list[coordinate]]]`, rather than a `list[list[coordinate]]`.

```

Processing AFGr108...
[[[ [61.91029739300001, 31.939151763000154],
  [61.90763473600009, 31.92839813200004],
  [61.889488219999976, 31.897674562000134],
  [61.873565674000076, 31.861076355000023],
  [61.865787505000185, 31.84001350500006],
  [61.84838104200014, 31.777658463000023],
  [61.82876205500003, 31.733200073000035],
  [61.809722900000054, 31.707509996000113],
  [61.795295716000055, 31.693006515000093],
  [61.77845001200018, 31.676336288000016],
  [61.75602340700004, 31.66171264700006],
  [61.72592544600013, 31.65002441500019],
  [61.702384948000145, 31.642864227000132],
  [61.686256410000055, 31.634950638000078],
  [61.678058623000084, 31.626251221000132],
  [61.67567825400005, 31.613782884000102],
  [61.67888641400009, 31.607639313000107],
  [61.69254684500015, 31.598247528000115],
  [61.70100784300013, 31.590129853000178],
  [61.70471573000009, 31.574106217000065],
  [61.70099258500011, 31.560598374000165],
  [61.69563674900019, 31.553159713000127],
  [61.684169769000164, 31.543054580000046],
  [61.65518188400006, 31.529300690000184],
  [61.64990615900007, 31.518861771000047],
  [61.65106964100005, 31.504491805000157],
  [61.661556245000156, 31.48198890800012],
  [61.66323852600016, 31.472545623000144],
```

Figure 3.18: `list[list[list[coordinate]]]` debug print statement

Therefore, my initial assumption on how the regions must be stored is incorrect. After looking into the `.geojson` documentation, I found that coordinates can be stored in 1 of 4 ways:

1. '`Coordinate`' (as `coordinate`): A point on the Earth's Surface.
2. '`LineSegment`' (as `list[coordinate]`): A set of '`Coordinate`'s, joined together.
3. '`Polygon`' (as `list[list[coordinate]]`): A set of '`LineSegment`'s, to create a polygon.
4. '`MultiPolygon`' (as `list[list[list[coordinate]]]`): A collection of '`Polygon`'s.

where `coordinate` is a `list[float]`. This means that regions AFGr101 to AFGr107 were of type '`Polygon`', but region AFGr108 is of type '`MultiPolygon`'.

In my initial assumption, I had not accounted for the existence of the '`LineSegment`', which is why I was off by 1 dimension of `lists`.

data_processing / compile_data.py

```

25      # find the main bit of the region (if it has multiple)
26      shapes = region['geometry']['coordinates']
27      if region['geometry']['type'] == 'Polygon':
28          mainShape = shapes[0]
29      else:
30          lengths = [len(polygon[0]) for polygon in shapes]
31          mainShape = shapes[lengths.index(max(lengths))][0]
32      mainShape = [coordinate[::-1] for coordinate in mainShape]
```

Code Snippet 3.27: Correctly finding the Main Area

Each region must either be a '`Polygon`' or '`MultiPolygon`', with each '`Polygon`' being 1 '`LineSegment`', as nothing else makes any sense.

If the region is a '`Polygon`', then it will just be a single '`LineSegment`', and so we can find it at the index 0.

Otherwise, it must be a '`MultiPolygon`', and so this is where we iterate over the lengths and find the main shape. We must also use the 0 index here, to retrieve the first (and only) '`LineSegment`' from the '`Polygon`'.

Afterwards, we must of course still reverse each coordinate, as they are still the wrong way around.

No.	Test	Input	Type	Expectation	Output
t_{19}	Writing Full Record	The AFGri08 region	—	HDI and all factors written to .csv	As Expected

Table 3.26: Testing Table for Code Snippet 3.27

After doing this, I found that it was easier to specify the most recent region, instead of the number of regions done, when not starting from the beginning:

```
data_processing / compile_data.py
```

```
17 MOST_RECENT_REGION = 'AFGr108'
18 started = False
```

```
data_processing / compile_data.py
```

```
23     if not started:
24         if region['properties']['gdlcode'] == MOST_RECENT_REGION:
25             started = True
26         continue
```

Code Snippet 3.28: Specifying the most recent region

No.	Test	Input	Type	Expectation	Output
t_{16}	Skipping the first n regions	Most recent region	—	Starts on the next region	As Expected

Table 3.27: Testing Table for Code Snippet 3.28

Everything was downloading successfully until I got to Fiji, where it threw another HTTP 400 error, which meant there was something wrong with my prompt again.

When pasting it into the Overpass Turbo frontend, it came back with an error, saying that I could not have coordinates whose longitude is greater than 180° , or less than -180° . This makes sense as to why this error only came about when processing Fiji, as it almost lies on the international dateline.

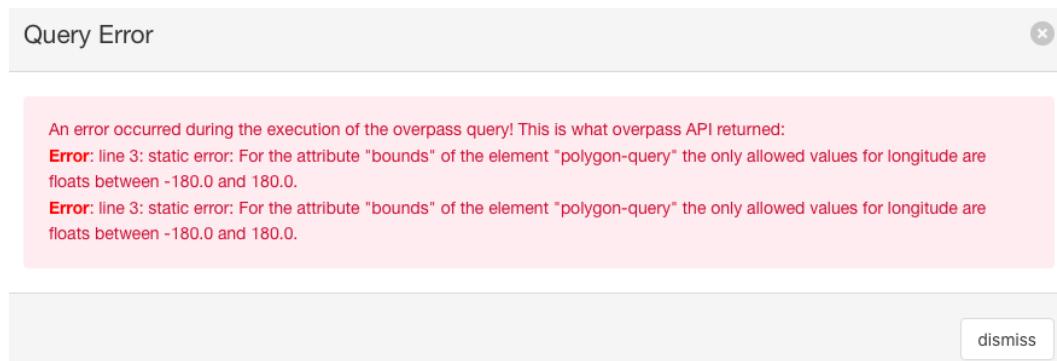


Figure 3.19: Longitude Error Message

To fix this, we can add 180° to it, and then mod 360° , and then subtract 180° again, such that $-180^\circ \leq Lo \leq 180^\circ$.

```
data_processing / get_osm_data.py / def get_osm_data
```

```
7     bounding_coords = [(latitude, (longitude + 180) % 360 - 180) for
    ↵   latitude, longitude in bounding_coords]
```

Code Snippet 3.29: Finding the right Longitude

No.	Test	Input	Type	Expectation	Output
t_{20}	Finding the right longitude	Longitude not inbetween -180° and 180°	–	Equivalent longitude inbetween -180° and 180°	As Expected

Table 3.28: Testing Table for Code Snippet 3.29

Again, everything was downloading successfully, until this time Scotland caused a problem, throwing yet another HTTP 400 error.

When I went to copy and paste the prompt into Overpass Turbo, in hopes of getting a similar error message, I quickly discovered that this prompt was much larger than the others, and had trouble copy and pasting it, as it was approximately 50 megabytes of text.

This was because Scotland was stored as a polygon with more than 500,000 vertices, most likely due to its irregular shape and jagged coastline, and so Overpass Turbo must have not been able to handle such a massive prompt.

Therefore, I decided to truncate regions with more than 100,000 vertices, by removing every other vertex until there were less than 100,000.

```
data_processing / get_osm_data.py / def get_osm_data
```

```
7     while len(bounding_coords) >= 100000:
8         print(f'truncating region with {len(bounding_coords)}')
    ↵   vertices...') # log message
9         bounding_coords = [bounding_coord for n, bounding_coord in
    ↵   enumerate(bounding_coords) if n % 2 == 0]
```

Code Snippet 3.30: Truncating Detailed Regions

No.	Test	Input	Type	Expectation	Output
t_{21}	Truncating Detailed Regions	Region with more than 100,000 vertices	–	Less detailed region of the same area	As Expected

Table 3.29: Testing Table for Code Snippet 3.30

The rest of the process went smoothly, with no other errors, except for the occasional time out.

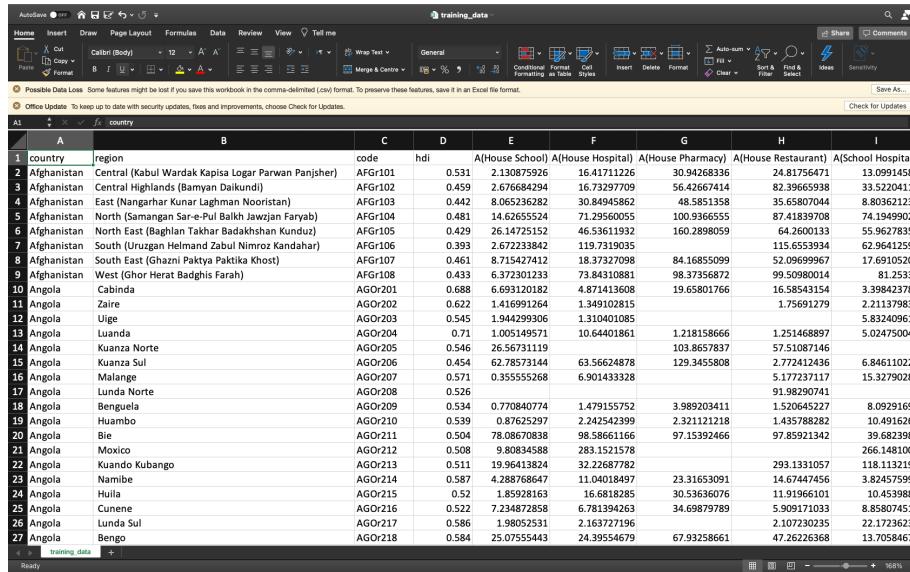


Figure 3.20: Final Training Data

Figure 3.20 shows the final .csv file, containing the training data for the neural network, which is what I had hoped to have achieved by the end of this milestone.

I can now therefore delete `region_coords.json` and `hdi.csv`, as I only needed them to create `training_data.csv`.

3.1.7 Review

No.	Test	Inputs	Pass / Fail
T_1	Overpass Turbo Query	Closed Polygon, Open Polygon, Valid object type, Invalid object type	Pass
T_2	Formula for distance between 2 points	2 Points in the same hemisphere, 2 Points in opposite hemispheres, 2 Points either side of the international date line, 2 Points whose shortest distance crosses the North Pole	Pass
T_3	$A(x, y)$	Less than 500 x -objects, 500 x -objects, More than 500 x -objects, 0 x -objects	Pass

T_4	$D(x)$	Region drawn Clockwise, Region drawn Anticlockwise, Region drawn in multiple loops of the Earth, Region that self-intersects	Pass
-------	--------	---	------

Table 3.30: Passed Testing Data for Milestone 1

After some successful debugging, all 4 of the main tests in Milestone 1 passed. As a result of that, the following success criteria has been met:

No.	Success Criterion	Justification	Success
S_1	Find suitable HDI training data for the Neural Network	The neural network must train on pre-calculated HDI values for certain areas, along with their respective density and average distance factors.	✓
S_2	Retrieve OSM data for a given area using Overpass Turbo	This will allow me to calculate useful geographical factors to predict the HDI from instead of more human ones (which are harder to determine).	✓
S_3	Calculate average distance factors from a given area	The average distance from <i>some object</i> to <i>some other object</i> is also a good measure to consider. For example <i>A</i> (House, School) is a useful factor as school children shouldn't have to travel long distances for education.	✓
S_4	Calculate density factors from a given area	The number of <i>some object</i> per unit area is a good way of measuring how many of that object are in a particular area while removing the bias of larger areas.	✓

Table 3.31: Achieved Success Criteria for Milestone 1

I also have a full set of training data, so overall, I believe this milestone has been a success.

3.2 Milestone 2 – Multilayer Perceptron

3.2.1 Success Criteria

By the end of this milestone, I hope to have a fully trained neural network, which is able of predicting HDI from a set of factors.

No.	Success Criterion	Justification
S_5	Successfully Implement the Feedforward Algorithm	This algorithm will allow us to make predictions on the HDI given a set of factors.
S_6	Successfully Implement the Backpropagation Algorithm	This algorithm will allow us to train the neural network base on a set of training examples.
S_7	Successfully train the Neural Network	In order to accurately predict the HDI of a given area, the neural network must train on existing examples to see what makes a high HDI.

Table 3.32: Success Criteria for Milestone 2

To determine if I have met this success criteria, I will be using the following test:

No.	Test	Inputs	Expected Output/Justification
T_5	Neural Network works	Using a <i>different</i> data set, e.g. handwritten digits, which certainly has enough training data	This is to ensure that the neural network works, and if it doesn't work for predicting HDI, it is as a result of a lack of training data.

Table 3.33: Testing Data for Milestone 2

Again, determining if the neural network is actually working is very difficult, so I intend to train a neural network to do something else, like recognise handwritten digits, which certainly has enough training data, to determine if it is working in principle.

3.2.2 Part 6 – Implementing the Neural Network Structure

We must first define the constructor method for a Multilayer Perceptron.

neural_network.py

```

1 import numpy as np
2
3 class MultilayerPerceptron(object):
4     # constructor method
5     def __init__(self, layer_sizes):
6         # initialise layer sizes constant
7         self.layer_sizes = layer_sizes
8         self.L = len(layer_sizes)-1

```

Code Snippet 3.31: Defining the Constructor Method

Here, I am importing `numpy`, as it will be very useful in all the calculations we will be doing in this class.

Next, we define the `class MultilayerPerceptron`, which inherits from the generic python `object`. As stated before, we must only pass the `layer_sizes` into the constructor method (as well as `self`), as everything else can be defined implicitly from that.

The `layer_sizes` attribute will obviously be set to the `layer_sizes` parameter, and the `L` attribute is the number of layers, not including the input layer, so we subtract 1 from the length of the `layer_sizes` `list`.

neural_network.py / `class MultilayerPerceptron / def __init__`

```

9     # initialise matrices for the different quantities
10    self.activations = [np.matrix(np.zeros(shape=(layer_size,1))) for
11        ↳ layer_size in layer_sizes]
12    self.weights =
13        ↳ [np.matrix(np.random.randn(layer_sizes[index+1],layer_size))
14            ↳ for index, layer_size in enumerate(layer_sizes[:-1])]
15    self.biases = [np.matrix(np.random.randn(layer_size,1)) for
16        ↳ layer_size in layer_sizes]
17    self.z = [np.matrix(np.zeros(shape=(layer_size,1))) for layer_size
18        ↳ in layer_sizes]
19    self.errors = [np.matrix(np.zeros(shape=(layer_size,1))) for
20        ↳ layer_size in layer_sizes]

```

Code Snippet 3.32: Initialising the Matrices

Next, I added the attributes of all the `lists` of matrices, and column vectors, representing the activations, weights, biases, z -values and errors within the network.

The activations of the neurons are stored as a `list` of column vectors, where the size of each column vector is that layer's size. The activations will initially be set to 0, so we can use the `np.zeros` function to get a multidimensional array of 0s. The size of multidimensional

array we want is `layer_size` rows and 1 column, for some `layer_size`. To turn this into a column vector, we pass this into the `np.matrix` function, as done before. To get the full `list`, we must iterate this process for each of the `layer_sizes` in `layer_sizes`. The `z` and `errors` attributes are set to the same lists, as they are also quantities stored in neurons and should also be 0s to begin with.

Bias is also a quantity stored in the neurons, but should not be set to 0 to begin with, as it is a parameter of the system. Instead it should be set to a random number, which has been picked from a random distribution. To get a sensible range of values, I used the standard normal distribution (using `np.random.randn` in place of `np.zeros`):

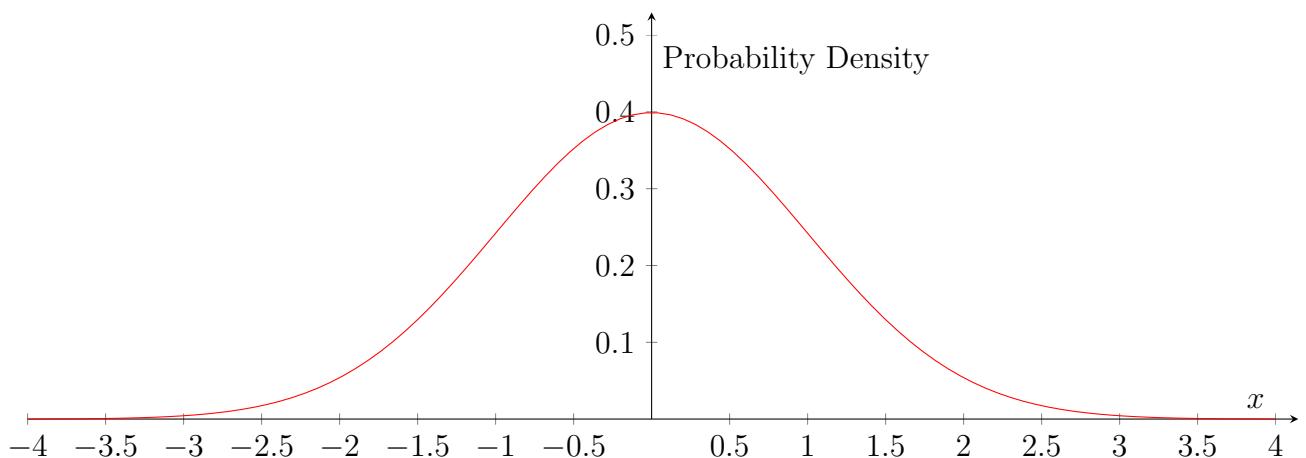


Figure 3.21: Graph of the Standard Normal Distribution

Figure 3.21 visually shows the normal distribution, and its bell-shaped curve. Most of the random values should therefore be in between -2 and 2 , with very few being outside that range.

Weights are also parameters of the system, and so should also get random values from the normal distribution. However, weights should be stored as a matrix between 2 layers, and so the shape must be different. Here, the number of rows must be the size of the next layer (`layer_sizes[index+1]`), and the number of columns must be the size of the current layer (`layer_size`). This time, instead of iterating over all of the layer sizes, we must only go up to 1 before the end, again due to the fact that weights go in between the layers.

```

activations
[matrix([[0.],
          [0.]]),
 matrix([[0.],
          [0.],
          [0.]]),
 matrix([[0.],
          [0.],
          [0.],
          [0.]]])
weights
[[matrix([[ 0.00724576, -0.79401028],
           [-0.30978017,  0.20706976],
           [-0.0887502 ,  0.68012537]]),
  matrix([[-1.8620002 ,  0.23609408,  0.94124899],
         [-0.56005494, -0.11782026, -0.1842248 ],
         [-0.99255937, -0.75466923, -0.76560838],
         [ 1.13878057,  2.803111 ,  1.37908258]])]
biases
[[matrix([[-1.63383171],
          [ 0.84018748]]),
  matrix([[ 0.94904602],
         [-0.7174148 ],
         [ 0.37237926]]),
  matrix([[-0.60068549],
         [-1.26667354],
         [-0.19706229],
         [ 1.89735122]])]

```

Figure 3.22: Printing activations, weights and biases

Figure 3.22 shows the output if we print `self.activations`, `self.weights` and `self.biases`, for a neural network with `layer_sizes = [2,3,4]` (`self.z` and `self.errors` will be the same as `self.activations`).

The final attributes we must define are the copies of the activations, z -values and errors used in training.

`neural_network.py`

```

2 import copy

neural_network.py / class MultilayerPerceptron / def __init__

16     # initialise lists for training states
17     self.training_activations = []
18     self.training_z = []
19     self.training_errors = []

```

Code Snippet 3.33: Initialising the Training Lists

Right now, they are set as empty lists, as we have not done any training yet.

In anticipation of making copies of the attributes, I have also imported the `copy` library.

No.	Test	Input	Type	Expectation	Output
t_{22}	Setting the activations <code>list</code>	(for example) Layer sizes = 2, 3, 4	–	A column vector of size 2, a column vector of size 3 and a column vector of size 4, filled with 0s	As Expected
t_{23}	Setting the weights <code>list</code>	(for example) Layer sizes = 2, 3, 4	–	A matrix of size 3×2 and a matrix of size 4×3 , filled with random numbers from the normal distribution	As Expected
t_{24}	Setting the biases <code>list</code>	(for example) Layer sizes = 2, 3, 4	–	A column vector of size 2, a column vector of size 3 and a column vector of size 4, filled with random numbers from the normal distribution	As Expected
t_{25}	Setting the training <code>lists</code>	–	–	Empty <code>lists</code>	As Expected

Table 3.34: Testing Table for Code Snippets 3.31, 3.32 & 3.33

We must also define the sigmoid function, to use as our activation function.

`neural_network.py`

```

21 # activation function
22 def sigmoid(x):
23     return 1/(1+np.exp(-x))

```

Code Snippet 3.34: Defining the Sigmoid Function

This matches with the definition stated in equation 2.22. In the class diagram (Figure 2.29), I had made this function a method in the `class MultilayerPerceptron`, but now I believe that it makes more sense for it to be a stand-alone function, as it does not use any of the attributes from the class.

We must also define the derivative of the sigmoid function, as it will be used in the back-propogation algorithm. We can differentiate it using the quotient rule, and then rearrange it so that it is in terms of $\sigma(x)$:

$$\begin{aligned}
 \sigma'(x) &= \frac{d}{dx} \left(\frac{1}{1+e^{-x}} \right) \\
 &= \frac{(0)(1+e^{-x}) - (1)(-e^{-x})}{(1+e^{-x})^2} \\
 &= \frac{e^{-x}}{(1+e^{-x})^2} \\
 &= \left(\frac{e^{-x}}{1+e^{-x}} \right) \left(\frac{1}{1+e^{-x}} \right) \\
 &= \left(\frac{1+e^{-x}-1}{1+e^{-x}} \right) \left(\frac{1}{1+e^{-x}} \right) \\
 &= \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}} \right) \left(\frac{1}{1+e^{-x}} \right) \\
 &= (1 - \sigma(x))(\sigma(x))
 \end{aligned} \tag{3.1}$$

`neural_network.py`

```

25 # derivative of the activation function
26 def sigmoid_prime(x):
27     return sigmoid(x)*(1-sigmoid(x))

```

Code Snippet 3.35: Defining the derivative of the Sigmoid Function

Again, this was previously planned to be a method in the `class MultilayerPerceptron`, but I have since decided to move it out.

No.	Test	Input	Type	Expectation	Output
t_{26}	Sigmoid Function	Real numbers (for example, 0, 5, -2.73)	-	For those examples: 0.5, 0.9931, 0.0612	As Expected
t_{27}	Sigmoid Prime Function	Real numbers (for example, 0, -1, 1.78)	-	For those examples: 0.25, 0.1966, 0.1235	As Expected

Table 3.35: Testing Table for Code Snippets 3.34 & 3.35

We must also create methods for saving and loading weights and biases.

`neural_network.py`

```

3 import pickle

neural_network.py / class MultilayerPerceptron

22 # saves weights and biases to an external file in the models folder
23 def save_model(self, filename):
24     parameters = {
25         'weights': self.weights,
26         'biases': self.biases
27     }
28     with open(f'models/{filename}.pkl', 'wb') as file:
29         pickle.dump(parameters, file)

```

Code Snippet 3.36: Saving weights and biases

As stated before, we can save the weights and biases attributes to a python `dict`, and then write that dictionary to a file. I believe the most suitable file type is the .pkl binary file, which can be used to store variables. As this is a binary file, we must open it in '`'wb'`' mode (w for write, b for binary), and then dump the `parameters` dictionary using the `pickle` module (which we must import first).

`neural_network.py / class MultilayerPerceptron`

```

31 # loads a model from a file path
32 def load_model(self, file_path):
33     with open(file_path, 'rb') as file:
34         parameters = pickle.load(file)
35         self.weights = parameters['weights']
36         self.biases = parameters['biases']

```

Code Snippet 3.37: Loading Models

To load the model, we can do the same thing but in reverse. First, open the file in '`'rb'`' mode (r for read, b for binary) and load the dictionary into a `parameters` variable. We can then set the `weights` and `biases` attributes to the respective elements in the dictionary.

No.	Test	Input	Type	Expectation	Output
t_{28}	Saving a Model	–	–	Weights and biases saved to an external binary file	As Expected
t_{29}	Loading a Model	–	Valid	Attributes set to the same weights and biases	As Expected

Table 3.36: Testing Table for Code Snippets 3.36 & 3.37

We should only be able to load weights and biases if they originally came from a network with the same layer sizes.

```
neural_network.py / class MultilayerPerceptron / def save_model
```

```
24     parameters = {
25         'weights': self.weights,
26         'biases': self.biases,
27         'layer_sizes': self.layer_sizes
28     }
```

```
neural_network.py / class MultilayerPerceptron / def load_model
```

```
36     if self.layer_sizes != parameters['layer_sizes']:
37         raise Exception('layer sizes do not match!')
```

Code Snippet 3.38: Ensuring Layer Sizes are the Same

To do this, we can also store the `layer_sizes` attribute in the dictionary which is written to file, and then when loading, we can check if the `layer_sizes` in the dictionary matches this instance's `layer_sizes`, before we set the weights and biases. If it does not match, then we can `raise` an `Exception`, to say that the layer sizes do not match.

No.	Test	Input	Type	Expectation	Output
t_{30}	Loading a Model with different layer sizes	–	Invalid	Error is thrown	As Expected

Table 3.37: Testing Table for Code Snippet 3.38

3.2.3 Part 7 – Implementing the Feedforward Algorithm & Prediction

Now it's time to implement the Feedforward algorithm, as shown in the flowchart in Figure 2.23.

```
neural_network.py / class MultilayerPerceptron
```

```
22     # carry out the feedforward algorithm
23     def feedforward(self):
24         for layer in range(0, self.L):
25             self.z[layer+1] = np.matmul(self.weights[layer],
26                                     self.activations[layer]) + self.biases[layer+1]
27             self.activations[layer+1] = sigmoid(self.z[layer+1])
```

Code Snippet 3.39: Implementing the Feedforward Algorithm

In the flowchart and the class diagram, I had said the input layer would be an input/parameter to this function, but I have since decided it would make more sense to just directly access it from the `self.activations` attribute (it will be at index 0).

Next, according to the flowchart, we must iterate for l from 0 to $L-1$, which can be simply done with a python `for` loop (iterator variable renamed to `layer` for greater readability).

Then for each of the values for l (`layer`), we must compute $\mathbf{a}^{(l+1)} = \sigma(\mathbf{W}^{(l)}\mathbf{a}^{(l)} + \mathbf{b}^{(l+1)})$ (equation 2.29). As we will also need to get the various values for z during training, I have broken this expression up into $\mathbf{z}^{(l+1)} = \mathbf{W}^{(l)}\mathbf{a}^{(l)} + \mathbf{b}^{(l+1)}$ and $\mathbf{a}^{(l+1)} = \sigma(\mathbf{z}^{(l+1)})$, which you can see implemented on lines 25 and 26.

```
neural_network.py / class MultilayerPerceptron
```

```
28     # run the feedforward algorithm on a set of input data and return the
29     # result
30     def predict(self, input_data):
31         self.activations[0] = input_data
32         self.feedforward()
33         return self.activations[self.L]
```

Code Snippet 3.40: Making a Prediction

If we then want to more formally make a prediction, we must first get the input data (the parameter of this `def predict` function), then feed it forward through the network, and return the activations in the output layer (the layer at index L).

No.	Test	Input	Type	Expectation	Output
t_{31}	Feedforward Algorithm	—	—	All activations and z -values are updated	As Expected
t_{32}	Making a Prediction	—	—	Returns the output layer of neurons	As Expected

Table 3.38: Testing Table for Code Snippets 3.39 & 3.40

Again, due to the nature of the calculations involved in the network, it is difficult to test these algorithms. Therefore, I was only looking for the correct structure of the answers (e.g. the correct dimensions of matrix) to determine if the output is as expected.

3.2.4 Part 8 – Implementing the Backpropagation Algorithm & Training

If we actually try to make a prediction, the network will return absolute nonsense, and so we must be able to train it, using the backpropagation algorithm.

`neural_network.py / class MultilayerPerceptron`

```

28     # carry out the backpropagation algorithm
29     def backpropagate(self, examples):
30         # reset the training lists
31         # iterate over each example
32         for example in examples:
33             # reset the activations, z-values and errors
34             # make a feedforward pass
35             # calculate the error in the output layer
36             # backpropagate the error to previous layers
37             # save the activations, z-values and errors
38             # use gradient descent to update the weights and biases

```

Code Snippet 3.41: Backpropagation Algorithm Structure

Here, in Code Snippet 3.41, I have outlined the general backpropagation algorithm, as shown in the flowchart in Figure 2.26, with some additional steps we need to consider when practically carrying it out (for example, resetting the lists).

```

neural_network.py / class MultilayerPerceptron / def backpropagate

30     # reset the training lists
31     self.training_activations = []
32     self.training_z = []
33     self.training_errors = []
34     # iterate over each example
35     for example in examples:
36         # reset the activations, z-values and errors
37         self.activations = [np.matrix(np.zeros(shape=(layer_size,1)))
38             ↪ for layer_size in self.layer_sizes]
39         self.z = [np.matrix(np.zeros(shape=(layer_size,1))) for
40             ↪ layer_size in self.layer_sizes]
41         self.errors = [np.matrix(np.zeros(shape=(layer_size,1))) for
42             ↪ layer_size in self.layer_sizes]

```

Code Snippet 3.42: Resetting the Lists

Resetting these lists is easy enough, as all we need to do is ensure the training lists are empty, and reset the activations, z -values and errors to 0 when processing each example (by doing the same things as in the constructor method).

```

neural_network.py / class MultilayerPerceptron / def backpropagate

40     # make a feedforward pass
41     self.activations[0] = example['input']
42     self.feedforward()
43     # calculate the error in the output layer
44     self.errors[self.L] = np.multiply(
45         self.activations[self.L] - example['output'],
46         sigmoid_prime(self.z[self.L]))
47

```

Code Snippet 3.43: Calculating the error in the output layer

Next, we must feedforward the example's input, to get the network's current prediction, and then compare that with the "correct" output to get the error.

We must use the formula $\delta^{(L)} = (\mathbf{a}^{(L)} - \mathbf{y}) \odot (\sigma'(\mathbf{z}^{(L)}))$ (equation 2.41) to calculate the error in the output layer, which has been implemented in lines 44 to 47 (the `np.multiply` function carries out the \odot operation).

```
neural_network.py / class MultilayerPerceptron / def backpropagate
```

```

48      # backpropogate the error to previous layers
49      for layer in range(self.L, 1, -1):
50          self.errors[layer-1] = np.multiply(
51              np.matmul(
52                  self.weights[layer-1].T,
53                  self.errors[layer]
54              ),
55              sigmoid_prime(self.z[layer-1])
56          )
57      # save the activations, z-values and errors
58      self.training_activations.append(
59          copy.deepcopy(self.activations))
60      self.training_z.append(copy.deepcopy(self.z))
61      self.training_errors.append(copy.deepcopy(self.errors))
```

Code Snippet 3.44: Backpropogating the Error

After this, we can backpropogate the error, using the formula $\delta^{(l-1)} = \left((\mathbf{W}^{(l-1)})^T \delta^{(l)} \right) \odot (\sigma'(\mathbf{z}^{(l-1)}))$ (equation 2.51), for each l from L to 2. The iteration is easily implemented with a `for` loop on line 49, and the equation is implemented in lines 50 to 56. The attribute `.T` of a `np.matrix` is the transpose of that matrix, as required by the formula.

Now that we have a full set of activations, z -values, and errors, we can save them to their respective lists to be able to compute the gradient of the cost function.

```

neural_network.py / class MultilayerPerceptron

29     def backpropogate(self, examples, learning_rate):

neural_network.py / class MultilayerPerceptron / def backpropogate

61         # use gradient descent to update the weights and biases
62         for layer in range(self.L, 0, -1):
63             self.weights[layer-1] -= (learning_rate/len(examples)) * sum(
64                 [np.matmul(
65                     self.training_errors[example][layer],
66                     self.training_activations[example][layer-1].T
67                 ) for example in range(len(examples))],
68                 np.matrix(np.zeros(self.weights[layer-1].shape)) # (extra
69                 # argument in sum function to add matrices instead of
70                 # numbers)
71             )
72             self.biases[layer] -= (learning_rate/len(examples)) * sum(
73                 [self.training_errors[example][layer] for example in
74                  range(len(examples))],
75                 np.matrix(np.zeros(self.biases[layer].shape)) # (extra
76                 # argument in sum function to add matrices instead of
77                 # numbers)
78             )

```

Code Snippet 3.45: Updating the weights and biases

Finally, we must update the weights using the formula $\mathbf{W}^{(l-1)} = \mathbf{W}^{(l-1)} - \frac{\eta}{m} \sum_{x=1}^m \delta_x^{(l)} (\mathbf{a}_x^{(l-1)})^T$ (equation 2.56), implemented on lines 63 to 69.

And we must update the biases using the formula $\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \frac{\eta}{m} \sum_{x=1}^m \delta_x^{(l)}$ (equation 2.57), implemented on lines 70 to 73. (for each of the layers from the final layer, to the input layer).

One of the terms in these equations is η , the learning rate, which is just a constant to determine how fast the network should learn. I had forgotten to add this as a parameter of the `def backpropogate` function in the class diagram, but I have since added it here.

Both of these formulas also involve summing over multiple matrices, which I have implemented with the default python `sum` function. As we are summing matrices instead of numbers, we must also pass in a matrix of 0s, of the same dimension as the matrix we are updating, for the sum to work.

This should be the full backpropagation algorithm, and so we are ready to test it. All I am looking for at the moment is that it does not throw an error, as it is impossible to determine if the answer is correct or not without carrying out the algorithm again.

No.	Test	Input	Type	Expectation	Output
t_{33}	Backpropogation algorithm doesn't throw an error	—	—	No errors when carrying out the backpropogation algorithm	An Error

Table 3.39: Testing Table for Code Snippets 3.41, 3.42, 3.43, 3.44 & 3.45

```
File "/Users/luke/Desktop/FOLDERS/School/Subjects/Computing/A-Level CS/Project
      return sigmoid(x)*(1-sigmoid(x))
File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-
      return N.dot(self, asmatrix(other))
ValueError: shapes (3,1) and (3,1) not aligned: 1 (dim 1) != 3 (dim 0)
```

Figure 3.23: Backpropogation Error

The error seems to be that we are attempty to carry out matrix multiplication between $\sigma(x)$ and $1 - \sigma(x)$.

`neural_network.py`

```
100 # derivative of the activation function
101 def sigmoid_prime(x):
102     return np.multiply(sigmoid(x), 1-sigmoid(x))
```

Code Snippet 3.46: Fixing the σ' function

We can fix this by specifying we want to do an element-wise multiplication `np.multiply` instead of a matrix multiplication.

No.	Test	Input	Type	Expectation	Output
t_{33}	Backpropogation algorithm doesn't throw an error	—	—	No errors when carrying out the backpropogation algorithm	As Expected

Table 3.40: Testing Table for Code Snippet 3.46

The backpropogation algorithm now does not crash. Whether or not it actually works will be tested shortly, in the next section.

`neural_network.py`

```

4   import random

neural_network.py / class MultilayerPerceptron

82  # carry out stochastic gradient descent over a number of epochs to
     → train a model
83  def train(self, examples, mini_batch_size, num_epochs, learning_rate):
84      for epoch in range(1, num_epochs+1):
85          print(f'training epoch {epoch}/{num_epochs}...') # log message
86          # randomly split into mini batches
87          random.shuffle(examples)
88          mini_batches = [examples[(batch_number *
     → mini_batch_size):(batch_number+1) * mini_batch_size]] for
     → batch_number in range(int(np.ceil(len(examples) /
     → mini_batch_size)))]
89          # backpropogate for each mini batch
90          for mini_batch in mini_batches:
91              self.backpropagate(mini_batch, learning_rate)

```

Code Snippet 3.47: Implementing Stochastic Gradient Descent

In order to fully train the network, we must split the training examples into mini batches of a particular size, and then carry out the backpropogation algorithm on each of those. We then may repeat the process over a number of “epochs”, to further train the model.

No.	Test	Input	Type	Expectation	Output
t_{34}	Stochastic Gradient Descent doesn’t throw an error	—	—	No errors when carrying out the full training algorithm	As Expected

Table 3.41: Testing Table for Code Snippet 3.47

3.2.5 Part 9 – Testing the Neural Network

The `class MultilayerPerceptron` is now complete, and can be used to make predictions.

Before training it on my HDI training data, I am going to first test that it actually works, by training it on something else with a lot of training data. Both Grant Sanderson and Micheal Nielsen (who wrote the resources I used to learn about neural networks) used the MNIST handwritten digits data set to test their networks, so I have decided to do the

same thing. This is because it has been proven that this method works to make accurate predictions on these handwritten digits.

The MNIST data set contains 70000 images of handwritten digits, each attached with a number 0 to 9, to say which digit it is supposed to be. Each image is grayscale and 28 by 28 pixels, and so is therefore made of $28^2 = 784$ numbers between 0 and 1, which represent the brightness of each pixel.

	image	label
0	4	4
1	1	1
2	0	0
3	7	7
4	8	8
5	1	1
6	2	2
7	7	7
8	1	1
9	6	6

Figure 3.24: Examples from the MNIST Data Set

Figure 3.24 shows 10 of the handwritten digits from the MNIST data set, and their respective labels.

I first needed to download the dataset and read it. I decided to retrieve it and some code to read it from Micheal Nielsen's code repository [9], to ensure that I get the correct data.

(This code has been adapted from Micheal Nielsen's code repository [9])

`network_training / load_digits_data.py`

```

1 import pickle
2 import gzip
3
4 # loads the MNIST data set
5 def load_data():
6     with gzip.open('data/mnist.pkl.gz', 'rb') as file:
7         unpickled = pickle._Unpickler(file)
8         unpickled.encoding = 'latin1'
9         training_data, validation_data, test_data = unpickled.load()
10    return (training_data, validation_data, test_data)

```

Code Snippet 3.48: Loading in the MNIST data set

The MNIST data set seems to be in the `.pkl.gz` format, which means we must open it with the `gzip` library, and then unpickle it with the `pickle` module.

We can then load the training data, validation data, and testing data from the unpickled and uncompressed file. The 70000 examples are split up into 50000 training examples, 10000 validation examples and 10000 testing examples. I will only be using the 50000 training examples and the 10000 testing examples for the purposes of testing my network.

We now must convert this data into the correct format, to feed into our neural network.

```
network_training / load_digits_data.py
```

```
3 import numpy as np
```

(This code has been adapted from Micheal Nielsen's code repository [9])

```
network_training / load_digits_data.py
```

```
13 # makes a vector for the output layer from the correct output label
14 def convert_to_vector(correct_output):
15     output_layer = np.zeros((10, 1))
16     output_layer[correct_output] = 1
17     return output_layer
18
19 training_data, _, test_data = load_data()
20 # convert training data into the right format
21 training_inputs = [np.reshape(input_data, (784, 1)) for input_data in
22     ↪ training_data[0]]
22 training_outputs = [convert_to_vector(output_data) for output_data in
23     ↪ training_data[1]]
23 training_data = [
24     {'input': training_inputs[example],
25      'output': training_outputs[example]}
26 } for example in range(len(training_inputs))]
27 # convert testing data into the right format
28 test_inputs = [np.reshape(input_data, (784, 1)) for input_data in
29     ↪ test_data[0]]
29 test_outputs = test_data[1]
30 test_data = [
31     {'input': test_inputs[example],
32      'output': test_outputs[example]}
33 } for example in range(len(test_inputs))]
```

Code Snippet 3.49: Converting the MNIST data to the right format

First, we must load the data using the function defined in the previous code snippet. It returns all 3 data sets, but I only need the training one and the testing one, so in place of

the validation data I have used an underscore, to show that that return variable will not be used.

As each image is made of 784 grayscale pixels, that means our input layer must have 784 neurons, and so should therefore be a matrix with 1 column and 784 rows (hence why we `np.reshape` it).

The neural network will have 10 neurons in the output layer, representing the confidence in each digit. For example, an output layer of [0, 0, 0.9, 0, 0, 0.1, 0, 0, 0, 0] would suggest that the network believe that particular digit is a 2, as 0.9 > all of the other activations.

Therefore, we must write a function, `def convert_to_vector`, which takes in the correct output number, and creates the output layer vector from it. This will be a matrix with 10 rows and 1 column, filled with 0s, except for at the index of the correct answer, where it will be 1. This function must then be applied to all items in the training outputs.

We can then save these 2 variables, `training_inputs` and `training_outputs` to a dictionary ready for the neural network to read.

This process must be repeated for the test data, except we don't need to convert the output to a vector, as we will instead be converting the vector generated by the network into a formal prediction (1 digit).

`network_training / load_digits_data.py`

```

29 # save data to a file
30 with open(f'data/digits.pkl', 'wb') as file:
31     pickle.dump({'training': training_data, 'testing': test_data}, file)

```

Code Snippet 3.50: Saving the MNIST data to a file

Once we have all of the training data and testing data, we can save it to a file, again using `pickle`.

No.	Test	Input	Type	Expectation	Output
t_{35}	Convert MNIST training and testing data to right format	Strange format	—	Correct dimensions of matrix, in a dictionary with input and output keys	As Expected

Table 3.42: Testing Table for Code Snippets 3.48, 3.49 & 3.50

network_training / train_digits.py

```

1 import pickle
2
3 # load the training data from the file
4 with open('data/digits.pkl', 'rb') as file:
5     data = pickle.load(file)
6 training_data = data['training']
7
8 # import the class MultilayerPerceptron from the neural_network.py file
9 from neural_network import MultilayerPerceptron

```

Code Snippet 3.51: Loading the MNIST Training Data

In a different file, for training the network, we can then read this data and select only the training data from the dictionary.

We then need to import the `class MultilayerPerceptron` from the `neural_network.py` file. Unfortunately, this returned a `ModuleNotFoundError`.

```

train_digits.py", line 14, in <module>
    from neural_network import MultilayerPerceptron
ModuleNotFoundError: No module named 'neural_network'

```

Figure 3.25: `ModuleNotFoundError`

This is because the current file (`train_digits.py`) is in another directory (`network_training`) from `neural_network.py`. We must therefore import from a parent directory.

I did not know how to resolve this issue, but after researching the problem, I found a GeeksforGeeks solution [10] on how to import from a parent directory.

(This code has been adapted from this GeeksforGeeks solution: [10])

network_training / train_digits.py

```

8 # import the class MultilayerPerceptron from the neural_network.py file
9 import os
10 import sys
11 current = os.path.dirname(os.path.realpath(__file__))
12 parent = os.path.dirname(current)
13 sys.path.append(parent)
14 from neural_network import MultilayerPerceptron

```

Code Snippet 3.52: Importing from a Parent Directory

This solution seems to be getting around the problem by finding the “real path” of the

current file (`__file__`), finding its parent directory, and adding that to the `sys.path`, which means we can now access files from the parent directory (namely, `neural_network.py`).

No.	Test	Input	Type	Expectation	Output
t_{36}	Loading the MNIST training data	–	–	Loads the MNIST training data and imports the Multilayer Perceptron class	As Expected

Table 3.43: Testing Table for Code Snippets 3.51 & 3.52

`network_training / train_digits.py`

```

16 # train the network
17 network = MultilayerPerceptron([784, 16, 16, 10])
18 network.train(training_data, 1000, 10, 1)
19 network.save_model('digits')
```

Code Snippet 3.53: Training a Neural Network on the MNIST Data Set

Now that we have successfully imported the `class MultilayerPerceptron`, we can attempt to train it. We must first instantiate a `MultilayerPerceptron`, with 784 neurons in the input layer and 10 neurons in the output layer. The number of neurons in the hidden layer(s), and the number of hidden layers have no requirements, so I have arbitrarily decided to make 2 hidden layers, each with 16 neurons.

We can then call the `train` method, passing in the training data. Again, I have arbitrarily decided that the mini batch size is 1000, the number of epochs is 10, and the learning rate is 1.

Once it is fully trained, we can save the weights and biases to a file, by calling the `save_model` method.

```

neural_network.py / class MultilayerPerceptron / def train
90     for num, mini_batch in enumerate(mini_batches):
91         print(f' mini batch {num}/{len(mini_batches)}...') # log
         ↵   message
```

Code Snippet 3.54: Adding another log message

Each epoch was taking a very long time, so I decided to add another log message, for each mini batch.

No.	Test	Input	Type	Expectation	Output
t_{37}	Training the neural network on MNIST	–	–	No errors when carrying out the full training algorithm	As Expected

Table 3.44: Testing Table for Code Snippets 3.53 & 3.54

network_training / test_digits.py

```

1 import pickle
2
3 # load the test data from the file
4 with open('data/digits.pkl', 'rb') as file:
5     data = pickle.load(file)
6 testing_data = data['testing']
7
8 # import the class MultilayerPerceptron from the neural_network.py file
9 import os
10 import sys
11 current = os.path.dirname(os.path.realpath(__file__))
12 parent = os.path.dirname(current)
13 sys.path.append(parent)
14 from neural_network import MultilayerPerceptron

```

Code Snippet 3.55: Loading the MNIST Testing Data

In order to test our trained neural network, we must first do the same thing as in the training file, which is loading the testing data from the file, and then importing the **class MultilayerPerceptron** from the parent directory.

No.	Test	Input	Type	Expectation	Output
t_{38}	Loading the MNIST testing data	–	–	Loads the MNIST testing data and imports the Multilayer Perceptron class	As Expected

Table 3.45: Testing Table for Code Snippet 3.55

network_training / test_digits.py

```

16 # iterate over each of the testing examples and see how many it gets
17     ↵ right
18 network = MultilayerPerceptron([784, 16, 16, 10])
19 network.load_model('models/digits.pkl')
20 success = 0
21 for total_so_far, example in enumerate(testing_data):
22     prediction = network.predict(example['input'])
23     list_of_probabilities = [prediction.item(x,0) for x in range(10)]
24     if list_of_probabilities.index(max(list_of_probabilities)) ==
25         ↵ example['output']:
26         success += 1
27 print(f'after {total_so_far+1} examples, the success rate is
28     ↵ {(success/(total_so_far+1))*100}%')

```

Code Snippet 3.56: Testing the Neural Network on the MNIST Data Set

To test it, again we must first instantiate a `MultilayerPerceptron`, with the same layer sizes, and then use the `load` method to set the weights and biases to that of the trained network.

Then, we must iterate over each of the pieces of testing data, and make a prediction. Currently, this prediction is a column vector with 10 entries, one of which should hopefully be much larger than the rest, which is the network's prediction. To get this prediction, we must find the index of the maximum value of the column vector. If it is the same as the example's output (what the number is supposed to be), then the network has successfully predicted the digit.

After each prediction, we can print the current success rate, to see how it evolves over time.

No.	Test	Input	Type	Expectation	Output
t_{39}	Testing the neural network on MNIST	—	—	Cumulative % Success printed after each test	As Expected

Table 3.46: Testing Table for Code Snippet 3.56

```
after 1 examples, the success rate is 100.0%
after 2 examples, the success rate is 100.0%
after 3 examples, the success rate is 100.0%
after 4 examples, the success rate is 100.0%
after 5 examples, the success rate is 80.0%
after 6 examples, the success rate is 83.3333333333334%
after 7 examples, the success rate is 71.42857142857143%
```

Figure 3.26: Output for the first 7 training examples

Figure 3.26 shows the output for the first 7 examples. This shows that it got the first 4 correct, then 1 wrong, then another one correct, and then another one wrong.

```
after 9994 examples, the success rate is 53.53211927156294%
after 9995 examples, the success rate is 53.536768384192094%
after 9996 examples, the success rate is 53.54141656662665%
after 9997 examples, the success rate is 53.546063819145736%
after 9998 examples, the success rate is 53.5507101420284%
after 9999 examples, the success rate is 53.54535453545355%
after 10000 examples, the success rate is 53.55%
```

Figure 3.27: Output for the last 7 training examples

Figure 3.27 shows the output for the last 7 examples. This shows that it has converged on a success rate of around 53.55%. This is a good result, but not great as it still got about half of them wrong.

`network_training / train_digits.py`

18 `network.train(training_data, 10, 30, 3)`

Code Snippet 3.57: Improving the Training Parameters

If we adjust the training parameters, to have smaller mini batches, more epochs and a greater learning rate, the success rate will increase. (To get a new model, we must of course re-run `train_digits.py` and `test_digits.py`)

```
after 9994 examples, the success rate is 93.52611566940165%
after 9995 examples, the success rate is 93.52676338169084%
after 9996 examples, the success rate is 93.52741096438577%
after 9997 examples, the success rate is 93.52805841752526%
after 9998 examples, the success rate is 93.52870574114823%
after 9999 examples, the success rate is 93.52935293529353%
after 10000 examples, the success rate is 93.53%
```

Figure 3.28: Improved Output for the last 7 training examples

Figure 3.28 shows the output for the last 7 examples, with new training parameters. We now converge on around 93.53%, which is much better.

If the training didn't work, then the success rate should be around 10%, as it would just be picking digits at random, and so it would have a $\frac{1}{10}$ chance of getting it right. 93.53% is much greater than 10%, and so I believe there is sufficient evidence to suggest that the neural network, and the backpropogation algorithm is in fact working.

No.	Test	Input	Type	Expectation	Output
T_5	Neural Network works	MNIST Hand-written Digits	—	A Success rate much larger than 10%	As Expected

Table 3.47: T_5 Testing Table (for Code Snippet 3.57)

3.2.6 Part 10 – Training the Neural Network

Now it is time to trian the neural network to predict the HDI of a given area.

`network_training / load_hdi_data.py`

```

1 import csv
2 import numpy as np
3
4 # read csv file
5 with open('data/training_data.csv', 'r') as file:
6     reader = csv.DictReader(file)
7     hdiData = []
8     for record in reader:
9         # convert it to the right format
10        hdiData.append({
11            "input": np.matrix([[1000 if factor == '' else float(factor)]
12                             for factor in list(record.values())[4:]]),
13            "output": np.matrix([[float(record['hdi'])]])
14        })

```

Code Snippet 3.58: Loading the HDI Data

First, we must convert the HDI data into the right format. We can read the .csv file using the same methods as before, except when we go to append each record, we must slightly change the format.

Currently, each record is a dictionary of all the factors, along with some extra information (such as the name of the region). The input data should be a list of factors (as a column

vector), which currently start at index 4. Each of these factors is currently of type `str`, so we must convert it to a `float`. Some of the factors are also currently an empty string, which correspond to average distance factors in regions that have 0 of that building. I have set these to 1000, as that is larger than every other average distance factor.

The output should just be the HDI, which is located at key '`'hdi'`'.

`network_training / load_hdi_data.py`

```
3 import random
4 import pickle
```

`network_training / load_hdi_data.py`

```
17 # split the data set into training and testing
18 def test_train_split(data_set, proportion_train):
19     random.shuffle(data_set)
20     return {
21         "training": data_set[:int((len(data_set)*proportion_train)//1)],
22         "testing": data_set[int((len(data_set)*proportion_train)//1):]
23     }
24
25 hdiData = test_train_split(hdiData, 0.9)
26 # save data to a file
27 with open(f'data/hdi.pkl', 'wb') as file:
28     pickle.dump(hdiData, file)
```

Code Snippet 3.59: Splitting into Testing and Training Data

We then must randomly split the data set into training and testing data. For this, I have written a function, `def test_train_split`, which will take in a data set and a proportion which should be training data.

If we multiply the length of the data set by the proportion, we get the index of the final piece of data which should be for training. Therefore, everything before that should be training and everything after it should be testing.

After we have split the data set, we can again save it to file as we have done before.

No.	Test	Input	Type	Expectation	Output
t_{40}	Converting HDI Data to the correct format	.csv	—	Numpy Matrices	As Expected

t_{41}	Test Train Split	Full Data Set	-	First proportion_」 train% are training, rest is testing	As Expected
----------	------------------	---------------	---	---	-------------

Table 3.48: Testing Table for Code Snippets 3.58 & 3.59

network_training / train_hdi.py

```

1 import pickle
2
3 # load the training data from the file
4 with open('data/hdi.pkl', 'rb') as file:
5     data = pickle.load(file)
6 training_data = data['training']
7
8 # import the class MultilayerPerceptron from the neural_network.py file
9 import os
10 import sys
11 current = os.path.dirname(os.path.realpath(__file__))
12 parent = os.path.dirname(current)
13 sys.path.append(parent)
14 from neural_network import MultilayerPerceptron
15
16 # train the network
17 network = MultilayerPerceptron([24, 10, 10, 1])
18 network.train(training_data, 10, 30, 3)
19 network.save_model('hdi-temp')

```

Code Snippet 3.60: Training the Neural Network on HDI

We can then train the neural network on the HDI training data in exactly the same way as we did for the MNIST handwritten digits.

Except, this time we must have 24 neurons in the input layer (for each of the 24 factors), and 1 neuron in the output layer. I initially decided to have 2 hidden layers, each with 10 neurons, and to keep the remaining training parameters the same. (mini batch size = 10, number of epochs = 30, learning rate = 3)

No.	Test	Input	Type	Expectation	Output
t_{42}	Training the neural network on HDI	–	–	No errors when carrying out the full training algorithm	As Expected

Table 3.49: Testing Table for Code Snippet 3.60

network_training / test_hdi.py

```

1 import pickle
2
3 # load the test data from the file
4 with open('data/hdi.pkl', 'rb') as file:
5     data = pickle.load(file)
6 testing_data = data['testing']
7
8 # import the class MultilayerPerceptron from the neural_network.py file
9 import os
10 import sys
11 current = os.path.dirname(os.path.realpath(__file__))
12 parent = os.path.dirname(current)
13 sys.path.append(parent)
14 from neural_network import MultilayerPerceptron
15
16 # iterate over each of the testing examples and see how many it gets
17 # right
17 network = MultilayerPerceptron([24, 10, 10, 1])
18 network.load_model('models/hdi-temp.pkl')

```

Code Snippet 3.61: Preparing to Test on the HDI

When we test the network, we should again load it in exactly the same way as with the MNIST Data Set, with the same changes to the hidden layers.

network_training / test_hdi.py

```

19 differences = []
20 for num, example in enumerate(testing_data):
21     correctAnswer = example["output"].item(0,0)
22     prediction = round(network.predict(example['input']).item(0,0), 3)
23     difference = round(abs(correctAnswer-prediction), 3)
24     print(f'({num}) correct output: {correctAnswer}, prediction:
25         {prediction}, difference: {difference}')
26     differences.append(difference)
27
28 import numpy as np
29 print(f'average difference: {np.mean(differences)}')

```

Code Snippet 3.62: Testing the Neural Network on HDI

As the output for this network is slightly different, we should test it in a different way. The correct HDI is given by the '`output`' of the testing example, and we can compare that with the prediction.

At each stage, we round our values to 3 decimal places, as this is what HDI is most commonly quoted to. At the end, we can take the mean to find an average difference.

No.	Test	Input	Type	Expectation	Output
t_{43}	Testing the neural network on HDI	—	—	Difference for each prediction calculated, as well as average difference	As Expected

Table 3.50: Testing Table for Code Snippets 3.61 & 3.62

(130) correct output: 0.777, prediction	0.789,	difference: 0.012
(131) correct output: 0.794, prediction	0.789,	difference: 0.005
(132) correct output: 0.763, prediction	0.789,	difference: 0.026
(133) correct output: 0.672, prediction	0.789,	difference: 0.117
(134) correct output: 0.836, prediction	0.789,	difference: 0.047
(135) correct output: 0.565, prediction	0.804,	difference: 0.239
(136) correct output: 0.929, prediction	0.597,	difference: 0.332
(137) correct output: 0.683, prediction	0.789,	difference: 0.106
(138) correct output: 0.474, prediction	0.761,	difference: 0.287
(139) correct output: 0.738, prediction	0.789,	difference: 0.051
(140) correct output: 0.686, prediction	0.789,	difference: 0.103
(141) correct output: 0.375, prediction	0.544,	difference: 0.169
(142) correct output: 0.904, prediction	0.789,	difference: 0.115
(143) correct output: 0.471, prediction	0.675,	difference: 0.204
(144) correct output: 0.739, prediction	0.789,	difference: 0.05
(145) correct output: 0.888, prediction	0.789,	difference: 0.099
(146) correct output: 0.908, prediction	0.789,	difference: 0.119
(147) correct output: 0.635, prediction	0.789,	difference: 0.154
(148) correct output: 0.445, prediction	0.789,	difference: 0.344
(149) correct output: 0.696, prediction	0.56,	difference: 0.136
(150) correct output: 0.91, prediction:	0.789,	difference: 0.121
(151) correct output: 0.564, prediction	0.789.	difference: 0.225

Figure 3.29: Some HDI Predictions

Figure 3.29 shows some of the network's predictions. As you can see, most of them are the same, with a prediction of 0.789, which is obviously not ideal.

The first thing I thought of to amend this is to change the scale of the factors.

network_training / load_hdi_data.py

```

11     # convert it to the right format
12     hdiData.append({
13         "input": np.matrix([[100 if factor == '' else float(factor)/10
14             ↪ if index <= 11 else float(factor)] for index, factor in
15             ↪ enumerate(list(record.values())[4:])]),
16         "output": np.matrix([[float(record['hdi'])]])
17     })

```

Code Snippet 3.63: Dividing some factors by 10

Each of the average distance factors often take values in between 1 and 10, but this particular type of neural network often works best with numbers between 0 and 1. Therefore, I have divided each of the average distance factors (at indexes 0 to 11) by 10, to get most of them in this range.

No.	Test	Input	Type	Expectation	Output
t_{44}	Scaling Average Distance Factors	Average Distance Factor	—	Divided by 10	As Expected

Table 3.51: Testing Table for Code Snippet 3.63

The network was still not predicting very well, so I had to adjust the training parameters.

network_training / train_hdi.py

```

16 # train the network
17 network = MultilayerPerceptron([24, 18, 12, 6, 3, 1])
18 network.train(training_data, 5, 50, 0.2)
19 network.save_model('hdi-temp')
```

network_training / test_hdi.py

```

16 # iterate over each of the testing examples and see how many it gets
  ↵ right
17 network = MultilayerPerceptron([24, 18, 12, 6, 3, 1])
18 network.load_model('models/hdi-temp.pkl')
```

Code Snippet 3.64: Adjusting the Training Parameters

After many attempts, I ended up with a neural network with 4 hidden layers, each with 18, 12, 6 and 3 neurons respectively, which was trained with a mini batch size of 5, over 50 epochs and a learning rate of 0.2.

No.	Test	Input	Type	Expectation	Output
t_{42}	Training the neural network on HDI	—	—	No errors when carrying out the full training algorithm	As Expected

Table 3.52: Testing Table for Code Snippet 3.64

```
(159) correct output: 0.707, prediction: 0.694, difference: 0.013
(160) correct output: 0.534, prediction: 0.539, difference: 0.005
(161) correct output: 0.496, prediction: 0.601, difference: 0.105
(162) correct output: 0.543, prediction: 0.675, difference: 0.132
(163) correct output: 0.856, prediction: 0.812, difference: 0.044
(164) correct output: 0.858, prediction: 0.697, difference: 0.161
(165) correct output: 0.605, prediction: 0.595, difference: 0.01
(166) correct output: 0.577, prediction: 0.669, difference: 0.092
(167) correct output: 0.494, prediction: 0.669, difference: 0.175
(168) correct output: 0.762, prediction: 0.695, difference: 0.067
(169) correct output: 0.733, prediction: 0.695, difference: 0.038
(170) correct output: 0.491, prediction: 0.565, difference: 0.074
(171) correct output: 0.769, prediction: 0.69, difference: 0.079
(172) correct output: 0.8, prediction: 0.829, difference: 0.029
(173) correct output: 0.811, prediction: 0.726, difference: 0.085
(174) correct output: 0.593, prediction: 0.669, difference: 0.076
(175) correct output: 0.864, prediction: 0.81, difference: 0.054
(176) correct output: 0.933, prediction: 0.821, difference: 0.112
(177) correct output: 0.793, prediction: 0.767, difference: 0.026
average difference: 0.08447191011235955
```

Figure 3.30: Improved HDI Predictions

The variance in these predictions was very high, and the average difference is very low, which is what we need.

3.2.7 Review

No.	Test	Inputs	Pass / Fail
T_5	Neural Network works	Using a <i>different</i> data set, e.g. handwritten digits, which certainly has enough training data	Pass

Table 3.53: Passed Testing Data for Milestone 2

The fact that we successfully predicted 93.53% of MNIST handwritten digits suggests that T_5 has been passed, which implies the Success Criteria S_5 and S_6 have been met

No.	Success Criterion	Justification	Success
S_5	Successfully Implement the Feedforward Algorithm	This algorithm will allow us to make predictions on the HDI given a set of factors.	✓

S_6	Successfully Implement the Backpropagation Algorithm	This algorithm will allow us to train the neural network base on a set of training examples.	✓
S_7	Successfully train the Neural Network	In order to accurately predict the HDI of a given area, the neural network must train on existing examples to see what makes a high HDI.	✓

Table 3.54: Achieved Success Criteria for Milestone 2

I believe S_7 has also been met, as we have been able to train a neural network which can (vaguely) predict the HDI of a given area.

3.3 Milestone 3 – Initial User Interface

3.3.1 Success Criteria

By the end of this milestone, I hope to have a fully functioning user interface, in which the user can select their region on a map, predict the HDI of it, and have suggestions made to improve it.

No.	Success Criterion	Justification
S_8	User can select an area on the map for which they want to analyse	This will allow the user to choose which area they want to predict the HDI of by drawing on the map.
S_9	Neural Network accurately predicts HDI from those factors	Once the user has selected their area, the HDI must be predicted based on factors calculated from that area.
S_{10}	Accurately calculate where a new building would need to be in order to increase the HDI the most	The specific placement of new buildings (as suggestions to increase HDI) is important to consider, so a suitable place must be chosen in order to decrease the average distance to it as much as possible.
S_{11}	Changes are suggested to increase the HDI	This will allow the user to make an informed decision on what to do in order to improve their area.

Table 3.55: Success Criteria for Milestone 3

To determine if I have met this success criteria, I will be using the following tests:

No.	Test	Inputs	Expected Output/Justification
T_6	Overall Flow	User Draws Region and then Predicts, User tries to predict before selecting Region, User draws region and doesn't predict	The only valid sequence of steps the user should be able to take is drawing the region and then making a prediction.
T_7	Optimal Place for New Building	Building for which there exists 1 Factor, Building for which there exists more than 1 Factor, Building for which there exists 0 factors	This function should take an average over all of the factors the building is involved in, while throwing an error if it has 0 factors.
T_8	Suggestions fed back into Neural Network	All 8 suggestions for where to place new buildings	A new predicted HDI for each suggestion

Table 3.56: Testing Data for Milestone 3

3.3.2 Part 11 – Drawing on the Map

I believe that having a map to draw on is the most important thing in the GUI, so I will be implemented that first.

app.py

```

1 import gradio as gr
2
3 # structure of the UI
4 with gr.Blocks() as app:
5     pass
6
7 # launch the UI
8 app.launch()
```

Code Snippet 3.65: Initial Gradio Framework

Here, I am using the `gradio` library to begin to define the interface. Gradio uses context managers (`with` statement) to make the structure of the GUI very readable and easy to understand (this will become more apparent when we add more and more elements).

Gradio offers many types of interface, but the one I will be using is the `Blocks` interface, which allows for the most customisation. For now, we have no structure to add, so we can just `pass`.

Once we have defined the GUI, we can call the `launch` method on the app.

```
Running on local URL: http://127.0.0.1:7860
To create a public link, set `share=True` in `launch()`.
```

Figure 3.31: Launching the Gradio App

Running this code produces the output shown in Figure 3.31. When we called the `launch` method, it initialised the gradio interface on port 7860, of the localhost IP address (127.0.0.1). It also tells us that we could create a public link, which I will be doing later to share with my stakeholders.

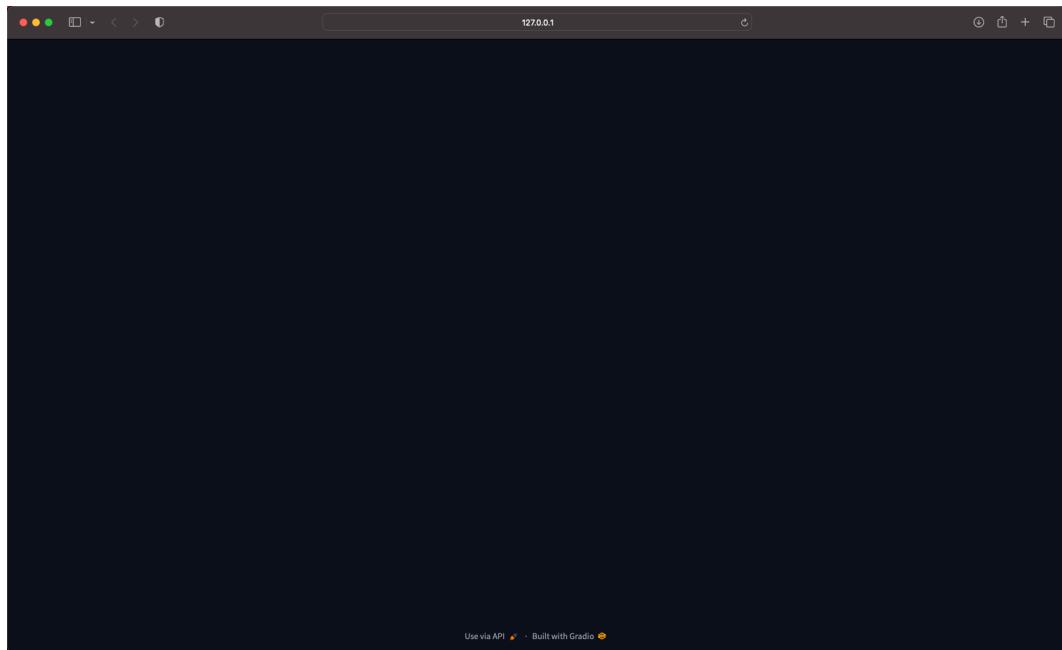


Figure 3.32: Empty Gradio UI

Following the link `http://127.0.0.1:7860` takes us to this empty gradio interface. Once we add components, they will show up here.

No.	Test	Input	Type	Expectation	Output
t_{45}	Initialise Gradio Interface	–	–	Empty Gradio Interface	As Expected

Table 3.57: Testing Table for Code Snippet 3.65

app.py

```

2 import folium
3 from folium.plugins import Draw
4
5 # initialise the map for drawing on
6 foliumMap = folium.Map()
7 draw = Draw(
8     export=False,
9     position='topleft',
10    draw_options={
11        'polyline': False,
12        'circlemarker': False,
13        'polygon': {'allowIntersection': False},
14        'marker': False,
15        'circle': False,
16        'rectangle': False,
17    },
18    edit_options={'poly': {'allowIntersection': False}}
19)
20 draw.add_to(foliumMap)

```

Code Snippet 3.66: Initialising the Map

Next, we must also import the `folium` library, which lets us create interactive maps.

First, we can make an instance of the `class Map`, and save that to a variable. To be able to draw on it, we must also make an instance of the `class Draw`, and then add that to the map at the end.

The `class Draw` has many attributes which must be set. Namely, specifying `position` to be `'topleft'` will make the drawing tools appear on the top left. Setting most of the `draw_options` to `False` except for `'polygon'` (which in turn has `'allowIntersection'` set to `False`) ensures that the user can only draw polygons, and the polygons should not be able to self intersect.

app.py

```

2 from gradio_folium import Foliump

```

app.py

```

23 # structure of the UI
24 with gr.Blocks() as app:
25     map = Foliump(value=foliumMap)

```

Code Snippet 3.67: Adding the Map to the GUI

To add this to the gradio interface, we must first import the `Folium` element from the `gradio_folium` library, and then add this to our `Blocks` interface. If we save this to a variable, `map`, we will be able to access and update it. We also have zoom in and out buttons in the top left as well.

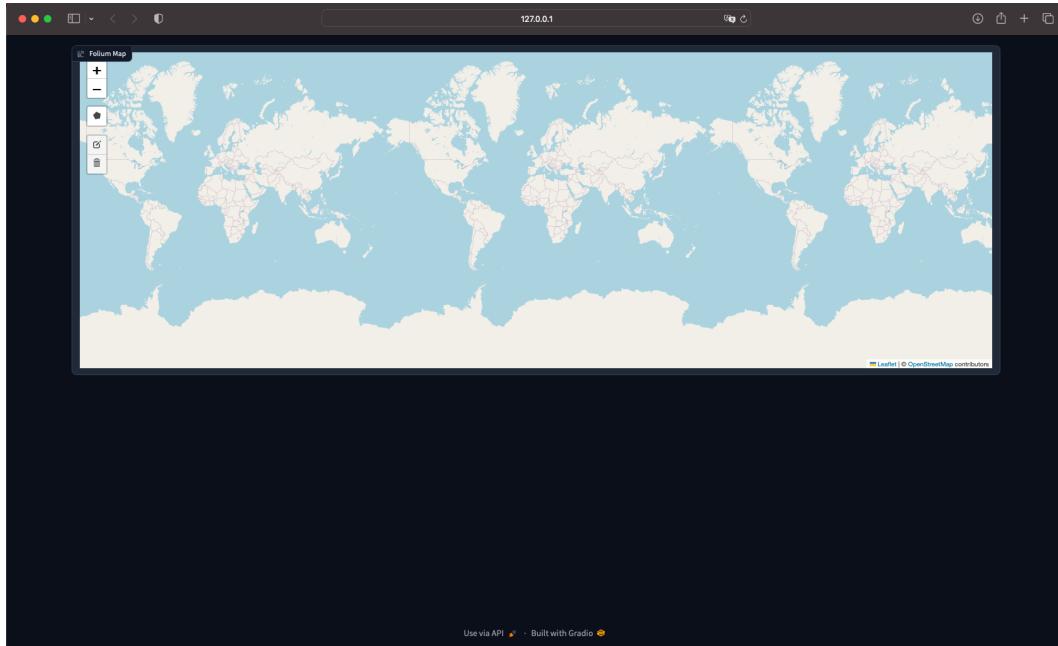


Figure 3.33: Map Added to GUI

Figure 3.33 shows the current output for the program, with the drawing tools shown in the top left of the map.

Users can also use the scroll wheel to zoom in and out of the map if they wish. By default, the mouse will drag around the map, but if the user clicks the pentagon icon, using the mouse will then begin to draw a polygon.

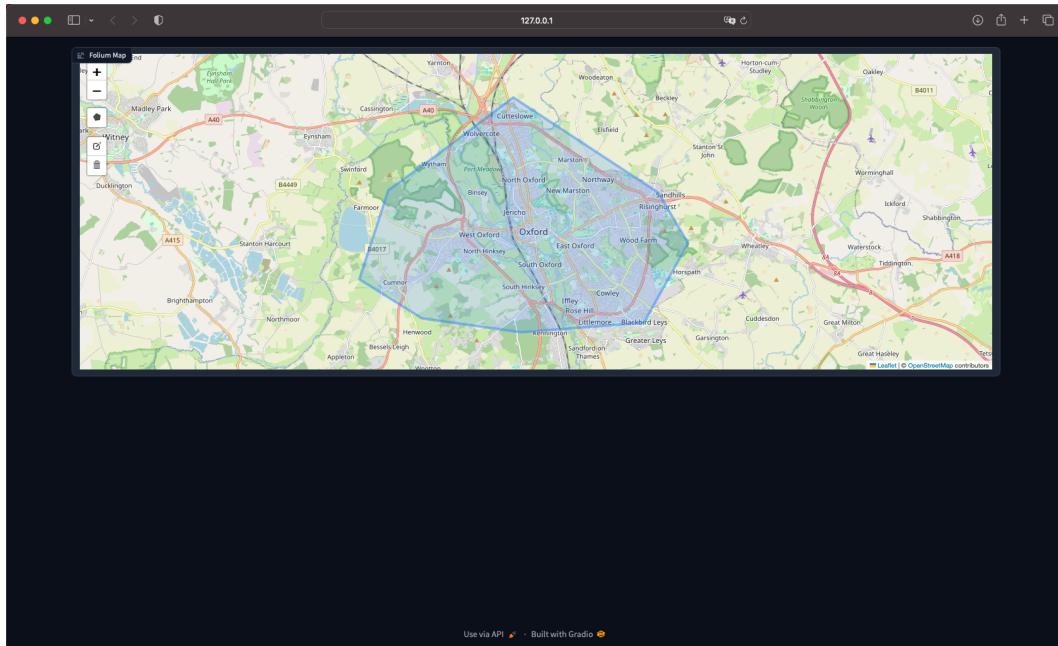


Figure 3.34: Drawing a Region

No.	Test	Input	Type	Expectation	Output
t_{46}	Map added to Interface	—	—	Gradio Interface with Map + Controls	As Expected
t_{47}	Drawing on Map	—	—	Can only draw polygons	As Expected

Table 3.58: Testing Table for Code Snippets 3.66 & 3.67

Instead of the map starting very zoomed out, I would like for the map to start at a random city.

app.py

```

5 import random
6
7 # define some starting locations
8 locations = [[51.5360, -0.1196], [40.8093, -73.9678], [34.0069,
   ↵ -118.2304], [25.7617, -80.1918], [48.8488, 2.3470], [41.8840,
   ↵ 12.4790], [52.5078, 13.4003], [37.9964, 23.7293], [59.3265, 18.0680],
   ↵ [40.4172, -3.6867], [41.3940, 2.1606], [38.7253, -9.1403], [55.7476,
   ↵ 37.6209], [31.2230, 121.4860], [39.8958, 116.4000], [37.5467,
   ↵ 126.9901], [35.6777, 139.7685], [1.3294, 103.8081], [-33.8782,
   ↵ 151.1754], [30.0465, 31.2246], [6.5047, 3.3732], [-1.2873, 36.8198],
   ↵ [-33.9425, 18.5065], [19.3916, -99.1279], [-23.5727, -46.6207],
   ↵ [36.1458, -115.1832], [38.8939, -77.0372], [49.2501, -123.1164],
   ↵ [43.6888, -79.4190]]
9
10 # initialise the map for drawing on
11 foliumMap = folium.Map(location=random.choice(locations))

```

Code Snippet 3.68: Setting the Initial Location to a Random City

Here, the `list` `locations` specifies the latitude/longitude coordinates of many famous cities across the world. To pick a random one, we must first import the `random` library and get a random choice from the `locations`. If we set the attribute `location` of the `Map` class to this, then the map will initialise at that location.

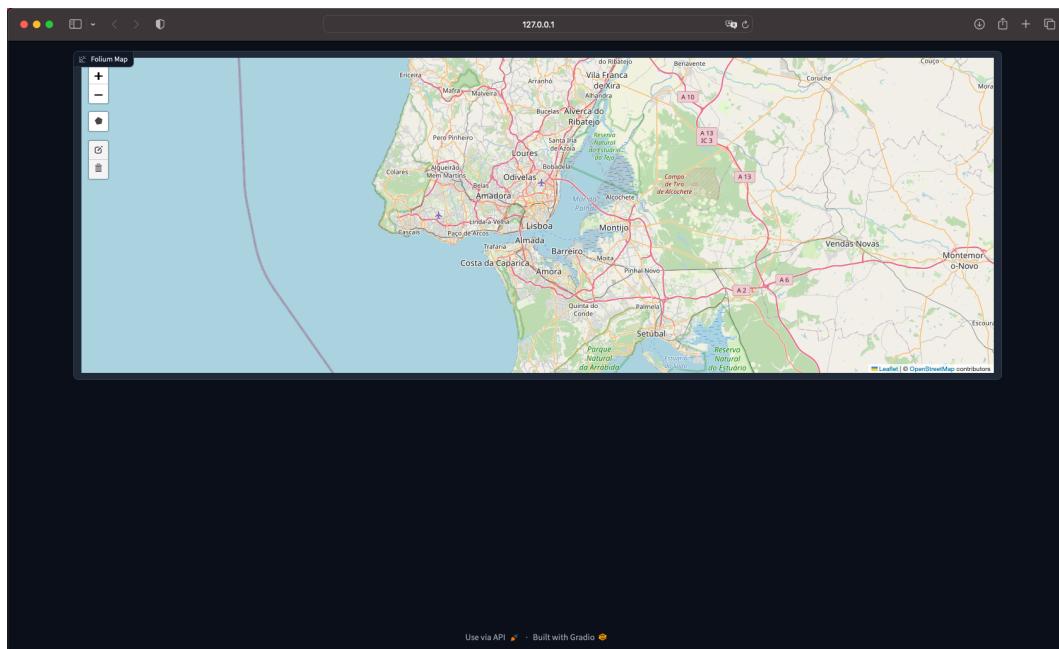


Figure 3.35: Map Starting at a Random City

No.	Test	Input	Type	Expectation	Output
t_{48}	Starting at a Random Location	-	-	Map Initialises at a Random City	As Expected

Table 3.59: Testing Table for Code Snippet 3.68

Now I need to retrieve the region the user draws on the map, in order to process it and make predictions. I did not know how to do this, so I began to research the problem.

After looking through many forums and StackOverflow pages, it seems that accessing the region drawn in the folium map dynamically is impossible.

app.py

```
13     export=True,
14     filename='region.geojson',
```

app.py

```
28 # structure of the UI
29 with gr.Blocks() as app:
30     map = Folium(value=foliumMap)
31     uploadButton = gr.UploadButton(label='Upload the GeoJSON file',
→     file_count='single')
```

Code Snippet 3.69: Reading the Map for the Region

Instead, I have decided to set the `export` attribute (in `Draw`) to `True`, meaning that there will now be an export button on the top right of the map. This will download a `.geojson` file to the user's computer containing the region.

The user can then upload their `.geojson` file to be read and processed.

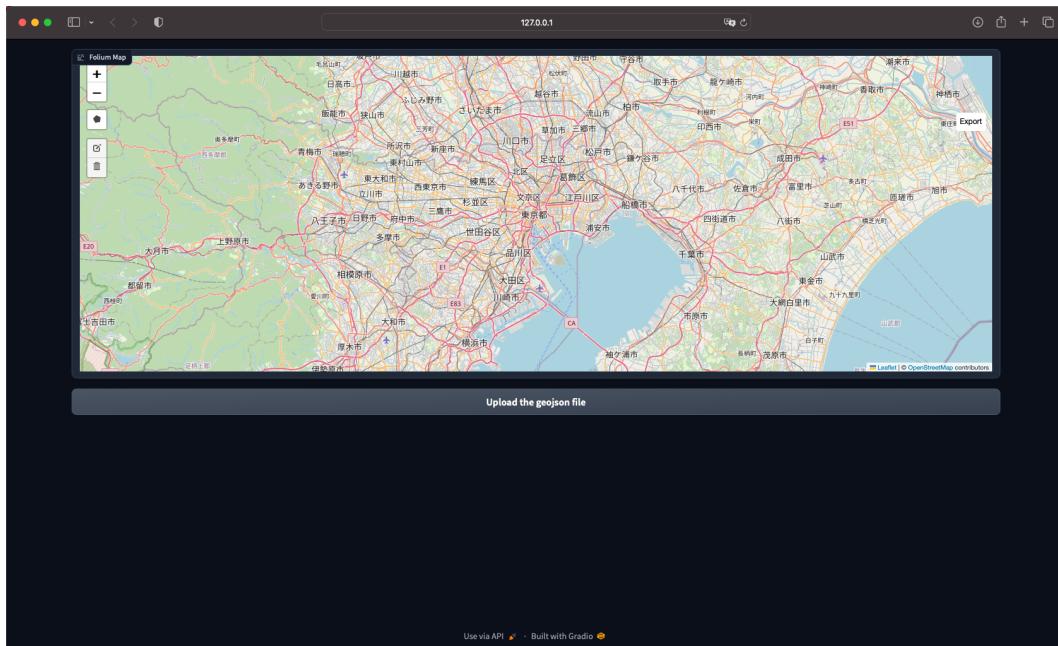


Figure 3.36: Upload Button Added

Figure 3.36 shows the output with these new changes. This is not an ideal solution, as the way the user uses the main function of the app has become a bit cumbersome. However, I cannot see a way around this, for now at least.

No.	Test	Input	Type	Expectation	Output
t_{49}	Export GeoJSON	—	—	User can export GeoJSON from map	As Expected

Table 3.60: Testing Table for Code Snippet 3.69

The button to export the `.geojson` file does not work and will be implemented in the next part.

3.3.3 Part 12 – Predicting the HDI

`app.py`

```
32     log = gr.Textbox(label='Log Messages', value='', interactive=False)
```

Code Snippet 3.70: Adding a Log Textbox

I first wanted to add a Log Textbox to the GUI, which will show messages when executing operations successfully. For this, we can use the gradio `Textbox`, with the `interactive` argument set to `False`. This will make it so that the user cannot write in the text box.

This was added below the other elements, and should appear below them on the screen.

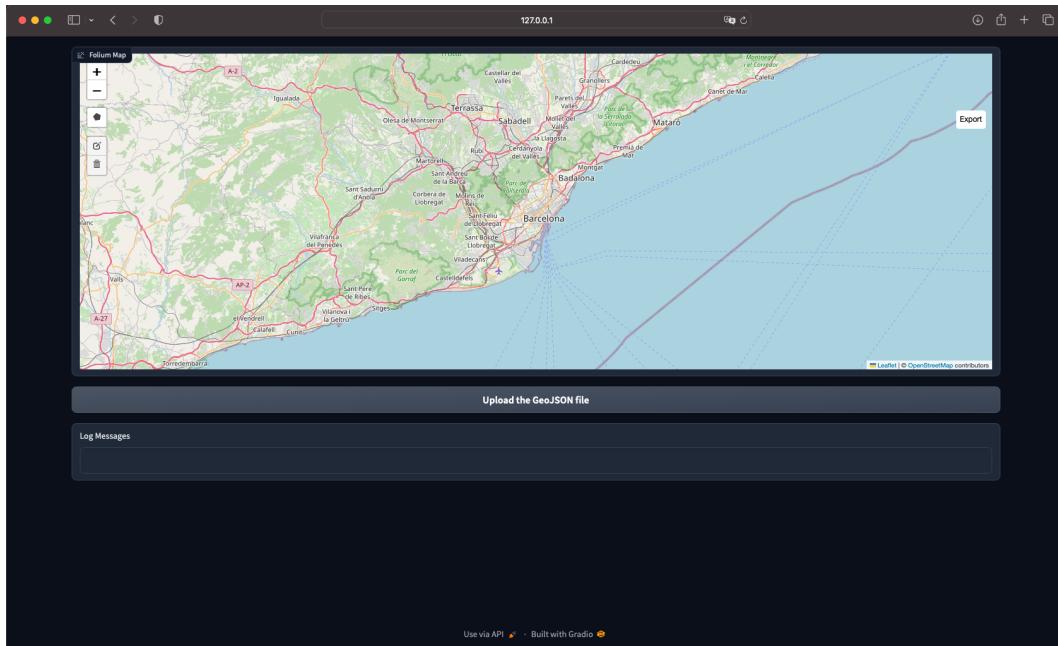


Figure 3.37: Log Textbox Added

No.	Test	Input	Type	Expectation	Output
t_{50}	Can't Write in Log Textbox	—	—	Log Textbox is read only	As Expected

Table 3.61: Testing Table for Code Snippet 3.70

Now we must add functionality to the upload button.

app.py

```

6 import json
7
8 # define global vars
9 currentRegion = []
10
11 # code to process the user uploading their geojson file
12 def uploadGeoJSON(filename):
13     global currentRegion
14     with open(filename, 'r') as file:
15         data = json.load(file)
16         currentRegion = [coordinate[::-1] for coordinate in
17                         ↳ data['features'][0]['geometry']['coordinates'][0]]
18     return f'Successfully Uploaded {filename[filename.rindex("//")+1:]}'
```

app.py

```

46 # functionality
47 uploadButton.upload(uploadGeoJSON, inputs=[uploadButton],
    ↳ outputs=[log])
```

Code Snippet 3.71: Adding Functionality to the Upload Button

All of the functionalities in the gradio interface will be of the format: `component.action(function, inputs=[...], outputs=[...])`, where the inputs and outputs are lists of components, and still contained within the `with gr.Blocks()` statement. Therefore, the code to make the upload button interactive is that shown on line 47, where `uploadGeoJSON` is the identifier of some function.

To store the current region, we can use a `global` variable, `currentRegion`. Recalling how `.geojson` files are structured, to get the coordinates into the right format, we must reverse each coordinate before storing it.

Once we have read the contents of the uploaded file, we can return a string to be displayed in the log textbox. The filename given by the parameter will be its full path, so we only want to take a substring of that starting with the character after the last `'/'`.

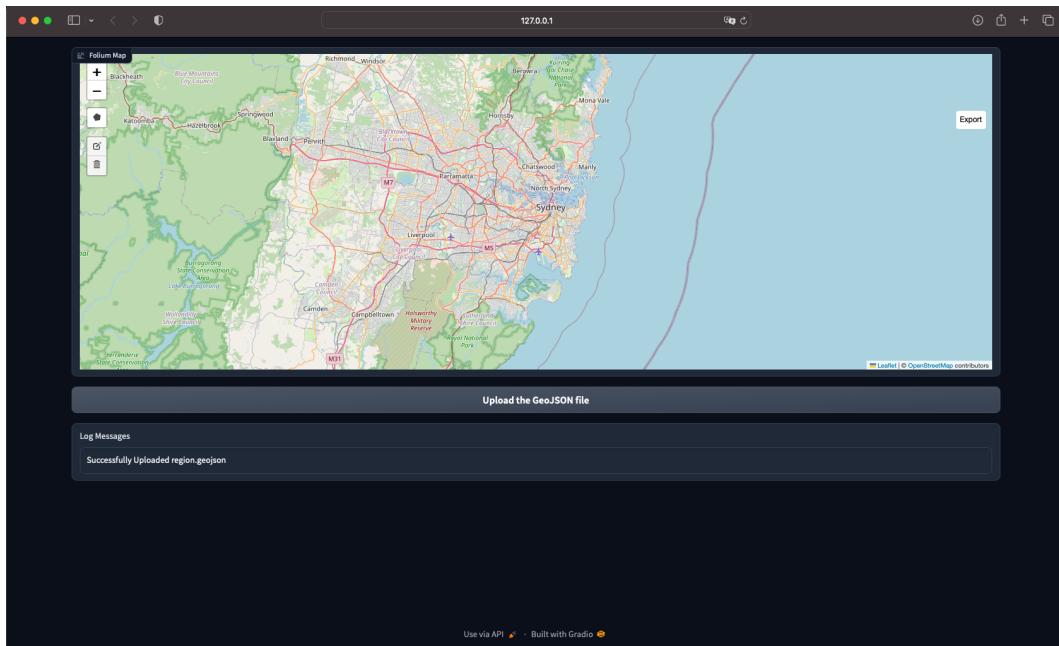


Figure 3.38: Uploading GeoJSON Files

No.	Test	Input	Type	Expectation	Output
t_{51}	Upload GeoJSON Files	region.geojson	—	Coordinates saved to <code>currentRegion</code> , and appropriate message shown	As Expected

Table 3.62: Testing Table for Code Snippet 3.71

app.py

```

40 # structure of the UI
41 with gr.Blocks() as app:
42     with gr.Row():
43         with gr.Column(scale=3):
44             map = Folium(value=foliumMap)
45             uploadButton = gr.UploadButton(label='Upload the GeoJSON
46             ↵ file', file_count='single')
47             with gr.Column(scale=1):
48                 predictHDIbutton = gr.Button(value='Predict HDI',
49                 ↵ variant='primary')
50                 HDIprediction = gr.Textbox(label='I believe that the HDI of
51                 ↵ this region is...', value='', interactive=False)
52             log = gr.Textbox(label='Log Messages', value='', interactive=False)

```

Code Snippet 3.72: Placing the Predict Button

In order to place the prediction button as shown by the Interface Sketch (Figure 2.30), we must make use of `gradio Rows` and `Columns`.

As we want our map and upload button to be arranged vertically, we can place them in a `Column`. The same goes for the button to predict HDI and the output textbox for the HDI prediction. As we want these two columns to be arranged side by side, we can put them in a `Row`. As the log textbox should be below everything, it should be placed outside of the `Row`. In my opinion, this use of context managers (`with gr.Row()` and `with gr.Column()`) makes the code extremely readable, and easy to understand where elements will end up.

If we want the width of the first column (containing the map) to be larger than the width of the second, we can use the `scale` attribute, to specify the ratio of the widths. Here, I have specified that the width of the map column should be 3 times the width of the prediction column.

For the actual button, setting the `variant` attribute to '`primary`' makes the button appear in the primary colour of the app, which is orange.

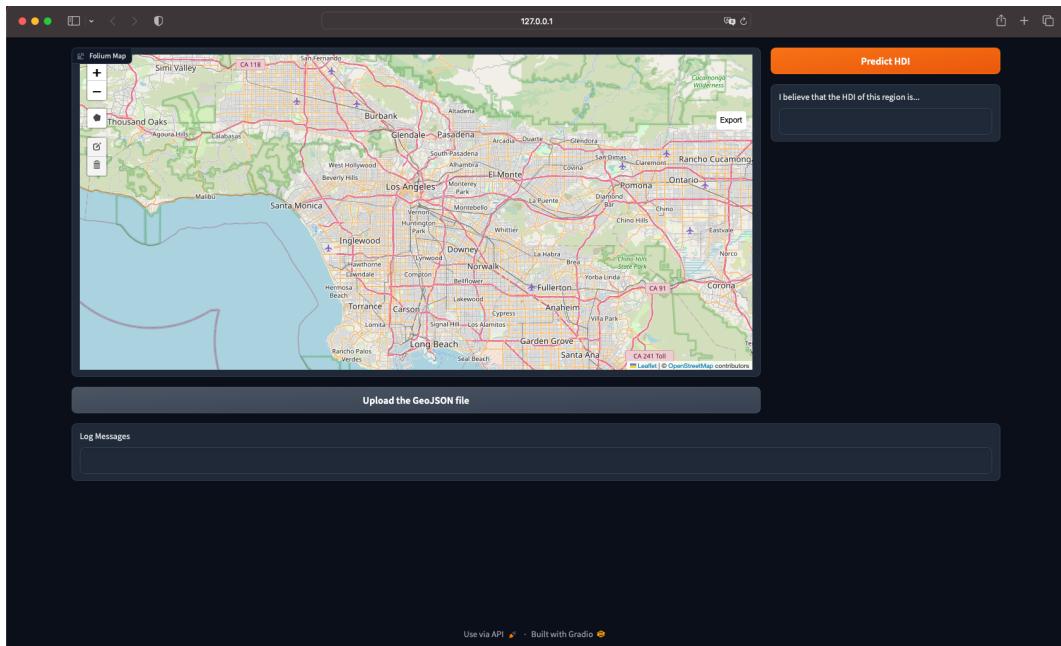


Figure 3.39: New Structure of the GUI

No.	Test	Input	Type	Expectation	Output
t_{52}	Column Structure of GUI	—	—	Width of map is 3 times the width of the prediction button	As Expected

Table 3.63: Testing Table for Code Snippet 3.72

Now it is time to implement the functionality of the predict HDI button.

app.py

```

19 # predict the HDI of the given region
20 def processPrediction(progress=gr.Progress()):
21     global currentRegion
22     if currentRegion == []:
23         return 'You have not uploaded a region!'
24     progress(0, desc='Starting...')
```

Code Snippet 3.73: Processing the Prediction

Again, we must define a function, to pass into the gradio interface, which will carry out the required processes.

We must first determine if the user had in fact uploaded a region. If they haven't, it will still be an empty `list`, and so we should return an error message before any calculations are carried out.

I imagine this function will take a lot of time to run, and so I would like for it to show a progress bar to the user. We can do this by using a `gr.Progress()` bar. This allows us to set the progress bar to different percentages at different times within the function (as shown on line 24).

The rest of this function will be very similar to stuff we have written before, mainly `def getAllFactors`, with some small changes.

`app.py`

```
7   from data_processing.get_osm_data import get_osm_data
```

`app.py`

```
11  all_objects = {}
```

`app.py / def processPrediction`

```
23  global all_objects
```

`app.py / def processPrediction`

```
27  # retrieve all relevant osm data
28  object_types = ['house', 'school', 'hospital', 'pharmacy',
29    ↳ 'restaurant', 'place_of_worship', 'bank', 'slot_machines',
30    ↳ 'fast_food', 'toilets', 'police', 'university', 'library',
31    ↳ 'post_box', 'vending_machine', 'bench', 'tree']
32  all_objects = {}
33  for num, object_type in enumerate(object_types):
34      progress(0.3*(num/len(object_types)), desc=f'Downloading
35        ↳ {object_type} data...')
36      print(f'getting {object_type} data...') # log message
37      all_objects[object_type] = get_osm_data(object_type,
38        ↳ currentRegion)
```

Code Snippet 3.74: Retrieving All Relevant OSM Data

The first thing we must do is download all of the different objects from OpenStreetMaps, by calling the `def get_osm_data` function written in Section 3.1.3. We must also initialise a global dictionary to contain all of these objects, as we will need to use them later, to improve the HDI.

Again, we can use the `progress` function to set the length of the progress bar according to how far through the list of object types to download we are.

No.	Test	Input	Type	Expectation	Output
t_{12}	Collecting all objects	–	–	Each list saved in a <code>dict</code>	As Expected

Table 3.64: Testing Table for Code Snippets 3.73 & 3.74

app.py

```
8 from calc_factors import averageDistance
```

app.py

```
13 averageDistanceFactors = [['house', 'school'], ['house', 'hospital'],
    ← ['house', 'pharmacy'], ['house', 'restaurant'], ['school',
    ← 'hospital'], ['police', 'hospital'], ['house', 'place_of_worship'],
    ← ['bank', 'slot_machines'], ['fast_food', 'toilets'], ['house',
    ← 'police'], ['university', 'library'], ['house', 'library']]
```

app.py / def processPrediction

```
36     # calculate average distances
37     allFactors = []
38     for num, averageDistanceFactor in enumerate(averageDistanceFactors):
39         progress(0.3+0.6*(num/len(averageDistanceFactors)),
40                 desc=f'Calculating Average Distance between
41                 {averageDistanceFactor[0]} and {averageDistanceFactor[1]}... ')
42         print(f'finding A({averageDistanceFactor[0]},\n
43               {averageDistanceFactor[1]})...') # log message
44         allFactors.append(averageDistance(
45             ← all_objects[averageDistanceFactor[0]],
46             ← all_objects[averageDistanceFactor[1]]))
```

Code Snippet 3.75: Calculating $A(x, y)$ Factors

Next, we must calculate each of the average distance factors. I have defined a `list` of the pairs of each object type, and so they can just be fed into the function to calculate the average distance. We can again use the `progress()` function to update the progress bar.

At this point, I decided to make a copy of the `calc_factors.py` file, and placed it in the main directory, instead of the `data_processing` directory. This is because I plan to make small edits to how some of these functions are going to work.

app.py

```
8  from calc_factors import averageDistance, calcArea, density
```

app.py

```
14 densityFactors = ['school', 'hospital', 'pharmacy', 'police', 'library',
    ↵ 'toilets', 'restaurant', 'place_of_worship', 'post_box',
    ↵ 'vending_machine', 'bench', 'tree']
```

app.py / def processPrediction

```
43     # calculate area
44     progress(0.9, desc='Calculating Area...')
45     print('calculating area...') # log message
46     area = calcArea(currentRegion)
47     # calculate density factors
48     for num, densityFactor in enumerate(densityFactors):
49         progress(0.91+0.05*(num/len(densityFactors)), desc=f'Calculating
    ↵ {densityFactor} Density...')
50         print(f'finding D({densityFactor})') # log message
51         allFactors.append(density(all_objects[densityFactor], area))
```

Code Snippet 3.76: Calculating $D(x)$ Factors

We can also calculate the density factors in a similar way, by defining a **list** of all the factors, and iterating over them. We must first however calculate the area of the region, to pass into the **def density** function.

No.	Test	Input	Type	Expectation	Output
t_{13}	Each factor is calculated	-	-	We have a dict of all the factors of a given region	As Expected

Table 3.65: Testing Table for Code Snippets 3.75 & 3.76

Now that we have a full list of factors, we can predict the HDI

app.py

```

9 import numpy as np
10 from neural_network import MultilayerPerceptron
11
12 # initialise neural network
13 network = MultilayerPerceptron([24, 18, 12, 6, 3, 1])
14 network.load_model('models/hdi.pkl')

```

app.py / def processPrediction

```

58     # predict HDI
59     progress(0.96, desc='Predicting HDI...')
60     inputLayer = np.matrix([[100 if factor == None else float(factor)/10
61         ↳ if index <= 11 else float(factor)] for index, factor in
62         ↳ enumerate(allFactors)])
63     prediction = round(network.predict(inputLayer).item(0,0), 3)
64     return prediction

```

Code Snippet 3.77: Predicting the HDI

For this, we must first import the `MultilayerPerceptron` from `neural_network.py`, and set the layer sizes to the right values. We can then use the `load_model` method to load the model we trained in Section 3.2.6.

Then, at the end of the `def processPrediction` function, we can get a matrix for the input layer (dividing the average distance factors by 10 in the same way as done before), and then we can make a prediction, to be returned.

app.py / def processPrediction

```

99     predictHDIbutton.click(processPrediction, inputs=[], 
    ↳ outputs=[HDIprediction])

```

Code Snippet 3.78: Adding Functionality to the Prediction Button

To make the button actually work, we must add a gradio functionality. This line of code makes it so that when the `predictHDIbutton` is `clicked`, it calls the `processPrediction` function, passing in no inputs (as the region is already a global variable), and outputting to the `HDIprediction` textbox.

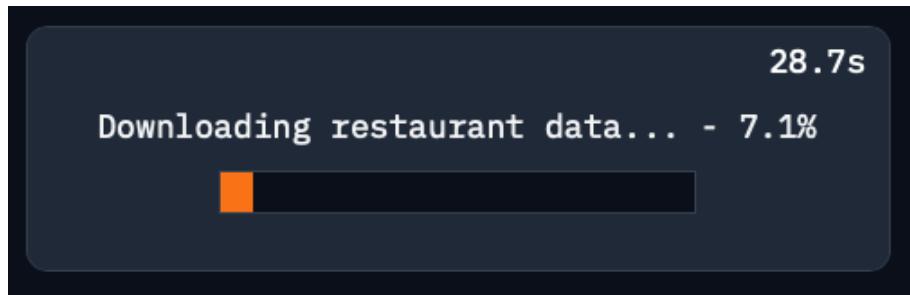


Figure 3.40: Progress Bar

Figure 3.40 shows the progress bar as the prediction is taking place. Gradio, by default, also provides a timer for these processes (as shown in the top right).

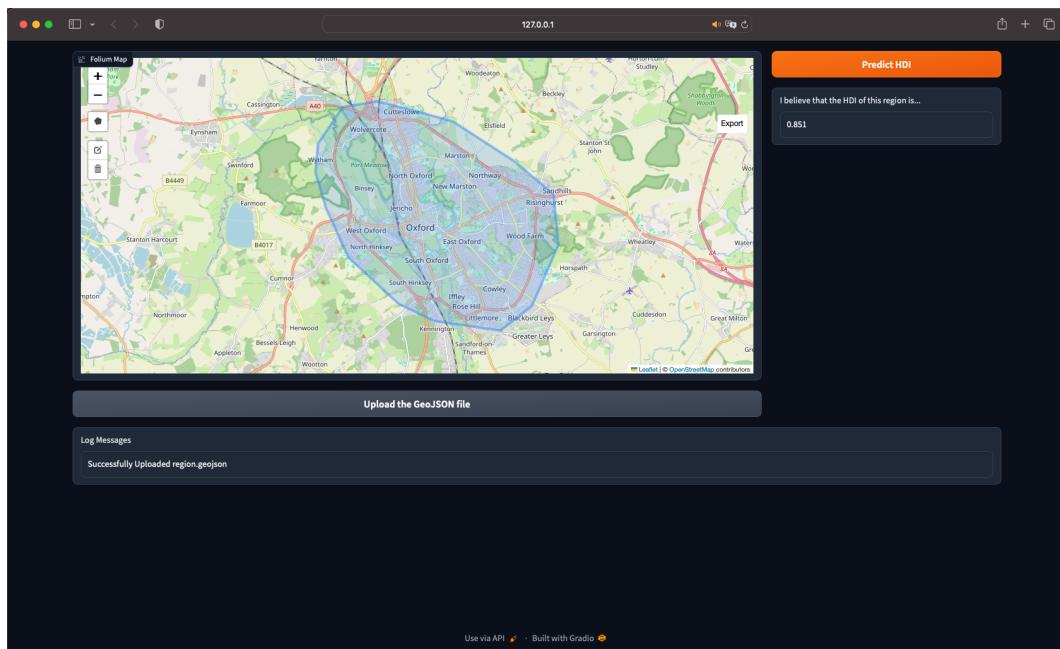


Figure 3.41: Successful Prediction from a Region

Figure 3.41 shows the prediction for Oxford, which is pretty close.

No.	Test	Input	Type	Expectation	Output
$T_{6.1}$	Overall Flow	User Draws Region and then Predicts	Valid	Factors are calculated, HDI is predicted and displayed	As Expected
$T_{6.2}$	Overall Flow	User attempts to predict before uploading a region	Invalid	Error is thrown in the prediction output	As Expected

$T_{6.3}$	Overall Flow	User Draws Region and doesn't predict	Valid	Region is saved, Factors are never calculated and HDI is never predicted	As Expected
-----------	--------------	---------------------------------------	-------	--	-------------

Table 3.66: T_6 Testing Table (for Code Snippets 3.77 & 3.78)

Unfortunately however, this took about 9 minutes to fully complete, which is too much time to wait. By looking at the progress bar, I found that the majority of the time taken was spent in calculating the average distance factors. Therefore, I plan to let the user change the `max_x_objects` variable with a slider, along with a new `max_y_objects` variable, which must first be implemented.

`calc_factors.py`

```
18 # finds the average distance between 2 types of objects, A(x,y)
19 def averageDistance(x_objects, y_objects, max_x_objects=2000,
→ max_y_objects=2000):
```

`calc_factors.py / def averageDistance`

```
58     if len(y_objects) > max_y_objects:
59         y_objects_in_consideration = random.sample(y_objects,
→ max_y_objects)
60     else:
61         y_objects_in_consideration = y_objects
62     for y_object in y_objects_in_consideration:
63         dist_to_ys.append(distBetween2Points(x_object, y_object))
```

Code Snippet 3.79: Implementing a Maximum for y -objects

In the `def averageDistance` function (in the new `calc_factors.py` file), we can pass in `max_y_objects` in as a parameter, with a default value of 2000. When we iterate for each `x_object`, we can random sample the `y_objects` if necessary.

This means that we are no longer finding the absolute Average Shortest Distance from x to y , as the y -objects we select may not include the closest one to the current x -object. However, this may actually make the metric more realistic, for a reason which is best shown with an example. The closest school to my house is my primary school, which I no longer attend. Therefore, when I travel to school, I must travel further than the distance from my house to the closest school. This would be reflected in the code by the fact that my primary school may not be selected in the random sample of all the schools (y -objects in this example).

No.	Test	Input	Type	Expectation	Output
t_{53}	Maximum objects	y -	Set of y -objects	–	Random sample of y -objects for each x -object As Expected

Table 3.67: Testing Table for Code Snippet 3.79

app.py

```
28 # predict the HDI of the given region
29 def processPrediction(max_x_objects, max_y_objects,
→ progress=gr.Progress()):
```

app.py / def processPrediction

```
48     allFactors.append(averageDistance(
→       all_objects[averageDistanceFactor[0]],
→       all_objects[averageDistanceFactor[1]], max_x_objects,
→       max_y_objects))
```

Code Snippet 3.80: Passing in the max_y_objects

To then take these variables into account when calculating factors, we must pass the `max_x_objects` and `max_y_objects` variables into the `def processPrediction` function.

Then, inside that function, we must pass them into the the `def averageDistance` function.

app.py

```

93     predictHDIbutton = gr.Button(value='Predict HDI',
94         ↪ variant='primary')
95     with gr.Group():
96         max_x_objectsSlider = gr.Slider(minimum=10, maximum=5000,
97             ↪ value=500, label='Maximum x-buildings', info='When
98             ↪ calculating the average distance between 2 types of
99             ↪ building, it will find the average of this many pairs.
             ↪ Increasing this value may make the prediction take
             ↪ much longer, but may make the prediction more
             ↪ accurate.')
50         max_y_objectsSlider = gr.Slider(minimum=10, maximum=5000,
51             ↪ value=500, label='Maximum y-buildings', info='When
52             ↪ calculating the average distance between 2 types of
53             ↪ building, it will find the closest of the second type
54             ↪ of building out of this many options. Increasing this
55             ↪ value may make the prediction take much longer, but
56             ↪ may make the prediction more accurate.')
57     with gr.Group():
58         HDIprediction = gr.Textbox(label='I believe that the HDI
59             ↪ of this region is...', value='', interactive=False)
60         similarHDI = gr.Textbox(label='That\'s a similar HDI
61             ↪ to...', value='', interactive=False)

```

Code Snippet 3.81: Adding Sliders

To let the user decide the values for `max_x_objects` and `max_y_objects`, we can add some sliders. For each of them, I have set the minimum value to 10 and the maximum value to 5000, with a default value of 500. They also each come with an information label, as their function may not be clear to the user.

The use of the `gr.Group()` block will place these elements together, as shown in Figure 3.42. I have also added an output textbox for the similar HDI region, but I will not be implementing this until the 2nd prototype.

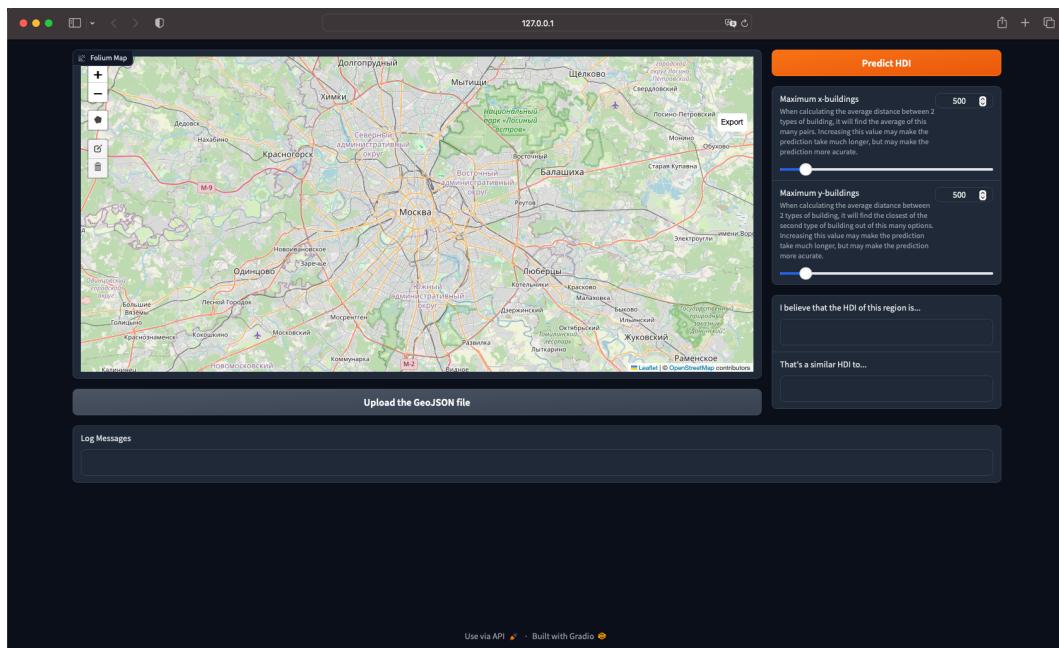


Figure 3.42: `max_x_objects` and `max_y_objects` Sliders

`app.py`

```
104 predictHDIbutton.click(processPrediction, inputs=[max_x_objectsSlider,
    ↪ max_y_objectsSlider], outputs=[HDIprediction])
```

Code Snippet 3.82: Updating Functionality for Predicting HDI Button

To take these slider values into account, we must now pass them as inputs to the `predictHDIbutton.click` method.

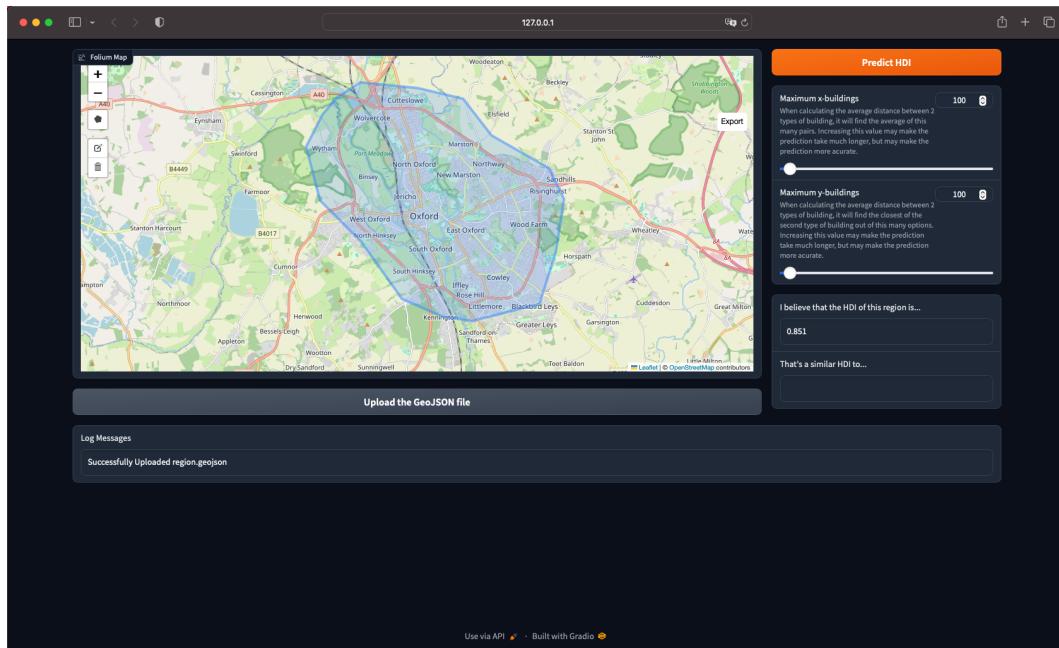


Figure 3.43: Prediction with lower `max_x_objects` and `max_y_objects` values

No.	Test	Input	Type	Expectation	Output
t_{54}	Prediction with lower <code>max_x_objects</code> and <code>max_y_objects</code> values	<code>max_x_objects</code> , <code>max_y_objects</code>	—	Less Accurate Values for $A(x, y)$, still gives accurate prediction	As Expected

Table 3.68: Testing Table for Code Snippets 3.80, 3.81 & 3.82

This prediction 1 minute instead of 9, 50 seconds of which were spent downloading the locations of all the objects, which I have no control over how long it takes.

3.3.4 Part 13 – Suggesting Improvements

In order to make suggestions, we must first define the inputs and the outputs the user will be interacting with.

app.py

```
100 makeSuggestionsButton = gr.Button(value='Make Suggestions')
101 suggestionsTable = gr.Dataframe(label='Suggestions', headers=['New
→ Building', 'Coordinates', 'New HDI', 'Change in HDI'],
→ type='array', interactive=False)
```

Code Snippet 3.83: Adding a Suggestions Table

Here, I am defining a button in the same way as before, along with a `Dataframe`. This will create a table for us to output to, with headers that we can specify. The `type` of this `Dataframe` is set to `'array'`, as I will be returning a python `list` (as opposed to a `np.ndarray` or similar). To ensure that the user cannot edit the data in each cell, or add new rows and columns, `interactive` has been set to `False`.

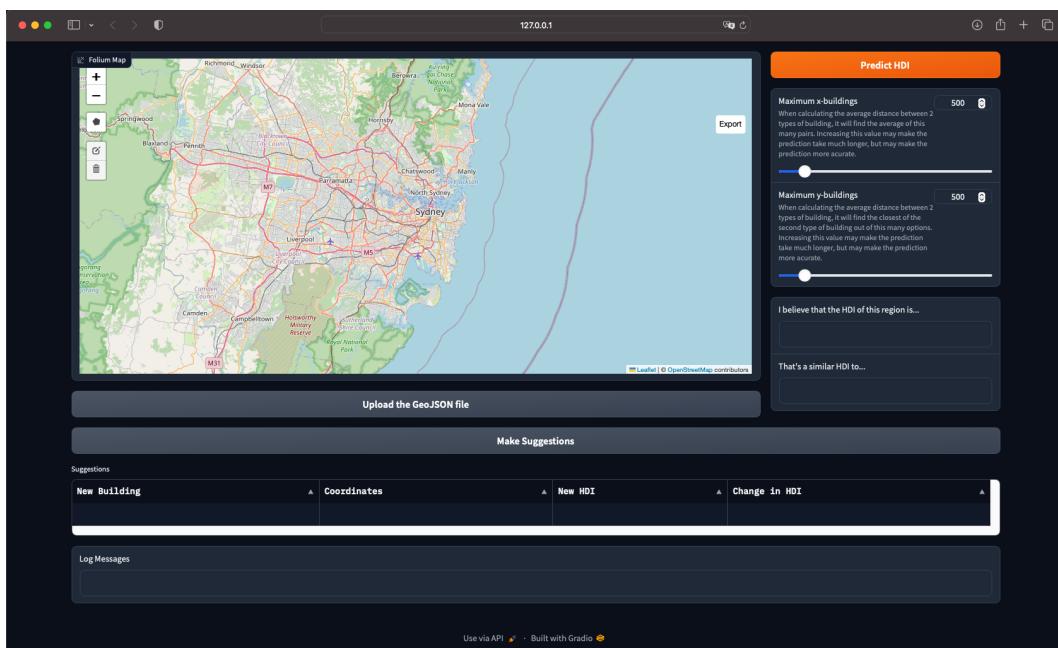


Figure 3.44: Suggestions Table

Figure 3.44 shows the user interface with these new additions. Unfortunately, there is a small bug in `gradio`, which causes the `Dataframe` to have a small white border for scroll bars, even when they are not required (as in this case). Unfortunately, I cannot do anything about this.

No.	Test	Input	Type	Expectation	Output
t_{55}	Suggestions Table and button in the right place	–	–	At the bottom of the page, stacked on top of each other (ignoring the scroll bar bug)	As Expected

Table 3.69: Testing Table for Code Snippet 3.83

Now we must add functionality to this button.

app.py

```

65 # make suggestions from a prediction
66 def makeSuggestions(max_x_objects, max_y_objects, progress=gr.Progress()):
67     global all_objects
68     suggestions = ['school', 'hospital', 'place_of_worship', 'police',
69                     'restaurant', 'slot_machines', 'library', 'pharmacy']
70     returnTable = []
71     for num, suggestion in enumerate(suggestions):
72         progress(num/len(suggestions), desc=f'Suggesting building a
73             {suggestion}...')
```

Code Snippet 3.84: Defining the Suggestions Function

Here, I am defining the `def makeSuggestions` function, which will be called when the user clicks the make suggestions button. It must take in `max_x_objects` and `max_y_objects` again, as we will be computing average distances. It will also use `gr.Progress` again, as it may take a while.

To avoid downloading all of the objects again, we can use the global variable, `all_objects`, which we saved them to, when predicting the HDI. Then, we must define each of the suggestions, as well as the return list, which will be displayed in the `Dataframe`. We will then iterate over each suggestion and calculate its new HDI (for now however, all we can do is update the progress bar).

No.	Test	Input	Type	Expectation	Output
t_{56}	Iterating over Suggestions	A <code>list</code> of object types	–	Iterate over each one, and update the progress bar	As Expected

Table 3.70: Testing Table for Code Snippet 3.84

The first thing we must do for each suggestion is find the optimal place for it to go.

app.py

```
65 # helper function to calculate mean of coordinates
66 def calcMeanOfCoords(coords):
67     return [np.mean([coord[0] for coord in coords]), np.mean([coord[1] for
68     ↳ coord in coords])]
```

Code Snippet 3.85: Finding the mean of coordinates

To do this, we must be able to calculate the mean of coordinates (that is, the latitude will be the mean of all the latitudes, and the longitude will be the mean of all the longitudes).

app.py

```
69 # calculate the optimal place for a new building to go when suggesting
70 def calcOptimalPlaceFor(building):
71     global all_objects
72     means = []
73     for averageDistanceFactor in averageDistanceFactors:
74         if averageDistanceFactor[1] == building:
75             means.append(calcMeanOfCoords(
76                 ↳ all_objects[averageDistanceFactor[0]]))
77     return calcMeanOfCoords(means)
```

app.py / def makeSuggestions

```
85     # find optimal place
86     progress((num/len(suggestions))+(0*(1/32)), desc=f'Suggesting
87     ↳ building a {suggestion} (finding optimal position)...')
88     position = calcOptimalPlaceFor(suggestion)
```

Code Snippet 3.86: Finding the Optimal Place for a New Building

To then find the optimal place (according to the algorithm shown in Figure 2.32), we must iterate over each average distance factor, for which the second parameter is the new building, and find the mean position of the first parameter. The optimal position will then be the mean of those means.

We then must call this function in the actual `def makeSuggestions` function for each of the suggestions.

No.	Test	Input	Type	Expectation	Output
$T_{7.1}$	Optimal Place for New Building	Building for which there exists 1 Factor	Valid	Calculates mean position of the other building in that factor	As Expected
$T_{7.2}$	Optimal Place for New Building	Building for which there exists more than 1 Factor	Valid	Calculates the mean position of the mean positions of the other buildings in those factors	As Expected
$T_{7.3}$	Optimal Place for New Building	Building for which there exists 0 Factors	Invalid	Function is never ran with this option	As Expected

Table 3.71: T_7 Testing Table (for Code Snippets 3.85 & 3.86)

The next step is to calculate the new average distances. As we are making use of random sampling, it would not guaranteed that the new building is considered, if we just add it to the list. Therefore, we must add it to the considered objects after we random sample, which will require some extra parameters:

`calc_factors.py`

```
18 # finds the average distance between 2 types of objects, A(x,y)
19 def averageDistance(x_objects, y_objects, max_x_objects=2000,
→ max_y_objects=2000, must_include_x=None, must_include_y=None):
```

`calc_factors.py / def averageDistance`

```
24     if must_include_x != None: # include the new building when making
→     suggestions
25         x_objects[0] = must_include_x
```

`calc_factors.py / def averageDistance`

```
34     if must_include_y != None: # include the new building when making
→     suggestions
35         y_objects_in_consideration[0] = must_include_y
```

Code Snippet 3.87: Updating the Average Distance Function

Here, in the `def averageDistance` function, the parameters `must_include_x` and `must_include_y` are set to `None` by default (as to not break the previous function calls). If we

pass something in however, we want to include it in the calculations. We can do this by setting the first item in the list to be the new building, as each element in those lists are random ones anyway, so it is irrelevant which one we override.

```
app.py / def makeSuggestions

88     # calculate average distances
89     progress((num/len(suggestions))+(1*(1/32)), desc=f'Suggesting
90         ↵ building a {suggestion} (calculating new average
91         ↵ distances)...')
92     allFactors = []
93     for averageDistanceFactor in averageDistanceFactors:
94         # ensure that we include the new building
95         if suggestion == averageDistanceFactor[0]:
96             allFactors.append(averageDistance(
97                 ↵ all_objects[averageDistanceFactor[0]],
98                 ↵ all_objects[averageDistanceFactor[1]], max_x_objects,
99                 ↵ max_y_objects, must_include_x=position))
100            elif suggestion == averageDistanceFactor[1]:
101                allFactors.append(averageDistance(
102                    ↵ all_objects[averageDistanceFactor[0]],
103                    ↵ all_objects[averageDistanceFactor[1]], max_x_objects,
104                    ↵ max_y_objects, must_include_y=position))
105            else:
106                allFactors.append(averageDistance(
107                    ↵ all_objects[averageDistanceFactor[0]],
108                    ↵ all_objects[averageDistanceFactor[1]], max_x_objects,
109                    ↵ max_y_objects))
```

Code Snippet 3.88: Calculating the New Average Distances

Then, when we calculate the new average distances, we can pass in the position of the new building as required (either as an `x_object`, `y_object` or neither).

No.	Test	Input	Type	Expectation	Output
t_{57}	Average Distance with New Building	Position of New Building	-	Always taken into account, after random sampling	As Expected

Table 3.72: Testing Table for Code Snippets 3.87 & 3.88

app.py / def makeSuggestions

```

99      # calculate density factors
100     progress((num/len(suggestions))+(2*(1/32)), desc=f'Suggesting
101       ↵ building a {suggestion} (calculating new densities)...')
102     area = calcArea(currentRegion)
103     for densityFactor in densityFactors:
104       # ensure that we include the new building
105       if suggestion == densityFactor:
106         allFactors.append(density(all_objects[densityFactor] +
107           ↵ [position], area))
108       else:
109         allFactors.append(density(all_objects[densityFactor],
110           ↵ area))

```

Code Snippet 3.89: Calculating the New Densities

When calculating the density factors, we must add the position of the new building to the list if and only if the current density factor we are calculating is about the new building. Otherwise, we can just calculate it as normal.

No.	Test	Input	Type	Expectation	Output
t_{58}	Densities with New Building	Position of New Building	—	Considered if we are on that density factor	As Expected

Table 3.73: Testing Table for Code Snippet 3.89

app.py / def makeSuggestions

```

108      # predict HDI
109     progress((num/len(suggestions))+(3*(1/32)), desc=f'Suggesting
110       ↵ building a {suggestion} (predicting new HDI)...')
111     inputLayer = np.matrix([[100 if factor == None else
112       ↵ float(factor)/10 if index <= 11 else float(factor)] for index,
113       ↵ factor in enumerate(allFactors)])
114     prediction = round(network.predict(inputLayer).item(0,0), 3)

```

Code Snippet 3.90: Predicting the New HDI

Predicting the HDI from these factors is exactly the same again.

No.	Test	Input	Type	Expectation	Output
T_8	Suggestions fed back into Neural Network	Suggested Building	—	Location is Calculated, Average Distances are Calculated, Densities are Calculated, HDI is Calculated	As Expected

Table 3.74: T_8 Testing Table (for Code Snippet 3.90)

We now must return these HDI predictions to the table.

app.py

```
19 currentHDI = -1
```

```
35 app.py / def processPrediction
```

```
global currentHDI
```

```
65 app.py / def processPrediction
```

```
currentHDI = prediction
```

```
115 app.py / def makeSuggestions
```

```
returnTable.append([suggestion, position, prediction,
    ↵ round(prediction-currentHDI, 3)])
```

```
116 return returnTable
```

Code Snippet 3.91: Storing the Current HDI Globally & Returning a Table

One of the columns of the table is the change in HDI. In order to calculate this, we must know the predicted HDI of the region. To do this I have used a `global` variable, `currentHDI`. It is initially set to `-1`, and updated when we make a prediction.

Once we have this, we can create a row of the table, with all of the required information, and append it to the `return` variable defined at the start of the function (`returnTable`).

app.py

```
160     makeSuggestionsButton.click(makeSuggestions,
  ↳   inputs=[max_x_objectsSlider, max_y_objectsSlider],
  ↳   outputs=[suggestionsTable])
```

Code Snippet 3.92: Adding Functionality to the Suggestions Button

To make the button actually work, we can add a functionality, to call the `def makeSuggestions` function, passing in the values held by the sliders, and outputting to the table.

No.	Test	Input	Type	Expectation	Output
t_{59}	Suggestions Returned to Table	—	—	Columns for building type, location, new HDI, and change in HDI	JSON Error

Table 3.75: Testing Table for Code Snippets 3.91 & 3.92

```
+----- 1 -----
| Traceback (most recent call last):
|   File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/starlette/responses.py", line 253, in wrap
|     await func()
|   File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/starlette/responses.py", line 242, in stream_response
|     async for chunk in self.body_iterator:
|       File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/gradio/routes.py", line 938, in sse_stream
|         raise e
|       File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/gradio/routes.py", line 902, in sse_stream
|         response = process_msg(message)
|       File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/gradio/routes.py", line 858, in process_msg
|         return f"data: {json.dumps(message.model_dump())}\n\n"
|   TypeError: Type is not JSON serializable: numpy.float64
```

Figure 3.45: Invalid Table Format

This returned an error, which made me realise I had forgotten to format the location.

app.py / def makeSuggestions

```
115     returnTable.append([suggestion, f'{round(position[0], 7)}°N',
  ↳   {round(position[1], 7)}°E', str(prediction),
  ↳   str(round(prediction-currentHDI, 3))])
```

Code Snippet 3.93: Creating a Readable Format for the Coordinates

This is the amended line for creating a new row in the table, with the coordinate being formatted as a string, stating the latitude (in degrees North) and longitude (in degrees East).

No.	Test	Input	Type	Expectation	Output
t_{59}	Suggestions Returned to Table	—	—	Columns for building type, location, new HDI, and change in HDI	As Expected

Table 3.76: Testing Table for Code Snippet 3.93

Suggestions				
New Building	Coordinates	New HDI	Change in HDI	
school	51.4762373°N, -0.0687613°E	0.851	0.0	
hospital	51.4863596°N, -0.107484°E	0.851	0.0	
place_of_worship	51.4762373°N, -0.0687613°E	0.851	0.0	
police	51.4762373°N, -0.0687613°E	0.851	0.0	
restaurant	51.4762373°N, -0.0687613°E	0.851	0.0	
slot_machines	51.4927741°N, -0.1342361°E	0.851	0.0	
library	51.5007017°N, -0.1086843°E	0.851	0.0	
pharmacy	51.4762373°N, -0.0687613°E	0.851	0.0	

Figure 3.46: Suggestions Table Successfully Generated

Figure 3.46 shows a table of suggestions. This has *functionally* worked perfectly, however you may notice that for each suggestion, the predicted HDI has not changed. I will therefore need to be improving the way we make suggestions in the next prototype.

The user may also attempt to press the suggest button before predicting the HDI, which should not be able to happen.

```
app.py / def makeSuggestions
```

```
84     if currentHDI == -1:
85         return [['You have not yet predicted an HDI!', None, None, None]]
```

Code Snippet 3.94: Validating the `def makeSuggestions` function

If the HDI has not yet been predicted, the value of the `currentHDI` variable will be `-1`. It is impossible for it to ever be `-1` again, as HDI will be between 0 and 1. Therefore, at the start of the `def makeSuggestions` function, if the user has not predicted the HDI yet, we can return an error message. The message is formatted in this list so that it is in the right format for the table.

Suggestions				
New Building	Coordinates	New HDI	Change in HDI	
You have not yet predicted an HDI!				

Figure 3.47: Error Message Shown in Table

No.	Test	Input	Type	Expectation	Output
t_{60}	Validation for Suggestions	User attempts to make suggestions before predicting	Invalid	Error is thrown to the table	As Expected

Table 3.77: Testing Table for Code Snippet 3.94

The final thing I would like to do in this prototype is letting the user select one of the suggestions locations, and show it on the map.

app.py

```
11 import copy
12
13
14
15
16
17
18
19
20
21 # place a pin on the map when user clicks the suggestion
22 def selectSuggestionsTable(evt: gr.SelectData):
23     cellData = evt.value
24     newMap = copy.deepcopy(foliumMap)
25     if '°' in cellData: # it is a coordinate
26         coordinates = [float(coord[:-2]) for coord in cellData.split(', '
27             ↴ ')] # convert it to floats, remove formatting
28         folium.Marker(location=coordinates).add_to(newMap)
29         newMap.location=coordinates
30
31     return newMap
```

Code Snippet 3.95: Placing a Pin on the Map

To do this, we can write a function, which takes in some `gr.SelectData`, which is the cell the user has clicked on. We will update the map by changing the value of the `map` component, to have a marker at the new location. To do this, we will be returning a copy of the map, (which must be copied with the `copy` module).

This should only happen if the user selects a cell which has coordinates. This will be the case only when it includes the `'°'` character. If they have selected a valid cell, we can extract the coordinates from the string, and add a marker to the new map. We can also set the map's location to the marker, so that the user doesn't have to scroll around to find it.

app.py / def makeSuggestions

```
174 suggestionsTable.select(selectSuggestionsTable, inputs=[],  
→ outputs=[map])
```

Code Snippet 3.96: Adding Functionality to the Suggestions Table

We call this function, when the `suggestionsTable` is selected, and we want to return to the `map` component.

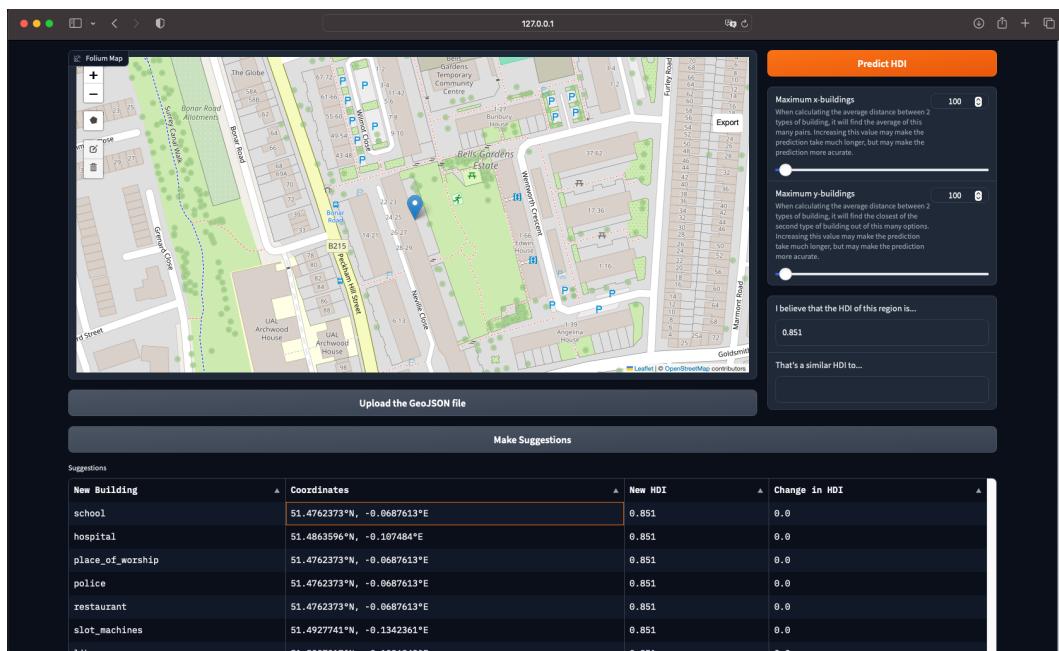


Figure 3.48: Pin Shown on Map for Suggestion

No.	Test	Input	Type	Expectation	Output
t_{61}	User clicks coordinate to see map	—	—	Marker added to map and shows that area	As Expected

Table 3.78: Testing Table for Code Snippets 3.95 & 3.96

3.3.5 Review

No.	Test	Inputs	Pass / Fail
T_6	Overall Flow	User Draws Region and then Predicts, User tries to predict before selecting Region, User draws region and doesn't predict	Pass
T_7	Optimal Place for New Building	Building for which there exists 1 Factor, Building for which there exists more than 1 Factor, Building for which there exists 0 factors	Pass
T_8	Suggestions fed back into Neural Network	All 8 suggestions for where to place new buildings	Pass

Table 3.79: Passed Testing Data for Milestone 3

All 3 of the main tests in Milestone 3 passed. As a result of that, the following success criteria has been met:

No.	Success Criterion	Justification	Success
S_8	User can select an area on the map for which they want to analyse	This will allow the user to choose which area they want to predict the HDI of by drawing on the map.	✓
S_9	Neural Network accurately predicts HDI from those factors	Once the user has selected their area, the HDI must be predicted based on factors calculated from that area.	Partial ✓
S_{10}	Accurately calculate where a new building would need to be in order to increase the HDI the most	The specific placement of new buildings (as suggestions to increase HDI) is important to consider, so a suitable place must be chosen in order to decrease the average distance to it as much as possible.	✓
S_{11}	Changes are suggested to increase the HDI	This will allow the user to make an informed decision on what to do in order to improve their area.	Partial ✓

Table 3.80: Achieved Success Criteria for Milestone 3

S_9 and S_{11} have only been given partial ✓'s, as I believe that the predictions and suggestions could be more accurate, if I spent more time training the network.

I also have an initial user interface, which allows for the user to draw any region on a map, predict the HDI of it, and make suggestions on how to improve it, so overall, I believe this milestone has been a success.

However, the suggestions made often do not make any change to the predicted HDI, so in the next milestone, I plan to come up with a better method for generating suggestions.

3.4 Feedback from Stakeholders

In order for my stakeholders to give feedback, I need to be able to share this application with them. Gradio again makes this very easy, as all I have to do is pass `share=True` into `app.launch()` (in Code Snippet 3.65), and a public link (as opposed to the localhost IP address I have been using) will be generated.

```
Running on local URL: http://127.0.0.1:7860
Running on public URL: https://d6c11289fbc6a86896.gradio.live
```

Figure 3.49: Output for when we set `share=True`

I then shared this link with my stakeholders, and asked for general feedback about the experience of using the app, and what could be better. (Again, coloured quotes will represent each stakeholder, as shown in the Stakeholders Table (Table 1.1)).

“I thought that the user interface had a very modern and clean design, which made it pretty easy to use. However, the fact that I had to export the map and then upload it made it a little harder to understand. The processes were quite slow, but in the end made fairly accurate predictions. The main thing that needs improving is the suggestions, as all of them returned no change in HDI.”

“I did not think that it was very usable, and it took me a bit to figure out the part where you export and upload the region. I think it would be helpful to add a help screen, which gives step by step instructions on how to use the various features, similar to how the sliders are labelled. While they are slightly inaccurate, the predictions are still very impressive, considering the limited data I inputted.”

“My first impressions of the user interface were pretty good, as it looked quite professional. The thing I would improve most is the time taken to make predictions, but I understand not much more can be done about it. I therefore think it would be a good idea to let the user have the option to manually enter these factors, in case they have problems (such as a poor internet connection).”

“Overall, I thought that the experience of using this app was pretty good, except for the fact that I ran into an error when predicting the HDI. I was told that this was caused by timing out with the Overpass Turbo server, and that I just had to try again, which then worked. I think that the user should be made aware of this, but otherwise everything went well.”

Overall, it seems like the main thing I need to improve from this prototype is the quality of suggestions. The stakeholders also gave some ideas on what to further add to the next prototype, such as a help menu and a way to enter factors manually.

4. Design of the 2nd Prototype

4.1 Structure Diagram

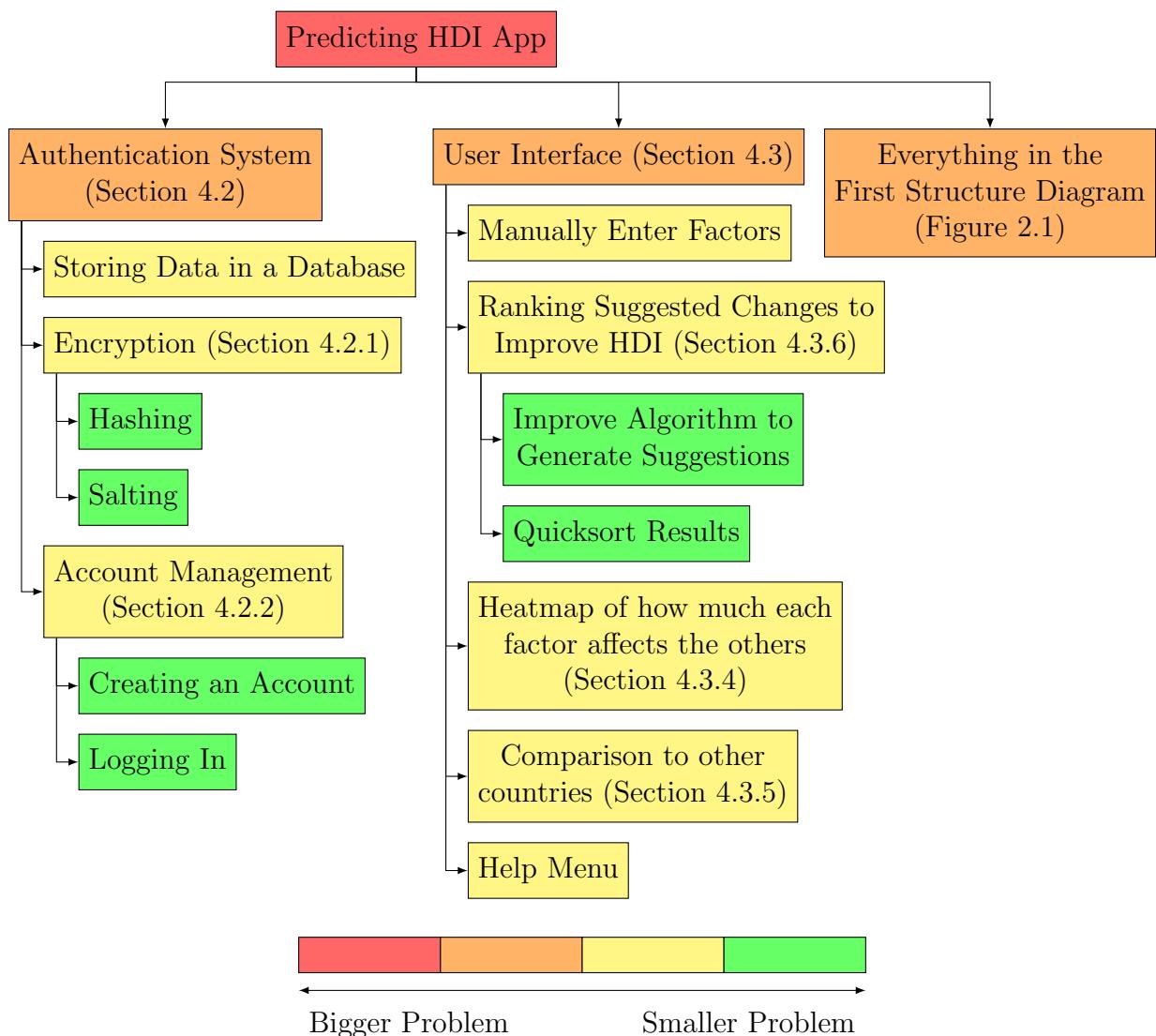


Figure 4.1: Structure Diagram (2nd prototype)

Figure 4.1 shows this project’s structure, breaking down the larger problem of the entire app, into a series of smaller problems which are easier to consider. Some of the problems listed above were present in the first structure diagram, but they have been added to this one as well as I have not yet implemented them. I will go into more detail for each of the smaller problems above (further decomposing them), in the approximate order I would be implementing them.

4.2 Authentication System

4.2.1 Storing Data & Encryption

In the second prototype, users will be able to create and log into an account, which they can use to save regions they have predicted. I will be using MongoDB to store this data, which allows for `json` objects to be stored in “collections” (tables).

Most of the data I will be entering can just be stored as it is, but the user’s password must be encrypted, to ensure security. One thing we can do to encrypt the password is to pass it through a hashing function.

Hashing functions (such as `sha256` or `MD5`) take in an input and generate a unique, fixed-length key as an output, obscuring the original meaning. Hashing functions are also impossible to reverse-engineer, meaning that if a hacker gets a hold of hashed passwords, they will not be able to find the original password.

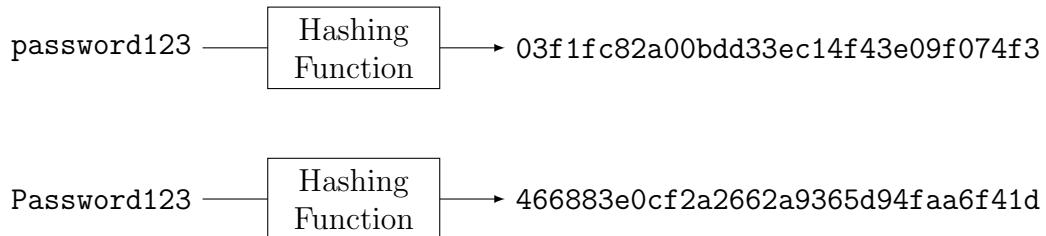


Figure 4.2: Hashing Examples

As the hashed output is based on the entire input, changing the input only slightly (for example, “p” to “P”) will have a big effect on the output, which makes it harder for hackers to look for patterns.

However, humans are often predictable, and many of them use the same, easy to guess, password (even I have done it in the example). Therefore, if a hacker has a list of common passwords, they can pass them through the hashing function, and compare them against the database. Alternatively, they could use a “rainbow table”, which is a table of common passwords, along with the hashed versions, which means they don’t need to compute any hashes.

To avoid this, we can add salt to the passwords, before we hash them. This is where we append a random set of characters to the beginning of the password, before hashing. The salt (random characters) will be stored alongside the password, so that we can reconstruct the hashed password.

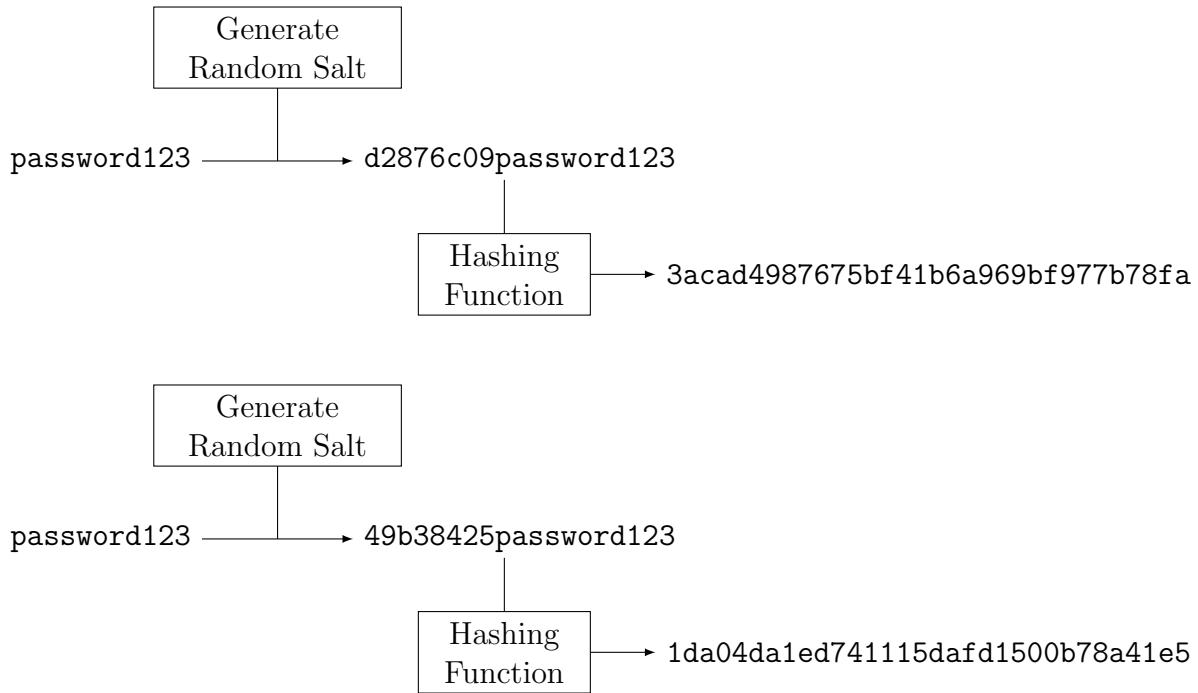


Figure 4.3: Salting Examples

In the first example, what would be stored in the database would be `d2876c09` for the salt and `3acad4987675bf41b6a969bf977b78fa` for the hashed password.

This protects against the rainbow tables, as the hackers will then have to add the salt to each of their common passwords, and then run the hashing function, instead of just using a table. However, we should also attempt to protect against doing that, as it is still pretty easy.

One way to do this, is to ensure that we use a slow hashing algorithm. Many high-end GPUs can compute billions of hashes per second, if we don't use a slow hashing algorithm, but can only compute thousands of them if we do. The slow hashing algorithm I will be using is called `bcrypt`.

4.2.2 Account Management

Creating an Account

The first thing the user must do when creating an account is select a username, but we must check that the username they select does not already exist. If it doesn't, they can write their password, validating with double entry validation. If they match, the user can submit and the password will be salted and hashed. The username, the hashed password, and the salt will be stored in the MongoDB.

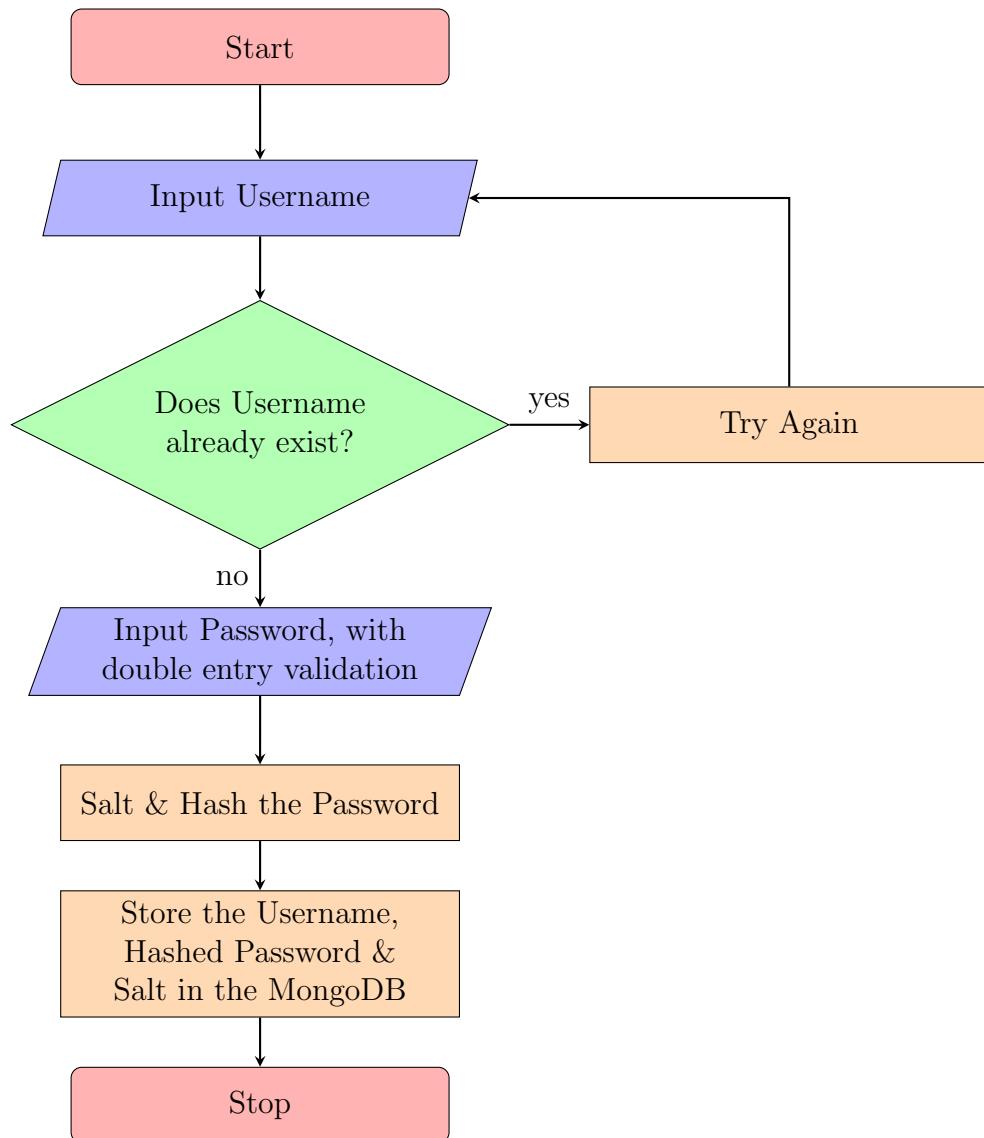


Figure 4.4: Creating an Account Flowchart

Logging in

When logging in to an existing account, the user will enter their username and password. We must check if the username exists, and if it does, we can retrieve the salt, add it to the password and hash it, and compare that against the hashed password in the database. If it matches, we can grant access, otherwise they can try again.

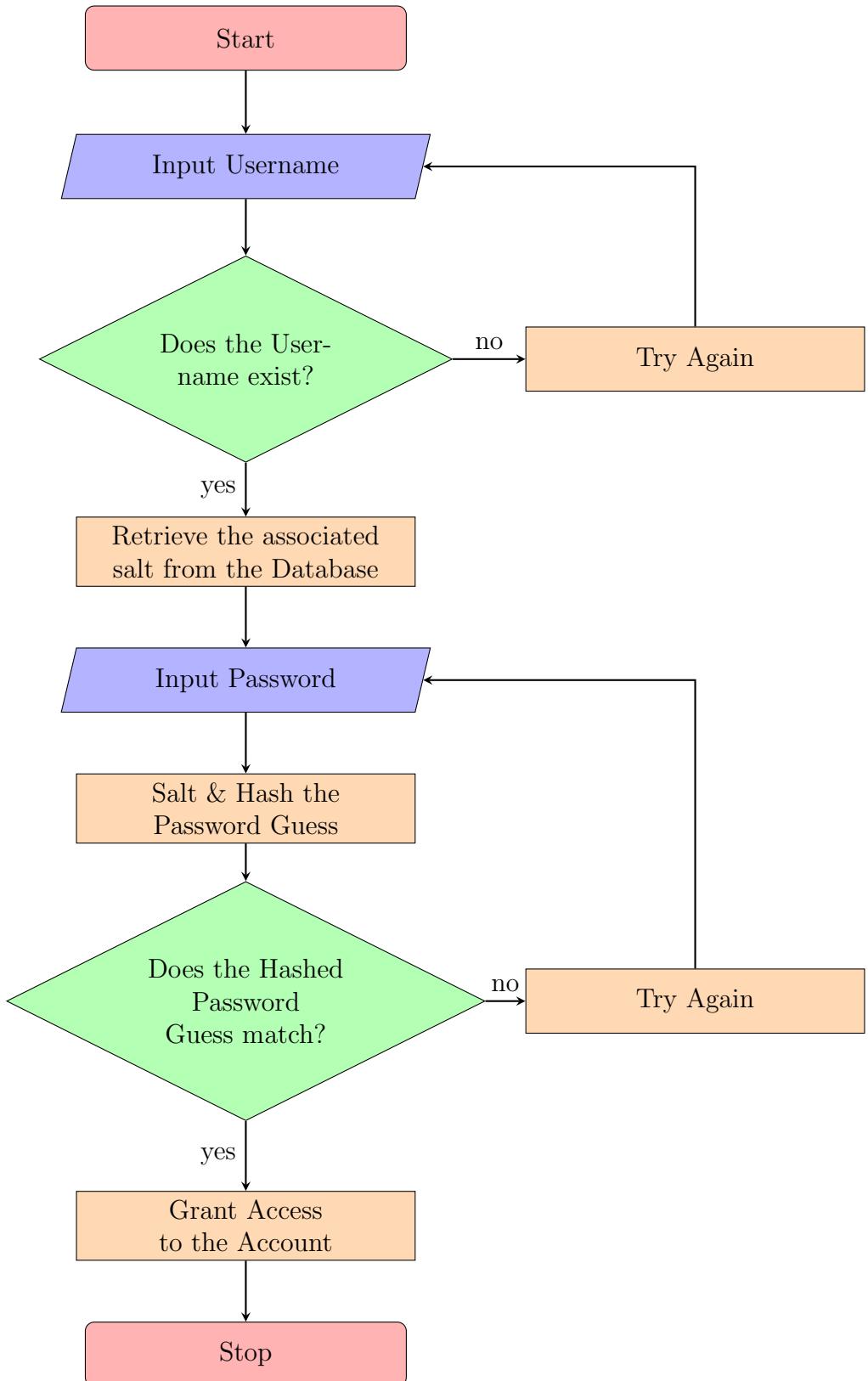


Figure 4.5: Logging In Flowchart

4.2.3 List of Key Variables

Variables in Creating an Account

Identifier	Data Type	Explanation	Justification
username	str	Stores what the user entered as their username	Must be checked against the database to see if it exists already
password	str	Stores what the user entered as their password	All accounts should have a password for authentication
password2	str	Stores what the user entered as their password, in the second field	This ensures double entry validation
salt	str (hex)	Stores the random characters that will be added to the password	The salt must be added to the database with the password
hashed_password	str (hex)	Stores the password after it has been salted and hashed	This is what will be stored in the database for us to check against

Table 4.1: List of Key Variables in Creating an Account

Variables in Logging In

Identifier	Data Type	Explanation	Justification
username	str	Stores what the user entered as their username	Must be checked against the database to see if it exists
salt	str (hex)	Stores the random characters that were added to the password	Retrieved from the database, to get the same hash
password	str	Stores what the user entered as their password	All accounts should have a password for authentication
hashed_password	str (hex)	Stores the salted and hashed version of what the user entered as their password	Used to check if what they have entered is correct

<code>correct_ hashed_ password</code>	<code>str</code> (hex)	Stores the correct salted and hashed password	Used to check against the password guess, if they match we can grant access to the account
--	------------------------	---	--

Table 4.2: List of Key Variables in Logging In

Other Variables

Identifier	Data Type	Explanation	Justification
<code>database</code>	<code>object</code>	Stores the database from MongoDB	We must be able to retrieve values from it, in order to authenticate the user

Table 4.3: List of Other Key Variables in Authentication System

4.3 User Interface

4.3.1 Interface Sketch – A Tabbed Interface

As seen before, when implementing the GUI, Gradio offers different ways to arrange elements (For example, I have used `gr.Row()`, `gr.Column()` and `gr.Group()`). Another one that we can make use of is `gr.Tab()`, which will place all of its elements inside a tab. Other `gr.Tab()` elements on the same level will appear next to each other, for the user to switch between.

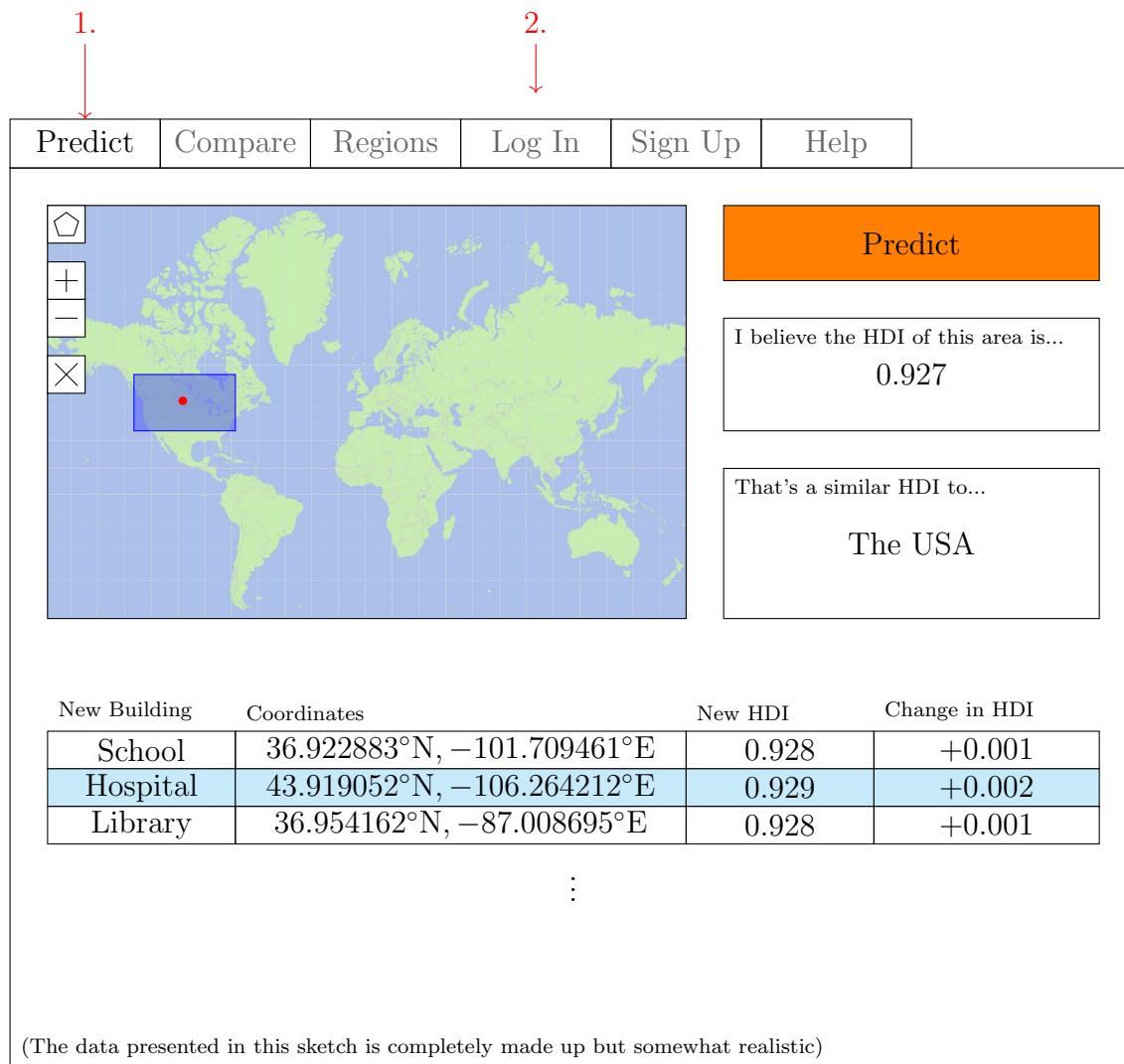
Figure 4.6: Technical Interface Sketch (2nd prototype, main page)

Figure 4.6 shows the main prediction page for the app, which I have already implemented, and so most of these elements are not labelled. The elements that are labelled are:

1. Currently Selected Tab
2. Other Tabs

The contents of the other 5 tabs are illustrated below, in Figures 4.7 to 4.11.

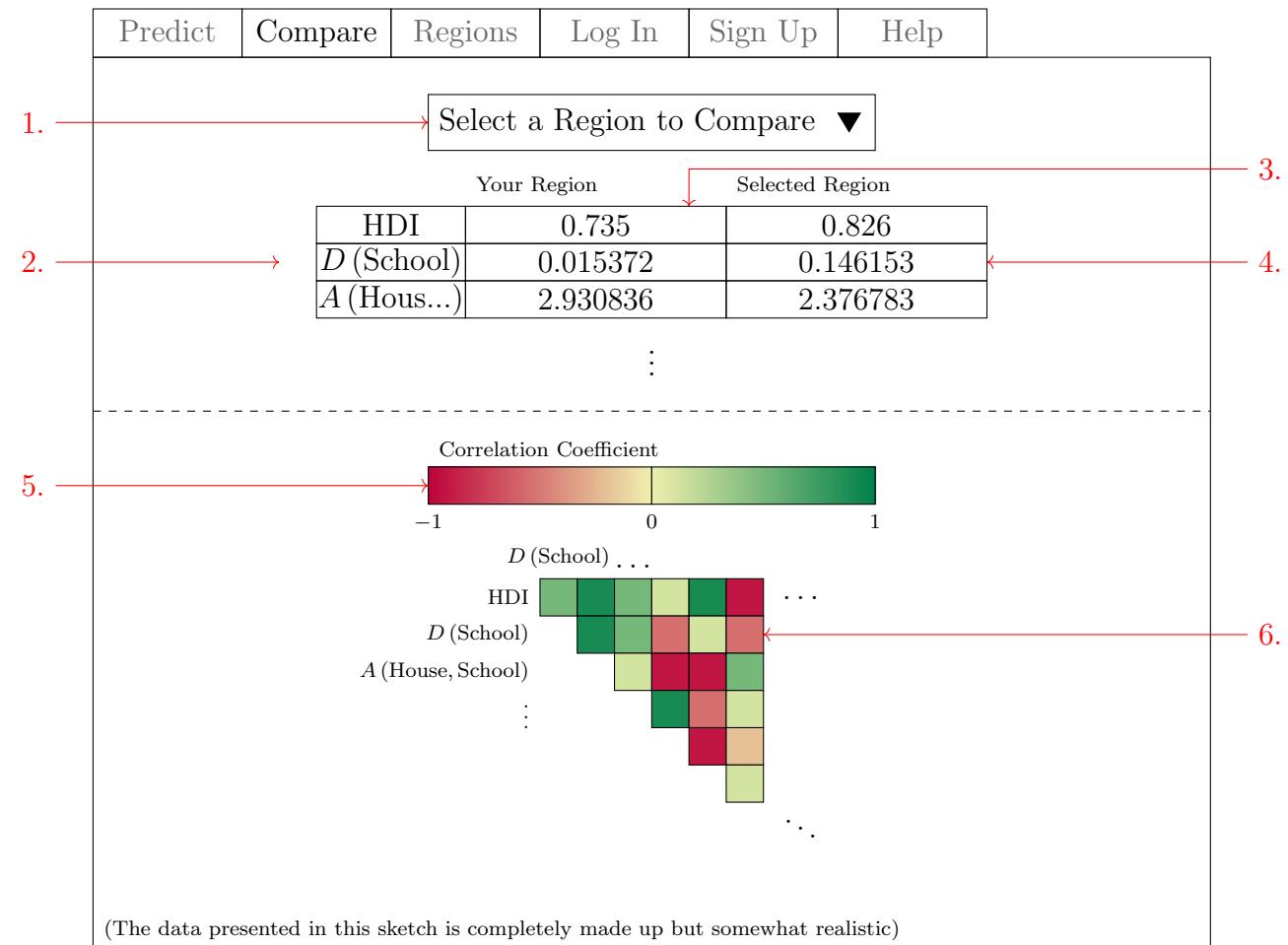


Figure 4.7: Technical Interface Sketch (2nd prototype, compare page)

The labelled items shown on Figure 4.7 are:

1. Dropdown menu to Select a Region to Compare
2. Table showing the HDI and each of the factors for your region and the chosen region
3. Column for your region
4. Column for the chosen region
5. Key for the correlation between factors and HDI
6. Correlation Heatmap (Analysis of how much each factor affects the others)

I will go into more detail about the meaning of the key & heatmap in Section 4.3.4.

Predict	Compare	Regions	Log In	Sign Up	Help																														
<p style="text-align: center;">Current Region</p> <div style="border: 1px solid black; padding: 2px; width: fit-content;">London</div>																																			
1.																																			
2.	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Name</th> <th>D (School)</th> <th>D (Hospital)</th> <th>D (Tree)</th> <th>D (Restaurant)</th> </tr> </thead> <tbody> <tr> <td>Oxford</td> <td>1.1923</td> <td>0.0917</td> <td>62.569</td> <td>2.8799</td> </tr> <tr> <td>London</td> <td>1.5946</td> <td>0.1248</td> <td>32.861</td> <td>3.3566</td> </tr> <tr> <td>region-3</td> <td>0.5673</td> <td>0.0132</td> <td>10.483</td> <td>1.4084</td> </tr> <tr> <td>region-4</td> <td>0.9191</td> <td>0.0342</td> <td>29.589</td> <td>5.4273</td> </tr> <tr> <td>region</td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table>					Name	D (School)	D (Hospital)	D (Tree)	D (Restaurant)	Oxford	1.1923	0.0917	62.569	2.8799	London	1.5946	0.1248	32.861	3.3566	region-3	0.5673	0.0132	10.483	1.4084	region-4	0.9191	0.0342	29.589	5.4273	region				
Name	D (School)	D (Hospital)	D (Tree)	D (Restaurant)																															
Oxford	1.1923	0.0917	62.569	2.8799																															
London	1.5946	0.1248	32.861	3.3566																															
region-3	0.5673	0.0132	10.483	1.4084																															
region-4	0.9191	0.0342	29.589	5.4273																															
region																																			
3.	...																																		
4.	←																																		
5.	←																																		
(The data presented in this sketch is completely made up but somewhat realistic)																																			

Figure 4.8: Technical Interface Sketch (2nd prototype, regions page)

The labelled items shown on Figure 4.8 are:

1. A Textbox showing the region the user has currently selected (it may not always be obvious)
2. Table for each of the user's regions. The user can choose the name of each one, by typing it into the "Name" column.
3. Examples of regions where the user has not chosen a name
4. Example of a region where the user has not yet entered the factors
5. Button to add a row to the table, so that the user can manually enter factors

In this page, the user will be able to switch between the different regions they have processed before. If they are logged in with an account, regions they have processed in a previous session will also appear here. Users can also use this to manually enter factors, by adding rows to this table.

Predict	Compare	Regions	Log In	Sign Up	Help
---------	---------	---------	--------	---------	------

1.  Username

2.  Password

3.  Log In

4.  Result

Figure 4.9: Technical Interface Sketch (2nd prototype, log in page)

The labelled items shown on Figure 4.9 are:

1. Field for the user to enter their username
2. Field for the user to enter their password
3. Log in Button
4. Output Button showing the result of pressing the button (e.g. Login Success, Username does not exist, or Incorrect Password)

Predict	Compare	Regions	Log In	Sign Up	Help
---------	---------	---------	--------	---------	------

Username
1.

Password
2.

Re-Enter Password
3.

Create Account
4.

Result
5.

Figure 4.10: Technical Interface Sketch (2nd prototype, sign up page)

The labelled items shown on Figure 4.10 are:

1. Field for the user to enter their username
2. Field for the user to enter their password
3. Field for the user to re-enter their password (for double-entry validation)
4. Sign up Button
5. Output Button showing the result of pressing the button (e.g. Sign Up Success, Username already exists, or Password not Strong Enough, Passwords Do Not Match)

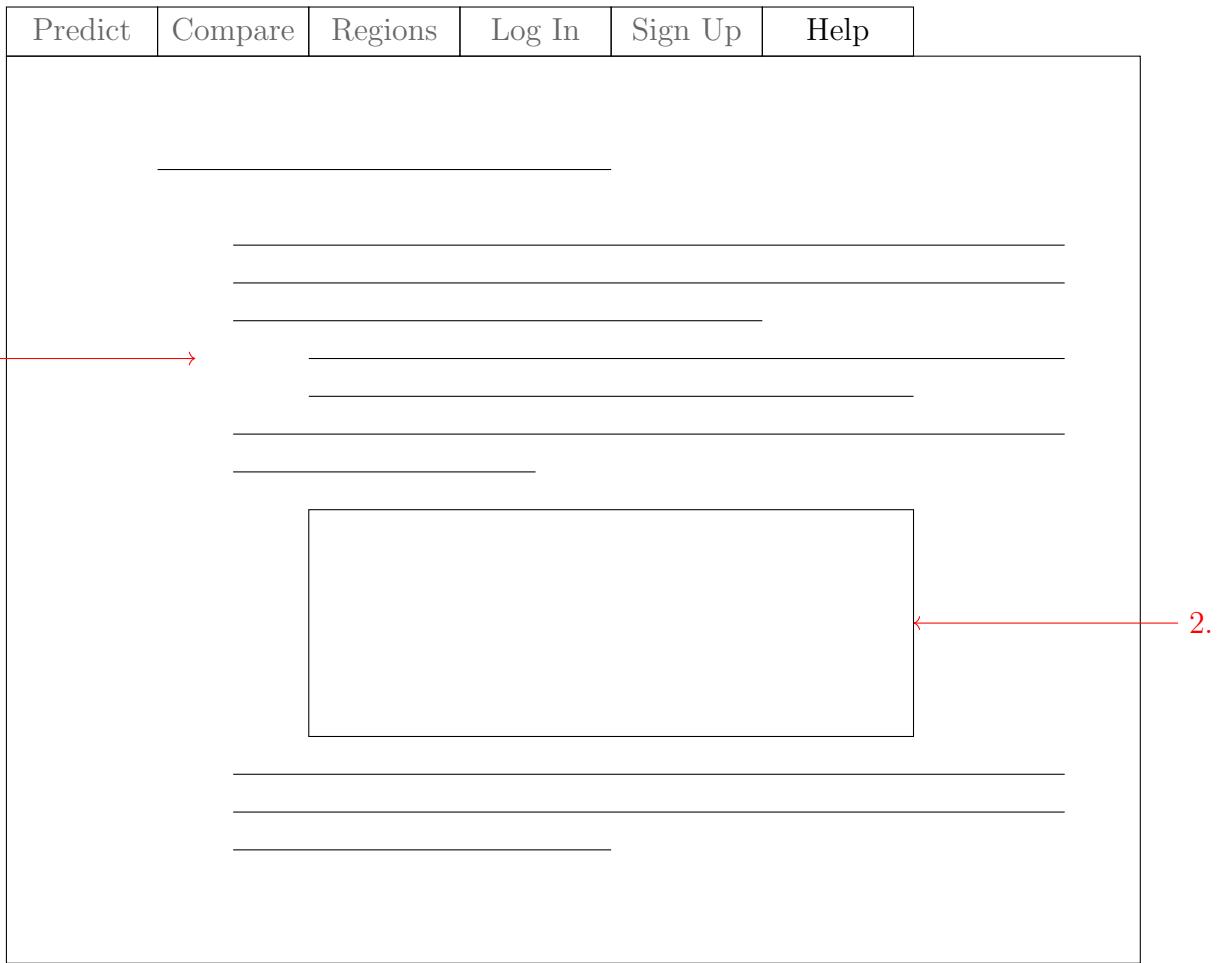


Figure 4.11: Technical Interface Sketch (2nd prototype, help page)

The labelled items shown on Figure 4.11 are:

1. Text describing the features of the app (Actual help text not included in this sketch, as I will be writing it as part of the development)
2. An Image or Diagram to help the understanding

This tab will be written in markdown (which gradio supports), as it allows for basic text formatting, such as different levels of headings, bullet points, ordered lists and tables.

4.3.2 Input Validation

Authentication System

When the user creates an account, they will first be prompted to enter a username, which must be a string. Gradio textboxes can only hold strings, so no matter what the user inputs, it will be validated. They must then enter a password, and then re-enter their password, for double entry validation. This is where the user enters data twice, to ensure they have

entered the correct data (if the 2 copies do not match, they may have not entered the correct data). In order to ensure that the user uses a secure password, I will be enforcing that the password must be at least 8 characters, using a length check, and that it contains at least 1 number and 1 special character, using a format check. I will also be using a presence check to see if all 3 fields have been filled, otherwise they cannot create an account.

When the user logs into an account, they will again be prompted to enter a username and password. The only input validation that must be done here is a presence check, to ensure they have entered both a username and password.

Manually Entering Regions

On the Regions tab (Figure 4.8), I will be using a gradio Dataframe for the regions table (which is the same thing we used for the suggestions table), except this time it will be interactive. This means that the user will be able to enter data into any of the cells, as well as create new rows.

To ensure that the data in each column is of the right type (`str` for the Name column, and `float` for everything else), we can specify the datatype of each column. When the user creates a new region by drawing on the map, the name of it will be the same as the filename they uploaded. When the user creates a new region by adding a new row to the dataframe, it will be called “region”.

Comparing Regions

When comparing regions, the only thing the user has to input is which region they wish to compare to. It should only be one of the ~ 2000 regions processed in Milestone 1 (Section 3.1), and not anything else. I will therefore be using a dropdown menu, instead of something like a textbox. The gradio dropdown menu however can act like a textbox, where the user types to filter the options.

4.3.3 Usability Features

Comparison Page

On the comparison page, users will be able to choose a region from the training data to compare their region to. They will be presented with a dropdown menu, which will be the top element in the screen (to draw their attention to it), to choose an element from. On the right side of the dropdown menu, there will be a small triangle pointing down, to suggest that clicking on this element will show a dropdown menu. A cursor will also appear when the element is selected, which suggests that they can type to search for a particular region.

The rest of the elements on this page (the table, the correlation coefficient key and the correlation heatmap) will be not be interactive, as all they do is convey information (as opposed to take in information from the user). Everything will be labelled accordingly, and if the user does not understand something (for example, the correlation heatmap), they can go to the help page.

In short, a correlation coefficient of 1 means the 2 variables have a very strong positive correlation, a correlation coefficient of -1 means the 2 variables have a very strong negative

correlation, and a correlation coefficient of 0 means they have no correlation. I have therefore chosen the colour to represent 1 to be green and the colour to represent -1 to be deep red, as green often symbolises good and red bad. In the middle, for 0, there is a neutral off-white yellow, to make the gradient from red to green smoother.

Regions Page

On the regions page, users will be able to switch between different regions they have previously processed, by selecting rows in the table. They will also be able to add new regions without drawing on the map by adding new rows to the table. It may not be obvious which of their regions is the one they have currently selected, so there will be a textbox at the top of the screen showing which one they have chosen.

Users will know they can add new regions by the fact that there will be a “+” button below the table, which suggests they can add a new row. When they add a new row, it will be empty, which will lead them to the fact that they can edit cells and change the factors.

Log In & Sign Up Pages

On the log in page, the user will only be presented with 2 fields to fill in, a username and a password. Typically, the username comes first, so I have put that above the password. The password textbox will have the characters hidden (each character will appear as a dot instead of itself), so that any onlookers cannot see the user’s password as they type it in.

Below both the username and password fields will be the log in button. It will be coloured in the same “call to action” colour as the predict HDI button on the main page, to suggest that this is the main function of the page. Finally, at the bottom, is a results textbox, which will tell the user if they have access, or if they have entered incorrect details.

The sign in page is very similar, with fields on top of each other, followed by a “call to action” button to create the account, followed by the result box. Except here, there will be 2 password fields to ensure double entry verification. The second one will have a slightly different label (“Re-Enter Password” as opposed to “Password”), to show the need for this field. This is also typical when creating accounts, so once the user reads this, they will understand why they must enter it again.

Help Page

On the help page, users can read about each feature in more detail. This will allow them to have a greater understanding of how to use the app, if they were confused. It will also clear up any misconceptions the user may have about what they can or cannot do. The help page will also include various images, to further get any points across, for those who prefer to learn things visually. Nothing on this page will be interactive, so the user must only be able to read or look at the pictures or diagrams to use it.

Other Usability Features

I also plan on adding emojis to various pieces of text all across the app. This can help in many different ways, such as further conveying a call to action, or describing further what a

particular section is about.

4.3.4 Analysis of how much Each Factor affects the Others

In order to see how much each factor affects the others, and how much each factor affects the HDI, we can calculate the Product Moment Correlation Coefficient, ρ , for each pair of factors. ρ is a number between -1 and 1 , which tells us how correlated 2 variables are. It is important to note that, if 2 variables are correlated, it is not the case that increasing 1 will *cause* the other to increase. It may be that (and most likely will be that) they both increase due to a 3rd variable (for example, both D (School) and A (House, School) may increase together when a new school is built, but increasing A (House, School) on its own may not increase D (School)).

If $\rho = 1$, that means the 2 variables have strong positive correlation, if $\rho = -1$, that means the 2 variables have strong negative correlation, and if $\rho = 0$, that means the 2 variables have no correlation. This follows for any values in between -1 and 1 (for example, $\rho = -0.3$ would mean the 2 variables have weak negative correlation).

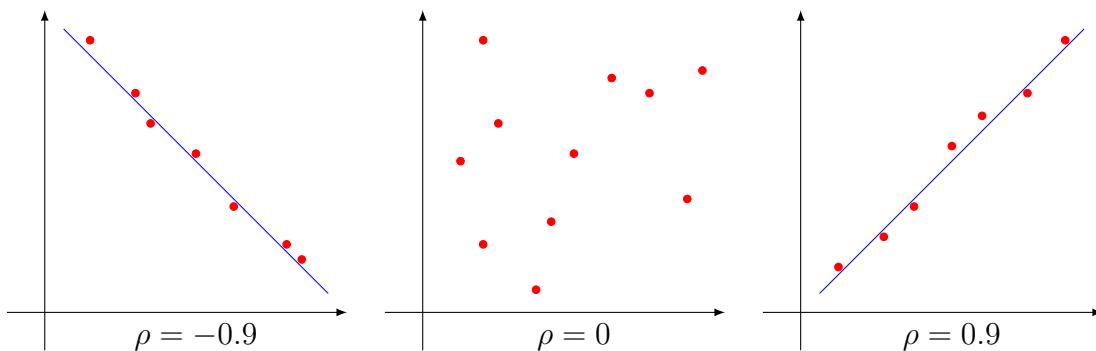


Figure 4.12: PMCC Examples

To calculate the PMCC, we can use the formula:

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}} \quad (4.1)$$

where x and y are the lists of our variables we are comparing (and, for example, x_i is the i^{th} element in that list and \bar{x} denotes the mean of that list).

Once we have the values for ρ for each of the pairs of factors, we can use a library for generating graphs and charts, such as `matplotlib` to create the heatmap. This will be done in advance, instead of each time the program is run, similar to how we generated the training data and the trained model.

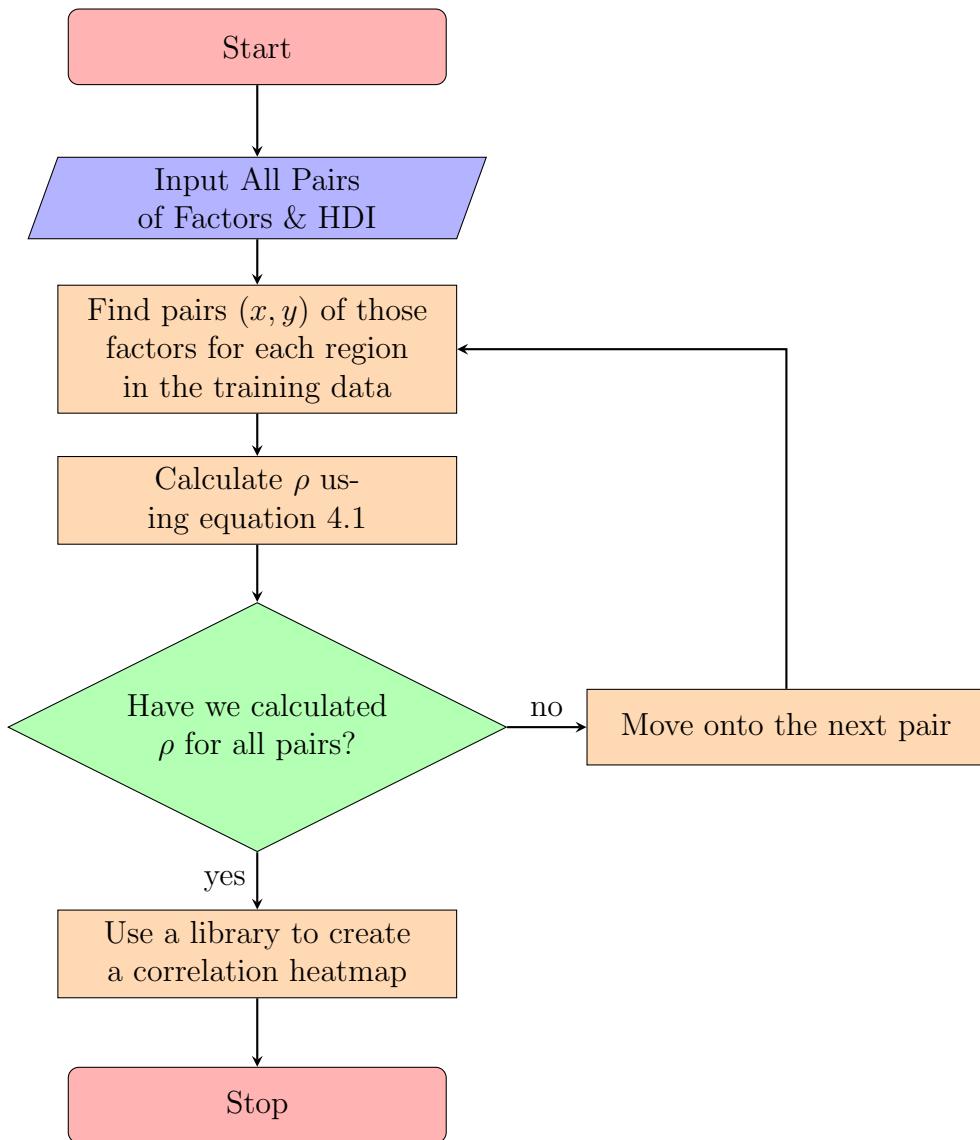


Figure 4.13: Analysis of how much Each Factor Affects the Others Flowchart

Once the image has been generated, I will add it to the gradio interface, at the bottom of the Compare tab.

4.3.5 Comparison to Similar Regions

As well as directly comparing 2 regions in the Compare tab, I would also like to show the user 1 similar region on the main page (I have already added the textbox for this output, but have not done any of the functionality yet).

In order to choose a region with a similar HDI, we should first see if there are any that have exactly the same HDI, and if there are we can randomly choose from them. If there aren't we can find whichever one is closest. If there is a tie for that, we can again choose randomly.

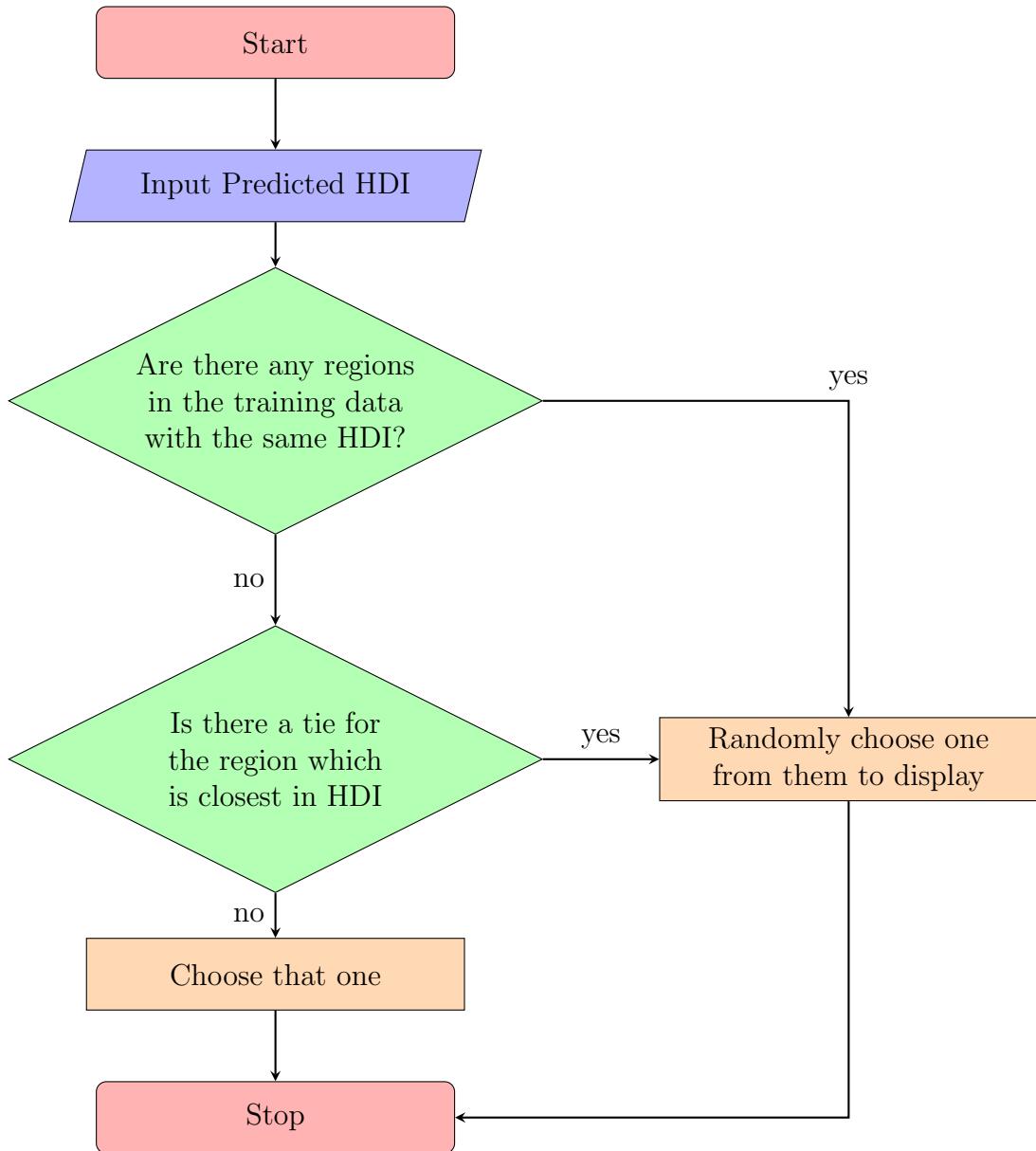


Figure 4.14: Finding Similar Regions Flowchart

4.3.6 Improved Algorithm for Suggestions

The first improvement I would like to make for suggestions is placing multiple new buildings instead of just 1. The user will be able to choose how many, with another slider (by default, I believe 5 is a good number). However, with the current algorithm for finding where to place new buildings, all 5 of them will end up in the same place. Furthermore, some positions were unable to be calculated, as they depended on the positions of buildings such as universities, which smaller regions may not have. Therefore, we need a new algorithm to find where to place a new building.

Again, placing a certain number of buildings in a region will increase the corresponding

density factor by the same amount no matter where we place them, so we should focus on improving the average distance factors. For the remainder of this explanation, I will be using A (House, School), as an example. Therefore, the problem we are trying to solve is that of placing a new school, considering the existing positions of all houses and schools.

The current algorithm for finding where to place this school to just place it in the mean position of all the houses. This does not take into account where there are schools missing, and often just puts schools in the middle of the region. We should therefore give more weight to those houses who are further from the closest school when considering the mean position.

If we therefore let the weight we give each house inversely proportional to the distance from it to the nearest school, we can say that the new position for the school should be given by:

$$\text{position} = \frac{\sum_i \frac{x_i}{\text{dist}(x_i, \text{nearest } y)}}{\sum_i \frac{1}{\text{dist}(x_i, \text{nearest } y)}} \quad (4.2)$$

where x_i is the position of the i^{th} x -object (in this case, house), nearest y is the position of the closest y -object (in this case, school), to that x -object, and $\text{dist}(a, b)$ is the distance from a to b , as given by equation 2.7.

We will therefore need to know beforehand which of the y -objects is closest to each x -object before we carry out this calculation, which we can find out in a similar way as we do in finding the average distance factors.

Again, if there are multiple factors which will have an influence, we can repeat this process and take a mean of each of the values.

This will hopefully lead to some better suggestions, but it still does not fix the issue where the new position was not able to be calculated due to the fact that there were no buildings to consider. In this case, we can place one of the required buildings in the middle of the region.

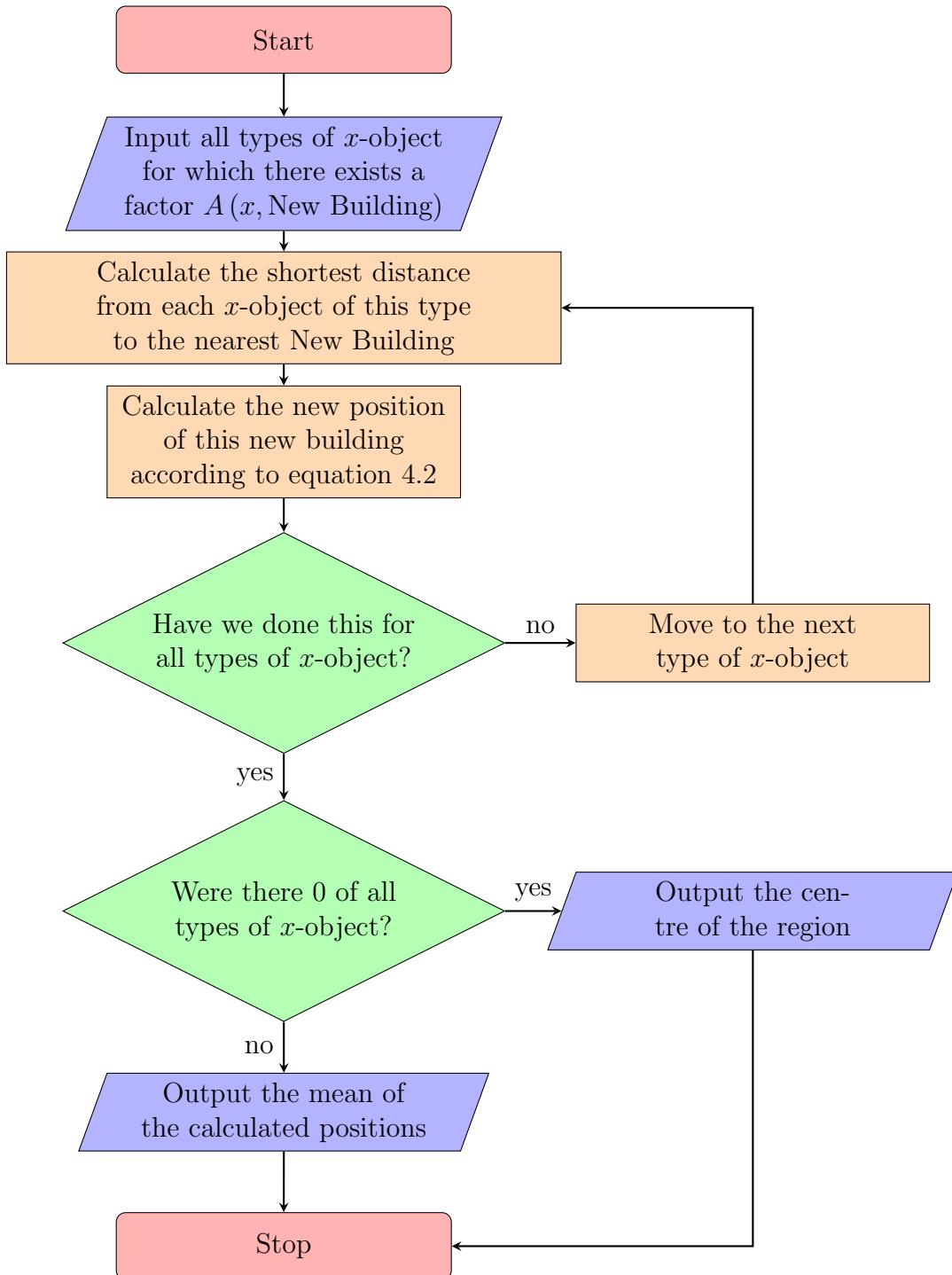


Figure 4.15: Finding a Better Optimal Place for New Building Flowchart

Now that we can calculate the position of new buildings, in order to make suggestions we must calculate the top n positions to place the new building, where n is the number of new buildings, as selected by the user, recalculate the factors, and re-predict the HDI, which is similar for what we are already doing.

4.3.7 List of Key Variables

Variables in Calculating PMCC

Identifier	Data Type	Explanation	Justification
training_data	<code>list[dict[float]]</code>	Stores the data we used to train the network, as it has all of the factors for each region	We must have many pairs of values to calculate an accurate value for ρ , and the training data had many
all_pairs	<code>list[list[str]]</code>	A <code>list</code> of all possible pairs of factors	We must iterate over this list, and calculate the PMCC for each pair, to display in the heatmap
pmcc_grid	<code>list[list[float]]</code>	A 2-dimensional <code>list</code> , which will store each of the values for ρ	This will be passed into the graphics library, to create the chart. If we print this variable, it will look similar to how the heatmap will look, but without the colours.
file_path	<code>str</code>	A path to store the image of the heatmap	Once we generate the heatmap, we must save it to an image, so that we can add it to the gradio interface

Table 4.4: List of Key Variables in Calculating PMCC

Variables in Finding Similar Regions

Identifier	Data Type	Explanation	Justification
predictedHDI	<code>float</code>	Stores the predicted HDI ad given by the neural network	We must have a value for the HDI which we are trying to match

<code>possible_choices</code>	<code>list[str]</code>	A <code>list</code> of regions, all of the same HDI, which are may either be the same as <code>predictedHDI</code> or a group that is tied for closest	We will be randomly choosing a region from this list to display
<code>similar_region</code>	<code>str</code>	The randomly chosen region from <code>possible_choices</code>	Will be returned to the “similar region” textbox in the main page

Table 4.5: List of Key Variables in Finding Similar Regions

Variables in Finding an Optimal Place for a New Building

Identifier	Data Type	Explanation	Justification
<code>new_building_type</code>	<code>str</code>	Holds the type of new building to be placed.	This is used to determine if there are any other factors which contain this type of building.
<code>other_objects</code>	<code>list[str]</code>	Holds each of the other types of object to consider.	We must iterate over this to find an optimal place for a new building.
<code>current_object</code>	<code>str</code>	Iterator variable for <code>other_objects</code>	This will be used to keep track of which object we are currently considering
<code>new_positions</code>	<code>list[tuple[float]]</code>	Holds the new calculated positions when considering each of the other objects	This is a list of latitude and longitude coordinates, so a tuple is used as before.
<code>final_position</code>	<code>tuple[float]</code>	Holds the mean position of those newly calculated positions	If there are multiple factors to consider, we will need to find a middle position to satisfy all of them

Table 4.6: List of Key Variables in Finding an Optimal Place for a New Building

Other Variables

Identifier	Data Type	Explanation	Justification
number_new_buildings	int	Stores the number of new buildings to generate when making suggestions	Building more will hopefully lead to better suggestions, but will take longer to make. Users should therefore be able to change this value with a slider.
region_to_compare	str	Stores the region the user is comparing theirs with, in the Compare page.	We must use this, to get the right row from the training data

Table 4.7: List of Other Key Variables in User Interface

4.4 Data

4.4.1 Storing Data in the MongoDB

MongoDB uses JSON format, instead of SQL. Therefore, each record can be represented with a python `dict`, with the keys being the field names. The dictionary structure I will be using will be: `{"username": str, "hashedPassword": str, "salt": str, "regions": list}`. This is because we must store the user's login information, as well as their previously processed regions.

When the user logs into an account, the previously processed regions list will be added to the table in the Regions tab. Similarly, when a new region is added in the regions tab, it will also be added in to their account in the MongoDB.

4.4.2 Testing Data for Development

I plan to have done all the tests under each milestone by the time I reach it. Once all the tests have gone successfully, across all the milestones, I will ask for my stakeholders' opinions, to help with my evaluation.

I will still be testing code in small increments as I am developing, to ensure that I do not make any silly mistakes.

Milestone 4 – Authentication System

No.	Test	Inputs	Expected Output/Justification
T_9	Creating an Account – Username	Username which does not yet exist, Username which already exists, No Username	There should be no duplicate usernames, so users should only be able to pick a username that does not exist yet. The username field must also not be empty.
T_{10}	Creating an Account – Password	Password which contains no numbers, Password which contains no special characters, Password which is less than 8 characters, Re-Entered Password does not match, Password with at least 1 number and 1 special character, with length of at least 8 characters, and double entry verified, No Password	Passwords should specify all of the requirements, to ensure security for the user, and that they have correctly chosen a password
T_{11}	Encryption Algorithm	'password123', 'Password123', 'password123'	The resulting hashed password should be very different for each of these 3 inputs, to ensure that both the salting and hashing is working as expected
T_{12}	Logging In	Username which does not exist, Incorrect Password, Existing username and Correct Password, No username or password	The user should only be granted access to the account if they have entered an existing username with the correct password

Table 4.8: Testing Data for Milestone 4

Milestone 5 – Final User Interface

No.	Test	Inputs	Expected Output/Justification
T_{13}	Calculating PMCC	The Training Data, which includes regions with full sets of factors, and some with some missing factors	If a certain region does not include both factors we are considering, that region should not be included in that PMCC calculation.
T_{14}	Finding a Similar Region	HDI in the training data, HDI not in the training data, HDI not in the training data, with a tie for closest	We must select a region with the closest HDI to the predicted one, to show the user a similar region.
T_{15}	Validating User Manually Entering Factors	Region with no name, Region with duplicate name, String in Density Factor, Non-Empty String in Average Distance Factor	When entering data manually on the regions tab, we must ensure that there are no duplicate regions, and no erroneous data.
T_{16}	Improved Suggestions	Region for which there exists 0 of each type of x -object, Region for which there exists 0 of a particular type of y -object, Region for which there exists an x -object in the same place as a y -object, Region which does not satisfy any of the above edge cases	If the region has 0 of each type of x -object, then the output should just be the centre of the region.

Table 4.9: Testing Data for Milestone 5

4.4.3 Testing Data for Post-Development

At the end of development, I will be testing for function, robustness, and usability.

Testing for Function

In testing for function, I will be checking that each feature functions as intended, when the correct inputs are carried out. Incorrect inputs will be tested as part of testing for robustness.

No.	Test	Expected Output/Justification
FT_1	User can Select an Area (S_8)	User should be able to draw their region on the map, with little restrictions. It can be any polygon that doesn't self-intersect. If the user makes a mistake, they can undo or reset.
FT_2	Neural Network Accurately Predicts HDI (S_9)	Once the user has selected their region, they should be able to predict the HDI of it. An accurate prediction will reflect the region drawn, so I will be looking for regions with less infrastructure to have a lower HDI than those with more infrastructure.
FT_3	Changes are Suggested to Increase HDI ($S_{10} \& S_{11}$)	Once the user has predicted the HDI of a region, they should also be able to make suggestions on how to improve it. The program should calculate a number of positions to put new buildings, and re-predict the HDI. I will be looking for distinct locations, as well as predictions that increase the HDI by a significant amount.
FT_4	User can Create an Account (S_{12})	Users should be able to create an account, to store their regions in. The account must have a username and password, which will be hashed. I will be looking inside the database to see if this is successful.
FT_5	User can Log Into an Account (S_{13})	Users should also be able to log into existing accounts, given they have the correct username and password. If they do, the regions in that account will be loaded into the Regions Tab.
FT_6	User can Compare a Region to one in the Training Data (S_{14})	Users should be able to compare their region to one in the training data. That is, they will see a table containing the HDI and each of the factors for their and any other region.
FT_7	User can See how much Each Factor affects the Others (S_{15})	Users should also be able to see a heatmap about how much each factor affects the others. This heatmap is not interactive, so there is not much testing that needs to be done here.
FT_8	Predicted HDI is Ranked among Other Countries with a Similar HDI (S_{16})	When predicting the HDI of a region, the user should be able to get a sense of what this HDI means, by comparing it to another region.

FT_9	User can Save Regions (S_{17})	Even if the user is not logged in, they should be able to save regions. Regions that the user predicts the HDI of should be saved to the Regions Tab, where they can switch between them.
FT_{10}	User can Manually Enter Factors (S_{18})	Users should also be able to manually enter factors, whether it is in a region that already exists, or in a new region. Users should also be made aware that any regions they edit like this can no longer have suggestions made on them.

Table 4.10: Final Testing Data for Function

Each final test matches with a success criterion, from S_8 to S_{18} . Note, this is not all of the success criteria, as S_1 to S_7 are to do with the backend, and S_{19} is to be determined during the evaluation.

Testing for Robustness

In testing for robustness, I will be checking that the program cannot break or crash at any point, no matter what the user does. The app should handle any invalid and erroneous inputs, and instead return a suitable message for the user to fix their error.

No.	Test	Inputs	Expected Output/Justification
RT_1	User cannot predict the HDI of no region	Press the Predict HDI Button Before Uploading Any Regions	If there was no validation, this would throw an error, as the region data would be undefined at this point. Instead, it should return a prompt to upload a region first.
RT_2	User cannot make suggestions before predicting HDI	Press the Make Suggestions Button Before Predicting HDI	This would also throw an error if there was no validation, as this time the HDI is undefined. It should instead retrun a prompt to predict the HDI first.

RT_3	User cannot upload 2 regions with the same name	Upload a region called <code>region.geojson</code> twice	Regions with the same name will cause issues down the line, in saving regions. Therefore, if the user uploads a region with the same name as an existing one, they should be prompted to change the name.
RT_4	User cannot create an account with insufficient credentials	<ol style="list-style-type: none"> 1. Empty fields – all combinations of empty fields will be attempted, with non-empty fields being filled with random characters. 2. Username which already exists – username: '<code>notlukewilliams</code>', passwords: random characters. 3. Non-Matching Passwords – username: '<code>lukewilliams</code>', password: '<code>a</code>', password re-attempt: '<code>aa</code>'. 4. Short Password – username: '<code>lukewilliams</code>', passwords: '<code>short</code>'. 5. Password with no numbers – username: '<code>lukewilliams</code>', passwords: '<code>nonumbers</code>'. 6. Password with no Special Characters – username: '<code>lukewilliams</code>', passwords: '<code>01number</code>'. 	Users must create accounts with unique usernames and secure passwords. If the user has done any of these things, they should be prompted to choose another username/password that satisfies the rules.

<i>RT₅</i>	User cannot log into account with incorrect credentials	<ol style="list-style-type: none"> 1. Empty fields – all combinations of empty fields will be attempted, with non-empty fields being filled with random characters. 2. Username which does not exist – username: 'luke', password: random characters. 3. Incorrect password – username: 'lukewilliams', password: random characters. 	Users should only be able to log into accounts if they have the correct username and password. If they do not, they should be prompted to enter the correct information.
<i>RT₆</i>	User cannot enter an invalid item into the dropdown menu	Type a region into the dropdown menu that does not exits: ' this region does not exist '	Users should only be able to compare their region with one that exists. If they try to type one that does not, no comparison will take place
<i>RT₇</i>	User cannot submit invalid data in the regions table	Enter random characters into any column that is not the name column.	When the user edits their regions in the table, the values in the table should be float, except for the average distance factors which may be empty strings, and the names which must be non-empty, unique strings.
<i>RT₈</i>	User cannot make suggestions on a region which has been edited in the regions table	Press the Make Suggestions Button after editing a region in the Regions Tab	Regions that have been edited in the regions tab no longer reflect the objects that contributed to the factors. The objects have therefore been deleted and so suggestions should not be made. Instead, a prompt to select a different region should be returned.

Table 4.11: Final Testing Data for Robustness

Testing for Usability

In testing for usability, I will be asking my stakeholders a series of questions, on how usable they believe the app is. They will be asked to rank the following features on a scale from 1 to 10. The scale having an even number of options (1 to 10) was intentional, as that way the stakeholders cannot submit a choose opinion (5 is slightly lower than neutral and 6 is slightly above neutral).

No.	Test	Expected Output/Justification
UT_1	How easy is it to predict the HDI of a region?	This is the main function of the app, and so it should be as usable as possible. However, the fact that the user has to export and upload the region will make it significantly less usable.
UT_2	How easy is it to make suggestions on how to improve a region?	This is the second function of the app, and so it should be almost as usable as the first.
UT_3	How easy is it to compare your region with another?	The user should also be able to compare their region with another one that already exists. There is little to get confused about here, so I believe this should be pretty usable.
UT_4	How easy is it to switch between regions?	If the user has uploaded multiple regions, it should be easy and intuitive to switch between them. The user should also be able to tell which region they currently have selected.
UT_5	How easy is it to manually create your own region?	If the user wished to manually define their own region, they should also do that. This is not a primary feature, and so I won't be focusing on it as much, but it should still be relatively easy to use.
UT_6	How easy is it to create an account?	The user should be made aware of the restrictions on their password, before they make one. If there is any problem with their account, they should be made aware of what it is.
UT_7	How easy is it to log into an account?	When logging in, if the user has not entered their details correctly, they should be aware of if their username is incorrect or if their password is incorrect.

UT_8	How much did the help tab improve the usability?	When I implement the Help Tab, I intend for it to be a source of information which the user can look to, if they are confused about a certain feature. It should clear up any misconceptions they may have, making it more usable.
--------	--	--

Table 4.12: Final Testing Data for Usability

After I have gotten a response for each stakeholder, I will add up the scores they gave each feature to get a total out of 40. Tests that end up getting higher scores will therefore be more usable.

5. Developing the Coded Solution of the 2nd Prototype

5.1 Milestone 4 – Authentication System

5.1.1 Success Criteria

By the end of this milestone, I hope to have a fully functioning authentication system, where the user can create an account and log in.

No.	Success Criterion	Justification
S_{12}	User can create an account	Eventually, I would like the user to be able to create an account, which will allow them to save different areas they have already predicted the HDI of.
S_{13}	User can log into an account	Once the account has been created, the user must be able to log into it, in order to make use of the features.

Table 5.1: Success Criteria for Milestone 4

To determine if I have met this success criteria, I will be using the following tests:

No.	Test	Inputs	Expected Output/Justification
T_9	Creating an Account – Username	Username which does not yet exist, Username which already exists, No Username	There should be no duplicate usernames, so users should only be able to pick a username that does not exist yet. The username field must also not be empty.

T_{10}	Creating an Account – Password	Password which contains no numbers, Password which contains no special characters, Password which is less than 8 characters, Re-Entered Password does not match, Password with at least 1 number and 1 special character, with length of at least 8 characters, and double entry verified, No Password	Passwords should specify all of the requirements, to ensure security for the user, and that they have correctly chosen a password
T_{11}	Encryption Algorithm	'password123', 'Password123', 'password123'	The resulting hashed password should be very different for each of these 3 inputs, to ensure that both the salting and hashing is working as expected
T_{12}	Logging In	Username which does not exist, Incorrect Password, Existing username and Correct Password, No username or password	The user should only be granted access to the account if they have entered an existing username with the correct password

Table 5.2: Testing Data for Milestone 4

5.1.2 Part 14 – Authentication Frontend

The first thing we must do is implement the tab structure, as shown in the interface sketches.

app.py

```

152 # structure of the UI
153 with gr.Blocks() as app:
154     with gr.Tab(label='Predict'):
155         (the entirety of the ui so far, indented)

```

Code Snippet 5.1: Embedding the current UI into a Tab

In the `gr.Blocks` environment, if we put the entirety of the user interface so far into a `gr.Tab`, everything will be incased in a tab, with a label as specified.

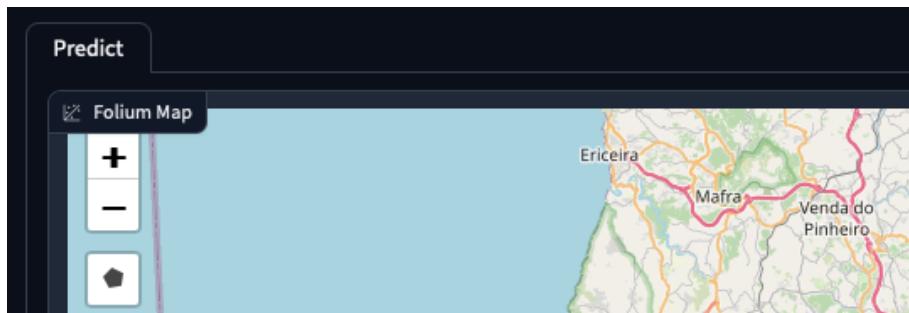


Figure 5.1: Predict Tab

Figure 5.1 shows how the tab has been added in the top left of the screen.

`app.py`

```
170     with gr.Tab(label='Log In'):
171         gr.Markdown('there\'s nothing here yet!') # placeholder text
172     with gr.Tab(label='Sign Up'):
173         gr.Markdown('there\'s nothing here yet!') # placeholder text
```

Code Snippet 5.2: Adding the Log In & Sign Up Tabs

To then add new tabs, we can add them on the same level as the first one (1 indent, inside the `gr.Blocks`), after the content of the first tab. We must specify at least one element in the tab, otherwise we will have an indentation error (the python interpreter will expect at least 1 line of code to be indented by 2, after each `with` statement). For now, I have added a `gr.Markdown` element, which will allow us to add text to the UI. I plan on using this element for the help tab, as well as any headers that may be useful.

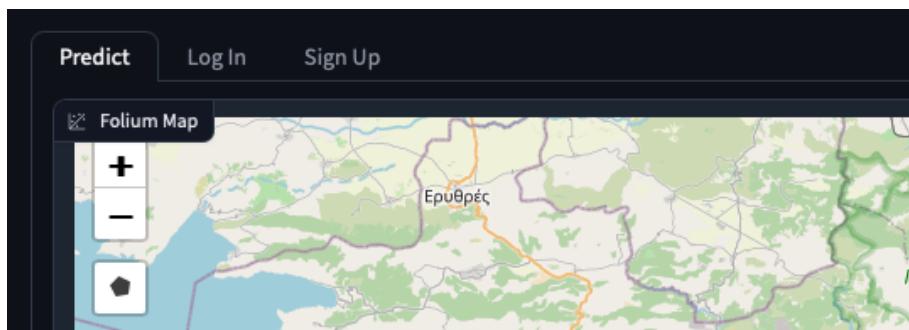


Figure 5.2: Log In & Sign Up Tabs

Figure 5.2 shows how the new tabs have been added next to the first one

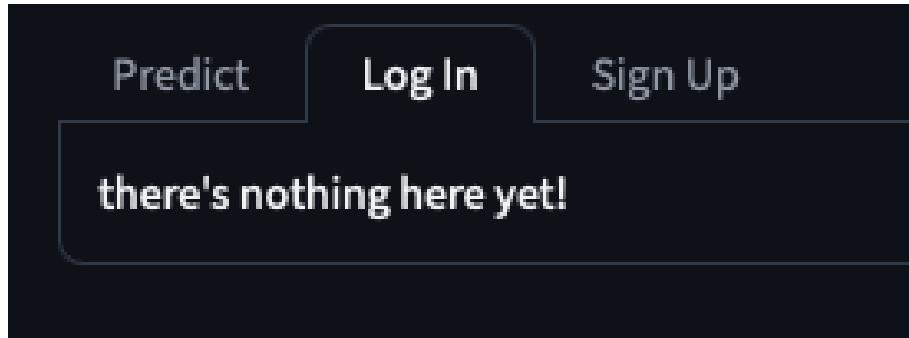


Figure 5.3: Placeholder Markdown

If we click on one of the tabs, we see the text as specified by the markdown (“there’s nothing here yet!”).

No.	Test	Input	Type	Expectation	Output
t_{62}	Tabs Appearance	–	–	Tabs appear in the top right, users can switch between them	As Expected
t_{63}	Gradio Markdown	“there’s nothing here yet!”	–	Displays the text passed into gr.Markdown()	As Expected

Table 5.3: Testing Table for Code Snippets 5.1 & 5.2

app.py

```

172     with gr.Tab(label='Sign Up'):
173         signUpUsername = gr.Textbox(label='Username', placeholder='Enter
174             ↵ your username...')
175         signUpPassword = gr.Textbox(label='Password', placeholder='Enter
176             ↵ your password...', type='password')
177         signUpPassword2 = gr.Textbox(label='Re-Enter Password',
178             ↵ placeholder='Re-Enter your password...', type='password')
179         signUpButton = gr.Button(value='Create an Account',
180             ↵ variant='primary')
181         signUpResult = gr.Textbox(label='Result', interactive=False)

```

Code Snippet 5.3: Adding the Content of the Sign Up Tab

For the sign in tab, we must add 3 textboxes, one for the username, and 2 for the password. For each textbox, we can specify a label, to add text above the field, and a

placeholder, to add temporary, grayed-out text inside the field. For the 2 password fields, we can also specify `type='password'`, which will censor whatever the user inputs, by showing each character as a dot.

Below the 3 textboxes, we must have a button for the user to press to create an account, again coloured in the primary “call to action” colour, so we can set `variant='primary'`. Finally, below the button, we must have a textbox which is the user cannot type in, to show the result of the sign up. To make this happen, we can set `interactive=False`.

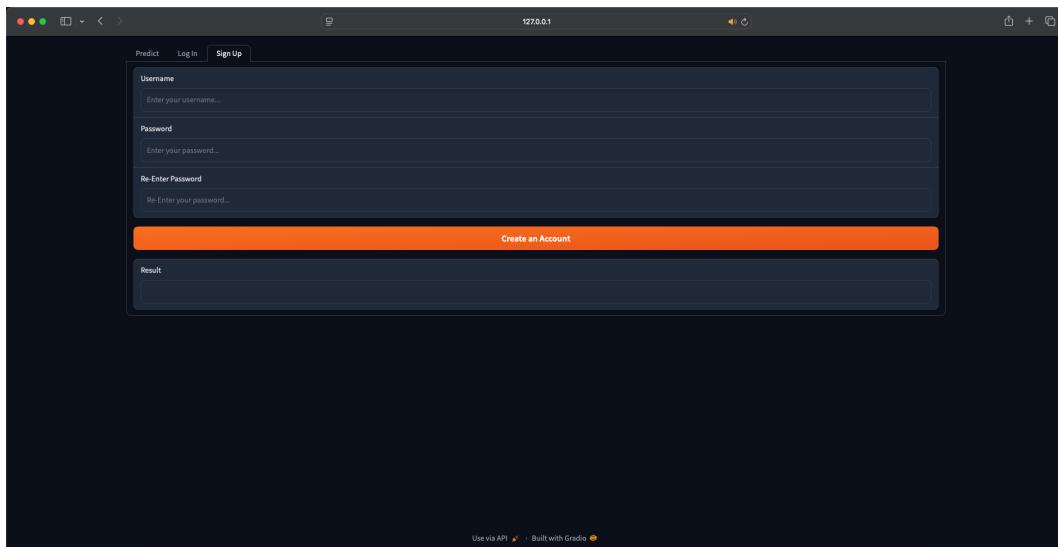


Figure 5.4: Complete Sign Up Tab

Figure 5.4 shows the complete sign up page, with all of the elements correctly positioned and labelled.

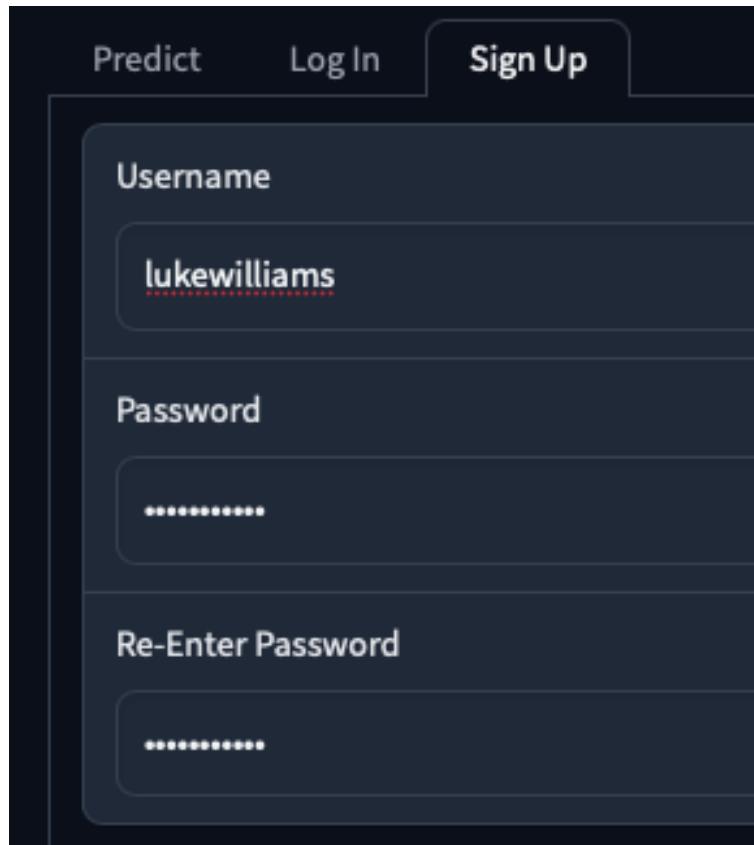


Figure 5.5: Censorship of the user's password

If we try to type in a password, we can see that the censorship also works

No.	Test	Input	Type	Expectation	Output
t_{64}	Sign Up Tab	–	–	3 Textbox inputs for the username, password, and password verification, a button, and a result textbox	As Expected
t_{65}	Password censorship	A password of length n	–	n dots	As Expected

Table 5.4: Testing Table for Code Snippet 5.3

app.py

```

170     with gr.Tab(label='Log In'):
171         logInUsername = gr.Textbox(label='Username', placeholder='Enter
172             ↵ your username...')
173         logInPassword = gr.Textbox(label='Password', placeholder='Enter
174             ↵ your password...', type='password')
175         logInButton = gr.Button(value='Log In', variant='primary')
176         logInResult = gr.Textbox(label='Result', interactive=False)

```

Code Snippet 5.4: Adding the Content of the Log In Tab

The log in page is very similar to the sign up page, except we only have 1 password field instead of 2. Once I deleted the second password textbox, all that needed doing was changing a few labels.

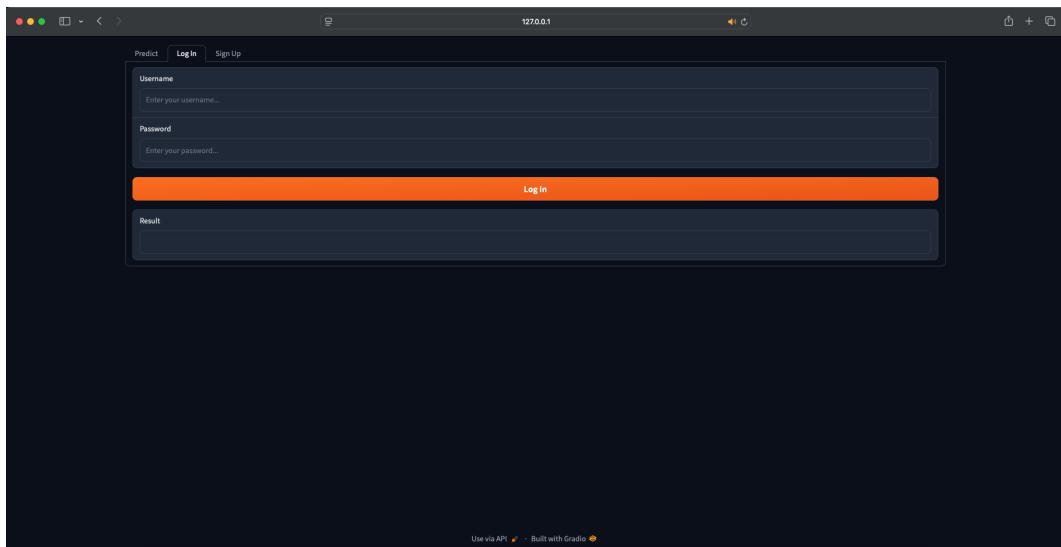


Figure 5.6: Complete Log In Tab

Figure 5.6 shows the complete log in page, with all of the elements correctly positioned and labelled.

No.	Test	Input	Type	Expectation	Output
t_{66}	Log In Tab	—	—	2 Textbox inputs for the username and password, a button, and a result textbox	As Expected

Table 5.5: Testing Table for Code Snippet 5.4

5.1.3 Part 15 – Sign Up Backend

In order to make the sign up page work, I need to create the MongoDB that will store the user's information.

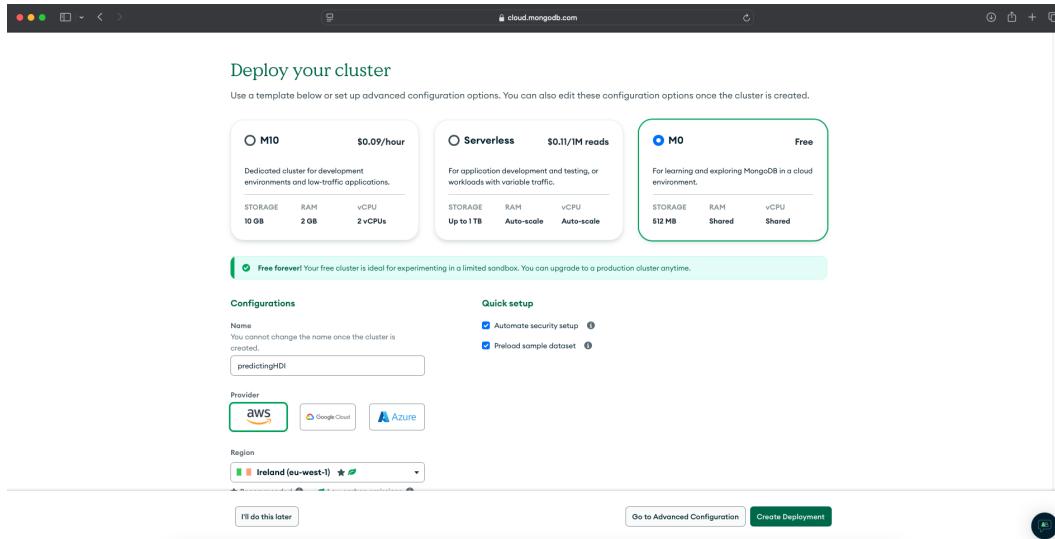


Figure 5.7: Creating the MongoDB

Here, I am creating a cluster on the MongoDB website. The free option should be enough, which allows for 512 megabytes of data. To ensure that the connection is as fast as possible, I chose the location to be Ireland, as it was the closest option to the UK.

Create a database user using a username and password. Users will be given the *read and write to any database privilege* by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password.

Username

Password

Figure 5.8: Adding a Database User (me)

In order to access the database and have read and write privileges, I had to create a database user. This information will be passed into the URI that I will use to connect to the database.

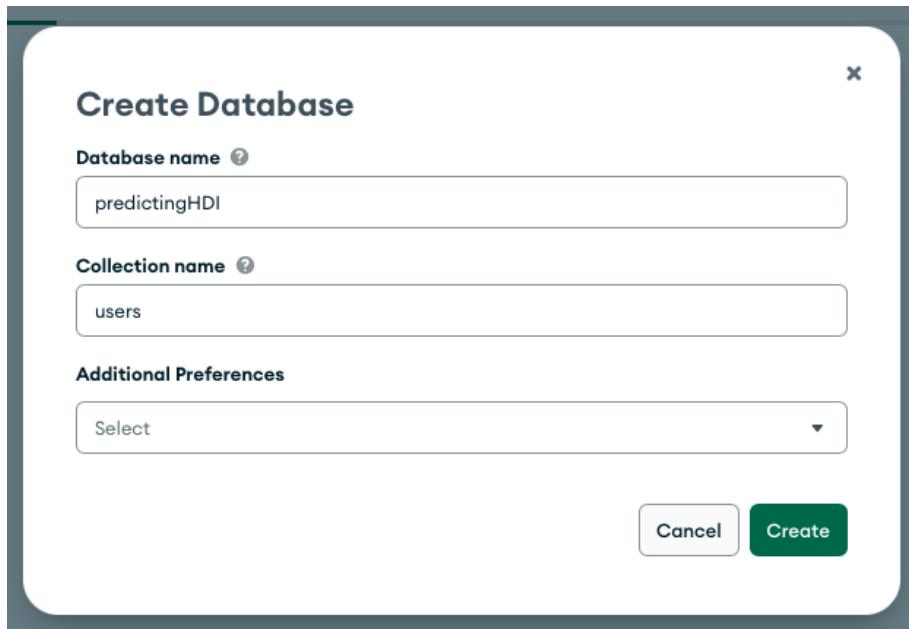


Figure 5.9: Creating a Database & Collection (table)

Once the cluster was set up, I created a database called `predictingHDI`, and a collection called `users`. Collections are similar to tables in traditional databases.

`app.py`

```
12 from pymongo.mongo_client import MongoClient
13 from pymongo.server_api import ServerApi
```

(On line 27, where I define `mongoURI`, my database password has been replaced by '`<db_password>`')

`app.py`

```
26 # connect to mongoDB
27 mongoURI =
28     "mongodb+srv://lukerhodri:<db_password>@predictinghdi.3nkqe.mongodb.net/?retryWrites=true&w=majority&appName=predictingHDI"
29 mongoClient = MongoClient(mongoURI, server_api=ServerApi('1'))
30 users = mongoClient['predictingHDI']['users']
31 # send a ping to confirm a successful connection
32 mongoClient.admin.command('ping')
33 print("Pinged your deployment. You successfully connected to MongoDB!")
```

Code Snippet 5.5: Connecting to the MongoDB

Here, I am connecting to the MongoDB cluster that I just set up. First, I defined the URI to connect to, which contains my database username and password. Then, we can establish

a client and use that to find the `users` table.

To check whether or not we have successfully connected, we can use the client to send the server a ping.

```
pymongo.errors.ServerSelectionTimeoutError: predictinghdi-shard-00-01.3nkqe.mongodb.net:27017: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1018) (configured timeouts: socketTimeoutMS: 20000.0ms, connectTimeoutMS: 20000.0ms),predictinghdi-shard-00-00.3nkqe.mongodb.net:27017: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1018) (configured timeouts: socketTimeoutMS: 20000.0ms, connectTimeoutMS: 20000.0ms),predictinghdi-shard-00-02.3nkqe.mongodb.net:27017: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1018) (configured timeouts: socketTimeoutMS: 20000.0ms, connectTimeoutMS: 20000.0ms), Timeout: 30s, Topology Description: <TopologyDescription id: 67768dd1fe8ec0f4cd6358d, topology_type: ReplicaSetNoPrimary, servers: [<ServerDescription ('predictinghdi-shard-00-00.3nkqe.mongodb.net', 27017) server_type: Unknown, rtt: None, error=AutoReconnect('predictinghdi-shard-00-00.3nkqe.mongodb.net:27017: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1018) (configured timeouts: socketTimeoutMS: 20000.0ms, connectTimeoutMS: 20000.0ms)'), <ServerDescription ('predictinghdi-shard-00-01.3nkqe.mongodb.net', 27017: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1018) (configured timeouts: socketTimeoutMS: 20000.0ms, connectTimeoutMS: 20000.0ms)'), <ServerDescription ('predictinghdi-shard-00-02.3nkqe.mongodb.net', 27017) server_type: Unknown, rtt: None, error=AutoReconnect('predictinghdi-shard-00-02.3nkqe.mongodb.net:27017: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1018) (configured timeouts: socketTimeoutMS: 20000.0ms, connectTimeoutMS: 20000.0ms)')>]
```

Figure 5.10: SSL Certificate Error

No.	Test	Input	Type	Expectation	Output
t_{67}	Connect to MongoDB	Ping the Sever	—	No Errors	SSL Certificate Error

Table 5.6: Testing Table for Code Snippet 5.5

Unfortunately, the ping was unsuccessful, and returned an SSL Certificate Error. After researching the problem, I found a StackOverflow solution [11] on this problem.

`app.py`

```
14 import certifi
```

(This code has been adapted from this StackOverflow solution: [11])

`app.py`

```
29 mongoClient = MongoClient(mongoURI, server_api=ServerApi('1'),  
    ↪ tlsCAFile=certifi.where())
```

Code Snippet 5.6: Fixing the SSL Certificate Error

It turns out I had to pass in this extra parameter, which also meant I had to import an additional module, `certifi`

```
Pinged your deployment. You successfully connected to MongoDB!  
Running on local URL: http://127.0.0.1:7860
```

Figure 5.11: Successful Connection to MongoDB

No.	Test	Input	Type	Expectation	Output
t_{67}	Connect to MongoDB	Ping the Sever	–	No Errors	As Expected

Table 5.7: Testing Table for Code Snippet 5.6

Now that I could connect to the MongoDB, it was time to make the sign up button work

app.py

```

142 # validate sign up & add account to mongoDB
143 def signUp(username, password, password2):
144     # validate username
145     # validate password
146     # add to mongoDB
147     return f'Successfully created an account, with username: {username}!'

```

app.py

```

205     signUpButton.click(signUp, inputs=[signUpUsername, signUpPassword,
→     signUpPassword2], outputs=[signUpResult])

```

Code Snippet 5.7: Adding the `def signUp` function

Here, I am adding a gradio functionality to the sign up button, where we pass in the 3 textboxes, and output to the result textbox. At the end of the function, if everything goes well, we will return that messsgage (obviously however, it is not true yet).

No.	Test	Input	Type	Expectation	Output
t_{68}	Sign Up Button	–	–	Success Message returned to result textbox	As Expected

Table 5.8: Testing Table for Code Snippet 5.7

app.py / `def signUp`

```

144 # validate no empty fields
145 if username == '' or password == '' or password2 == '':
146     return 'Please fill in all fields!'

```

Code Snippet 5.8: Ensuring no Empty Fields

The first thing we must do when validating this input, is a presence check on all 3 fields. If any of them are empty, then the user shouldn't be able to log in. If that is the case, we return a suitable message, skipping the rest of the function.

```
app.py / def signUp

147     # validate username - does not already exist
148     if users.count_documents({'username': username}) != 0:
149         return 'This username is already in use! Please choose another
          ↵ username.'
```

Code Snippet 5.9: Validating the Username

We must also check that the username does not already exist. We can do this by counting the documents (similar to records) in the `users` collection, where the '`'username'`' is equal to the username that the user typed. If there are not 0 of them, then that means that someone has already claimed an account with this name, and so we should return another suitable message.

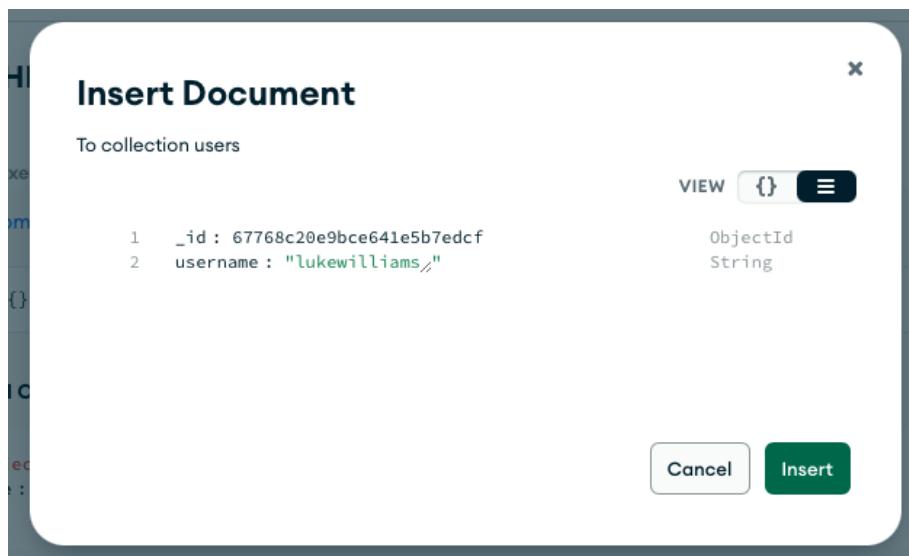


Figure 5.12: Creating a Temporary, Fake Account

In order to test that these validations are working, we must first add a temporary account to the MongoDB. This can be done directly on the MongoDB website, where we can add a document with an `_id` (similar to primary key) which is automatically generated, and a `username`.

The figure consists of three screenshots of a web-based sign-up form, labeled (a), (b), and (c). Each screenshot shows a dark-themed interface with a navigation bar at the top containing 'Predict', 'Log In', and 'Sign Up' buttons. Below the navigation bar are four input fields: 'Username', 'Password', 'Re-Enter Password', and a large orange 'Submit' button. A 'Result' section below the buttons displays validation messages.

- (a) Existing Username:** The 'Username' field contains 'lukewilliams'. The 'Result' section shows the message: "This username is already in use! Please choose another username."
- (b) Non-Existing Username:** The 'Username' field contains 'notlukewilliams'. The 'Result' section shows the message: "Successfully created an account, with username: notlukewilliams!"
- (c) No Username:** The 'Username' field is empty, showing the placeholder text "Enter your username...". The 'Result' section shows the message: "Please fill in all fields!"

Figure 5.13: Validating Username Input

No.	Test	Input	Type	Expectation	Output
$T_{9.1}$	Username Validation	Username which already exists (lukewilliams)	Invalid	Prompt to choose a username which doesn't exist	As Expected

$T_{9.2}$	Username Validation	Username which does not already exist (notluke williams)	Valid	Success Message	As Expected
$T_{9.3}$	Username Validation	No Username	Invalid	Prompt to enter a username	As Expected

Table 5.9: T_9 Testing Table (for Code Snippets 5.8 & 5.9)

Now we must validate the password.

```
app.py / def signUp

150     # validate password - double entry validation
151     if password != password2:
152         return 'The passwords do not match! Please ensure that you have
        ↵ entered the correct password.'
```

Code Snippet 5.10: Validating the Password – Double Entry Validation

Firstly, we must check if the 2 passwords they have entered are the same, for double entry validation. If they are not the same, they must have made a mistake when typing, and so we should return a suitable message.

```
app.py / def signUp

153     # validate password - at least 8 characters long
154     if len(password) < 8:
155         return 'Your password must have at least 8 characters!'
```

Code Snippet 5.11: Validating the Password – Length

Next, we can do a length check on the password to see if it is at least 8 characters long. If it is not, we should again return a suitable message.

```
app.py / def signUp
```

```
156     # validate password - at least 1 digit
157     digits = '0123456789'
158     hasDigit = False
159     for digit in digits:
160         if digit in password:
161             hasDigit = True
162     if not hasDigit:
163         return 'Your password must include at least 1 digit!'
```

Code Snippet 5.12: Validating the Password – Digits

The password must also contain at least 1 digit. To ensure this, we can define a string of all 10 digits, and create a flag `hasDigits`. If we assume the password has no digits, we can iterate over the digits and check if it is in the password. If we find that a certain digit is in the password, we can set the flag to `True`. Once we have checked all of the digits, we can look at the flag to see if there was a digit in the password or not. If there were no digits, we can again return a suitable message.

```
app.py / def signUp
```

```
164     # validate password - at least 1 special character
165     specialChars = '!#"#$%&\'()*+,.-./;<=>?@[\\"]^_`{|}~'
166     hasSpecialChar = False
167     for specialChar in specialChars:
168         if specialChar in password:
169             hasSpecialChar = True
170     if not hasSpecialChar:
171         return f'Your password must include at least 1 special character!
172             ↵ ({specialChars})'
```

Code Snippet 5.13: Validating the Password – Special Characters

The password must also contain at least 1 special character. To ensure this, we can do exactly the same thing as with the digits, but starting with a string of all special characters instead of a string of all digits. Note that for the single quote and backslash characters, we must use a backslash before them, to specify we want to use that character.

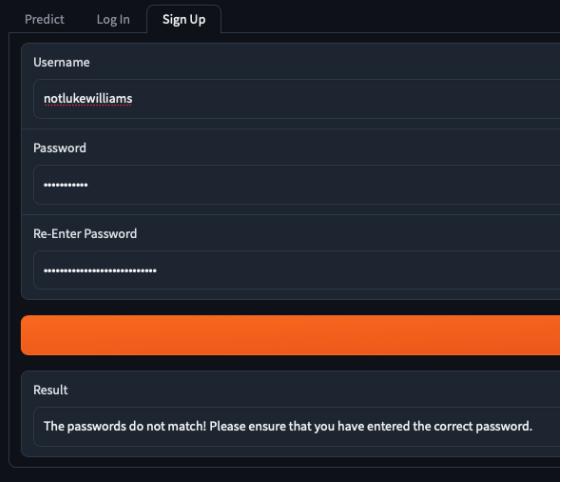
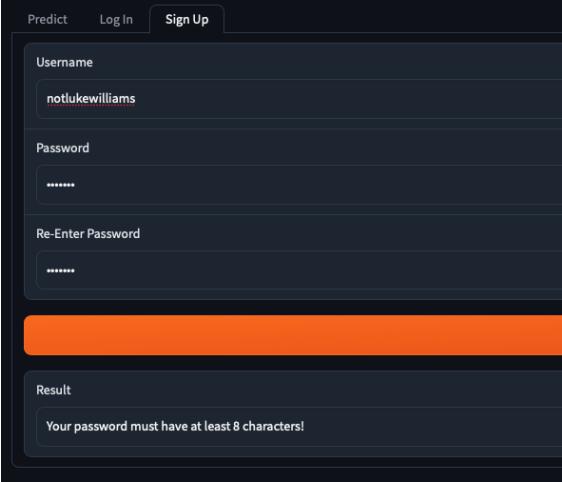
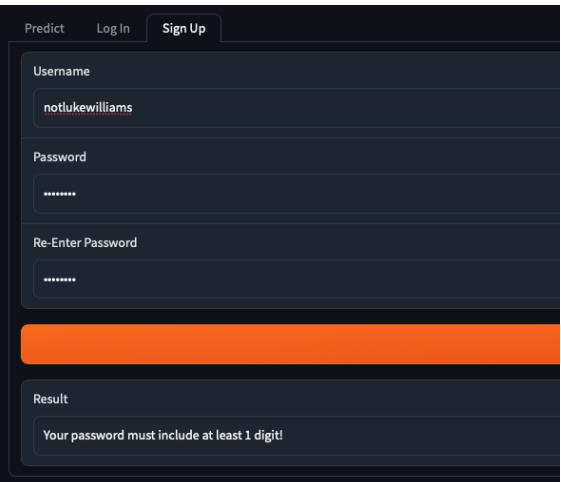
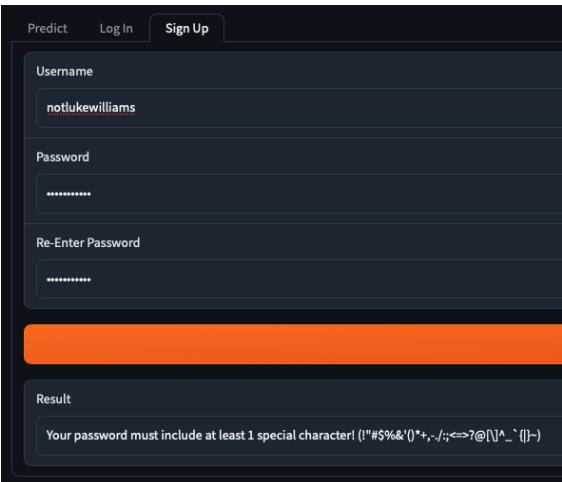
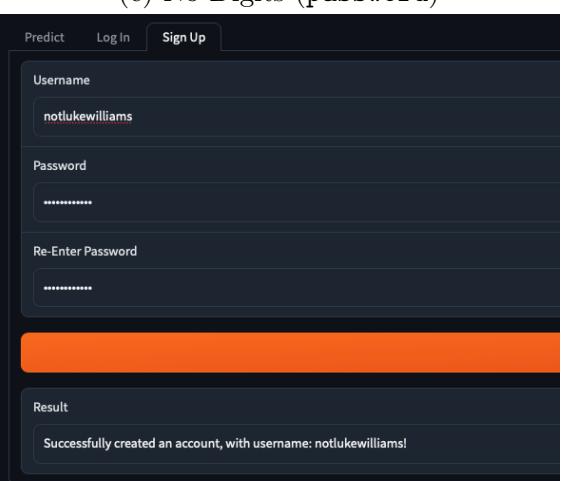
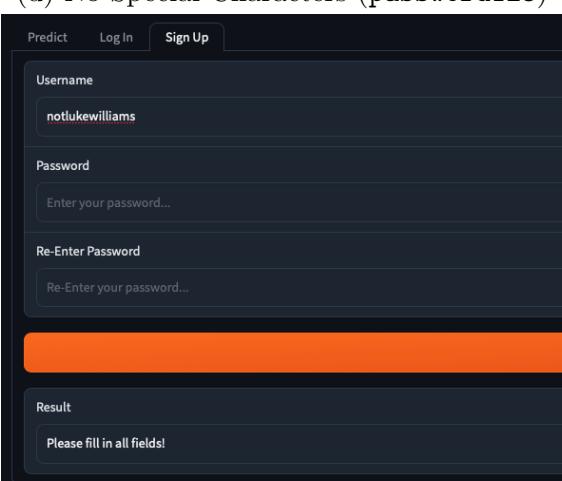
 <p>(a) Non-Matching Passwords</p>	 <p>(b) Less than 8 Characters (password)</p>
 <p>(c) No Digits (password)</p>	 <p>(d) No Special Characters (password123)</p>
 <p>(e) Valid Password (password123!)</p>	 <p>(f) No Password</p>

Figure 5.14: Validating Password Input

No.	Test	Input	Type	Expectation	Output
$T_{10.1}$	Password Validation	Passwords do not match	Invalid	Prompt to correctly enter passwords	As Expected
$T_{10.2}$	Password Validation	Password with less than 8 characters (<code>password</code>)	Invalid	Prompt to increase length of password	As Expected
$T_{10.3}$	Password Validation	Password which contains no numbers (<code>password</code>)	Invalid	Prompt to include a number	As Expected
$T_{10.4}$	Password Validation	Password which contains no special characters (<code>password123</code>)	Invalid	Prompt to include a special character	As Expected
$T_{10.5}$	Password Validation	Password with at least 1 number and 1 special character, with length of at least 8, double entry validated (<code>password123!</code>)	Valid	Success Message	As Expected
$T_{10.6}$	Password Validation	No Password	Invalid	Prompt to enter a password	As Expected

Table 5.10: T_{10} Testing Table (for Code Snippets 5.10, 5.11, 5.12 & 5.13)

Now that the user has entered correct and suitable values for their username and password, we can salt and hash it.

In the design section, I said I planned to use the `bcrypt` algorithm to do this. I didn't know much about how to implement it, so I researched the documentation about it. It turns out that the salt is saved with the hashed password, so we do not actually need to store it in a separate key in the database.

app.py

```

15 import bcrypt
app.py / def signUp

173     # salt & hash password
174     salt = bcrypt.gensalt()
175     hashed_password = bcrypt.hashpw(password.encode('utf-8'), salt)

```

Code Snippet 5.14: Salting & Hashing the Password

Here, I am importing the `bcrypt` module, which will allow us to generate the salt, and hash the password. We must first encode the password into '`'utf-8'`', as the `hashpw` function cannot take in python strings.

```
b'$2b$12$0Ur6q40ZFlnMEVouDAKQIec.W7iaB0V.l9DUQK.jgzB4.rTXXMlGW'
b'$2b$12$cnBAk6WT6gMi0vBW10hYVuiClWiZSvZOUpC4DnWosB2QD10svP/7.'
b'$2b$12$4RA31gvn.HI8bEmp3iSDoeEvUe9EInZ1hSBsV5rvkyheNkGwen8ee'
```

Figure 5.15: Hashed Password

Figure 5.15 shows the output of the hashing function for '`'password123'`', '`'Password123'`' and '`'password123'`'.

No.	Test	Input	Type	Expectation	Output
$T_{11.1}$	Encryption Algorithm	' <code>password123</code> '	—	Some Hash	As Expected
$T_{11.2}$	Encryption Algorithm	' <code>Password123</code> '	—	A Different Hash	As Expected
$T_{11.3}$	Encryption Algorithm	' <code>password123</code> '	—	Another Different Hash	As Expected

Table 5.11: T_{11} Testing Table (for Code Snippet 5.14)

Now that we have a salted and hashed password, we can add it and the username to the MongoDB.

```
app.py / def signUp
```

```
176     # add to mongoDB
177     user = {
178         'username': username,
179         'hashedPassword': hashed_password,
180         'regions': []
181     }
182     users.insert_one(user)
183     return f'Successfully created an account, with username: {username}!'
```

Code Snippet 5.15: Adding the User to MongoDB

To do this, we must define a `dict` to act as the document, with a key for the username, hashedPassword, and list of regions. For now, this list is empty, but when I implement saving regions, I will add change this to include the regions they have processed so far.

```
_id: ObjectId('6776a5028304fd0e6b8346bf')
username : "notlukewilliams"
hashedPassword : Binary.createFromBase64('3DjJDEyJGNGeTB1cjR0RVJweE9zV3cwNC95TmVMM1JJLjVuTxpWcTYzNThwZ3dLTVRZL09CTmxWeUzl', 0)
> regions : Array (empty)
```

Figure 5.16: New Account in MongoDB

If we check the database on the MongoDB website, we can see the account generated, with username: '`notlukewilliams`', a hashed password, and an empty array of regions.

No.	Test	Input	Type	Expectation	Output
t ₆₉	Insert into MongoDB	User Document	—	Username, Hashed Password, and regions saved to MongoDB	As Expected

Table 5.12: Testing Table for Code Snippet 5.15

I can now therefore delete the fake '`lukewilliams`' account, which has no hashed password and no regions array, as we now have a real account to test with.

5.1.4 Part 16 – Log In Backend

app.py

```
185 # validate & log in the user
186 def logIn(username, password):
187     pass
```

app.py

```
246 logInButton.click(logIn, inputs=[logInUsername, logInPassword],
→   outputs=[logInResult])
```

Code Snippet 5.16: Adding the `def logIn` function

Here, I am adding a gradio functionality to the log in button, where we pass in the 2 textboxes, and output to the result textbox. For now, I am passing, so that there are no indentation errors that follow.

No.	Test	Input	Type	Expectation	Output
t_{70}	Log In Button	–	–	<code>def logIn</code> function is executed (temporary print statement was added to test)	As Expected

Table 5.13: Testing Table for Code Snippet 5.16

app.py / `def logIn`

```
187 # validate no empty fields
188 if username == '' or password == '':
189     return 'Please fill in all fields!'
```

Code Snippet 5.17: Ensuring No Empty Fields in Logging In

For logging in, we must again first check if any of the fields have been left empty. If any have, we must return a suitable message, prompting the user to fill everything in.

```
app.py / def logIn

190     # check username exists
191     if users.count_documents({'username': username}) == 0:
192         return 'This username does not exist! Please ensure that you have
        ↵ entered the correct username.'
```

Code Snippet 5.18: Validating the Username's Existance

Next, we must check if the username they entered exists in the database. To do this, we can use the same `count_documents` method, but this time check if there *are* 0 instead of checking if there *aren't* 0. If there *are* 0, then they must have entered the wrong username, and so we should return a suitable message.

```
app.py / def logIn

193     # check password
194     user = users.find_one({'username': username})
195     if not bcrypt.checkpw(password.encode('utf-8'),
196         ↵ user['hashedPassword']): # the hashed passwords do not match
        return 'Incorrect Password! Please try again.'
```

Code Snippet 5.19: Checking the Password is Correct

Now we must check the password is correct. To do this, we must first retrieve the user's document from the MongoDB collection. Now that we have ensured that the username does exist, we can pass it into the `find_one` method, and be sure it won't return an error. It will instead return the dictionary we inserted into the MongoDB in the `def signIn` function.

Once we have the user's hashed password (from their dictionary), we must use the `checkpw` function from the `bcrypt` library. This is because the randomly generated salt is stored with the hashed password, and we must use `bcrypt` to retrieve it and check that it is correct. If it is not correct, we should return an error message, saying that the user has entered an incorrect password.

```
app.py / def logIn

197     # grant access to the account
198     return f'Successfully logged in as {username}!'
```

Code Snippet 5.20: Granting Access to the Account

If nothing has been returned yet, then that means the user has entered the correct information and can be granted access to the account. Currently this does nothing, as I

have not implemented saving regions yet. However, we can add the final return success message, to tell the user they have logged in.

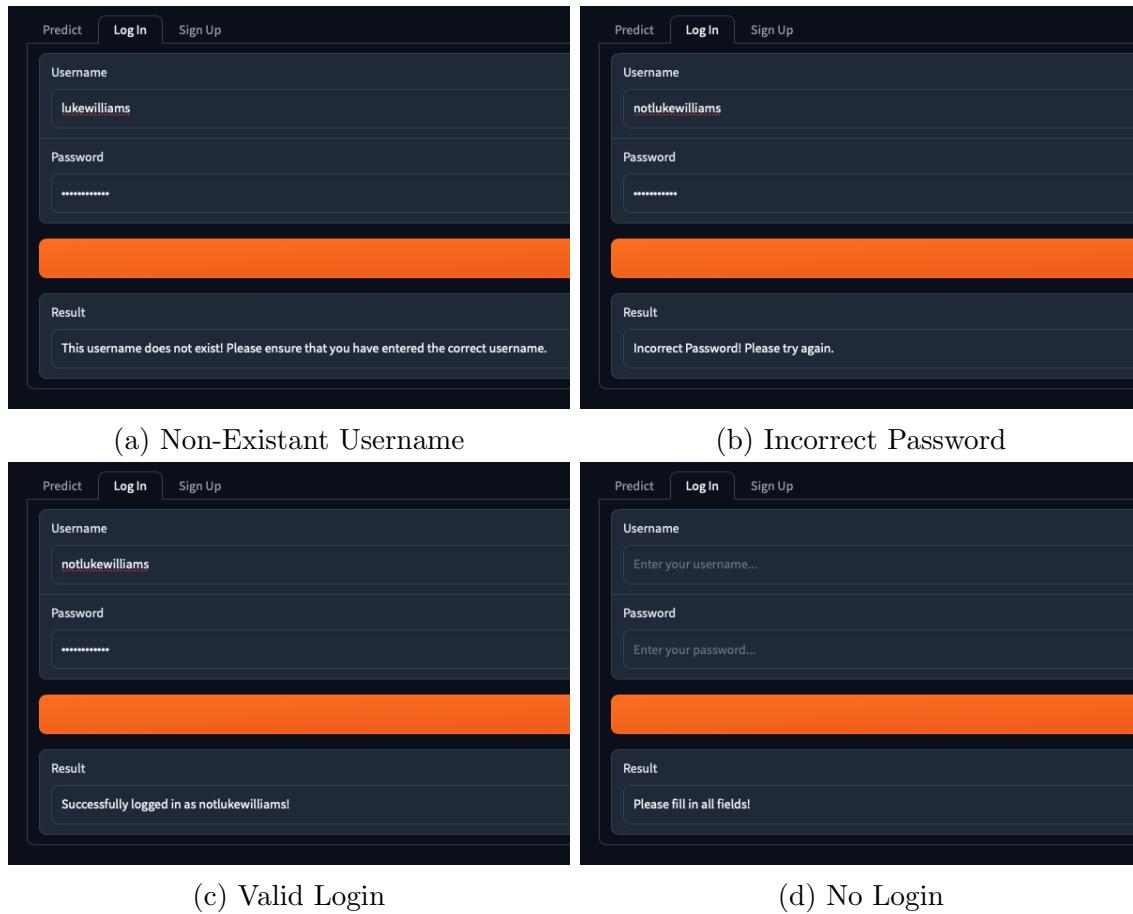


Figure 5.17: Validating Login

No.	Test	Input	Type	Expectation	Output
$T_{12.1}$	Logging In	Username which does not exist	Invalid	Prompt to enter an existing username	As Expected
$T_{12.2}$	Logging In	Incorrect Password	Invalid	Prompt to enter the correct password	As Expected
$T_{12.3}$	Logging In	Existing Username & Correct Password	Valid	Success Message	As Expected
$T_{12.4}$	Logging In	No Username or Password	Invalid	Prompt to enter a username and password	As Expected

Table 5.14: T_{12} Testing Table (for Code Snippets 5.17, 5.18, 5.19 & 5.20)

5.1.5 Review

No.	Test	Inputs	Pass / Fail
T_9	Creating an Account – Username	Username which does not yet exist, Username which already exists, No Username	Pass
T_{10}	Creating an Account – Password	Password which contains no numbers, Password which contains no special characters, Password which is less than 8 characters, Re-Entered Password does not match, Password with at least 1 number and 1 special character, with length of at least 8 characters, and double entry verified, No Password	Pass
T_{11}	Encryption Algorithm	'password123', 'Password123', 'password123'	Pass
T_{12}	Logging In	Username which does not exist, Incorrect Password, Existing username and Correct Password, No username or password	Pass

Table 5.15: Passed Testing Data for Milestone 4

All 4 of the main tests in Milestone 4 passed. As a result of that, the following success criteria has been met:

No.	Success Criterion	Justification	Success
S_{12}	User can create an account	Eventually, I would like the user to be able to create an account, which will allow them to save different areas they have already predicted the HDI of.	✓
S_{13}	User can log into an account	Once the account has been created, the user must be able to log into it, in order to make use of the features.	✓

Table 5.16: Achieved Success Criteria for Milestone 4

I also have a fully functioning authentication system, where the user can create and log into an account, so overall, I believe this milestone has been a success.

5.2 Milestone 5 – Final User Interface

5.2.1 Success Criteria

By the end of this milestone, I hope to have a full user interface, with all of the desired features.

No.	Success Criterion	Justification
S_{14}	User can compare a region to one in the training data	This will allow users to make a judgement on how certain regions have the HDI that they have, and how they can improve their own.
S_{15}	User can see how each factor can affect the others	This will allow the user to consider the secondary effects of their decisions, as the suggested changes may not only increase the HDI, but also some of the other factors.
S_{16}	Predicted HDI is ranked among other countries with a similar HDI	This will allow users who are not that familiar with HDI to get an idea of the HDI scale by comparing it to the gist of what they know.
S_{17}	User can save regions	This will allow users to save time and not re-download the buildings & re-calculate the factors. Users with an account should also be able to store regions there.
S_{18}	User can manually enter factors	If downloading & calculating the factors takes too long, the user may manually enter them instead.

Table 5.17: Success Criteria for Milestone 5

To determine if I have met this success criteria, I will be using the following tests:

No.	Test	Inputs	Expected Output/Justification
T_{13}	Calculating PMCC	The Training Data, which includes regions with full sets of factors, and some with some missing factors	If a certain region does not include both factors we are considering, that region should not be included in that PMCC calculation.
T_{14}	Finding a Similar Region	HDI in the training data, HDI not in the training data, HDI not in the training data, with a tie for closest	We must select a region with the closest HDI to the predicted one, to show the user a similar region.
T_{15}	Validating User Manually Entering Factors	Region with no name, Region with duplicate name, String in Density Factor, Non-Empty String in Average Distance Factor	When entering data manually on the regions tab, we must ensure that there are no duplicate regions, and no erroneous data.
T_{16}	Improved Suggestions	Region for which there exists 0 of each type of x -object, Region for which there exists 0 of a particular type of y -object, Region for which there exists an x -object in the same place as a y -object, Region which does not satisfy any of the above edge cases	If the region has 0 of each type of x -object, then the output should just be the centre of the region.

Table 5.18: Testing Data for Milestone 5

5.2.2 Part 17 – Comparing a Region to Training Data

The first thing I would like to do is implement the compare tab, where the user can select a region to compare theirs to.

app.py

```

239     with gr.Tab(label='Compare'):
240         compareDropdown = gr.Dropdown(choices=['a', 'b', 'c'],
241             label='Select a Region')
242         compareTable = gr.DataFrame(label='Comparison', headers=['Factor',
243             'Your Region', 'Selected Region'], type='array',
244             interactive=False)

```

Code Snippet 5.21: Adding the Compare Tab

Here, I am adding the gradio elements that will go inside the compare tab. The first, is a dropdown menu, which prompts the user to select a region. For now, the options are just '`a`', '`b`' and '`c`', to show how the element works. Below this, we have another DataFrame, which will act as our output table. Again, we must specify that this is not `interactive`, with type '`array`', and the headers for the table.

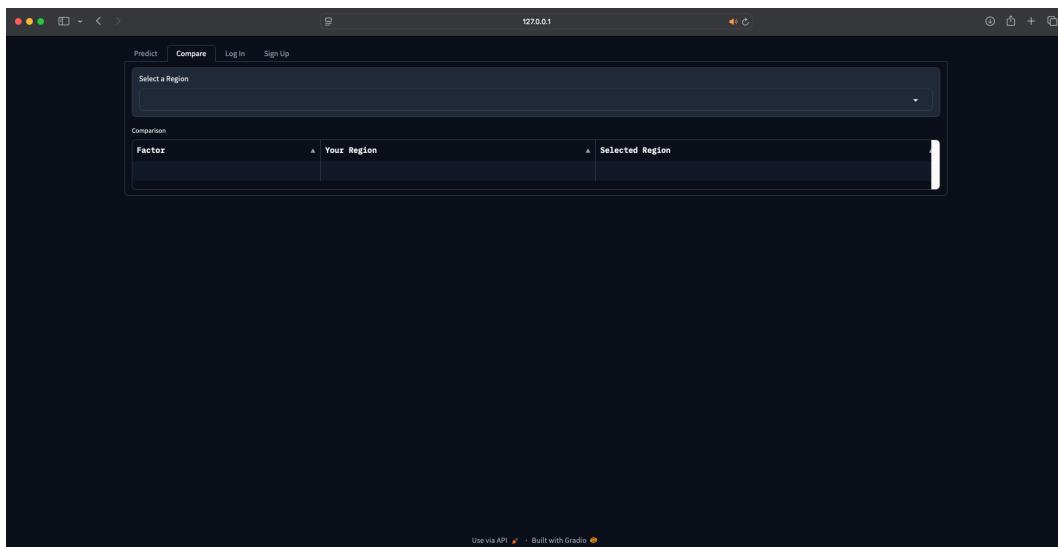


Figure 5.18: Compare Tab Interface

Figure 5.18 shows the structure of the Compare Tab. Currently, this is looking pretty empty, but when the table is filled, and the PMCC analysis is added below, the page will look more complete.

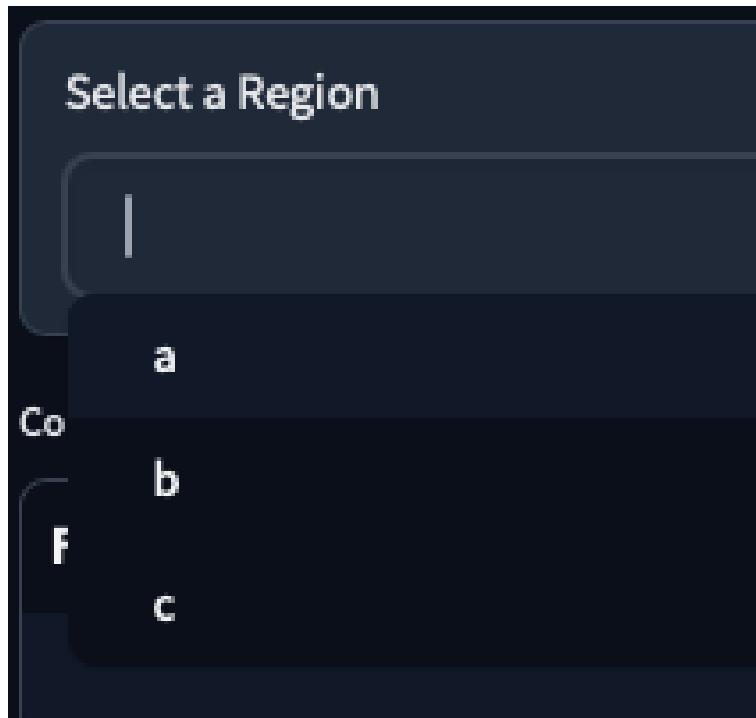


Figure 5.19: Example Options for Dropdown

Figure 5.19 shows what happens when you select the dropdown, which is that the options we specified appear.

No.	Test	Input	Type	Expectation	Output
t_{71}	Compare Tab	-	-	Dropdown Menu & Output Table	As Expected

Table 5.19: Testing Table for Code Snippet 5.21

To make the dropdown menu have the right options (the name of each region), we must first import the training data.

app.py

```
16 import csv
```

app.py

```
37 # load the training data
38 with open('data/training_data.csv', 'r') as file:
39     reader = csv.DictReader(file)
40     trainingData = []
41     for record in reader:
42         trainingData.append(record)
```

Code Snippet 5.22: Importing the Training Data

Here, I am reading the `csv` file of the training data in the same way as before, by defining a `DictReader`, iterating over the records, and appending each one to a list, `trainingData`.

app.py

```
248 compareDropdown = gr.Dropdown(choices=[f'{region["region"]}', 
    ↴ {region["country"]} for region in trainingData],
    ↴ label='Select a Region')
```

Code Snippet 5.23: Adding the Regions to the Dropdown

To construct the list of choices, we can iterate over the `trainingData` list, and format a string to have the name of the region, followed by the name of the country. This is so that users can search for regions by country in addition to region name.

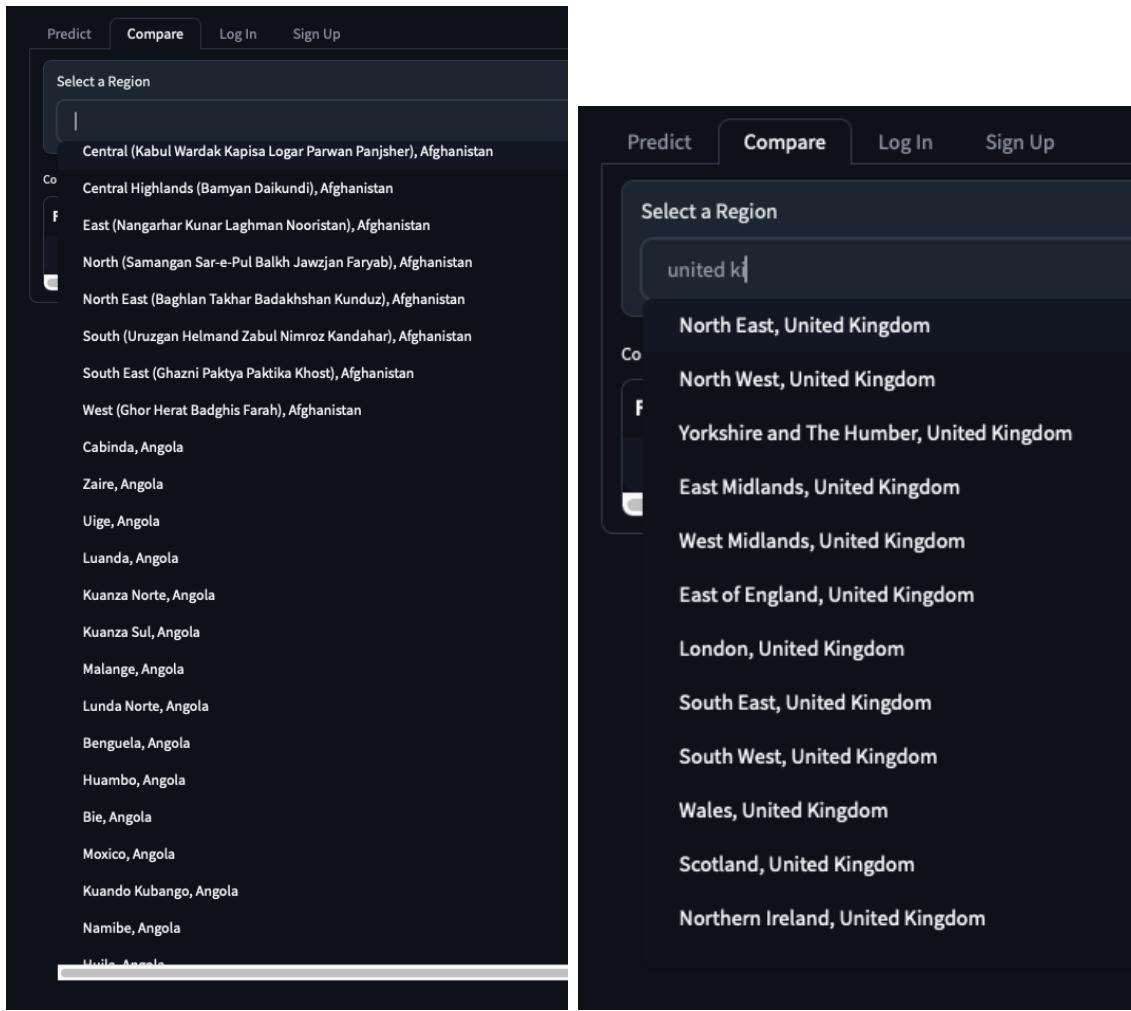


Figure 5.20: All Dropdown Options

Figure 5.20 shows the first few items of the full list, and how typing can narrow down the options.

No.	Test	Input	Type	Expectation	Output
t_{72}	Select a Region Dropdwon	—	—	Contains all regions from training data	As Expected
t_{73}	Searching for a Region	Beginning to type a name	—	Narrows down options include string	As Expected

Table 5.20: Testing Table for Code Snippets 5.22 & 5.23

Now we must add the functionality for this page.

app.py

```
25 allFactors = []

app.py / def processPrediction

58     global allFactors
```

Code Snippet 5.24: Making `allFactors` a `global` variable

Firstly, we must be able to access the factors we calculated for the user's region. To do this, I will be using a global variable. In the `def processPrediction` function, we already have a variable called `allFactors`, which stores the factors that are calculated. We can therefore define it outside the function, and make the one inside the function global.

There is also a variable called `allFactors` in the `def makeSuggestions` function, which stores the re-calculated factors for each suggestion. This must not be made global, as if it is, it will override the true `allFactors` list.

app.py

```
153 # user selects an item from the dropdown menu to compare to
154 def compareRegions(regionName):
155     # confirm the user has predicted
156     if allFactors == []:
157         return [['Please predict a region first!', None, None]]
```

app.py

```
275     compareDropdown.input(compareRegions, inputs=[compareDropdown],
→     outputs=[compareTable])
```

Code Snippet 5.25: Adding the Functionality to the Compare Tab

Here, I am defining the `def compareRegions` function, which will be executed when the value dropdown menu changes (which is when the user chooses one of the options).

In this function, we must first validate that the user has in fact got a region to compare, and if not, we can return a message to prompt them to make a prediction first. The return is in this format, because we are returning to a DataFrame, instead of a textbox.

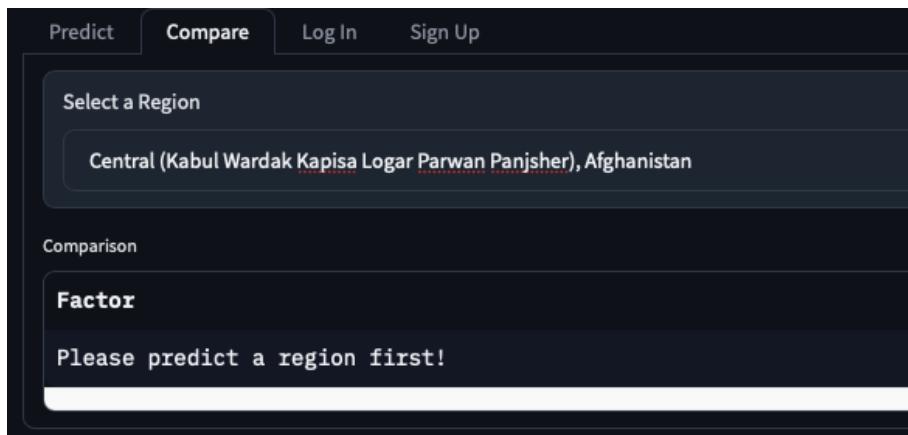


Figure 5.21: Output when the User has not Predicted Yet

No.	Test	Input	Type	Expectation	Output
t_{74}	User tries to compare before Predicting	–	–	Prompt to make a prediction first	As Expected

Table 5.21: Testing Table for Code Snippets 5.24 & 5.25

What is passed into the function is the text in the dropdown, which is a string containing both the region name and the country. We must therefore un-concatenate this string to get the country and region back.

```
app.py / def compareRegions
158     # undo the concatenation
159     split = regionName.split(' ', ' ')
160     country = split[-1]
161     region = ' '.join(split[:-1])
```

Code Snippet 5.26: Undoing the Concatenation

As no countries contain the substring: ' ', ' ', we can split the name at every ' ', ' ' and take the last element to be the country name. Everything else before that must have been the region name, so we must join those elements together, putting back the ' ', ' ' in between.

```
app.py / def compareRegions
```

```
162     # find the region record & get lists of factors
163     regionRecord = [record for record in trainingData if record['country']
164         == country and record['region'] == region][0]
165     yourFactors = ['undefined' if factor == None else round(float(factor),
166         5) for factor in allFactors]
167     selectedFactors = ['undefined' if factor == '' else
168         round(float(factor), 5) for factor in
169         list(regionRecord.values())[4:]]
```

Code Snippet 5.27: Getting Lists of Factors

Next, we must find the correct record from the `csv` file, which can easily be done by finding the record who's region and country matches with what we just found. From this, we can get a list of all factors, similar to the `allFactors` list. Except here, we must replace empty ones with '`'undefined'`' to make it more readable for the user.

app.py / def compareRegions

```

166     # construct the return list
167     factorNames = ['Average Distance from House to nearest School
168                     (km)', 'Average Distance from House to nearest Hospital
169                     (km)', 'Average Distance from House to nearest Pharmacy
170                     (km)', 'Average Distance from House to nearest Restaurant
171                     (km)', 'Average Distance from School to nearest Hospital
172                     (km)', 'Average Distance from Police Station to nearest Hospital
173                     (km)', 'Average Distance from House to nearest Place of Worship
174                     (km)', 'Average Distance from Bank to nearest Slot Machine
175                     (km)', 'Average Distance from Fast-Food Place to nearest Toilet
176                     (km)', 'Average Distance from House to nearest Police Station
177                     (km)', 'Average Distance from University to nearest Library
178                     (km)', 'Average Distance from House to nearest Library
179                     (km)', 'Number of Schools per km2', 'Number of Hospitals per
180                     km2', 'Number of Pharmcies per km2', 'Number of Police Stations per
181                     km2', 'Number of Libraries per km2', 'Number of Toilets per
182                     km2', 'Number of Restaurants per km2', 'Number of Places of Worship
183                     per km2', 'Number of Post Boxes per km2', 'Number of Vending
184                     Machines per km2', 'Number of Benches per km2', 'Number of Trees per
185                     km2']
186
187     returnList = [['HDI', currentHDI, regionRecord['hdi']]]
188     for num, factor in enumerate(factorNames):
189         returnList.append([factor, yourFactors[num],
190                           selectedFactors[num]])
191
192     return returnList

```

Code Snippet 5.28: Returning the 2D List of Factors

To then create the list for the table, we can create a list of the factor names, not in terms of A and D , to make sure the user understands what is meant by each row. After this, we can iterate over each factor name and get the corresponding values from the user's region and their selected region.

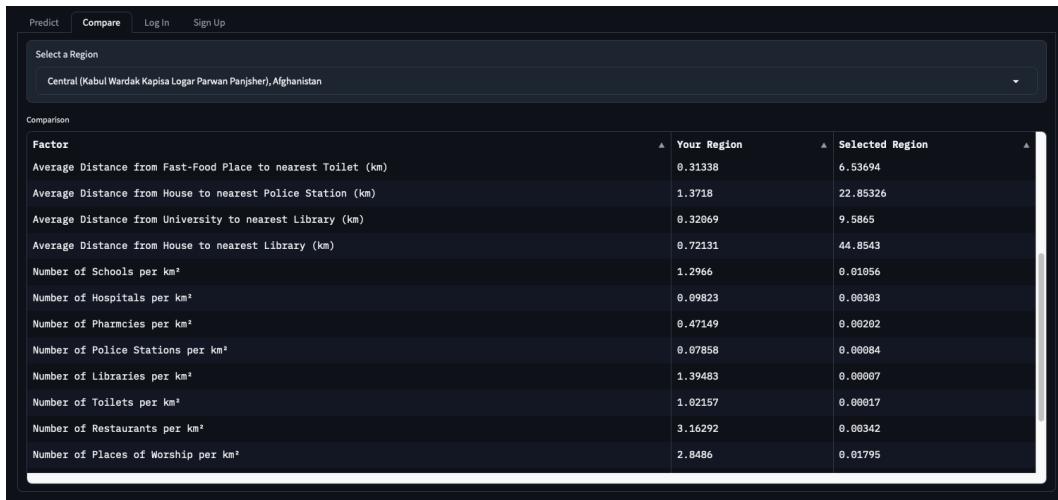


Figure 5.22: Full Comparison between User's and Selected Regions

No.	Test	Input	Type	Expectation	Output
t_{75}	Comparison to Another Region	Region Name	—	Table showing HDI and Factors for both regions	As Expected

Table 5.22: Testing Table for Code Snippets 5.26, 5.27 & 5.28

5.2.3 Part 18 – Analysis of how much Each Factor affects the Others

data_processing / pmcc.py

```

1 import csv
2
3 # load the training data
4 with open('data/training_data.csv', 'r') as file:
5     reader = csv.DictReader(file)
6     trainingData = []
7     for record in reader:
8         trainingData.append(record)

```

Code Snippet 5.29: Reading the Training Data

First, we must again retrieve the training data in the same way as we have done before.

data_processing / pmcc.py

```

10 # iterate over every pair of factors
11 factors = list(trainingData[0].keys())[3:]
12 pmcc_grid = []
13 for factor1 in factors:
14     pmcc_row = []
15     for factor2 in factors:
16         print(f'calculating pmcc of {factor1} and {factor2}...') # log
17         ↪ message
18         x_values = [record[factor1] for record in trainingData]
19         y_values = [record[factor2] for record in trainingData]
```

Code Snippet 5.30: Iterating over All the Factors Twice

In order to get every possible pair of factors, we must iterate over each of them twice. Here, I am using a nested `for` loop, to ensure that I cover all possible pairs. This will include factors paired with themselves, as well as double counting.

Once we have the 2 factors, we can retrieve all of the data points from the training data list. At the beginning of these `for` loops, we must also define variables to store the calculated values for ρ .

No.	Test	Input	Type	Expectation	Output
t_{76}	Retrieving Data Points for PMCC Calculation	Training Data	—	<code>csv</code> is read, and we iterate over each pair of factors	As Expected

Table 5.23: Testing Table for Code Snippets 5.29 & 5.30

However, some of these data points will be empty, as a result of some average distances being undefined. If a certain pair of values contains an empty string, it should be removed. Otherwise, both values may be converted into floats, ready to be used in calculations.

data_processing / pmcc.py

```

19      # remove pairs with empty values, iterating over the lists in
20      ↵ reverse as to not disturb the index numbers
21      for index in range(len(x_values)-1, -1, -1):
22          if x_values[index] == '' or y_values[index] == '':
23              x_values.pop(index)
24              y_values.pop(index)
25      # convert to float
26      x_values = [float(x) for x in x_values]
27      y_values = [float(y) for y in y_values]

```

Code Snippet 5.31: Ensuring Complete Pairs

Here, I am iterating an index over the length of the `x_values` list, and checking if either of the lists at that index are empty. If at least one of them is, then we must remove both the empty value, and the corresponding value in the other list. However, removing a value like this will cause the index number to no longer be synchronised with the length of the lists, and so we will run into an index out of range error. To get around this, we can iterate over the lists in reverse, so that removing elements will not change the indexes of the elements we have yet to search.

Once all of the empty strings have been removed, we can iterate over each of the lists and convert them to floats.

No.	Test	Input	Type	Expectation	Output
$T_{13.1}$	Discarding Empty Strings in PMCC Calculation	Non-empty x and Non-empty y	Valid	Not Removed	As Expected
$T_{13.2}$	Discarding Empty Strings in PMCC Calculation	Non-empty x and Empty y	Invalid	Removed	As Expected
$T_{13.3}$	Discarding Empty Strings in PMCC Calculation	Empty x and Non-empty y	Invalid	Removed	As Expected
$T_{13.4}$	Discarding Empty Strings in PMCC Calculation	Empty x and Empty y	Invalid	Removed	As Expected

Table 5.24: T_{13} Testing Table (for Code Snippet 5.31)

data_processing / pmcc.py

```
2 import numpy as np
```

data_processing / pmcc.py

```
28     # calculate pmcc and add it to grid
29     pmcc = sum([(x_values[index]-np.mean(x_values))*(y_values[index]
30             -np.mean(y_values)) for index in range(len(x_values))])/
31             np.sqrt((sum([(x_values[index]-np.mean(x_values))**2 for index
32             in range(len(x_values))]))*(sum([(y_values[index]
33             -np.mean(y_values))**2 for index in range(len(x_values))])))
34     pmcc_row.append(pmcc)
35     pmcc_grid.append(pmcc_row)
```

Code Snippet 5.32: Calculating the PMCC

Once we have a list of valid pairs, we can calculate the PMCC, using equation 4.1 (for which we need to import `numpy`). Once the PMCC has been calculated, we can add it to the row, and once the full row has been completed (the `for` loop has finished), we can add the row to the full grid.

No.	Test	Input	Type	Expectation	Output
t_{77}	Calculating PMCC	2 Sets of Values	-	A Number between -1 and 1	As Expected
t_{78}	Generate PMCC Grid	Training Data	-	2D List of PMCCs for each pair of Factors	As Expected

Table 5.25: Testing Table for Code Snippet 5.32

Now we must turn this 2D list into an image of a heatmap. In the design section, I had mentioned that a possible library to do this would be `matplotlib`, but after researching the topic, I believe `seaborn` is a better option.

data_processing / pmcc.py

```
3 import seaborn

data_processing / pmcc.py

34 # generate the image
35 heatmap = seaborn.heatmap(pmcc_grid, xticklabels=factors,
36   ↪ yticklabels=factors, vmin=-1, vmax=1, center=0, cmap='Spectral')
37 heatmap.get_figure().savefig('data/pmcc.png')
```

Code Snippet 5.33: Generating & Exporting the Figure

Here, I am creating a `seaborn.heatmap` of the `pmcc_grid`. The labels on the x -axis and the y -axis should be the factors, and the range should be -1 to 1 . For the colour (`cmap`), I believe I can only choose from certain options, instead of having my own colour scheme. I chose '`Spectral`', as I believe it is the closest to what I used in the design section. Once the heatmap is created, we can save the figure to a `png` file.

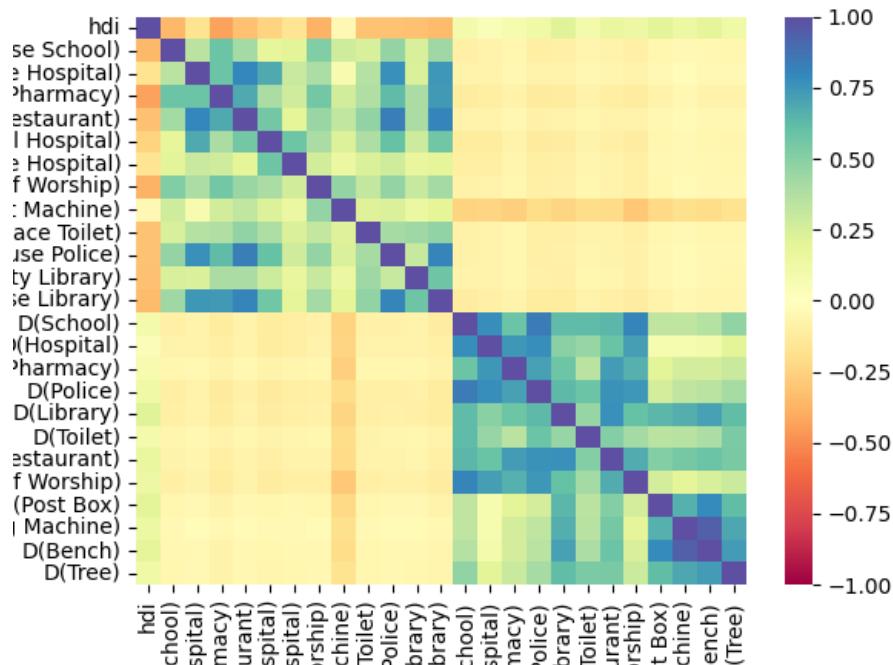


Figure 5.23: Initial Image of the Correlation Heatmap

Figure 5.23 shows the image generated, which unfortunately has the labels mostly cut off.

No.	Test	Input	Type	Expectation	Output
t_{79}	Generate Heatmap	PMCC Grid	-	Image of the heatmap, showing all factors	Image of the heatmap, with the labels cut off

Table 5.26: Testing Table for Code Snippet 5.33

data_processing / pmcc.py

```
36  heatmap.get_figure().savefig('data/pmcc.png', bbox_inches='tight',
    ↵   dpi=600)
```

Code Snippet 5.34: Improving the Quality of the Figure

After looking at the documentation, for the `savefig` method, I discovered you can pass in `bbox_inches = 'tight'` to ensure that the output includes everything. While I was there, I found that you can change the resolution of the output image, by changing the `dpi` parameter (I set it to an arbitrary 600).

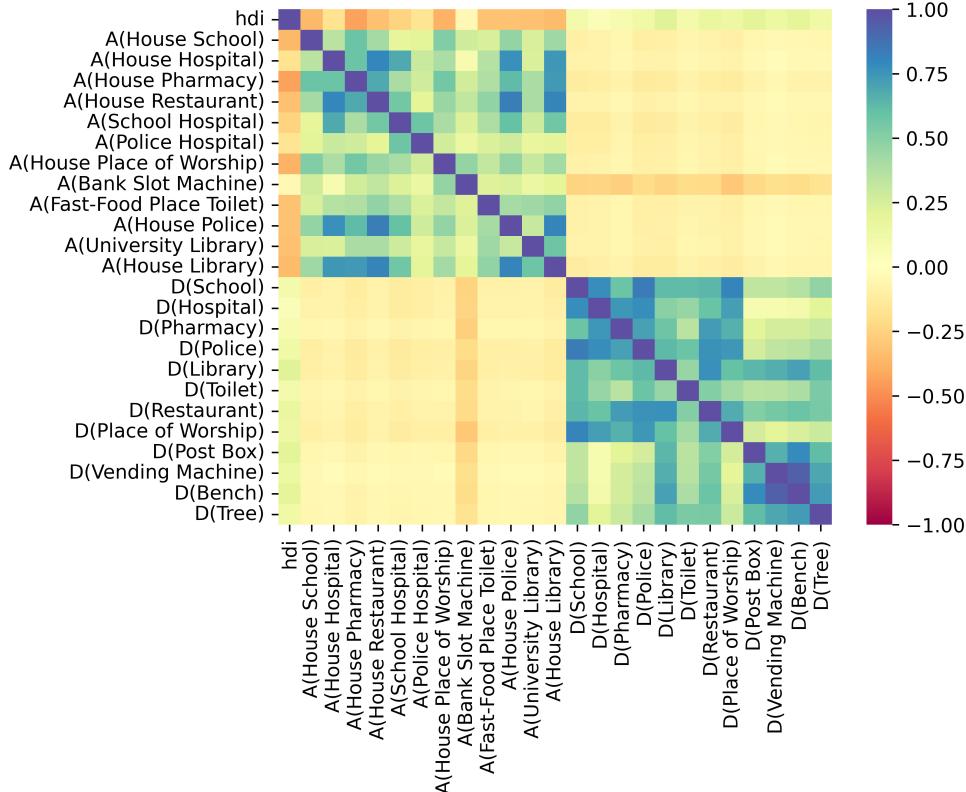


Figure 5.24: Final Image of the Correlation Heatmap

Figure 5.24 shows the new image generated, which is much better, and at much higher resolution.

No.	Test	Input	Type	Expectation	Output
t_{79}	Generate Heatmap	PMCC Grid	-	Image of the heatmap, showing all factors	As Expected

Table 5.27: Testing Table for Code Snippet 5.34

app.py

```
272     pmccImage = gr.Image('data/pmcc.png', label='Correlation between
    ↵ Factors', height=600, interactive=False)
```

Code Snippet 5.35: Adding the Image to the Gradio Interface

Finally, we can add the image to the gradio interface by making use of the `gr.Image` component. By default, this is used for users uploading images, but if we set `interactive=False`, we can use it to just display an image.

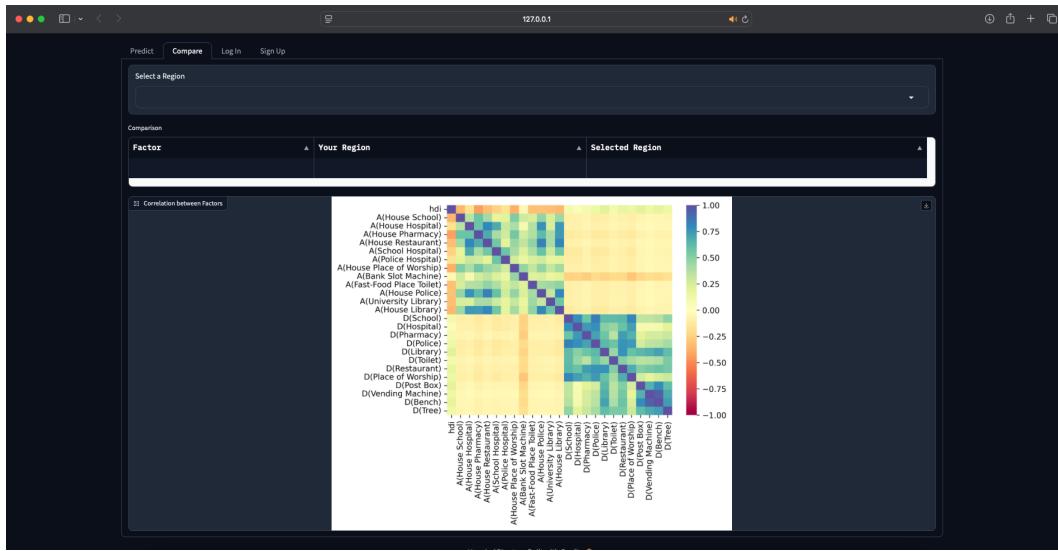


Figure 5.25: Correlation Heatmap in Gradio

Figure 5.25 shows the heatmap in the gradio interface. It seems that gradio also provides a download button on the image component, which allows for users to download this image and zoom in if they please.

No.	Test	Input	Type	Expectation	Output
t_{80}	Heatmap in Gra-dio Interface	—	—	Image of Heatmap at the Bottom of the Compare Tab	As Expected

Table 5.28: Testing Table for Code Snippet 5.35

5.2.4 Part 19 – Finding Similar Regions

In this part, I will be implementing the textbox on the main page, which will output a region which has a similar, if not identical, HDI to the user's. First, we must find a list of possible candidates to choose from.

app.py / def processPrediction

```

89      # find similar region
90      diff = 0
91      possible_choices = []
92      while len(possible_choices) == 0:
93          # search below & above (round to remove floating point errors)
94          possible_choices += [region for region in trainingData if
95              → region['hdi'] == str(round(prediction-diff, 3))]
96          possible_choices += [region for region in trainingData if
97              → region['hdi'] == str(round(prediction+diff, 3))]
# increment difference (round to remove floating point errors)
diff = round(diff + 0.001, 3)

```

Code Snippet 5.36: Iterating over Different Possible HDIs

To do this, I have defined a list of the possible choices, and we search further and further from the predicted HDI until we find at least one that matches. Most of the time, due to the vast number of regions, it will not go beyond 1 iteration.

When we add or subtract the difference, or increment the difference, we must round it to 3 decimal places, as there will often be a floating point error, causing the HDI that we are looking for to be slightly off from the ones in the data set.

```
app.py / def processPrediction
```

```
98     # choose one of the options at random
99     chosenRegionRecord = random.choice(possible_choices)
100    similar_region = f'{chosenRegionRecord["region"]},'
        ↳ {chosenRegionRecord["country"]}'
```

Code Snippet 5.37: Choosing a Region to Display

Once we have a list of possible regions, we can randomly choose from one to display. From there, we can format a string in the same way we did in the dropdown menu, when the user is comparing regions.

No.	Test	Input	Type	Expectation	Output
$T_{14.1}$	Choosing a Similar Region	HDI in the training data	Valid	Randomly chooses from those	As Expected
$T_{14.2}$	Choosing a Similar Region	HDI not in the training data	Valid	Choose the closest one	As Expected
$T_{14.3}$	Choosing a Similar Region	HDI not in the training data, with a tie for closest	Valid	Randomly chooses from those	As Expected

Table 5.29: T_{14} Testing Table (for Code Snippets 5.36 & 5.37)

```
app.py / def processPrediction
```

```
101     return prediction, similar_region
```

```
app.py
```

```
299     predictHDIbutton.click(processPrediction, inputs=[max_x_objectsSlider,
        ↳ max_y_objectsSlider], outputs=[HDIprediction, similarHDI])
```

Code Snippet 5.38: Returning to the Textbox

Then, at the end of the function, when we return the predicted HDI, we must also return this similar region. In the radio functionality, we must also accept another output, which is to the textbox stored in the variable `similarHDI` (This textbox was added in Code Snippet 3.81, but was never functional).

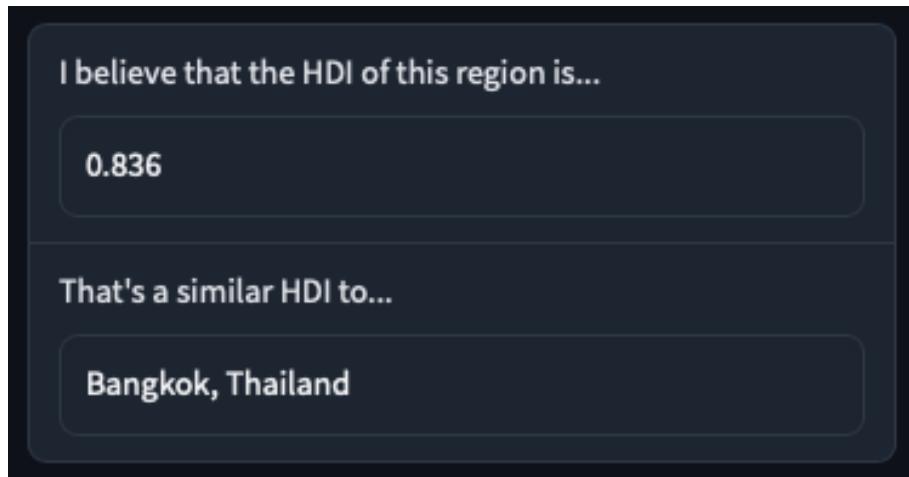


Figure 5.26: Similar Region Returned to Textbox

Figure 5.26 shows the output for Oxford.

No.	Test	Input	Type	Expectation	Output
t_{81}	Similar Textbox	HDI	Region Name	–	Returned to Textbox on Main Page

Table 5.30: Testing Table for Code Snippet 5.38

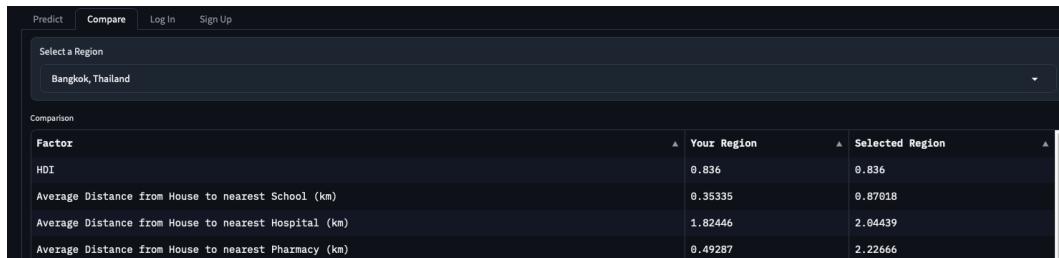


Figure 5.27: Comparison to Bangkok

When we look at the compare tab, and choose Bangkok to compare with, we can see that this has in fact worked, as Bangkok also has an HDI of 0.836.

5.2.5 Part 20 – Saving Regions

In this part, I will allow users to store different regions, and switch between which one they are predicting the HDI of.

```
app.py / def processPrediction
```

```
285     with gr.Tab(label='Regions'):
286         regionsDropdown = gr.Dropdown(choices=[], label='Current Region')
287         regionsTable = gr.DataFrame(label='Your Regions',
288             headers=['Name'] + list(trainingData[0].keys())[4:]),
289             type='array', col_count=(25, 'fixed'), interactive=True)
```

Code Snippet 5.39: Adding the Regions Tab

Here, I am creating the gradio interface for the Regions Tab. In the design section I had planned that the user would select their region by clicking in the table, and the interface sketch for this tab (Figure 4.8) had a textbox at the top to show which one was selected.

However, I now believe that it is better for the user to select which region they wish to process with a dropdown menu. At the start of the program running, this should have no possible options, as the user has not predicted any regions yet. Below this, is the table which will contain the regions. For this, we must set `interactive=True`, to ensure the user will be able to edit the contents (and hence manually enter their factors). This will also add buttons below the table, to create a new row and new column. I would like the user to create new rows (new regions), but not new columns. If we also set `col_count=(25, 'fixed')`, then we will ensure that the number of columns stays at 25, and the new column button will not be present. Finally, the headers of these 25 columns should be “Name”, followed by the names of all the factors, which we can get from the keys of a record from the training data.

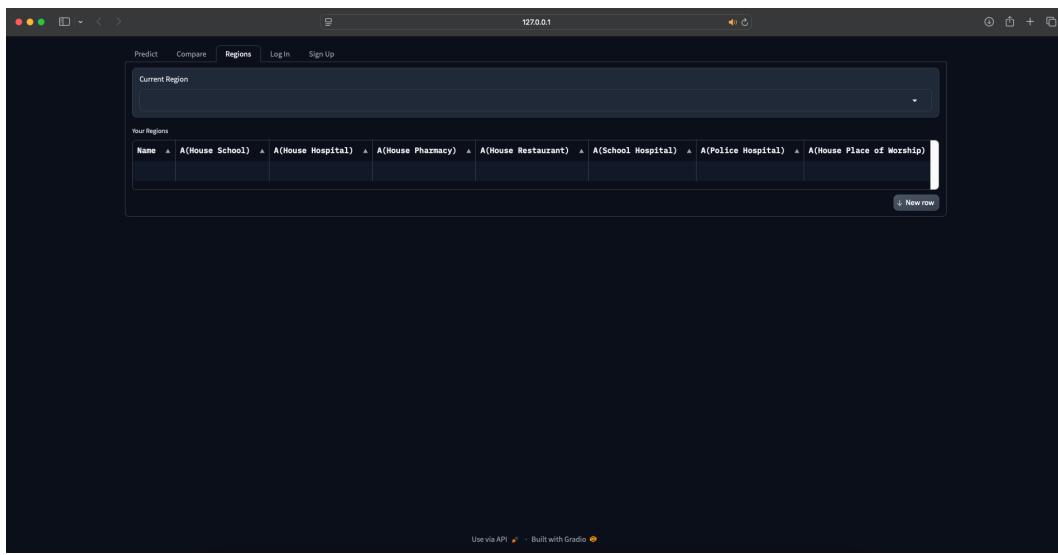


Figure 5.28: Regions Tab

No.	Test	Input	Type	Expectation	Output
t_{82}	Regions Tab	–	–	Dropdown Menu (no options) & Interactive Regions Table with New Row button	As Expected

Table 5.31: Testing Table for Code Snippet 5.39

In order to fill this table, we need to have some `global` variable, which stores a list of all regions. When the user uploads and predicts a region, It would be better for the entirety of this list to be updated at once, and not partially at different times (for example, adding the name of a region at a certain time, and then the factors later).

Currently, the name of the region is processed in the `def uploadGeoJSON` function, and the factors are processed in the `def processPrediction` function, and so the addition of these things will not be at the same time. Therefore, I have decided to move the part of the code that downloads all of the objects, and calculates each of the factors, to the `def uploadGeoJSON` function, which means that we can update this list of all regions at the end of this function.

This will also have the effect of making the upload button be really slow, and the prediction button really fast (as most of the time taken to predict is in downloading the objects). I believe that this is a good change, because users may be more used to uploading taking a while.

app.py

```
45 # code to process the user uploading their geojson file
46 def uploadGeoJSON(filename, max_x_objects, max_y_objects,
47     → progress=gr.Progress()):
48     global currentRegion
49     global all_objects
50     global allFactors
51     progress(0, desc='Starting...')
```

app.py / def uploadGeoJSON

```
54     shortFilename = filename[filename.rindex("/") + 1:]
```

app.py / def uploadGeoJSON

```
74         progress(0.91+0.09*(num/len(densityFactors)), desc=f'Calculating
    → {densityFactor} Density...')
```

app.py / def uploadGeoJSON

```
77     return f'Successfully Uploaded {shortFilename}'
```

app.py

```
79 # predict the HDI of the given region
80 def processPrediction():
81     global currentHDI
```

app.py

```
300     uploadButton.upload(uploadGeoJSON, inputs=[uploadButton,
    → max_x_objectsSlider, max_y_objectsSlider], outputs=[log])
301     predictHDIbutton.click(processPrediction, inputs=[],
    → outputs=[HDIprediction, similarHDI])
```

Code Snippet 5.40: Changes to Accommodate the Transition

In order to make it still work after this transition, we must make a few changes. The main change is what we pass into each function, where we now need to pass `max_x_objects` and `max_y_objects` into `def uploadGeoJSON` instead of `def processPrediction`. We therefore also need to update the radio functionalities, at the end of the program, to ensure we pass the values of the sliders in.

On top of these parameters, we must also now pass the `progress` into `def uploadGeoJSON`

instead of `def processPrediction`, as this is now the function that will take about a minute instead of `def processPrediction`. As this function does not contain the prediction part, we must also slightly change the numbers in the progress bar update when calculating the density factors, as well as removing the progress bar update in `def processPrediction` for predicting the HDI.

Each function must now also have slightly different `global` variables, as we now must make `all_objects` and `allFactors` global in the `def uploadGeoJSON` function, and so now the only variable that must be global in `def processPrediction` function is `currentHDI`.

Finally, while I was making these changes, I also decided to store the shorter filename in its own variable, as I will need to be using it again later.

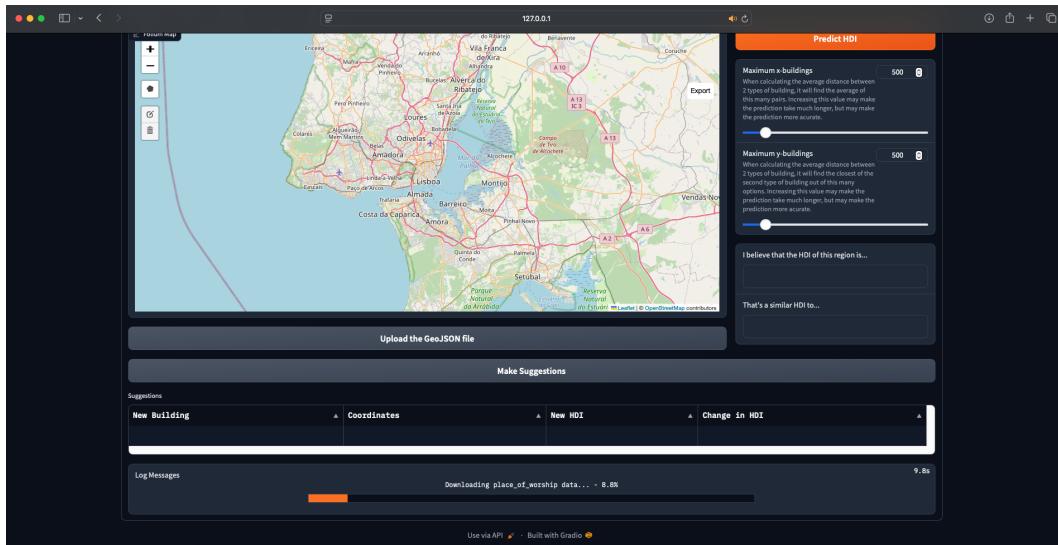


Figure 5.29: Progress Bar in the Log Messages

This also has the effect of making the progress bar appear over the log messages, instead of the output HDI box.

No.	Test	Input	Type	Expectation	Output
t_{83}	Predicting HDI Still Works	—	—	Uploading region file and then predicting HDI still results in the objects being downloaded, factors being calculated, and HDI being predicted	As Expected

Table 5.32: Testing Table for Code Snippet 5.40

app.py

```

27 allUserRegions = []

app.py / def uploadGeoJSON

51     global allUserRegions

app.py / def uploadGeoJSON

79         allUserRegions.append({
80             'name': shortFilename[:shortFilename.rindex('.')],
81             'objects': all_objects,
82             'factors': allFactors
83         })

```

Code Snippet 5.41: Storing New Regions when they Upload

Now that we have adjusted the code such that it processes the entire region in one function, we can create the `allUserRegions` list. In the `def uploadGeoJSON` function, we must specify this as a global variable, and at the end we can append the new region to it. The name of the region will be the name of the file, which is currently (for example) `'region.geojson'`. To get the right part of the name, we must take everything before the last `'.'`.

The next step is to now display this region in the table

app.py

```

191 # creates a 2d list to put in the regions table, from allUserRegions
192 def updateRegionsTable():
193     table = []
194     for region in allUserRegions:
195         table.append([region['name']] + region['factors'])
196     return table

```

Code Snippet 5.42: Creating a 2D List for the Regions Table

To do this, I have written a function, `def updateRegionsTable` to generate a 2D array from the `allUserRegions` list. The factors are stored as a `list` anyway, so we can just add this to a list containing the name and that will make up the table row. If we repeat this over all regions, we can construct a full table.

```
app.py / def uploadGeoJSON
```

```
84     return f'Successfully Uploaded {shortFilename}', updateRegionsTable()
```

```
app.py
```

```
314     uploadButton.upload(uploadGeoJSON, inputs=[uploadButton,
    ↵   max_x_objectsSlider, max_y_objectsSlider], outputs=[log,
    ↵   regionsTable])
```

Code Snippet 5.43: Updating the Regions Table

To update the table when we upload a region file, we can call the `def updateRegionsTable` function, to get the new table, and then return that in addition to the log message. In the gradio functionality, we should also specify that the second return value should be passed to the regions table.



Figure 5.30: Region added to the Table

Figure 5.30 shows the output for when I upload a file, called `region.geojson`, of Oxford.

No.	Test	Input	Type	Expectation	Output
t ₈₄	Upload New Region Added to Table	region.geojson	—	New row in the table with the name <code>region</code> , followed by 24 factors	As Expected

Table 5.33: Testing Table for Code Snippets 5.41, 5.42 & 5.43

```
app.py / def uploadGeoJSON
```

```
84     return f'Successfully Uploaded {shortFilename}', updateRegionsTable(),
    ↵     updateRegionsDropdown()
```

```
app.py
```

```
198 # updates regions dropdown, from allUserRegions
199 def updateRegionsDropdown():
200     names = [region['name'] for region in allUserRegions]
201     return gr.update(choices=names)
```

```
app.py
```

```
319 uploadButton.upload(uploadGeoJSON, inputs=[uploadButton,
    ↵ max_x_objectsSlider, max_y_objectsSlider], outputs=[log,
    ↵ regionsTable, regionsDropdown])
```

Code Snippet 5.44: Updating the Regions Dropdown

When the user uploads a new region, the options in the dropdown menu should also update, to reflect the table rows. To do this, I have written another similar function, `def updateRegionsDropdown`, which will generate a list of all the names of the regions, from the `allUserRegions` list.

Updating the choices is slightly more complicated however. Before, when we return something to a radio component, we have updated the `value` of it, which is the content held in the component. In the case of the dropdown menu, it would be the selected region. However, what we want to do here is change the `choices`, instead of the `value`. To do this, instead of just returning `names`, we can return `gr.update(choices=names)`, which will change the `choices` attribute of whatever component we return to to be `names`.

We must therefore also call `def updateRegionsDropdown` when we return in the `def uploadGeoJSON` function, and specify in the functionality that we want to return this update to the regions dropdown.

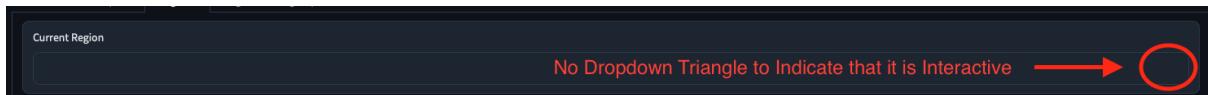


Figure 5.31: Updating the Dropdown Options

No.	Test	Input	Type	Expectation	Output
t_{85}	Updating Options in Regions Dropdown	region.geojson	—	New option in Regions Dropdown, <code>region</code>	Dropdown is no longer interactive, gradio assumes it's an output

Table 5.34: Testing Table for Code Snippet 5.44

Unfortunately, this has made the dropdown non-interactive. I believe this is because, if we have returned something to it, then gradio assumes it is meant to be an “output” component, and so shouldn’t be interactive.

`app.py`

```
304     regionsDropdown = gr.Dropdown(choices=[], label='Current Region',
    ↴   interactive=True)
```

Code Snippet 5.45: Ensuring the Dropdown is Interactive

We can fix this by just setting `interactive=True` in the regions dropdown component.

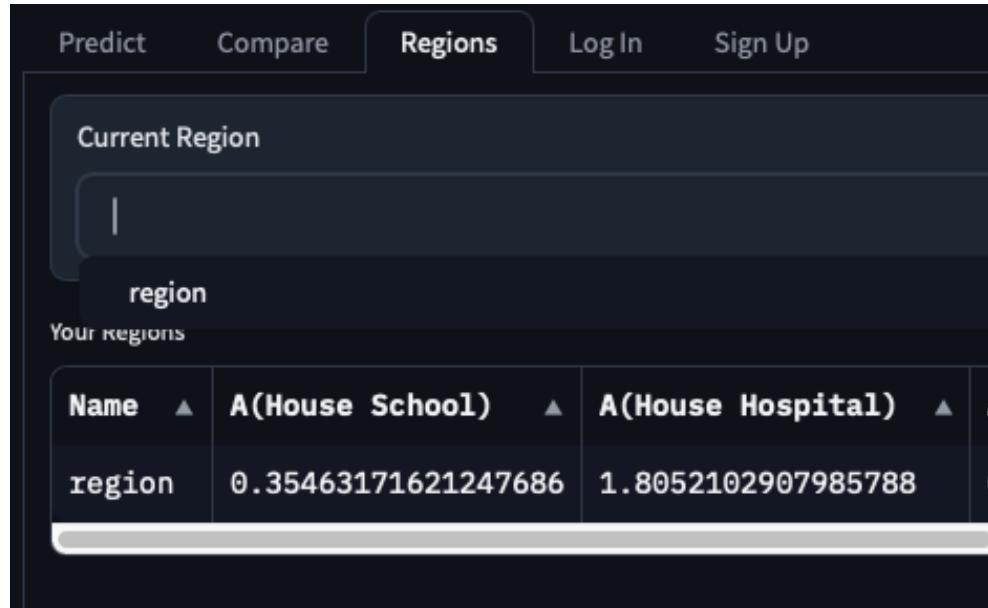


Figure 5.32: Dropdown Remains Interactive

No.	Test	Input	Type	Expectation	Output
t_{85}	Updating Options in Regions Dropdown	region.geojson	-	New option in Regions Dropdown, <code>region</code>	As Expected

Table 5.35: Testing Table for Code Snippet 5.45

If the user has just uploaded a region, the selected region (the value of the dropdown menu) should also change, to be the new region.

```
app.py / def uploadGeoJSON
```

```
84     return f'Successfully Uploaded {shortFilename}', updateRegionsTable(),
    ↵     updateRegionsDropdown(), shortFilename[:shortFilename.rindex('.')]]
```

```
app.py
```

```
319     uploadButton.upload(uploadGeoJSON, inputs=[uploadButton,
    ↵     max_x_objectsSlider, max_y_objectsSlider], outputs=[log,
    ↵     regionsTable, regionsDropdown, regionsDropdown])
```

Code Snippet 5.46: Updating the Value of the Regions Dropdown

We can do this, by just returning the name of the file to the dropdown menu. Note that this may look strange, as we are outputting to `regionsDropdown` twice, but remember one is to update the `choices`, and the other is now to change its value.

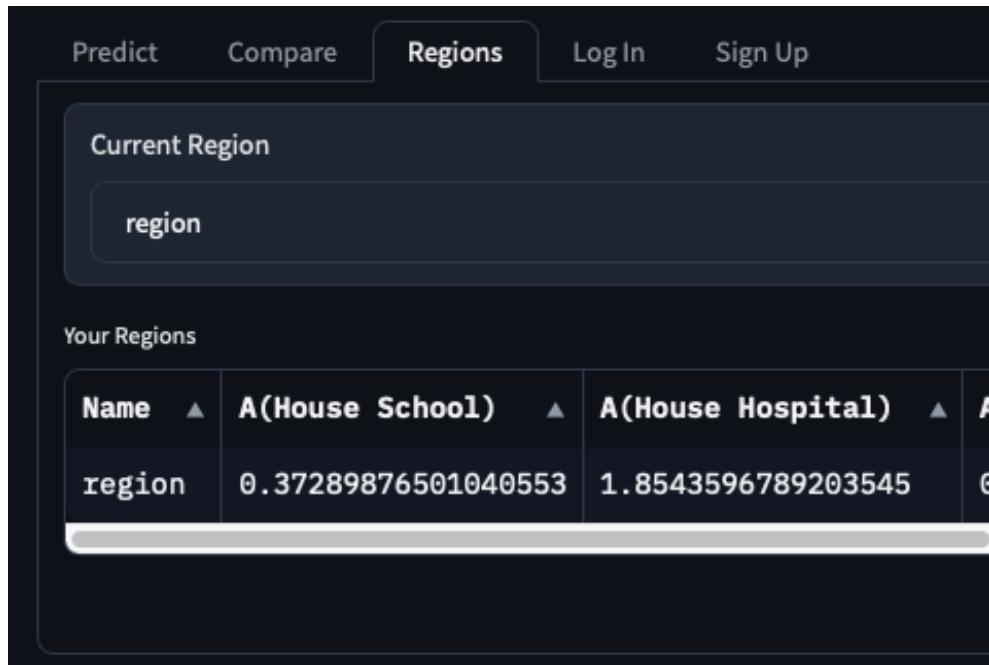


Figure 5.33: Dropdown Value Updated

No.	Test	Input	Type	Expectation	Output
t_{86}	Updating Value of Regions Drop-down	region.geojson	-	Current region becomes <code>region</code>	As Expected

Table 5.36: Testing Table for Code Snippet 5.46

The next thing to implement is updating the `allUserRegions` list whenever the user makes changes to the table. In the design section, I had planned for this to happen automatically, but now I believe it will be a better idea to let the user make their changes, and have a button to make all of the changes in the backend simultaneously.

`app.py`

```
306     submitEditsButton = gr.Button(value='Submit Table Changes',
307                                     variant='primary')
308     submitEditsResult = gr.Textbox(label='Result', interactive=False)
```

Code Snippet 5.47: Adding a Button to Submit Edits to the Table

Therefore, I need to add another button and result box, this time in the regions tab.

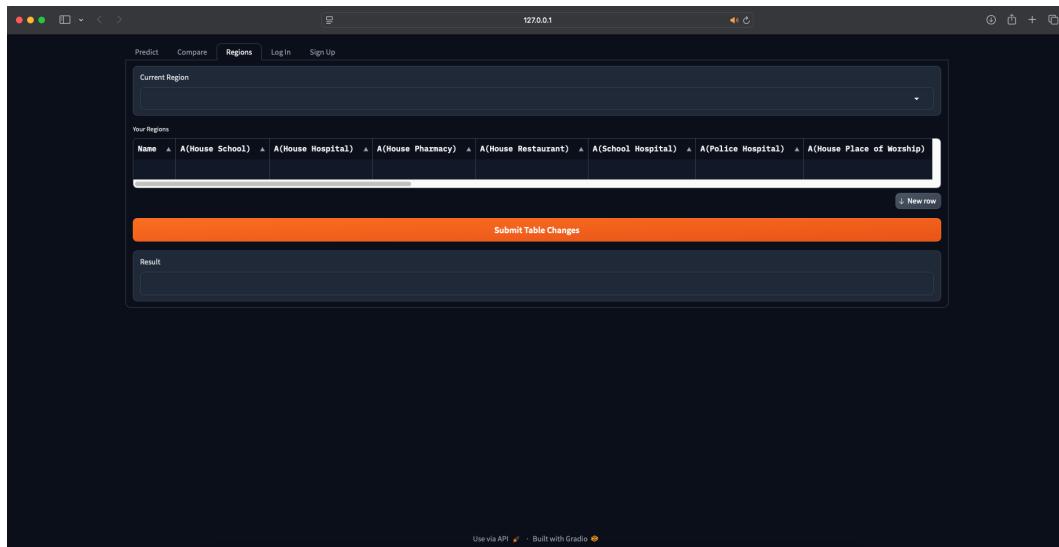


Figure 5.34: Submit Changes Button in Regions Tab

No.	Test	Input	Type	Expectation	Output
t_{87}	Full Regions Tab	—	—	Dropdown, Regions Table, Submit Changes Button, Submit Changes Result	As Expected

Table 5.37: Testing Table for Code Snippet 5.47

app.py

```
203 # updates the allUserRegions list, when the regions table is updated
204 def updateAllUserRegions(table, currentRegionName):
205     global allUserRegions
```

app.py

```
330 submitEditsButton.click(updateAllUserRegions, inputs=[regionsTable,
    ↴ regionsDropdown], outputs=[submitEditsResult, regionsDropdown,
    ↴ regionsDropdown])
```

Code Snippet 5.48: Adding the Submit Button Functionality

To make this button work, we must define another function, for another gradio functionality. On top of updating the `allUserRegions` variable, it should also update the dropdown

to choose regions from. That is why the inputs for this function are the table and the dropdown, and the outputs are the result textbox, and the region dropdown twice (again, once to update the choices and the other to update the value).

```
app.py / def updateAllUserRegions

206     # store the index of the current region
207     if currentRegionName != None:
208         currentRegionIndex = [region['name'] for region in
→      allUserRegions].index(currentRegionName)
```

Code Snippet 5.49: Saving the Index of the Current Region

In the case that the user has changed the name of their currently selected region, we must store the index of the row that this region is located it, so that we can keep it selected at the end of the function.

Before we update anything however, we must first validate that the user has entered correct data.

```
app.py / def updateAllUserRegions

209     # validate the input - names of regions
210     regionNames = [row[0] for row in table]
211     if '' in regionNames or len(set(regionNames)) != len(regionNames):
212         return 'Please ensure every region has a unique name!',
```

Code Snippet 5.50: Ensuring Unique Region Names

The first thing I am checking here is that they have correctly named their regions. Regions should not be named empty strings, and there should be no duplicate region names (which makes it easier to keep track of regions).

We can easily check if there are any empty names, but checking if they are all unique is a bit harder. The approach I took was to convert the list of names into a set, which can only hold unique values. If the length of the set is the same as the list, then there must have been no duplicate items. Otherwise, there are, and the user shouldn't be able to continue.

If the user has not satisfied these restrictions, we can return an error, with a message going to the result box, and calling the function to update the regions dropdown and returning the `currentRegionName` (which won't actually do anything, because we haven't changed `allUserRegions` yet. We must still return these however, as gradio is expecting something for each of the 3 components).

```

app.py / def updateAllUserRegions

213     # validate the input - type check
214     for rowIndex, row in enumerate(table):
215         for cellIndex, cell in enumerate(row):
216             if cellIndex != 0 and (cellIndex > 12 or cell != ''):
217                 try:
218                     table[rowIndex][cellIndex] = float(cell)
219                 except ValueError:
220                     return f'Please Enter the Data Correctly! Found an
221                         ↳ error at cell (row {rowIndex}, col {cellIndex})',
222                         ↳ updateRegionsDropdown(), currentRegionName

```

Code Snippet 5.51: Type Checking Every Other Cell

We then must ensure that the user has not entered any erroneous data, in any of the cells. We must therefore iterate over each of the cells, by using a nested `for` loop, and do a type check.

We have already validated the names of the regions (in column 0), so we don't need to validate them. The next 12 columns are the average distance factors, and the final 12 are the density factors. Average distance factors must either be a float, or an empty string, but density factors can only be floats (this is because average distances may be undefined).

Therefore, if we encounter a cell that is not in column 0, and that either we are considering a density factor, or it is not an empty string, then we must validate that it can be represented as a float.

To do this, I have made use of a `try except` block, which will catch any errors thrown in the `try` block, and then run the code in the `except` block. If we use the `try` block to try to turn it into a float, and it doesn't work (due to a `ValueError` caused by the user entering erroneous data), we can return an error, along with the same function call and current region name as before.

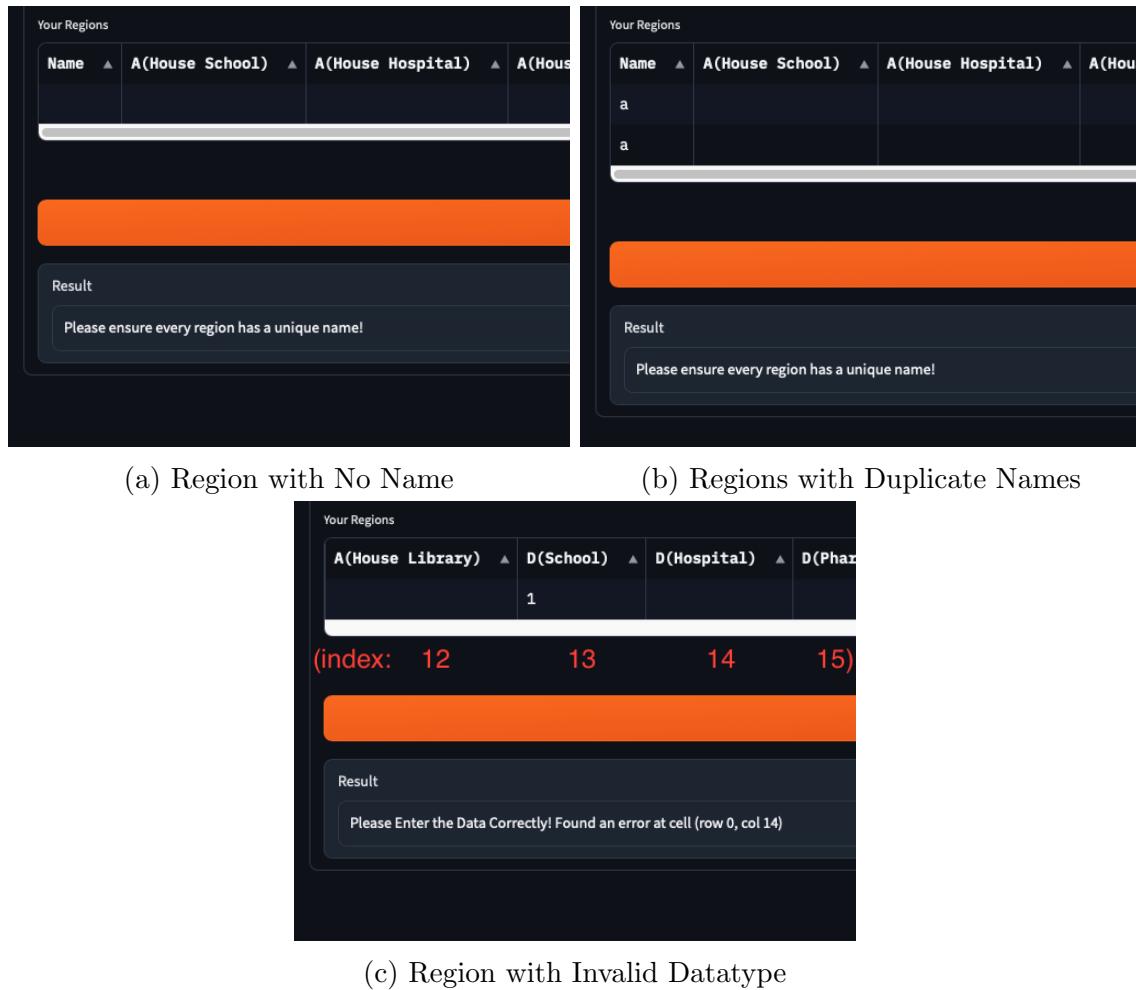


Figure 5.35: Validating Regions Table Input

No.	Test	Input	Type	Expectation	Output
$T_{15.1}$	Validating Regions Input	Region with No Name	Invalid	Prompt to ensure all regions have a name	As Expected
$T_{15.2}$	Validating Regions Input	Regions with Duplicate Names	Invalid	Prompt to ensure all regions have unique names	As Expected
$T_{15.3}$	Validating Regions Input	String in Density Factor	Invalid	Prompt to ensure all data is entered correctly	As Expected
$T_{15.4}$	Validating Regions Input	Non-Empty String in Average Distance Factor	Invalid	Prompt to ensure all data is entered correctly	As Expected

$T_{15.5}$	Validating Regions Input	Valid Entry	Valid	No Return (success message not implemented yet)	As Expected
------------	--------------------------	-------------	-------	---	-------------

Table 5.38: T_{15} Testing Table (for Code Snippets 5.48, 5.49, 5.50 & 5.51)

Now that the user's input has been validated, we can update `allUserRegions`.

```
app.py / def updateAllUserRegions

221     # update the allUserRegions list
222     initialLength = len(allUserRegions)
223     for index, row in enumerate(table):
224         factors = [None if value == '' else value for value in row[1:]]
225         # if the user has changed the factors of an existing list, the
226         # associated objects should be removed
227         if index < initialLength:
228             if row[1:] != allUserRegions[index]['factors']:
229                 allUserRegions[index] = {'name': row[0], 'objects': {}, 'factors': factors}
230             else:
231                 allUserRegions[index]['name'] = row[0]
232         # if the user has added a new region, it should have no
233         # associated objects
234     else:
235         allUserRegions.append({'name': row[0], 'objects': {}, 'factors': factors})
```

Code Snippet 5.52: Updating `allUserRegions`

Each row in the table corresponds to an element in the `allUserRegions` list, so we can just iterate over the rows in the table, and if we get to a point where we have exceeded the original length of the list, we know that these are new regions. It is important to know this information, as if the user has manually entered factors for a region, then there should be no objects associated with it, as the values will no longer reflect them. We should therefore also remove any objects from regions which the user edits in the table.

Therefore, if we haven't exceeded the original length yet, we must check if they have been edited. If they have, then we can replace the name and factors, and remove the objects. Otherwise, we just need to replace the name. If we have exceeded the length however, then we must append a new region, with the name, factors, and no object.

```
app.py / def updateAllUserRegions
```

```
234     # update the name of the selected region
235     if currentRegionName == None:
236         newRegionName = None
237     else:
238         newRegionName = allUserRegions[currentRegionIndex] ['name']
239     return 'Successfully Updated Regions', updateRegionsDropdown(),
→   newRegionName
```

Code Snippet 5.53: Updating the Name of the Selected Region

If the user has edited the name of their currently selected region, then we must also update the current value of the regions dropdown. If it was `None`, it should stay `None`, otherwise it should be the new name at the index of where the old name was (no matter if it has changed or not).

Once we have a name to return to the dropdown, we can return a success message, along with an update to the dropdown, and the new name of the selected region to the dropdown.

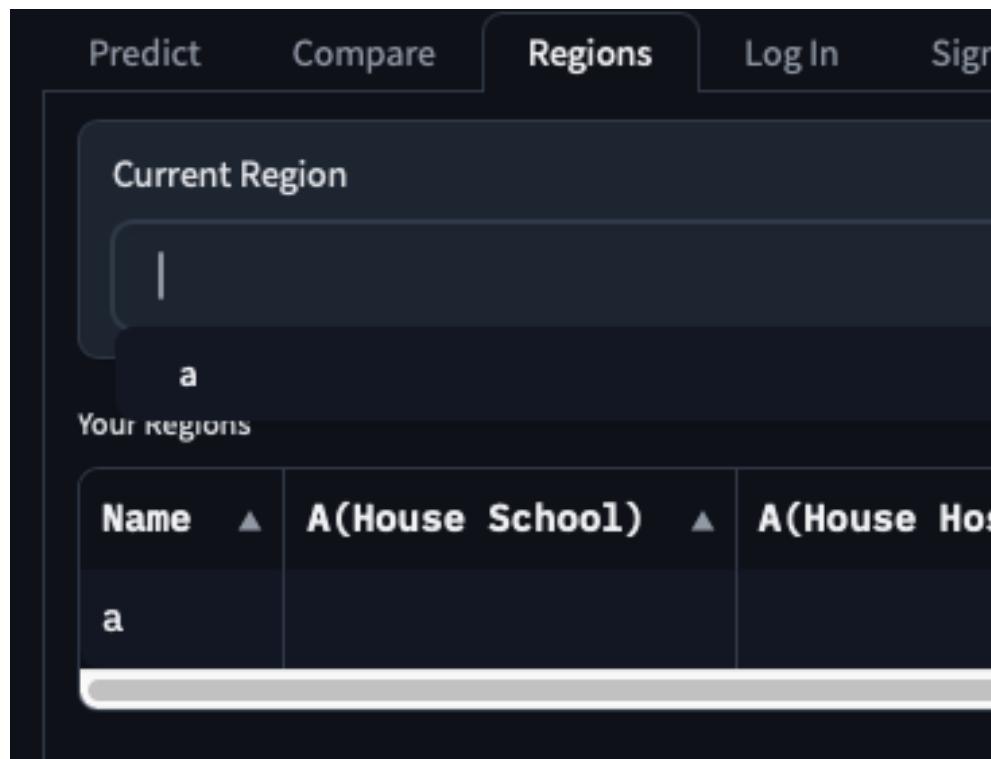


Figure 5.36: `allUserRegions` & Dropdown Updated from Table

Figure 5.36 shows a new region added to the dropdown. You may notice that the value of the dropdown is still empty, but this is in fact correct, because it was empty to begin with.

No.	Test	Input	Type	Expectation	Output
t_{88}	Updating Dropdown from Table	New region, with a name and density factors filled in	—	Region saved to <code>allUserRegions</code> , and name added to dropdown	As Expected

Table 5.39: Testing Table for Code Snippets 5.52 & 5.53

Now that we can add regions to the dropdown menu in multiple ways, it is time to actually implement the functionality dropdown menu.

app.py

```

241 # updates the current region, when the user switches it with the dropdown
242 → on the regions tab
243 def updateCurrentRegion(dropdownValue):
244     global all_objects
245     global allFactors
246     for region in allUserRegions:
247         if region['name'] == dropdownValue:
248             all_objects = region['objects']
              allFactors = region['factors']

```

app.py

```

373     regionsDropdown.change(updateCurrentRegion, inputs=[regionsDropdown],
374 →   outputs=[])

```

Code Snippet 5.54: Updating the Current Region

All the dropdown menu needs to do is change the current region, which should be easy enough. All we need to do is iterate over `allUserRegions`, and when we find the one that matched with the new dropdown value, we set the `all_objects` and `allFactors` variable to the values from the list.

This function should be run every time the value of the regions dropdown changes, for whatever reason (e.g. the user updates it, or it gets updated by a function).

No.	Test	Input	Type	Expectation	Output
t_{89}	User can switch between regions	—	—	Switching between 2 regions in the dropdown menu causes different predictions based on which one is selected	As Expected

Table 5.40: Testing Table for Code Snippet 5.54

app.py

```
46 # code to process the user uploading their geojson file
47 def uploadGeoJSON(filename, max_x_objects, max_y_objects,
→ currentRegionName, progress=gr.Progress()):
```

app.py / def uploadGeoJSON

```
57     if shortFilename[:shortFilename.rindex('.')] in [region['name'] for
→     region in allUserRegions] or
→     shortFilename[:shortFilename.rindex('.')] == '':
58         return 'There is already a region with this name! Please rename
→         the file first.', updateRegionsTable(),
→         updateRegionsDropdown(), currentRegionName
```

app.py

```
370     uploadButton.upload(uploadGeoJSON, inputs=[uploadButton,
→     max_x_objectsSlider, max_y_objectsSlider, regionsDropdown],
→     outputs=[log, regionsTable, regionsDropdown, regionsDropdown])
```

Code Snippet 5.55: Ensuring the User doesn't Upload a Duplicate Region Name

As the input from the table does not allow for empty region names or duplicate region names, the same should be true for when the user uploads a region file.

Here, I am checking if the name is in the list of existing names, or if the name is empty, and if it is, then we can return a suitable error message, along with the necessary updates (which again, will do nothing as we haven't changed anything yet).

To do this, we must also pass in the `currentRegionName`, so that we can return it back in this case if we need to.

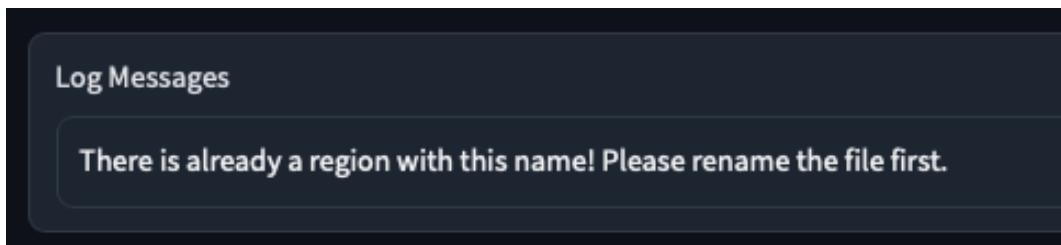


Figure 5.37: Cannot Upload Region with the Same Name

No.	Test	Input	Type	Expectation	Output
t_{90}	User cannot upload region with the same name	region.geojson, twice	–	Suitable error message in the log messages	As Expected

Table 5.41: Testing Table for Code Snippet 5.55

Now that I have implemented saving regions on a local level, it is time to bring this feature to accounts.

app.py

```
28     loggedInUsername = ''
```

```
297    app.py / def logIn
        global loggedInUsername
```

```
309    app.py / def logIn
        loggedInUsername = username
```

Code Snippet 5.56: Adding a Global Variable for Logging In

First, I need a way of keeping track of which account the user is currently logged into. Here, I have defined a variable, `loggedInUsername`, which keeps track of the username of the account the user is currently logged into. If it is empty, then that means that the user is not logged in, so therefore it should initially be empty.

When we log in, we must ensure that we are using the `global` variable, and then at the end, once we have verified everything is correct, we can set the `loggedInUsername` to the username that the user has entered.

```
app.py / def signUp
```

```
255     global loggedInUsername
```

```
app.py / def signUp
```

```
294     loggedInUsername = username
295     return f'Successfully created & logged into an account, with username:
→ {username}!', updateRegionsTable(), updateRegionsDropdown()
```

Code Snippet 5.57: Logging Into a Newly Created Account

The user should also be logged in when they sign up for an account. This can be easily done by again setting `loggedInUsername` to the username the user entered, at the end of the function.

No.	Test	Input	Type	Expectation	Output
t_{91}	Logging Into an Account	A username	—	<code>loggedInUsername</code> updates to that username	As Expected

Table 5.42: Testing Table for Code Snippets 5.56 & 5.57

```
app.py / def uploadGeoJSON
```

```
87     # save regions to account, if logged in
88     if loggedInUsername != '':
89         users.update_one({'username': loggedInUsername}, {'$set':
→   {'regions': allUserRegions}})
```

```
app.py / def updateAllUserRegions
```

```
240     # save regions to account, if logged in
241     if loggedInUsername != '':
242         users.update_one({'username': loggedInUsername}, {'$set':
→   {'regions': allUserRegions}})
```

Code Snippet 5.58: Adding Regions to the Account

When the user uploads a new region, or makes a change in the table, we should update the user's document in the database (if they are logged in). We can do this by calling the `update_one` method, and set the `'regions'` attribute to `allUserRegions`.

```
app.py / def logIn
```

```
306     global allUserRegions
```

```
app.py / def logIn
```

```
319     allUserRegions = user['regions']
```

Code Snippet 5.59: Reading Regions from the Account

When the user logs into an account, the regions should also be replaced by those in the account. This can be done by just setting `allUserRegions` to the user's '`regions`' at the end of the `def logIn` function.

```
app.py / def updateRegionsTable
```

```
202     # if allUserRegions is empty, return an empty table
203     if table == []:
204         table.append([''*25])
```

Code Snippet 5.60: Handling Empty Region Lists

This also made me realise about a slight oversight, which is that the regions table may be updated when `allUserRegions` is empty. In which case, we should return a `list`, containing a `list` of 25 empty strings.

```
_id: ObjectId('6776a5028304fd0e6b8346bf')
username: "notlukewilliams"
hashedPassword: Binary.createFromBase64('JDJiJDEyJGNGeTB1cjR0RVJweE9zV3cwNC95TmVMM1JJLjVuTXpWcTYzNThwZ3dLTVRZL09CTmxWeUzl', 0)
regions: Array (1)
  0: Object
    name: "Oxford"
    objects: Array (empty)
    factors: Array (24)
```

Figure 5.38: Region Added to MongoDB

Figure 5.38 shows a region added to my account, in MongoDB.

No.	Test	Input	Type	Expectation	Output
t_{92}	Saving Regions to an Account	Process a Region while Logged In	-	Region saved to MongoDB	As Expected

Table 5.43: Testing Table for Code Snippets 5.58, 5.59 & 5.60

However, I noticed that here the '`objects`' array is empty, where it shouldn't be, because I generated it by uploading a region. This lead me to believe that the `def updateRegionsTable` function had incorrectly identified that this had been edited, where in fact it had not.

```
app.py / def updateRegionsTable

236     if [float(factor) if factor != '' else None for factor in
237         ↵ row[1:]] != allUserRegions[index]['factors']:
238             allUserRegions[index] = {'name': row[0], 'objects': {}, 
239             ↵ 'factors': factors}
240         else:
241             allUserRegions[index]['name'] = row[0]
```

Code Snippet 5.61: Correctly Identifying when Table Rows have Changed

I fixed this by ensuring that we convert all factors to floats, and replace empty strings with `Nones`.

```
_id: ObjectId('6776a5028304fd0e6b8346bf')
username: "notlukewilliams"
hashedPassword: Binary.createFromBase64('JDJiJDEyJGNGeTBICjRORVJweE9zV3cwNC95TmVMM1JJLjVuTxpWcTYzNThwZ3dLTVRZL09CTmxWeUzl', 0)
regions: Array (2)
  ▶ 0: Object
  ▶ 1: Object
    name: "Oxford2"
    ▶ objects: Object
      ▶ house: Array (12738)
      ▶ school: Array (66)
      ▶ hospital: Array (5)
      ▶ pharmacy: Array (24)
      ▶ restaurant: Array (161)
      ▶ place_of_worship: Array (145)
      ▶ bank: Array (14)
      ▶ slot_machines: Array (empty)
      ▶ fast_food: Array (126)
      ▶ toilets: Array (52)
      ▶ police: Array (4)
      ▶ university: Array (83)
      ▶ library: Array (71)
      ▶ post_box: Array (199)
      ▶ vending_machine: Array (28)
      ▶ bench: Array (624)
      ▶ tree: Array (3411)
    ▶ factors: Array (24)
```

Figure 5.39: Region with Objects Added to MongoDB

Figure 5.39 shows the new region added to the MongoDB, which now correctly has all of the objects

No.	Test	Input	Type	Expectation	Output
t ₉₃	Saving Regions with Objects to an Account	Process a Region while Logged In	—	Region with Objects saved to MongoDB	As Expected

Table 5.44: Testing Table for Code Snippet 5.61

5.2.6 Part 21 – Improved Suggestions

By the end of this section, I hope to have better suggestions returned to the user.

`app.py`

```
361     with gr.Row():
362         makeSuggestionsButton = gr.Button(value='Make Suggestions')
363         newBuildingsSlider = gr.Slider(minimum=1, maximum=10, value=5,
364             step=1, label='Number of New Buildings', interactive=True)
```

Code Snippet 5.62: Adding a Slider for Number of New Buildings

Firstly, I needed to add a slider so that the user can decide how many new buildings they want to place (More buildings will lead to better suggestions, but will take longer to calculate).

Here, I have added a row, which contains the button to make suggestions, as well as this new slider. I set the minimum to be 1 (we must build at least 1 new building), and the maximum to be 10 (an arbitrary upper bound, to ensure it does not take too long). I set the default value to 5, which I believe is a good middle ground between the two. We also need to ensure that we are building an integer number of buildings, so I made the step equal to 1.

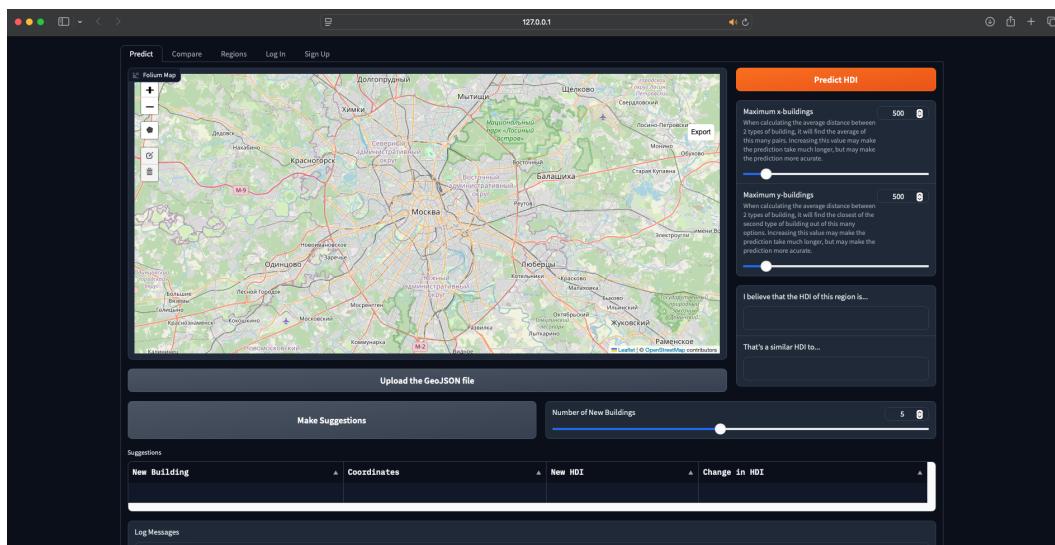


Figure 5.40: Predict Tab with New Slider

No.	Test	Input	Type	Expectation	Output
t_{94}	Appearance of Suggestions Section	–	–	Number of Buildings Slider next to Make Suggestions Button	As Expected

Table 5.45: Testing Table for Code Snippet 5.62

We now need to re-implement the `def calcOptimalPlaceFor` function, to include the new formula to calculate the optimal position of new buildings (equation 4.2), as well as lots of validation for various scenarios that could occur.

`app.py`

```

119 # calculate the optimal place for a new building to go when suggesting
120 def calcOptimalPlaceFor(building, extraBuildings):
121     global all_objects
122     # construct all_objects_in_consideration
123     all_objects_in_consideration = {}
124     for key in all_objects.keys():
125         if key in extraBuildings.keys():
126             all_objects_in_consideration[key] = all_objects[key] +
127                 extraBuildings[key]
128         else:
129             all_objects_in_consideration[key] = all_objects[key]
```

Code Snippet 5.63: Finding All Objects in Consideration

The first thing we must ensure is that we are considering the right lists of objects. This is slightly more complex as last time, as this time, we may be calling this function for the 2nd or 3rd (and so on) time, and so we must consider the previous buildings we have suggested.

To do this, I am adding an extra parameter, `extraBuildings`, which will be a dictionary, containing lists of the new buildings to consider. To construct a dictionary of all objects in consideration, we can iterate over the keys of the `all_objects` dictionary, and add the corresponding list from the `extraBuildings` dictionary if it exists. Otherwise, we can just set it to the original list in the `all_objects` dictionary.

```
app.py / def calcOptimalPlaceFor
```

```

129     # iterate over the factors
130     optimalPositions = []
131     zeroOfEverything = True
132     zeroOfThings = []
133     for averageDistanceFactor in averageDistanceFactors:
134         if averageDistanceFactor[1] == building:
135             # initialise variables
136             x_objects =
137                 ↵ all_objects_in_consideration[averageDistanceFactor[0]]
138             y_objects =
139                 ↵ all_objects_in_consideration[averageDistanceFactor[1]]
```

Code Snippet 5.64: Iterating over each Factor

We then need to iterate over each average distance factor, and see which ones are in relation to the new building we are suggesting. If it is, we can define a list of `x_objects` and `y_objects` to process. Before this however, we must define a list of `optimalPositions`, which will store the optimal position according to each factor we consider (we will be returning the mean of this).

We must also create a flag, `zeroOfEverything`, which will tell us if there are 0 of each type of *x*-object, as well as a list of the objects which there are 0 of (`zeroOfThings`). In the design section, I had said that if there was 0 of everything, I would return the centre of the region as the optimal place to put the new building, but now I realise this is not possible, as the only data we are given is the lists of the coordinates of `x_objects` and `y_objects`. We cannot even estimate the centre of the region from these lists, as if the user selects a very small region, they may be empty. Therefore, if there are 0 `x_objects`, I have decided to return another suggestion to the user, instead of a new HDI, which is that they should build some of those `x_objects` first. There may be multiple types of `x_objects` for which there are 0, which is why we must define the list, `zeroOfThings`, to keep track of all of them, and at the end we will return a random one.

```
app.py / def calcOptimalPlaceFor
```

```

139         if len(x_objects) != 0:
140             zeroOfEverything = False
141             # calculate the shortest distance from each
142             ↳ {averageDistanceFactor[0]} to
143             ↳ {averageDistanceFactor[1]}
144             shortest_dists = []
145             for x_object in x_objects:
146                 dists = []
147                 if len(y_objects) != 0:
148                     for y_object in y_objects:
149                         dists.append(distBetween2Points(x_object,
150                             ↳ y_object))
151                 else:
152                     dists.append(10000)
153                 if min(dists) == 0:
154                     shortest_dists.append(0.00001)
155                 else:
156                     shortest_dists.append(min(dists))
157             # calculate the optimal position
158             list_of_numpy_arrays = [np.array(x_object) for x_object in
159             ↳ x_objects]
160             optimalPosition = (sum([x_object/shortest_dists[index] for
161             ↳ index, x_object in enumerate(list_of_numpy_arrays)],
162             ↳ np.array([0,0]))/(sum([1/shortest_dist for
163             ↳ shortest_dist in shortest_dists])))
164             optimalPositions.append(optimalPosition)
165         else:
166             zeroOfThings.append(averageDistanceFactor[0])

```

Code Snippet 5.65: Calculating the Shortest Distances & Optimal Position

Here, I am calculating the optimal position of the new building. First, we must check if there are 0 *x*-objects, by checking the length of the `x_objects` list. If there are in fact 0, then we can add the name of this type of object to the `zeroOfThings` list, as said before. Otherwise, we can confirm that there isn't zero of *everything*, and so we can set `zeroOfEverything` to `False` and continue to find an optimal position.

The first step is finding the distance from each *x*-object, to its closest *y*-object. For each *x*-object, we must define a list of distances to each *y*-object, and find then minimum. If it turns out that there are no *y*-objects (`y_objects` is empty), then we can add an arbitrarily large distance (10000) to the list to act as the shortest distance. Another thing that may go wrong here is if the shortest distance ends up being 0, which will be a problem because we must divide by these numbers in the future. If it is 0, we can set the shortest distance to be

an arbitrarily small number (0.00001), otherwise it will be the minimum of the distances list.

Once we have a list of the shortest distances, corresponding to each x -object, we can implement equation 4.2. First however, we should convert each coordinate to a `numpy.array`. This is because if we add 2 `numpy` arrays, the result will be a `numpy` array of the same dimension, containing the element-wise sums of the values. For example, $[1, 2] + [3, 4] = [4, 6]$, which is what we need when adding coordinates. Once we have a list of `numpy` arrays, we can calculate the optimal position and add it to the `optimalPositions` list.

```
app.py / def calcOptimalPlaceFor
```

```
160     # return either something else to build, or the new position
161     if zeroOfEverything:
162         return random.choice(zeroOfThings)
163     return calcMeanOfCoords(optimalPositions)
```

Code Snippet 5.66: Returning the Optimal Position

Finally, if there was in fact `zeroOfEverything`, we can return a random choice from the `zeroOfThings` list, to be formatted into a string, such as '**'Build a new ... first!'**'. Otherwise, we can just return the mean position of the optimal positions, as done before.

No.	Test	Input	Type	Expectation	Output
$T_{16.1}$	Improved Suggestions	Region for which there exists 0 of each type of x -object	Edge Case	We instead return another suggestion, to build that type of x -object	As Expected
$T_{16.2}$	Improved Suggestions	Region for which there exists 0 of a particular type of y -object	Edge Case	An arbitrarily large value is used for the shortest distances	As Expected
$T_{16.3}$	Improved Suggestions	Region for which there exists an x -object in the same place as a y -object	Edge Case	An arbitrarily small value is used for the shortest distances	As Expected
$T_{16.4}$	Improved Suggestions	Region which does not satisfy any of the above edge cases	Valid	Optimal position calculated as normal	As Expected

Table 5.46: T_{16} Testing Table (for Code Snippets 5.63, 5.64, 5.65 & 5.66)

Now that we have a full function to find the optimal place to put a new building, we can adapt the `def makeSuggestions` function, to work with the new requirements.

app.py

```

165 # make suggestions from a prediction
166 def makeSuggestions(max_x_objects, max_y_objects, num_new_buildings,
167     ↪ progress=gr.Progress()):
168     global all_objects
169     if currentHDI == -1:
170         return [['You have not yet predicted an HDI!', None, None, None]]
171     if all_objects == {}:
172         return[['This region does not have any associated buildings!',
173             ↪ 'Please choose a different region', None, None]]
172     suggestions = ['school', 'hospital', 'place_of_worship', 'police',
173         ↪ 'restaurant', 'slot_machines', 'library', 'pharmacy']
173     returnTable = []

```

app.py

```

427     makeSuggestionsButton.click(makeSuggestions,
428         ↪ inputs=[max_x_objectsSlider, max_y_objectsSlider,
429             ↪ newBuildingsSlider], outputs=[suggestionsTable])

```

Code Snippet 5.67: New `def makeSuggestions` function

This time, we must pass `num_new_buildings` in, which will be an integer specifying how many new buildings we want to place. As before, we ensured that the user has predicted an HDI, but now we also must ensure that the current region has objects associated with it, as that is what we are using to place the new buildings. If it does not, we should return an error message, along with a prompt for the user to select a different region.

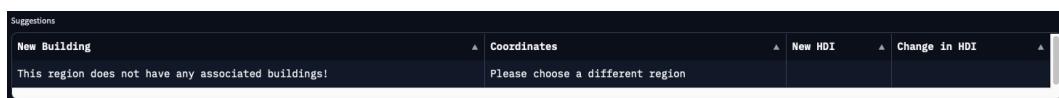


Figure 5.41: No Associated Buildings Error Message

No.	Test	Input	Type	Expectation	Output
t_{95}	Attempt to Suggest when Selected Region has no objects	-	Invalid	Prompt to select a different region returned to user	As Expected

Table 5.47: Testing Table for Code Snippet 5.67

app.py / def makeSuggestions

```

174     for num, suggestion in enumerate(suggestions):
175         extraBuildings = {suggestion: []}
176         progress(num/len(suggestions), desc=f'Suggesting building a
177             ↪ {suggestion}...')
178         # find optimal place
179         positions = []
180         for buildingNum in range(num_new_buildings):
181             # check for other suggestion
182             if len(positions) > 0:
183                 if type(positions[0]) == type(''):
184                     continue
185             # calc actual position
186             progress((num/len(suggestions))+(buildingNum*(1/8)
187                 ↪ *(1/(num_new_buildings+3))), desc=f'Suggesting building a
188                 ↪ {suggestion} (finding optimal position,
189                 ↪ {buildingNum+1}/{num_new_buildings})...')
190             position = calcOptimalPlaceFor(suggestion, extraBuildings)
191             positions.append(position)
192             extraBuildings[suggestion].append(position)

```

Code Snippet 5.68: Finding more than 1 Optimal Position

Now that we have validated that the user has a suitable region selected, we can make our suggestions. As usual, we iterate over each type of suggestion, and find the optimal places, but this time we must also iterate for `num_new_buildings` times, to find new suggestions. We can usually do each one as normal, by calling the `def calcOptimalPlaceFor` function, and then adding that to the `positions` list. However, if the first one returned another suggestion (as a result of there being 0 of all x -objects), then we should not attempt to find location again. Therefore, we must first check if we are not on the first iteration (as to avoid an index out of range error), and if its not, we can check if the first element in `positions` is a string. If it is, then that means we returned another suggestion instead of a location, and so we should `continue` (skip the rest of the `for` loop).

```
app.py / def makeSuggestions
```

```
189     if type(positions[0]) != type(''):
190         ... # (continue to calculate factors & predict HDI)
```

```
app.py / def makeSuggestions
```

```
215     else:
216         returnTable.append([suggestion, f'Build a new {positions[0]}'
217             ↪ first!', '--', '--'])
```

Code Snippet 5.69: Returning a Different Suggestion

Now that we have made a list (of length `num_new_buildings`) of new locations to place new buildings, we can re-calculate the factors and re-predict the HDI as usual. However, again we can only do this if we are not returning another suggestion (which we again check by checking the `type` against an empty string). If it turns out we are returning a different suggestion, we should instead append a string prompting them to build the new building, and no changes to HDI.

```
calc_factors.py / def averageDistance
```

```
24     if must_include_x != None: # include the new building when making
25         ↪ suggestions
25         x_objects += must_include_x
```

```
calc_factors.py / def averageDistance
```

```
34     if must_include_y != None: # include the new building when making
35         ↪ suggestions
35         y_objects_in_consideration += must_include_y
```

Code Snippet 5.70: Updating the `must_include` functionality in `def averageDistance`

The only thing we must change if we are actually predicting a new HDI is the `def averageDistance` function in `calc_factors.py`, by making it able to handle multiple new buildings. We can simply do this by adding the `must_include_x` list to the `x_objects` list we are considering, and similar for the `y-objects`.

app.py / def makeSuggestions

```
214     returnTable.append([suggestion, ',  
    ↪   '.join([f'{round(position[0], 7)}°N, {round(position[1],  
    ↪   7)}°E' for position in positions]), str(prediction),  
    ↪   str(round(prediction-currentHDI, 3))])
```

Code Snippet 5.71: New Return Format for Suggestions Table

Finally, when we return, we should surround the coordinates with brackets, and separate them with commas.

Suggestions	New HDI	Change in HDI
107°N, -0.7206731°E), (52.4042938°N, -0.7207446°E), (52.4042927°N, -0.7207564°E), (52.4042927°N, -0.7207509°E), (52.4042927°N, -0.720751°E)	0.668	0.047
357°N, -0.72333°E), (52.4008444°N, -0.7230271°E), (52.4008296°N, -0.7230098°E), (52.4008272°N, -0.723009°E), (52.400827°N, -0.7230089°E)	0.669	0.048
446°N, -0.721264°E), (52.4038433°N, -0.7212648°E), (52.4038433°N, -0.7212648°E), (52.4038433°N, -0.7212648°E)	0.668	0.047
7036°N, -0.7211326°E), (52.4034751°N, -0.720903°E), (52.4036269°N, -0.7209002°E), (52.4036608°N, -0.7209072°E), (52.4036608°N, -0.7209103°E)	0.825	0.264
7447°N, -0.7205597°E), (52.4042409°N, -0.7205889°E), (52.4042408°N, -0.7205903°E), (52.4042408°N, -0.7205903°E), (52.4042408°N, -0.7205903°E)	0.721	0.1
3809°N, -0.7281153°E), (52.3987864°N, -0.7280847°E), (52.3987361°N, -0.7280733°E), (52.3987067°N, -0.7280675°E), (52.3986886°N, -0.7280644°E)	0.721	0.1
7196°N, -0.7214112°E), (52.4036442°N, -0.7218645°E), (52.4037847°N, -0.7218347°E), (52.4038159°N, -0.7218342°E), (52.4038231°N, -0.7218351°E)	0.821	0.2
3544°N, -0.7209832°E), (52.4028357°N, -0.7209951°E), (52.4028648°N, -0.7210007°E), (52.4028698°N, -0.721002°E), (52.4028707°N, -0.7210022°E)	0.839	0.218

Figure 5.42: Output for a Region with Many Buildings

Figure 5.42 shows the output for a typical region, which contains many buildings. As you can see, these suggestions are much better than the ones from the 1st prototype, as they have much more range.

Suggestions	Coordinates	New HDI	Change in HDI
New Building			
school	Build a new house first!	--	--
hospital	Build a new school first!	--	--
place_of_worship	Build a new house first!	--	--
police	Build a new house first!	--	--
restaurant	Build a new house first!	--	--
slot_machines	Build a new bank first!	--	--
library	Build a new house first!	--	--
pharmacy	Build a new house first!	--	--

Figure 5.43: Output for a Region with No Buildings

Figure 5.43 shows the output for a region which is completely empty. For each suggestion, there are 0 of each type of *x*-object, so we should be returning different suggestions, with no improvement to HDI.

Note, that these 2 options are not exclusive, and may appear in one table. For example, in finding a place to put a new library, the region may not have any universities, and so nothing will be returned for that one, but it does have houses and so the majority of the other ones will have proper suggestions.

No.	Test	Input	Type	Expectation	Output
t_{96}	Suggestions for a Region with Many Objects	A Town	—	Multiple positions of buildings shown, HDI re-predicted to be better	As Expected
t_{97}	Suggestions for a Region with No Objects	A Field	—	Different Suggestions Shown	As Expected

Table 5.48: Testing Table for Code Snippets 5.68, 5.69, 5.70 & 5.71

The final thing we must implement is placing multiple pins on the map, when we select the table.

app.py

```

219 # place a pin on the map when user clicks the suggestion
220 def selectSuggestionsTable(evt: gr.SelectData):
221     cellData = evt.value
222     newMap = copy.deepcopy(foliumMap)
223     if '°' in cellData: # it is a coordinate
224         splitByCommas = cellData.split(', ')
225         coordinates = []
226         for halfIndex in range(int(len(splitByCommas)/2)):
227             coordinates.append([float(splitByCommas[2*halfIndex][1:-2]),
228                                 float(splitByCommas[2*halfIndex+1][:-3])])
229         for coordinate in coordinates:
230             folium.Marker(location=coordinate).add_to(newMap)
231         newMap.location=random.choice(coordinates)
232     return newMap

```

Code Snippet 5.72: Placing Multiple Pins on the Map

For this, we can do the same thing as before, but when we split every `' , '`, we have to be aware that both the coordinates, and the components of the coordinates have that same separator. Therefore, each coordinate will occupy 2 elements in this list generated by the `split` method. We must then iterate over a `halfIndex`, and add the elements of the list to a coordinates list (ensuring we remove any formatting using list splicing).

Once we have all of the coordinates, we can iteratively add them to the map, and set the location to one of them at random.

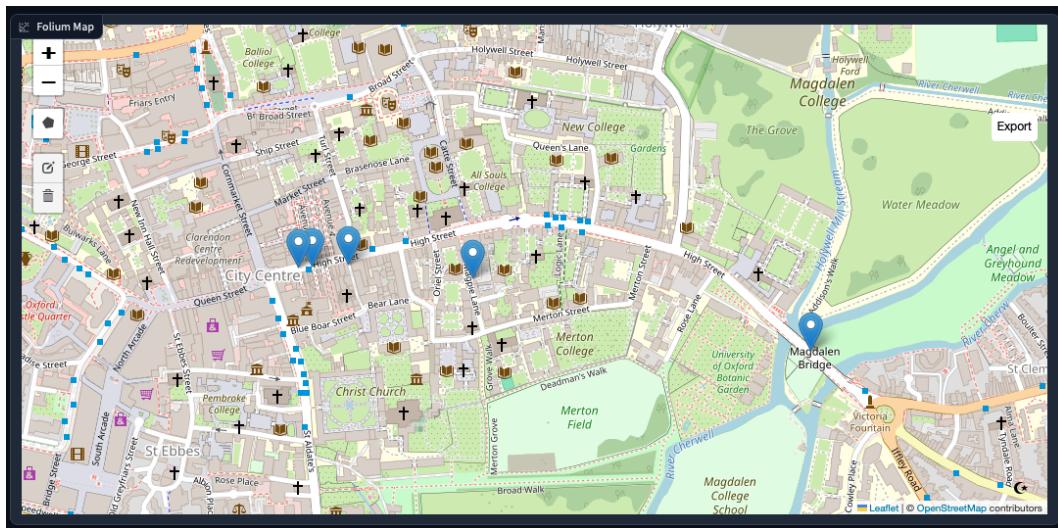


Figure 5.44: Output for Placing Pins on the Map

No.	Test	Input	Type	Expectation	Output
t_{98}	Placing Multiple Pins on the Map	—	—	Multiple Pins on the map in the right locations	As Expected

Table 5.49: Testing Table for Code Snippet 5.72

5.2.7 Part 22 – Finishing Touches

Bug Fix – Not Returning Sufficient Items

When testing out the program, I noticed that some functionalities were returning errors, instead of a prompt for the user to do something different.

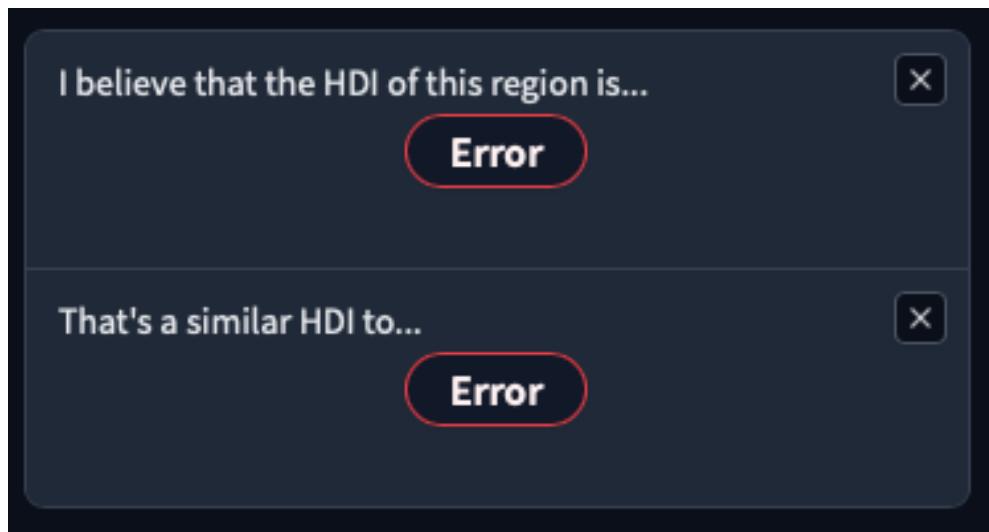


Figure 5.45: Screenshot of an Errored Output

Figure 5.45 shows the output for when we attempt to make a prediction before uploading any regions, which is an error, instead of a prompt. I then found that this was because I was not returning a sufficient number of values. For example, when we press the predict HDI button, we are expecting to output to the HDI textbox *and* the similar region textbox, and so we should return 2 items (e.g `return 'a', 'b'`).

app.py / `def makeSuggestions`
96 `return 'You have not uploaded a region!', ''`

Code Snippet 5.73: Fixing the Error for `def makeSuggestions`

The regular output, for when the user does everything correctly was returning 2 values, but the invalid output, for when the user has not yet uploaded a region only had 1 output, so I added another one to go to the similar regions textbox.

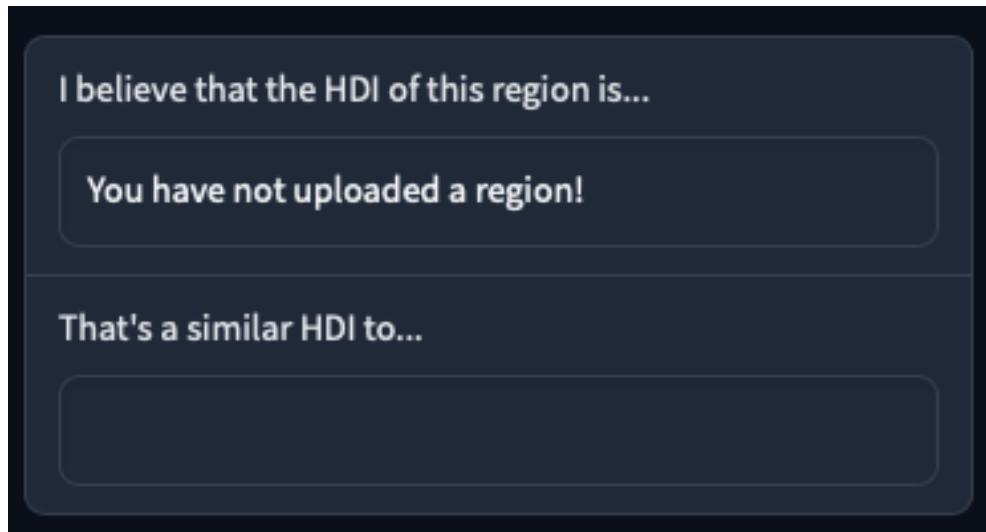


Figure 5.46: Screenshot of the Corrected Output (`def makeSuggestions`)

Figure 5.46 shows the corrected output.

No.	Test	Input	Type	Expectation	Output
t_{99}	Fixing Error on Predicting HDI before Uploading	—	—	Prompt to upload a region	As Expected

Table 5.50: Testing Table for Code Snippet 5.73

There was also a similar problem for the log in function, and the sign up function, where we were expecting to also update the regions table and the dropdown options, but only returned to the result box.

```
app.py / def signUp
```

```
323     return 'Please fill in all fields!', updateRegionsTable(),  
    ↵     updateRegionsDropdown()
```

```
app.py / def signUp
```

```
326     return 'This username is already in use! Please choose another  
    ↵     username.', updateRegionsTable(), updateRegionsDropdown()
```

```
app.py / def signUp
```

```
329     return 'The passwords do not match! Please ensure that you have  
    ↵     entered the correct password.', updateRegionsTable(),  
    ↵     updateRegionsDropdown()
```

```
app.py / def signUp
```

```
332     return 'Your password must have at least 8 characters!',  
    ↵     updateRegionsTable(), updateRegionsDropdown()
```

```
app.py / def signUp
```

```
340     return 'Your password must include at least 1 digit!',  
    ↵     updateRegionsTable(), updateRegionsDropdown()
```

```
app.py / def signUp
```

```
348     return f'Your password must include at least 1 special character!  
    ↵     ({specialChars})', updateRegionsTable(),  
    ↵     updateRegionsDropdown()
```

Code Snippet 5.74: Fixing the Error for `def signUp`

We can fix this by also returning the result of the `def updateRegionsTable` and `def updateRegionsDropdown` functions, on the invalid returns.

```
app.py / def logIn
```

```
368     return 'Please fill in all fields!', updateRegionsTable(),
    ↵     updateRegionsDropdown()
```

```
app.py / def logIn
```

```
371     return 'This username does not exist! Please ensure that you have
    ↵     entered the correct username.', updateRegionsTable(),
    ↵     updateRegionsDropdown()
```

```
app.py / def logIn
```

```
375     return 'Incorrect Password! Please try again.',
    ↵     updateRegionsTable(), updateRegionsDropdown()
```

Code Snippet 5.75: Fixing the Error for `def logIn`

The `def logIn` function had the exact same issue, so I fixed it in the exact same way

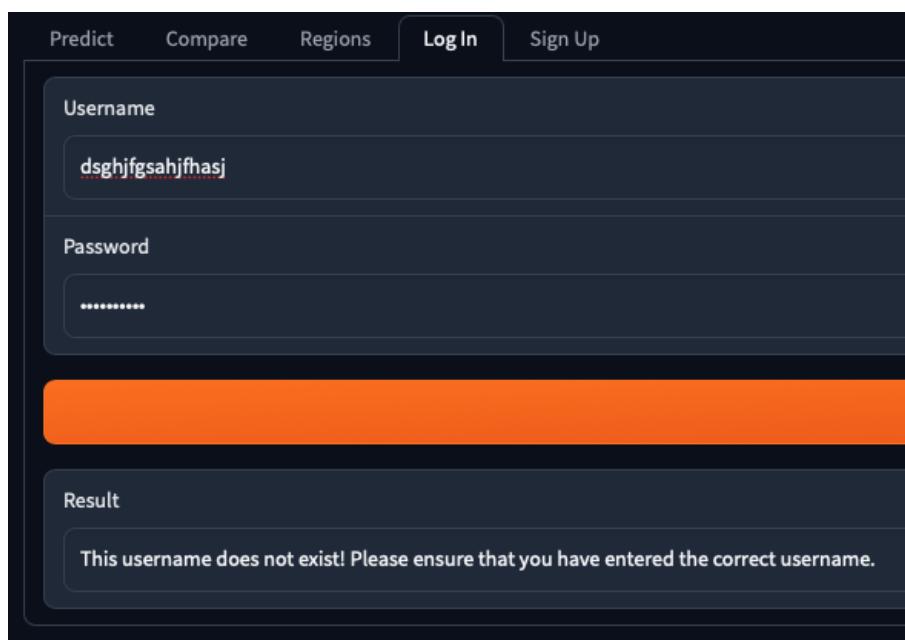


Figure 5.47: Screenshot of the Corrected Output (`def logIn`)

Figure 5.47 shows the corrected output for the `def logIn` function (which before was returning the same error).

No.	Test	Input	Type	Expectation	Output
t_{100}	Fixing Error on Signing Up & Logging In	–	–	Prompt to enter correct/valid credentials	As Expected

Table 5.51: Testing Table for Code Snippets 5.74 & 5.75

Bug Fix – New Account Doesn't Have Current Regions

I also noticed that creating a new account with existing regions did not add these regions to the account. I discovered that this was because I had accidentally kept '`regions': []`', when creating a new account, instead of updating it to `allUserRegions`, when I made that list.

`app.py / def signUp`

356

```
'regions': allUserRegions
```

Code Snippet 5.76: Adding Current Regions to the Account

To fix this, all I needed to do was pass `allUserRegions` into the dictionary instead of `[]`.

No.	Test	Input	Type	Expectation	Output
t_{101}	Signing Up Creates an Account with the User's Regions	–	–	Signing Up Creates an Account with the User's Regions	As Expected

Table 5.52: Testing Table for Code Snippet 5.76

Bug Fix – Switching Between Regions Doesn't Work with Comparing

Another bug that I noticed is that the Compare Tab doesn't function as intended after switching between regions on the Regions Tab.

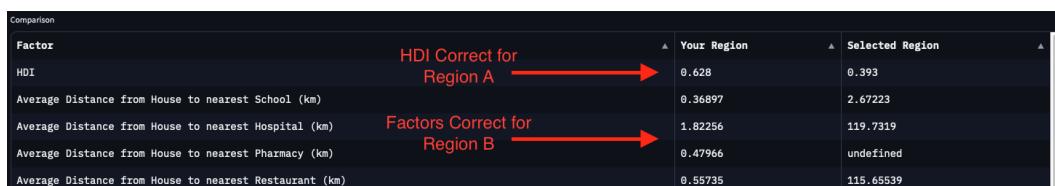


Figure 5.48: HDI is not updating when Comparing

Figure 5.48 illustrates the problem, where if I switch from region A to region B , and then compare the current region to one in the training data, the factors are correct for region B , but the HDI is for region A .

I found that this is because that cell that isn't being updated is based on the `global` variable, `currentHDI`, which is only updated when the user pressed the predict HDI button on the Predict Tab. Therefore, we should also update it when the current region is changed from the dropdown menu, in the `def updateCurrentRegion` function.

`app.py / def updateCurrentRegion`

```
313     global currentHDI
```

`app.py / def updateCurrentRegion`

```
318         # predict HDI - to update it internally
319         inputLayer = np.matrix([[100 if factor == None else
320             float(factor)/10 if index <= 11 else float(factor)] for
321             index, factor in enumerate(allFactors)])
320         prediction = round(network.predict(inputLayer).item(0,0), 3)
321         currentHDI = prediction
```

Code Snippet 5.77: Updating `currentHDI`

First, we must ensure that we are using the `global` variable, `currentHDI`, and then after we update `all_objects` and `allFactors`, we can re-predict the HDI in the same way as we have done before.

Comparison		HDI Correct for Region B	Your Region	Selected Region
Factor	HDI	0.836	0.433	
Average Distance from House to nearest School (km)		0.36466	6.3723	
Average Distance from House to nearest Hospital (km)	Factors Correct for Region B	1.80321	73.84311	
Average Distance from House to nearest Pharmacy (km)		0.50593	98.37357	
Average Distance from House to nearest Restaurant (km)		0.56786	99.5008	

Figure 5.49: HDI is now updating when Comparing

Figure 5.49 shows the output now, for the same regions A and B . The HDI it displays is now correct.

No.	Test	Input	Type	Expectation	Output
t_{102}	HDI Updates when we Switch Regions	Switch from region A to B	—	When Predicting, it displays region B 's HDI	As Expected

Table 5.53: Testing Table for Code Snippet 5.77

Info Popups

Gradio also supports the use of Info & Warning Popups, which will appear on the top right of the screen for a short number of seconds. We can program them to say whatever we want, for example, in the `def processPrediction` function, we can add the following lines right before we make a return:

```
app.py / def processPrediction
```

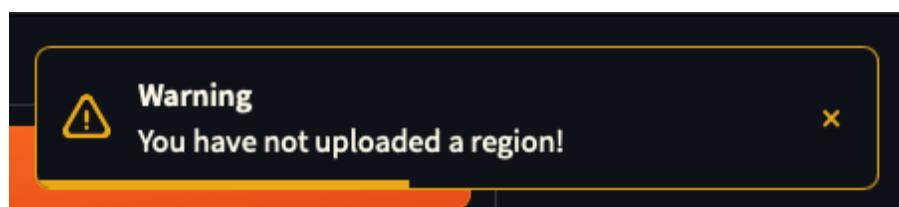
```
96     gr.Warning('You have not uploaded a region!')
```

```
app.py / def processPrediction
```

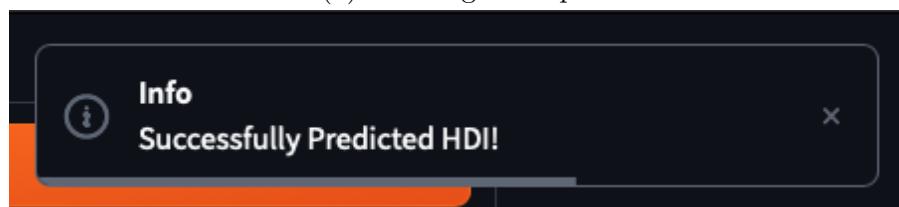
```
114    gr.Info('Successfully Predicted HDI!', 5)
```

Code Snippet 5.78: Info Popups in `def processPrediction`

The first being before the return when the user has not yet uploaded a region, and the second being if the function has successfully ran and the HDI has been predicted.



(a) Warning Example



(b) Info Example

Figure 5.50: Info Popup Examples

Figure 5.50 shows how the popups look. You may have also noticed the extra 5 being passed into the info popup, which is to specify how long it stays on the screen for. By default it is 10 seconds, but when the message just shows a success, the message does not need to stay for very long, as the user does not need to do anything about it (so I decreased it to 5 seconds). The bar at the bottom of the popup shows how long is left until it disappears, with the bar initially being full then moving to the left.

No.	Test	Input	Type	Expectation	Output
t_{103}	Warning Popup	An Invalid Input	—	Yellow Box appears in the Top Right Corner, for 10 seconds, displaying useful information	As Expected
t_{104}	Info Popup	An Valid Input	—	Gray Box appears in the Top Right Corner, for 5 seconds, displaying a success message	As Expected

Table 5.54: Testing Table for Code Snippet 5.78

However, while I was adding this feature, I found that if I logged into my account, and then selected a region from there, and then predicted the HDI (as opposed to uploading one using the upload button), I still got the yellow warning box, instead of a predicted HDI.

I found that this was because I was using `currentRegion` to check if the user has a region selected or not, which was sensible in the old function arrangement, but now it makes more sense for it to be `allFactors`, as that is what it is using to make a prediction.

```
app.py / def processPrediction
95     if allFactors == []:
96         gr.Warning('You have not uploaded a region!')
97         return 'You have not uploaded a region!', ''
```

Code Snippet 5.79: Ensuring `def processPrediction` works with Logging In

Replacing `currentRegion` with `allFactors` fixed this issue.

No.	Test	Input	Type	Expectation	Output
t_{105}	Predicting HDI from a region in an Account	Region in an account	—	HDI is predicted	As Expected

Table 5.55: Testing Table for Code Snippet 5.79

Now that I had fixed this minor error, I could add the popups for the rest of the functions.

```
app.py / def uploadGeoJSON
```

```
59     gr.Warning('There is already a region with this name! Please
    ↳   rename the file first.')
```

```
app.py / def uploadGeoJSON
```

```
91     gr.Info(f'Successfully Uploaded {shortFilename}', 5)
```

Code Snippet 5.80: Info Popups in `def uploadGeoJSON`

For `def uploadGeoJSON`, there was only the success message and the error for when the user attempts to upload a region with an existing name.

```
app.py / def makeSuggestions
```

```
173    gr.Warning('You have not yet predicted an HDI! Please predict one
    ↳   first.')
```

```
app.py / def makeSuggestions
```

```
176    gr.Warning('This region does not have any associated buildings!
    ↳   Please choose a different region')
```

```
app.py / def makeSuggestions
```

```
225    gr.Info('Successfully Made Suggestions!', 5)
```

Code Snippet 5.81: Info Popups in `def makeSuggestions`

For `def makeSuggestions`, there was the success message, as well as 2 possible errors, depending on the status of the current region.

```
app.py / def compareRegions
```

```
244    gr.Warning('Please predict a region first!')
```

Code Snippet 5.82: Info Popups in `def compareRegions`

For `def compareRegions`, I decided not to include a success popup, as the response from selecting a region from the dropdown menu is almost instant, where the other processes take

tens of seconds. The only popup I had to implement is the warning where the user hasn't got any region selected, and so cannot compare.

```
app.py / def updateAllUserRegions
```

```
285     gr.Warning('Please ensure every region has a unique name!')
```

```
app.py / def updateAllUserRegions
```

```
294     gr.Warning(f'Please Enter the Data Correctly! Found an  
      ↳ error at cell (row {rowIndex}, col {cellIndex})')
```

```
app.py / def updateAllUserRegions
```

```
317     gr.Info('Successfully Updated Regions', 5)
```

Code Snippet 5.83: Info Popups in `def updateAllUserRegions`

For `def updateAllUserRegions` (which is called when the user pressed the “Save Table Changes” button on the Regions Tab), I added the success message, as well as the 2 possible errors in validating the user’s input.

```
app.py / def signUp
```

```
339     gr.Warning('A field has been left empty! Please fill in all  
    ↳   fields.')
```

```
app.py / def signUp
```

```
343     gr.Warning('This username is already in use! Please choose another  
    ↳   username.')
```

```
app.py / def signUp
```

```
347     gr.Warning('The passwords do not match! Please ensure that you  
    ↳   have entered the correct password.')
```

```
app.py / def signUp
```

```
351     gr.Warning('Your password must have at least 8 characters!')
```

```
app.py / def signUp
```

```
360     gr.Warning('Your password must include at least 1 digit!')
```

```
app.py / def signUp
```

```
369     gr.Warning('Your password must include at least 1 special  
    ↳   character!')
```

```
app.py / def signUp
```

```
382     gr.Info('Successfully created & logged into an account', 5)
```

Code Snippet 5.84: Info Popups in `def signUp`

For `def signUp`, there are many restrictions on the user's username and password, which meant I had to add 6 warning popups, as well as a success info popup at the end.

```
app.py / def logIn
```

```
391     gr.Warning('A field has been left empty! Please fill in all
    ↵   fields.')
```

```
app.py / def logIn
```

```
395     gr.Warning('This username does not exist! Please ensure that you
    ↵   have entered the correct username.')
```

```
app.py / def logIn
```

```
400     gr.Warning('Incorrect Password! Please try again.')
```

```
app.py / def logIn
```

```
405     gr.Info(f'Successfully logged in as {username}!', 5)
```

Code Snippet 5.85: Info Popups in `def logIn`

For `def logIn`, there aren't as many things we need to check, but we still need to ensure that the user has entered the correct information, and if not, create a suitable warning popup. Otherwise, we can do another success message.

No.	Test	Input	Type	Expectation	Output
t_{103}	Warning Popup	An Invalid Input	–	Yellow Box appears in the Top Right Corner, for 10 seconds, displaying useful information	As Expected
t_{104}	Info Popup	An Valid Input	–	Gray Box appears in the Top Right Corner, for 5 seconds, displaying a success message	As Expected

Table 5.56: Testing Table for Code Snippets 5.80, 5.81, 5.82, 5.83, 5.84 & 5.85

Emojification

The next thing I wanted to do to improve the user interface is add emojis to each of the labels of the components. I found a website ([12]) which allowed me to search for emojis that matched a given keyword, even if that word wasn't part of the name of the emoji.

app.py

```
431     with gr.Tab(label='\U0001F52E Predict'):
```

app.py

```
449     with gr.Tab(label='\U0001F19A Compare'):
```

app.py

```
453     with gr.Tab(label='\U0001F30D Regions'):
```

app.py

```
458     with gr.Tab(label='\U0001F513 Log In'):
```

app.py

```
463     with gr.Tab(label='\U0001F4DD Sign Up'):
```

Code Snippet 5.86: Emojis in the Tabs

Using this website, I found suitable emojis for each of the tabs. I added the emoji at the start, by using their respective unicode codes, ensuring we have a space inbetween the emoji and the name (otherwise it will look too squished together).

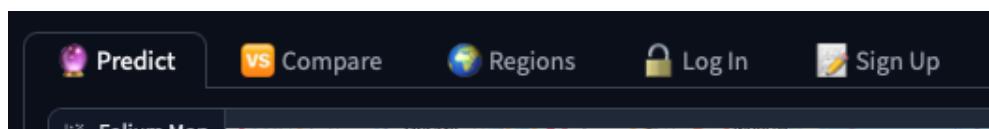


Figure 5.51: Emojis in Tabs

Figure 5.51 shows the new appearance of the tabs, with the emojis. I believe that this is a very good usability feature, as if the user does not speak English, they may now be able to have a vague understanding of what each tab may contain.

I was happy with how this looked, and so I added it to the rest of the content in the tabs.

(Note, the `info` attributes of the sliders on lines 439 and 440 have been removed for this code snippet, as they are quite long)

`app.py`

```

431     with gr.Tab(label='\U0001F52E Predict'):
432         with gr.Row():
433             with gr.Column(scale=3):
434                 map = Folium(value=foliumMap)
435                 uploadButton = gr.UploadButton(label='\U0001F4E4 Upload
436                     ↳ the GeoJSON file \U0001F4E4', file_count='single')
437                 with gr.Column(scale=1):
438                     predictHDIbutton = gr.Button(value='\U0001F52E Predict HDI
439                         ↳ \U0001F52E', variant='primary')
440                     with gr.Group():
441                         max_x_objectsSlider = gr.Slider(minimum=10,
442                             ↳ maximum=5000, value=500, label='Maximum
443                             ↳ x-buildings \U0001F3E0', info='...')
444                         max_y_objectsSlider = gr.Slider(minimum=10,
445                             ↳ maximum=5000, value=500, label='Maximum
446                             ↳ y-buildings \U0001F3EB', info='...')
447                     with gr.Group():
448                         HDIprediction = gr.Textbox(label='I believe that the
449                             ↳ HDI of this region is... \U00002728', value='',
450                             interactive=False)
451                         similarHDI = gr.Textbox(label='That\'s a similar HDI
452                             ↳ to... \U0001F30D', value='', interactive=False)
453                     with gr.Row():
454                         makeSuggestionsButton = gr.Button(value='\U0001F914 Make
455                             ↳ Suggestions \U0001F914')
456                         newBuildingsSlider = gr.Slider(minimum=1, maximum=10, value=5,
457                             ↳ step=1, label='Number of New Buildings \U0001F3D9',
458                             ↳ interactive=True)
459                         suggestionsTable = gr.Dataframe(label='Suggestions \U0001F4A1',
460                             ↳ headers=['New Building', 'Coordinates', 'New HDI', 'Change in
461                             ↳ HDI'], type='array', interactive=False)
462                         log = gr.Textbox(label='Log Messages \U0001F4C4', value='',
463                             ↳ interactive=False)

```

Code Snippet 5.87: Emojis in the Predict Tab

While adding the emojis for this tab, I decided on a standard, which is that tabs will have the emoji before the label, most components will have the emoji after the label, and buttons will have the emoji either side of the label. This is to draw attention to the buttons, which is the main source of input.

This was the biggest tab to fill with emojis, as I had to decide on one for the predict button, the maximum x -objects and y -objects sliders, the HDI output box, the similar region output box, the upload button, the make suggestions button, the number of new buildings slider, the suggestions table and the log messages.

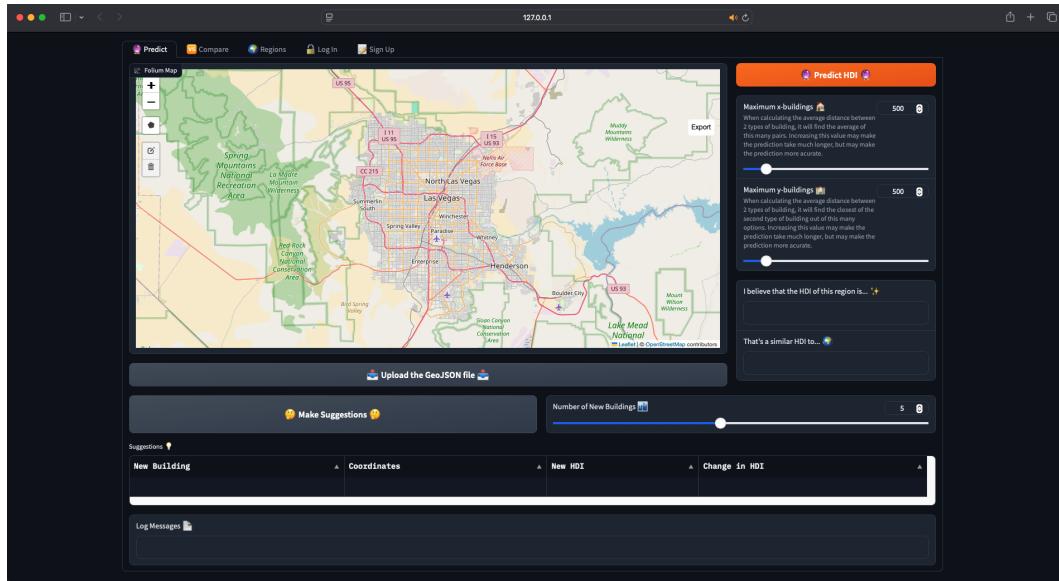


Figure 5.52: Emojis in the Predict Tab

app.py

```

449     with gr.Tab(label='\U0001F19A Compare'):
450         compareDropdown = gr.Dropdown(choices=[f'{region["region"]}', 
451                                         f'{region["country"]}' for region in trainingData], 
452                                         label='Select a Region \U0001F30D')
453         compareTable = gr.DataFrame(label='Comparison \U0001F4CA',
454                                     headers=['Factor', 'Your Region', 'Selected Region'],
455                                     type='array', interactive=False)
456         pmccImage = gr.Image('data/pmcc.png', label='Correlation between
457                               Factors \U0001F4C8', height=600, interactive=False)

```

Code Snippet 5.88: Emojis in the Compare Tab

For the compare tab, I only had to find ones for the region dropdown menu, the comparison table, and the factor heatmap. For the region dropdown menu, I decided to use the same emoji as for the regions tab, as I believe this will create a sense of cohesion around the app.

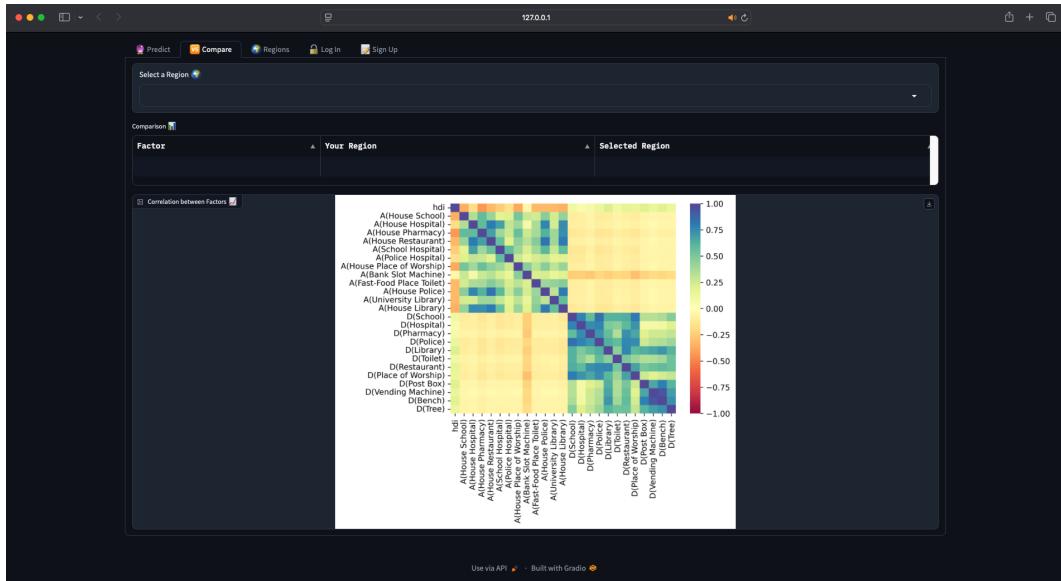


Figure 5.53: Emojis in the Compare Tab

app.py

```

453     with gr.Tab(label='\U0001F30D Regions'):
454         regionsDropdown = gr.Dropdown(choices=[], label='Current Region
455             \U0001F30D', interactive=True)
456         regionsTable = gr.DataFrame(label='Your Regions \U0001F304',
457             headers=['Name'] + list(trainingData[0].keys())[4:], type='array', col_count=(25, 'fixed'), interactive=True)
458         submitEditsButton = gr.Button(value='\U0001F504 Submit Table
459             Changes \U0001F504', variant='primary')
460         submitEditsResult = gr.Textbox(label='Result \U0001F4C4',
461             interactive=False)

```

Code Snippet 5.89: Emojis in the Regions Tab

For the regions tab, I needed emojis for the region dropdown menu, the table of the user's regions, the submit table changes button, and the result box. I again used the same emoji for the dropdown, and I also used the same emoji for the result box as the log messages in the Predict Tab, to further the continuity.

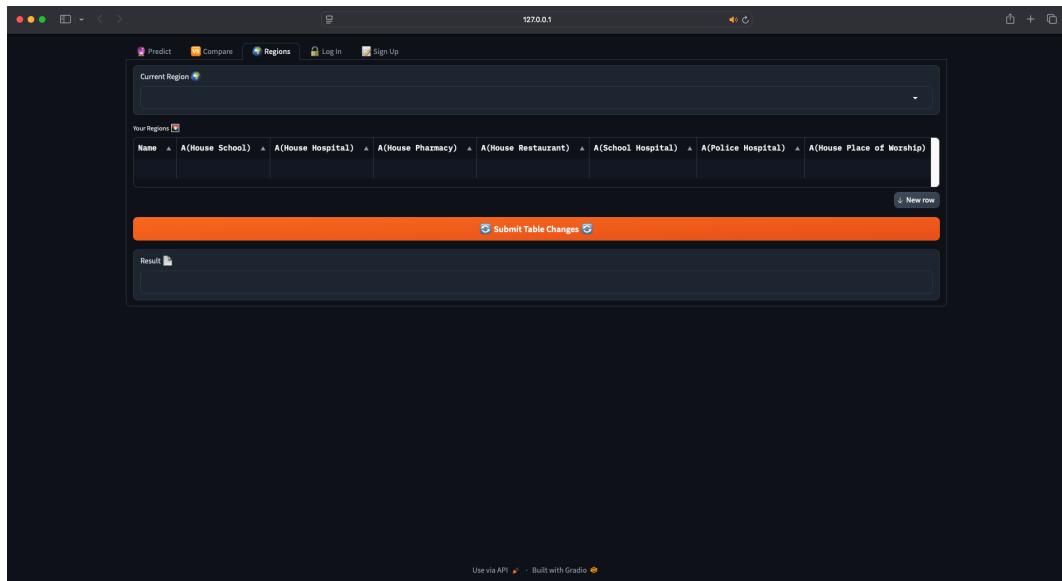


Figure 5.54: Emojis in the Regions Tab

app.py

```

458     with gr.Tab(label='\U0001F513 Log In'):
459         logInUsername = gr.Textbox(label='Username \U0001F50F',
460             placeholder='Enter your username... ')
461         logInPassword = gr.Textbox(label='Password \U0001F511',
462             placeholder='Enter your password...', type='password')
463         logInButton = gr.Button(value='\U0001F4F2 Log In \U0001F4F2',
464             variant='primary')
465         logInResult = gr.Textbox(label='Result \U0001F4C4',
466             interactive=False)

```

Code Snippet 5.90: Emojis in the Log In Tab

For the log in tab, I needed new emojis for the username, password and log in button (result box re-uses the same one from the region tab and the predict tab).

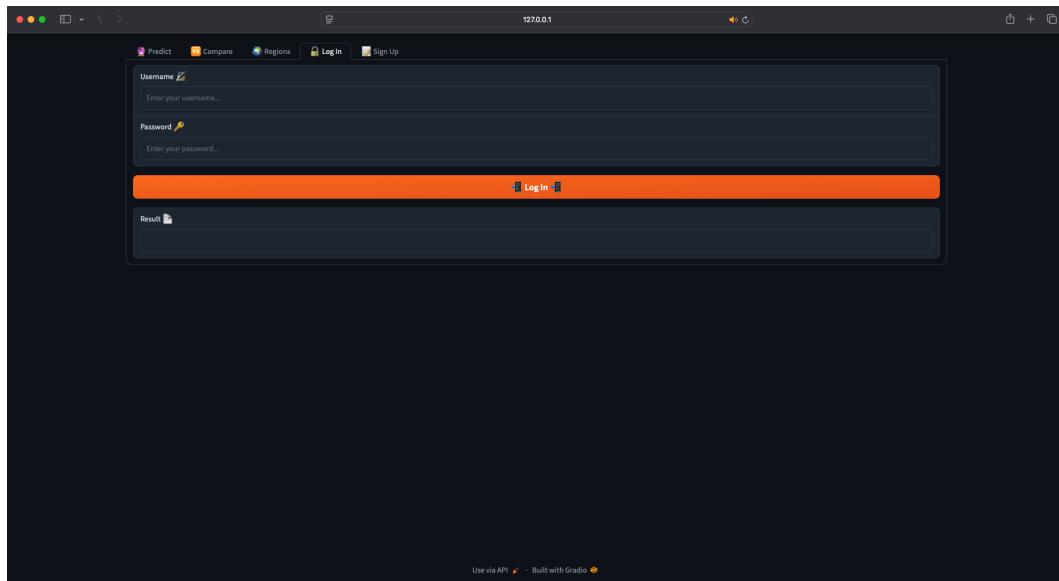


Figure 5.55: Emojis in the Log In Tab

app.py

```

463     with gr.Tab(label='\U0001F4DD Sign Up'):
464         signUpUsername = gr.Textbox(label='Username \U0001F50F',
465             placeholder='Enter your username... ')
466         signUpPassword = gr.Textbox(label='Password \U0001F511',
467             placeholder='Enter your password...', type='password')
468         signUpPassword2 = gr.Textbox(label='Re-Enter Password \U0001F510',
469             placeholder='Re-Enter your password...', type='password')
470         signUpButton = gr.Button(value='\U0001F680 Create an Account',
471             variant='primary')
472         signUpResult = gr.Textbox(label='Result \U0001F4C4',
473             interactive=False)

```

Code Snippet 5.91: Emojis in the Sign Up Tab

Finally, for the sign up tab, I only needed new emojis for the re-entering of the password, and the create account button (the others are re-used from the log in tab).

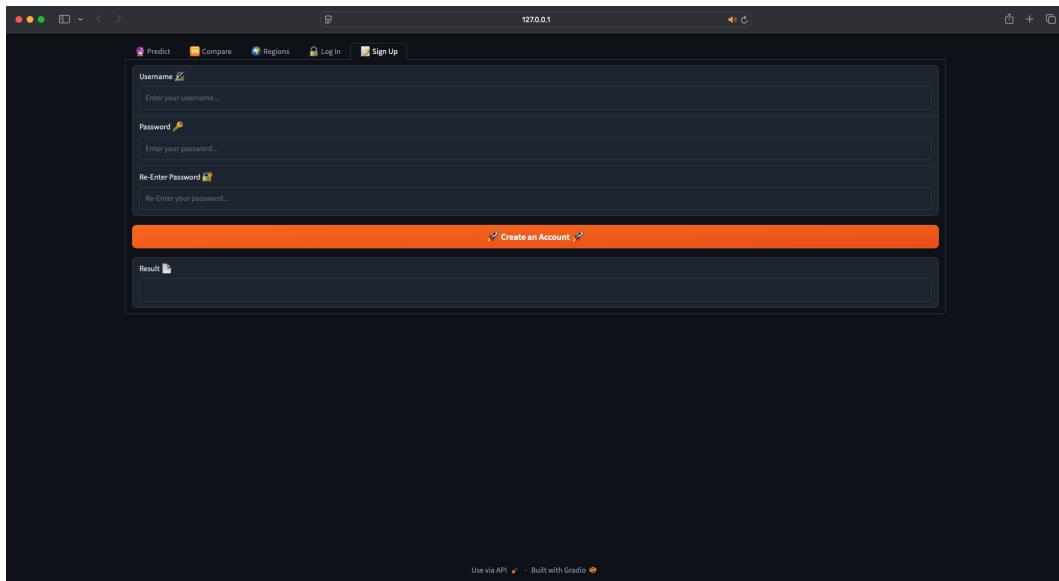


Figure 5.56: Emojis in the Sign Up Tab

No.	Test	Input	Type	Expectation	Output
t_{106}	Emojification of the User Interface	—	—	Each element in the interface as an emoji relating to its function	As Expected

Table 5.57: Testing Table for Code Snippets 5.86, 5.87, 5.88, 5.89, 5.90 & 5.91

Other Interface Changes

After I added all of the emojis, I thought that the log in & sign up tabs looked too plain and monotonous, as it was just 4 or 5 elements stacked on top of each other. I therefore decided that I wanted to put the “submit” button and the return textbox in a different column, next to the input fields.

app.py

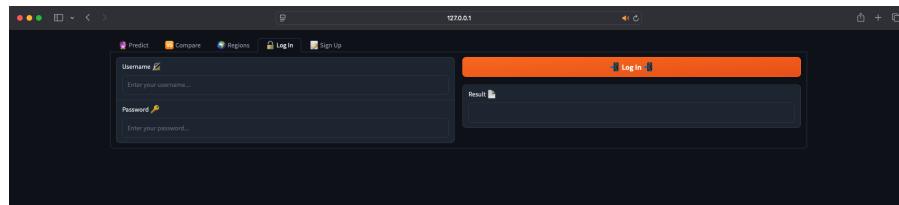
```

458     with gr.Tab(label='\U0001F513 Log In'):
459         with gr.Row():
460             with gr.Column():
461                 logInUsername = gr.Textbox(label='Username \U0001F50F',
462                     placeholder='Enter your username...')
462                 logInPassword = gr.Textbox(label='Password \U0001F511',
463                     placeholder='Enter your password...', type='password')
463             with gr.Column():
464                 logInButton = gr.Button(value='\U0001F4F2 Log In
465                     \U0001F4F2', variant='primary')
465                 logInResult = gr.Textbox(label='Result \U0001F4C4',
466                     interactive=False)
466         with gr.Tab(label='\U0001F4DD Sign Up'):
467             with gr.Row():
468                 with gr.Column():
469                     signUpUsername = gr.Textbox(label='Username \U0001F50F',
470                         placeholder='Enter your username...')
470                     signUpPassword = gr.Textbox(label='Password \U0001F511',
471                         placeholder='Enter your password...', type='password')
471                     signUpPassword2 = gr.Textbox(label='Re-Enter Password
472                         \U0001F510', placeholder='Re-Enter your password...', type='password')
472                 with gr.Column():
473                     signUpButton = gr.Button(value='\U0001F680 Create an
474                         \U0001F680', variant='primary')
474                     signUpResult = gr.Textbox(label='Result \U0001F4C4',
475                         interactive=False)

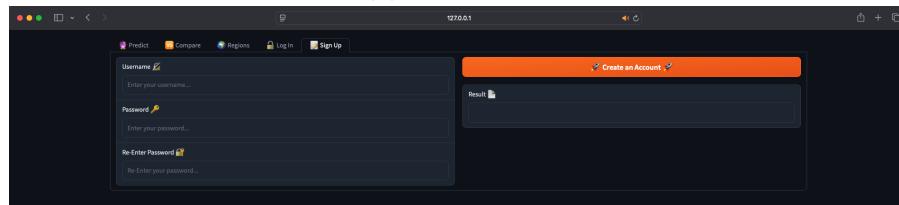
```

Code Snippet 5.92: Restructuring the Log In & Sign Up Tabs

To do this, we can put the button and the result textbox in a column, and the fields in a column, and put those columns in a row. The same can be done for the sign up page, as well as the log in page.



(a) Log In Tab



(b) Sign Up Tab

Figure 5.57: Restructuring the Log In & Sign Up Tabs

Figure 5.57 shows the new appearances of the log in and sign up tabs, which I believe are better and more interesting to look at.

No.	Test	Input	Type	Expectation	Output
t_{107}	Restructuring Authentication Appearance	—	—	Button and Result Textbox on the Side	As Expected

Table 5.58: Testing Table for Code Snippet 5.92

I also thought that I should add a few warnings, for some behaviour that may be unexpected.

app.py

```
457     gr.Markdown('`\U00002757 **WARNING:** You will no longer be able to  
    ↵   make suggestions from regions that you edit here. \U00002757`')
```

app.py

```
466     gr.Markdown('`\U00002757 **WARNING:** Logging in will  
    ↵   override the regions in the \U0001F30D Regions Tab,  
    ↵   and replace them with the ones in the account you log  
    ↵   into. \U00002757`')
```

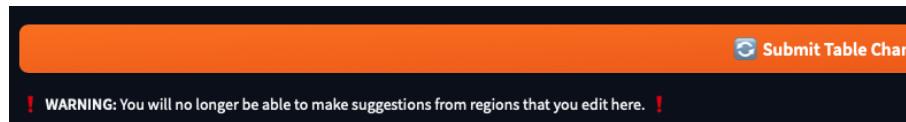
app.py

```
474         gr.Markdown(''  
475             Your password must:  
476                 - Be at least 8 characters  
477                 - Contain at least 1 digit  
478                 - Contain at least 1 special character  
479             '')')
```

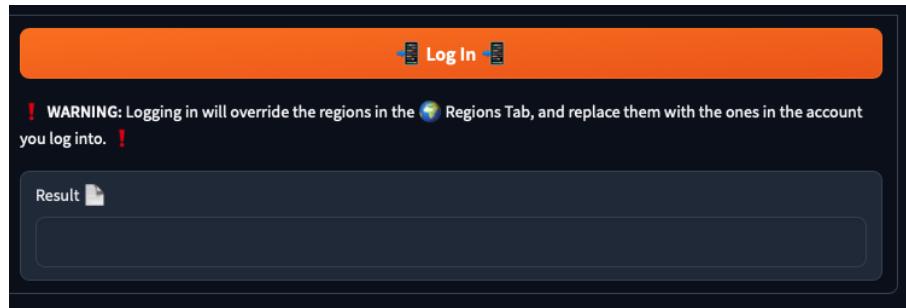
Code Snippet 5.93: Warning Messages

I thought that the user should know that editing regions in the table directly will cause them to no longer be able to make suggestions, and that if they log into an account, their regions will be replaced by the ones in the account. I also thought that the user should know the password requirements, before they type a password.

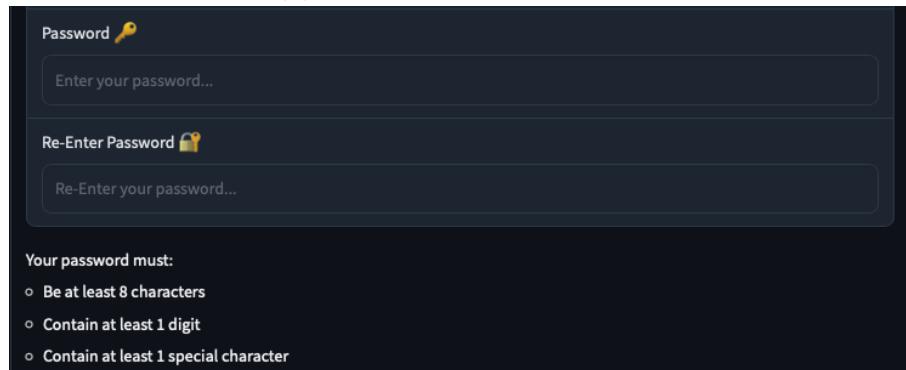
To do this, I used `gr.Markdown`, which allows us to add simple text to the interface, and format it with markdown (for example, here, writing `'**WARNING:**'` made “WARNING:” bold).



(a) Update Regions Table



(b) Logging Into an Account



(c) Sign Up Password Requirements

Figure 5.58: Adding Warnings

No.	Test	Input	Type	Expectation	Output
t_{108}	Warning Notes	–	–	Short Warning Message below buttons with potentially unexpected behaviour	As Expected

Table 5.59: Testing Table for Code Snippet 5.93

The Help Tab

The final thing to add to the user interface is the help tab. I wanted to write this last, as I wanted to be sure that all of the other features were in their final state.

app.py

```

429 # define markdown in help tab
430 markdown = '''
431 this is a test that this markdown has been rendered correctly
432
433 # this text should be a heading with the crystal ball emoji \U0001F52E
434
435 - this text should be in
436 - bullet points
437
438 ![] (data/pmcc.png "Alt Text")
439 '''

```

app.py

```

443     with gr.Tab(label='\U0001F64B Help'):
444         gr.Markdown(markdown)

```

Code Snippet 5.94: Adding the Help Tab

Here, I am adding the Help Tab. In the design section, I had put the help tab at the end of the interface, but I have since decided to put it as the first tab. This is because this is what the user will immediately see, and so they should use this to get started.

For now, I have written a small sample to test out some of the markdown formatting features. The test includes a regular paragraph, a heading, an emoji, bullet points, and an image with alt-text.

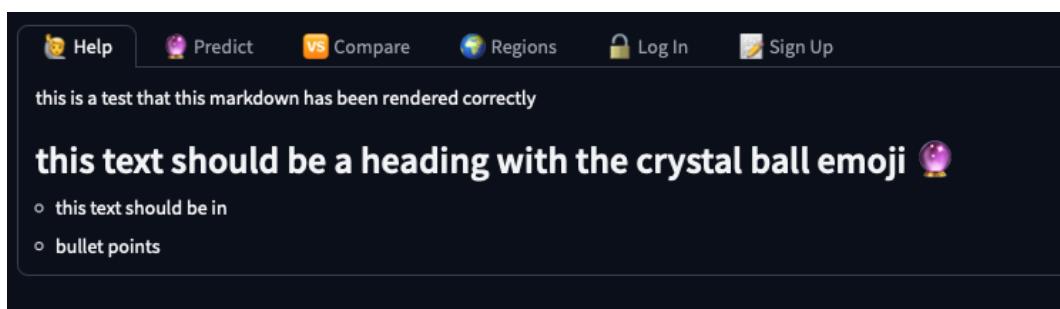


Figure 5.59: Markdown Test Initial Attempt

Figure 5.59 shows the current output for the help tab. Everything seems to have worked except for the image.

No.	Test	Input	Type	Expectation	Output
t_{109}	Help Tab	–	–	Markdown is Rendered Correctly	No Images

Table 5.60: Testing Table for Code Snippet 5.94

I did not know how to resolve this issue, so I researched the problem. I found someone who was having a similar problem to me, who had asked how to fix this problem in the gradio GitHub repository [13].

(Note, the declaration in line 437 is just to show that this is part of the string. In reality, this line is still line 430)

app.py

```
437 markdown = '''
438 
439 '''
```

app.py

```
509 # launch the UI
510 app.launch(allowed_paths=['/'])
```

Code Snippet 5.95: Allowing for Images in Markdown

After reading the responses, I found that I needed to add '`file/`' to the start of the file path, as well as pass `allowed_paths=['/']` into `app.launch()` at the very end of the program.

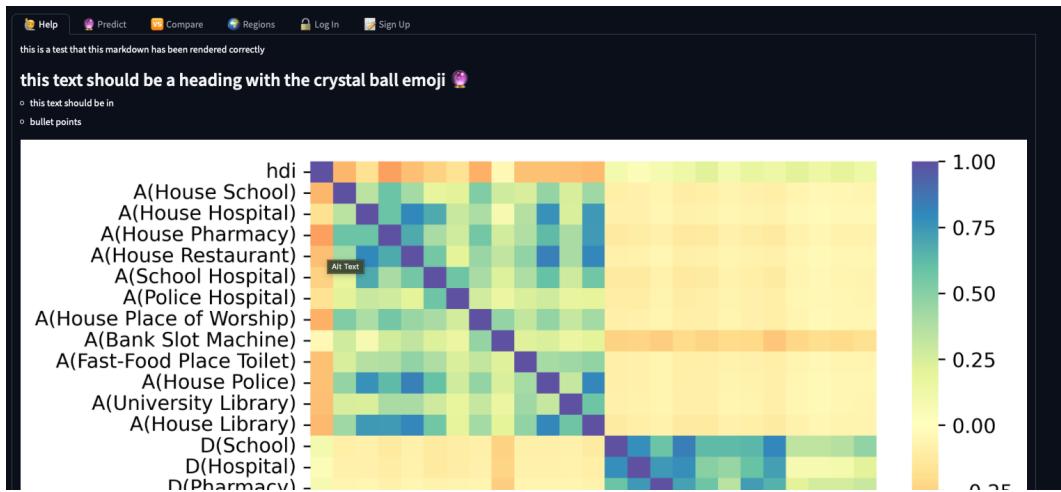


Figure 5.60: Markdown Test Successful

Figure 5.60 shows the correct output for this test.

No.	Test	Input	Type	Expectation	Output
t_{109}	Help Tab	—	—	Markdown is Rendered Correctly	As Expected

Table 5.61: Testing Table for Code Snippet 5.95

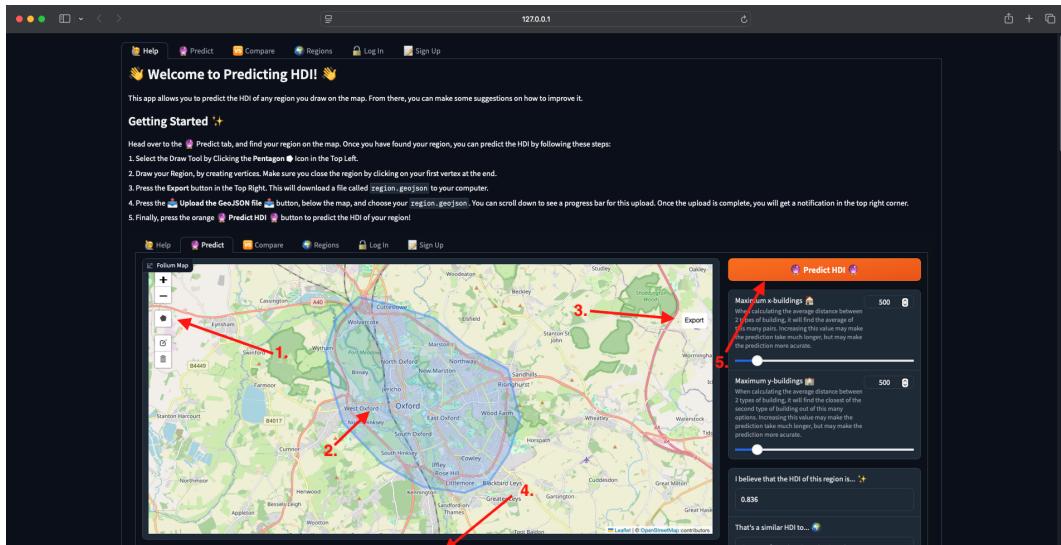


Figure 5.61: Help Tab

Figure 5.61 shows the top part of the appearance of the Help Tab, now that I have written the full text. I will not include it here, as it is quite long, but it will be included in the Final Code Section (Appendix A).

5.2.8 Review

No.	Test	Inputs	Pass / Fail
T_{13}	Calculating PMCC	The Training Data, which includes regions with full sets of factors, and some with some missing factors	Pass
T_{14}	Finding a Similar Region	HDI in the training data, HDI not in the training data, HDI not in the training data, with a tie for closest	Pass
T_{15}	Validating User Manually Entering Factors	Region with no name, Region with duplicate name, String in Density Factor, Non-Empty String in Average Distance Factor	Pass
T_{16}	Improved Suggestions	Region for which there exists 0 of each type of x -object, Region for which there exists 0 of a particular type of y -object, Region for which there exists an x -object in the same place as a y -object, Region which does not satisfy any of the above edge cases	Pass

Table 5.62: Passed Testing Data for Milestone 5

All 4 of the main tests in Milestone 5 passed. As a result of that, the following success criteria has been met:

No.	Success Criterion	Justification	Success
S_{14}	User can compare a region to one in the training data	This will allow users to make a judgement on how certain regions have the HDI that they have, and how they can improve their own.	✓
S_{15}	User can see how each factor can affect the others	This will allow the user to consider the secondary effects of their decisions, as the suggested changes may not only increase the HDI, but also some of the other factors.	✓

S_{16}	Predicted HDI is ranked among other countries with a similar HDI	This will allow users who are not that familiar with HDI to get an idea of the HDI scale by comparing it to the gist of what they know.	✓
S_{17}	User can save regions	This will allow users to save time and not re-download the buildings & re-calculate the factors. Users with an account should also be able to store regions there.	✓
S_{18}	User can manually enter factors	If downloading & calculating the factors takes too long, the user may manually enter them instead.	✓

Table 5.63: Achieved Success Criteria for Milestone 5

I also have a full user interface, with all of the desired features, so overall, I believe this milestone has been a success.

I would now ask my stakeholders for feedback, as I did at the end of the 1st prototype, but this will instead be done while Testing for Usability (Section 6.1.2).

6. Evaluation

6.1 Testing to Inform Evaluation

6.1.1 Testing for Function & Robustness

To carry out these tests, I recorded a short video, which can be found [here](#). The timestamps at which I do each test are in the table below. I mention some in the video, but did not for all of them.

Testing for Function

Again, each test for function matches with a success criterion, and will be used to determine if that success criterion has been met. Success criteria S_1 to S_7 are to do with the backend, and so are not tested here (but instead, during development). The fact that the features presented here work is enough evidence to suggest that the necessary backend requirements are met (for example, calculating the distance between 2 points).

No.	Test	Expected Output/Justification	Pass / Fail
FT_1	User can Select an Area (S_8)	User should be able to draw their region on the map, with little restrictions. It can be any polygon that doesn't self-intersect. If the user makes a mistake, they can undo or reset.	Pass, 1:11
FT_2	Neural Network Accurately Predicts HDI (S_9)	Once the user has selected their region, they should be able to predict the HDI of it. An accurate prediction will reflect the region drawn, so I will be looking for regions with less infrastructure to have a lower HDI than those with more infrastructure.	Partial Pass, 2:20

FT_3	Changes are Suggested to Increase HDI (S_{10} & S_{11})	Once the user has predicted the HDI of a region, they should also be able to make suggestions on how to improve it. The program should calculate a number of positions to put new buildings, and re-predict the HDI. I will be looking for distinct locations, as well as predictions that increase the HDI by a significant amount.	Partial Pass, 2:31
FT_4	User can Create an Account (S_{12})	Users should be able to create an account, to store their regions in. The account must have a username and password, which will be hashed. I will be looking inside the database to see if this is successful.	Pass, 6:06
FT_5	User can Log Into an Account (S_{13})	Users should also be able to log into existing accounts, given they have the correct username and password. If they do, the regions in that account will be loaded into the Regions Tab.	Pass, 7:54
FT_6	User can Compare a Region to one in the Training Data (S_{14})	Users should be able to compare their region to one in the training data. That is, they will see a table containing the HDI and each of the factors for their and any other region.	Pass, 8:38
FT_7	User can See how much Each Factor affects the Others (S_{15})	Users should also be able to see a heatmap about how much each factor affects the others. This heatmap is not interactive, so there is not much testing that needs to be done here.	Pass, 9:01
FT_8	Predicted HDI is Ranked among Other Countries with a Similar HDI (S_{16})	When predicting the HDI of a region, the user should be able to get a sense of what this HDI means, by comparing it to another region.	Pass, (not explicitly mentioned but you can see the output at 2:20)

FT_9	User can Save Regions (S_{17})	Even if the user is not logged in, they should be able to save regions. Regions that the user predicts the HDI of should be saved to the Regions Tab, where they can switch between them.	Pass, 10:27
FT_{10}	User can Manually Enter Factors (S_{18})	Users should also be able to manually enter factors, whether it is in a region that already exists, or in a new region. Users should also be made aware that any regions they edit like this can no longer have suggestions made on them.	Pass, 11:49

Table 6.1: Testing for Function to Inform Evaluation

All of the tests FT_1 to FT_{10} passed, except for FT_2 and FT_3 , which I decided to only give a partial pass. While the features do function correctly, the quality of the predictions and suggestions is not as good as I initially had hoped. This is most likely due to the limitations, outlined in Section 1.5.

Testing for Robustness

No.	Test	Inputs	Expected Output/Justification	Pass / Fail
RT_1	User cannot predict the HDI of no region	Press the Predict HDI Button Before Uploading Any Regions	If there was no validation, this would throw an error, as the region data would be undefined at this point. Instead, it should return a prompt to upload a region first.	Pass, 0:27
RT_2	User cannot make suggestions before predicting HDI	Press the Make Suggestions Button Before Predicting HDI	This would also throw an error if there was no validation, as this time the HDI is undefined. It should instead re-run a prompt to predict the HDI first.	Pass, 0:39

RT_3	User cannot upload 2 regions with the same name	Upload a region called <code>region.geojson</code> twice	Regions with the same name will cause issues down the line, in saving regions. Therefore, if the user uploads a region with the same name as an existing one, they should be prompted to change the name.	Pass, 2:05
RT_4	User cannot create an account with insufficient credentials	<p>1. Empty fields – all combinations of empty fields will be attempted, with non-empty fields being filled with random characters.</p> <p>2. Username which already exists – username: '<code>notlukewilliams</code>', passwords: random characters.</p> <p>3. Non-Matching Passwords – username: '<code>lukewilliams</code>', password: '<code>a</code>', password re-attempt: '<code>aa</code>'.</p> <p>4. Short Password – username: '<code>lukewilliams</code>', passwords: '<code>short</code>'.</p> <p>5. Password with no numbers – username: '<code>lukewilliams</code>', passwords: '<code>nonumbers</code>'.</p> <p>6. Password with no Special Characters – username: '<code>lukewilliams</code>', passwords: '<code>01number</code>'.</p>	<p>Users must create accounts with unique usernames and secure passwords. If the user has done any of these things, they should be prompted to choose another username/password that satisfies the rules.</p>	1. Pass, 3:23 2. Pass, 4:13 3. Pass, 4:39 4. Pass, 5:00 5. Pass, 5:19 6. Pass, 5:42

RT_5	User cannot log into account with incorrect credentials	<p>1. Empty fields – all combinations of empty fields will be attempted, with non-empty fields being filled with random characters.</p> <p>2. Username which does not exist – username: 'luke', password: random characters.</p> <p>3. Incorrect password – username: 'lukewilliams', password: random characters.</p>	Users should only be able to log into accounts if they have the correct username and password. If they do not, they should be prompted to enter the correct information.	1. Pass, 7:05 2. Pass, 7:23 3. Pass, 7:43
RT_6	User cannot enter an invalid item into the dropdown menu	Type a region into the dropdown menu that does not exits: ' this region does not exist '	Users should only be able to compare their region with one that exists. If they try to type one that does not, no comparison will take place	Pass, 9:13
RT_7	User cannot submit invalid data in the regions table	Enter random characters into any column that is not the name column.	When the user edits their regions in the table, the values in the table should be float, except for the average distance factors which may be empty strings, and the names which must be non-empty, unique strings.	Pass, 11:34
RT_8	User cannot make suggestions on a region which has been edited in the regions table	Press the Make Suggestions Button after editing a region in the Regions Tab	Regions that have been edited in the regions tab no longer reflect the objects that contributed to the factors. The objects have therefore been deleted and so suggestions should not be made. Instead, a prompt to select a different region shold be returned.	Pass, 12:10

Table 6.2: Testing for Robustness to Inform Evaluation

All of the tests RT_1 to RT_8 passed, so I believe that this app is robust.

6.1.2 Testing for Usability

I asked my stakeholders 8 questions (described in Table 4.12), on how usable they believe the app is. Again, coloured quotes will represent each stakeholder, as shown in the Stakeholders Table (Table 1.1).

Questions & Responses

1. “On a scale from 1 to 10, How easy is it to predict the HDI of a region?” (UT_1)
“5”, “5”, “6”, “5”
2. “On a scale from 1 to 10, How easy is it to make suggestions on how to improve a region?” (UT_2)
“8”, “9”, “8”, “10”
3. “On a scale from 1 to 10, How easy is it to compare your region with another?” (UT_3)
“10”, “10”, “9”, “10”
4. “On a scale from 1 to 10, How easy is it to switch between regions?” (UT_4)
“9”, “10”, “9”, “9”
5. “On a scale from 1 to 10, How easy is it to manually create your own region?” (UT_5)
“5”, “6”, “6”, “7”
6. “On a scale from 1 to 10, How easy is it to create an account?” (UT_6)
“8”, “9”, “8”, “10”
7. “On a scale from 1 to 10, How easy is it to log into an account?” (UT_7)
“9”, “9”, “8”, “10”
8. “On a scale from 1 to 10, How much did the help tab improve the usability?” (UT_8)
“9”, “10”, “9”, “10”

Total Scores

The total score for each usability test can be found by adding up the score each stakeholder gave. I will give any usability test that scores 30 or above a pass, a usability test that scores 20 or above a partial pass, and a usability test that scores below 20 a fail.

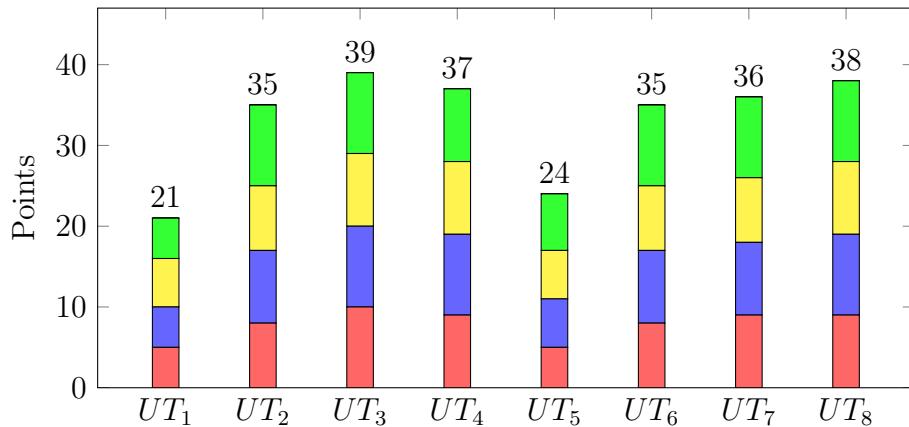


Figure 6.1: Scores for Usability Tests

As we can see from Figure 6.1, all usability tests passed, except for UT₁ and UT₅, which only partially passed.

No.	Test	Expected Output/Justification	Pass / Fail
UT ₁	How easy is it to predict the HDI of a region?	This is the main function of the app, and so it should be as usable as possible. However, the fact that the user has to export and upload the region will make it significantly less usable.	Partial Pass
UT ₂	How easy is it to make suggestions on how to improve a region?	This is the second function of the app, and so it should be almost as usable as the first.	Pass
UT ₃	How easy is it to compare your region with another?	The user should also be able to compare their region with another one that already exists. There is little to get confused about here, so I believe this should be pretty usable.	Pass

UT_4	How easy is it to switch between regions?	If the user has uploaded multiple regions, it should be easy and intuitive to switch between them. The user should also be able to tell which region they currently have selected.	Pass
UT_5	How easy is it to manually create your own region?	If the user wished to manually define their own region, they should also do that. This is not a primary feature, and so I won't be focusing on it as much, but it should still be relatively easy to use.	Partial Pass
UT_6	How easy is it to create an account?	The user should be made aware of the restrictions on their password, before they make one. If there is any problem with their account, they should be made aware of what it is.	Pass
UT_7	How easy is it to log into an account?	When logging in, if the user has not entered their details correctly, they should be aware of if their username is incorrect or if their password is incorrect.	Pass
UT_8	How much did the help tab improve the usability?	When I implement the Help Tab, I intend for it to be a source of information which the user can look to, if they are confused about a certain feature. It should clear up any misconceptions they may have, making it more usable.	Pass

Table 6.3: Testing for Usability to Inform Evaluation

I believe UT_1 only partially passed (and nearly failed) due to the fact that you have to export your region and the upload it, which seems unnecessary.

I believe UT_5 only partially passed because it may not have been obvious that you can type into the regions table and edit the values. The rules for what the cells can hold are also not made obvious to the user.

6.2 Success Criteria

6.2.1 Cross Referencing with Test Data

No.	Success Criterion	Success	Evidence & Explanation
S_1	Find suitable HDI training data for the Neural Network	✓	Section 3.1.2 – I needed to have the correct HDI for each region, and by the end of this section, I had done that.
S_2	Retrieve OSM data for a given area using Overpass Turbo	✓	T_1 (Table 3.5) – This test ensured that I could find a list of all buildings of a specific type, no matter what the type of building is.
S_3	Calculate average distance factors from a given area	✓	T_2 & T_3 (Tables 3.8 & 3.9) – T_2 ensured we could calculate the distance between any 2 points, and T_3 ensured we could calculate the average distance between 2 types of building.
S_4	Calculate density factors from a given area	✓	T_4 (Table 3.15) – This test ensured that we could find the number of a certain type of building per unit area, no matter how strangely the region is drawn.
S_5	Successfully Implement the Feedforward Algorithm	✓	Section 3.2.5 & T_5 (Table 3.47) – Testing the algorithms part of the neural network is difficult, so they were tested experimentally by training it to recognise handwritten digits in this section.
S_6	Successfully Implement the Backpropagation Algorithm	✓	Section 3.2.5 & T_5 (Table 3.47) – Testing the algorithms part of the neural network is difficult, so they were tested experimentally by training it to recognise handwritten digits in this section.
S_7	Successfully train the Neural Network	✓	Section 3.2.6 – This section was spent training the network, on predicting HDI. At the end, we had a set of weights and biases in the network which attempt to predict HDI.

S_8	User can select an area on the map for which they want to analyse	✓	FT_1 (Table 6.1) – This was tested as part of Final Testing for Function.
S_9	Neural Network accurately predicts HDI from those factors	Partial ✓	FT_2 (Table 6.1) – This was tested as part of Final Testing for Function.
S_{10}	Accurately calculate where a new building would need to be in order to increase the HDI the most	✓	T_{16} (Table 5.46) – This test ensured we could find the optimal place to put a new building, using the new algorithm introduced in the 2 nd prototype.
S_{11}	Changes are suggested to increase the HDI	Partial ✓	FT_3 (Table 6.1) – This was tested as part of Final Testing for Function.
S_{12}	User can create an account	✓	FT_4 (Table 6.1) – This was tested as part of Final Testing for Function.
S_{13}	User can log into an account	✓	FT_5 (Table 6.1) – This was tested as part of Final Testing for Function.
S_{14}	User can compare a region to one in the training data	✓	FT_6 (Table 6.1) – This was tested as part of Final Testing for Function.
S_{15}	User can see how each factor can affect the others	✓	FT_7 (Table 6.1) – This was tested as part of Final Testing for Function.
S_{16}	Predicted HDI is ranked among other countries with a similar HDI	✓	FT_8 (Table 6.1) – This was tested as part of Final Testing for Function.
S_{17}	User can save regions	✓	FT_9 (Table 6.1) – This was tested as part of Final Testing for Function.
S_{18}	User can manually enter factors	✓	FT_{10} (Table 6.1) – This was tested as part of Final Testing for Function.
S_{19}	App has a clean, simple and presentable GUI	Partial ✓	UT_1 to UT_8 (Table 6.3) – Final Testing for Usability was carried out to determine this success criterion. I gave it a partial success, as some of the individual tests were partial passes.

Table 6.4: Met (✓), Partially Met (Partial ✓) or Unmet (✗) Success Criteria

6.2.2 Reasoning Behind Outcome of Success Criteria

The success criteria to do with the backend (S_1 to S_7 & S_{10}) were not tested as part of final testing, and so the evidence was found during the development process. If these criteria were not met, no part of the app would function in the first place. Therefore, it was imperative that all 8 of these were fully met, which they were. This is because I implemented these things early on, before the user interface, so any problems that came up could be fixed immediately.

S_{19} is to do with the app's usability, and was determined from the responses to the usability tests. While the majority of the tests came back as pass, there were a few that came back as only a partial pass, and so I can only give the interface *as a whole* a partial pass.

The other success criteria (S_8 , S_9 & S_{11} to S_{18}) are to do with the various things the user can do within the app, and were tested as part of final testing for function. Each test corresponded to each of these success criteria, and if it got a pass, the criterion got a pass.

6.2.3 Addressing Partially Met Success Criteria in Further Development

Accurate Predictions (S_9)

In order to make better predictions, the simplest thing I could do is train the network for longer, by attempting different combinations of layer sizes, as well as adjusting the number of training epochs, the learning rate, and the mini batch size.

Furthermore, I could attempt to get more training data, by using unofficial values for HDI for regions smaller than sub-national regions. This will increase the number of training examples from 2000 to much more, which will allow the network to pick up on more patterns, and learn from them.

If none of the above work, a slightly more drastic approach may be to define some new factors, for the network to predict from. This will require slight adjustments all over the app, to accommodate the new factors (for example, the number of columns in the regions table may be different), and so this should only be done if there are no other options.

Accurate Suggestions (S_{11})

Making better suggestions is harder, as you cannot guarantee what the network will predict. If we assume that the network *generally* predicts a higher HDI for places with more infrastructure, we can always suggest building even more buildings (more than the 10 the user can do currently).

We could also suggest building different buildings, which would come with the different factors mentioned previously.

The suggestions themselves could also be more sophisticated, where instead of building just a hospital, we could instead suggest building a network of healthcare-related buildings.

User Interface (S_{19})

In general, this success criterion can be fully met when users have no problems with the usability of the app. Addressing poor usability features in further development is in Section 6.3.2.

6.3 Usability Features

In this section, I will first go through each of the interface sketches from the second design section (in Section 4.3.1), and compare them with the what the final interface looks like now. I will then go through each usability feature which has not worked out too well and describe how it can be improved in the future.

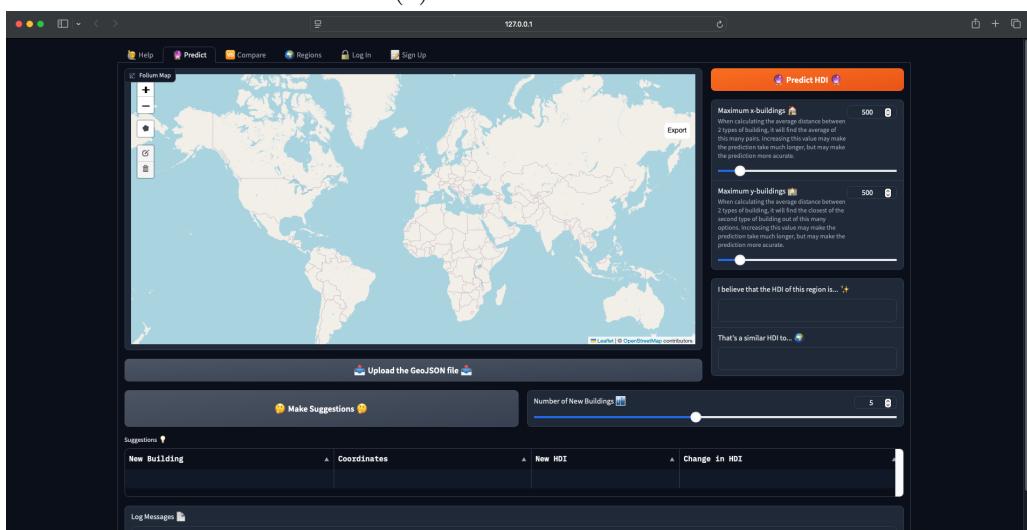
6.3.1 Comparison to Sketches & Effective Usability Features

Predict Tab

New Building	Coordinates	New HDI	Change in HDI
School	36.922883°N, -101.709461°E	0.928	+0.001
Hospital	43.919052°N, -106.264212°E	0.929	+0.002
Library	36.954162°N, -87.008695°E	0.928	+0.001
⋮			

(The data presented in this sketch is completely made up but somewhat realistic)

(a) Interface Sketch



(b) Result

Figure 6.2: Comparison to Interface Sketch, predict tab (Figure 4.6)

In the end, the sketch for this tab was pretty accurate. The map is in large on the left, and the prediction is in a column on the right, with the button to predict.

The main noticeable difference is the inclusion of the upload regions button, below the map. This is the least usable part of the app, but unfortunately it is necessary.

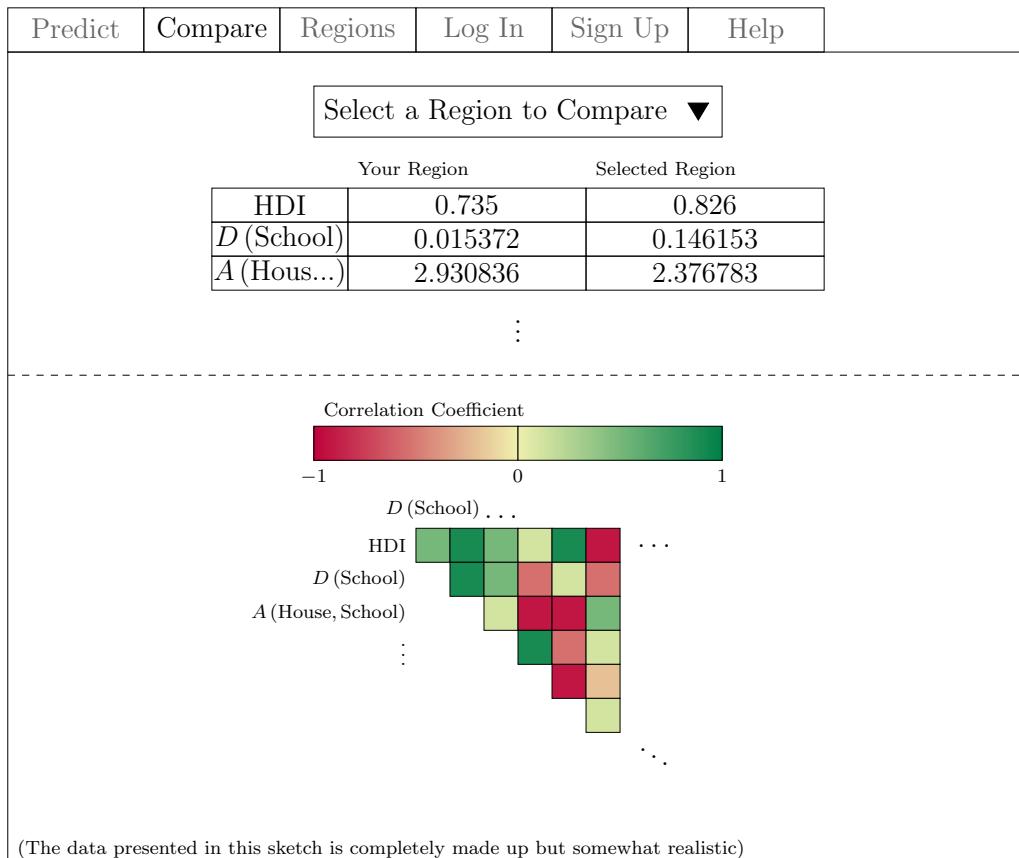
The sketch also omits the make suggestions button and the number of new buildings slider. At the time of design, I had intended that the suggestions would be made automatically, instead of pressing a button after the prediction has been made. During development, I decided that it should be a different button, to separate the processes that take a long time. I thought of the need for the number of new buildings slider after this sketch, which is why that is omitted.

The final thing missing from this sketch that is included in the final interface is the sliders for `max_x_objects` and `max_y_objects`. This is because I hadn't anticipated the need for them, as I didn't think that calculating the average distance factors would take so long. After I found that it did in fact take a while, I realised that I needed a way for the users to control how long they are willing to wait.

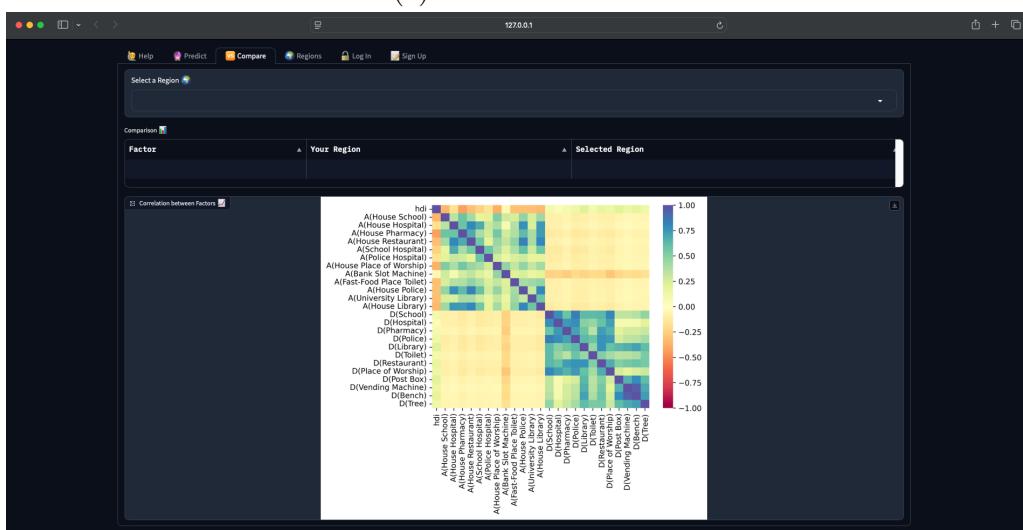
The sketch also shows each suggestion having 1 coordinate, but in the final app, it gives multiple.

Overall, I believe that this tab has turned out pretty good, as it is very similar to what I set out to do in the design section, with the exception of the upload region button.

Comapare & Region Tabs



(a) Interface Sketch



(b) Result

Figure 6.3: Comparison to Interface Sketch, compare tab (Figure 4.7)

Predict	Compare	Regions	Log In	Sign Up	Help
---------	---------	---------	--------	---------	------

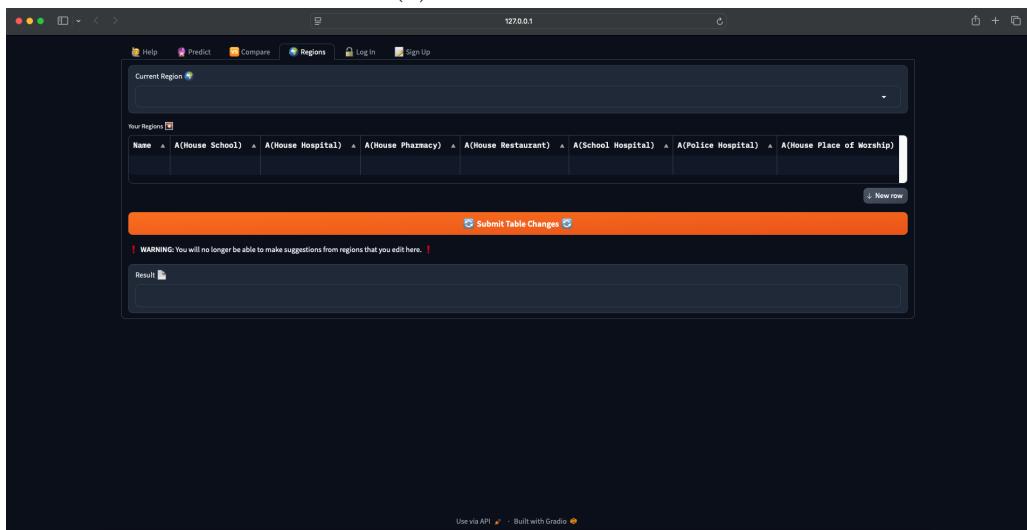
Current Region

London

Name	$D(\text{School})$	$D(\text{Hospital})$	$D(\text{Tree})$	$D(\text{Restaurant})$
Oxford	1.1923	0.0917	62.569	2.8799
London	1.5946	0.1248	32.861	3.3566
region-3	0.5673	0.0132	10.483	1.4084
region-4	0.9191	0.0342	29.589	5.4273
region				

(The data presented in this sketch is completely made up but somewhat realistic)

(a) Interface Sketch



(b) Result

Figure 6.4: Comparison to Interface Sketch, regions tab (Figure 4.8)

The compare tab is very similar to its sketch. At the top, there is a dropdown menu, then below that there is a table for comparison, and then below that is a heatmap showing the correlation between factors.

The only difference is that in the final interface, the heatmap is a full square, but in the design section I initially decided that it should be only a triangle. This is because the square will be symmetrical about that diagonal line (as you can see from the final interface), and so the bottom left half conveys no different information than the top right half, so there is no point in including it. When I was actually making the heatmap however, it was easier to just make the square, and so I decided to do that as the redundant information doesn't take up any space that would be used for anything anyway (if it were a triangle, those spaces would just be white).

Overall, I believe the compare tab was a success, as it is almost identical to the sketch. The only difference can be considered to be irrelevant, as no extra space has been wasted.

The regions tab looks a bit more different to its sketch than the compare tab. The only thing that is the same is the main feature of the tab, which is the table containing all of the user's regions, and its new row button below it on the right.

At the top, I had initially planned to have a textbox showing the current region, and the user would switch between regions by clicking on the table. When actually implementing this tab, I decided that it would be more usable if the user selects the current region from a dropdown menu.

The sketch also omits the button to submit the changes made to the table, and its result textbox. This is because I had initially planned for the changes to be made automatically, whenever the user changes the values in the table. In reality, this does not work, because changes would be made to the backend for every character the user types, which is not what the intended behaviour. Instead I decided to let the user to make all of their changes, and then press the submit button. I made the button the “call to action” colour, to draw attention to it, alerting the user that changes will not be made automatically.

Overall, I believe that the regions tab is the tab that turned out least similar to its initial design, due to the fact that I hadn't properly thought through how it would work in the backend. Once I realised, the necessary changes were made and in the end I believe it turned out alright.

Log In & Sign Up Tabs

Predict	Compare	Regions	Log In	Sign Up	Help
---------	---------	---------	---------------	---------	------

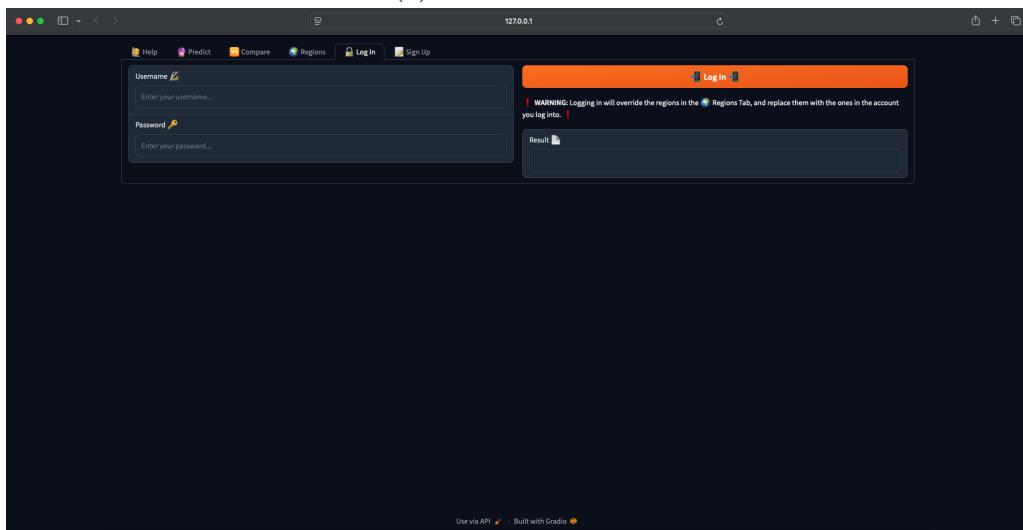
Username

Password

Log In

Result

(a) Interface Sketch



(b) Result

Figure 6.5: Comparison to Interface Sketch, log in tab (Figure 4.9)

Predict	Compare	Regions	Log In	Sign Up	Help
---------	---------	---------	--------	---------	------

Username

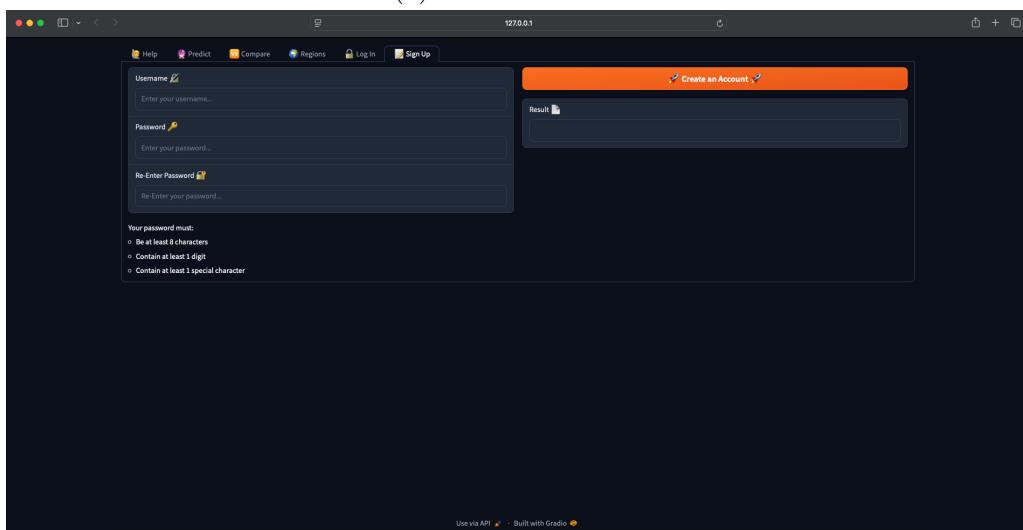
Password

Re-Enter Password

Create Account

Result

(a) Interface Sketch



(b) Result

Figure 6.6: Comparison to Interface Sketch, sign up tab (Figure 4.10)

The log in tab is quite simple, and so there is not much to change from the sketch to the final interface. The final interface still has a textbox for the user to enter their username, and their password. There is also a button to log in, given they have entered correct information,

as well as a result box.

The main difference between them is the structure. At the end, I decided to slightly change the structure of it, to put the button and the result box in a different column to the textboxes.

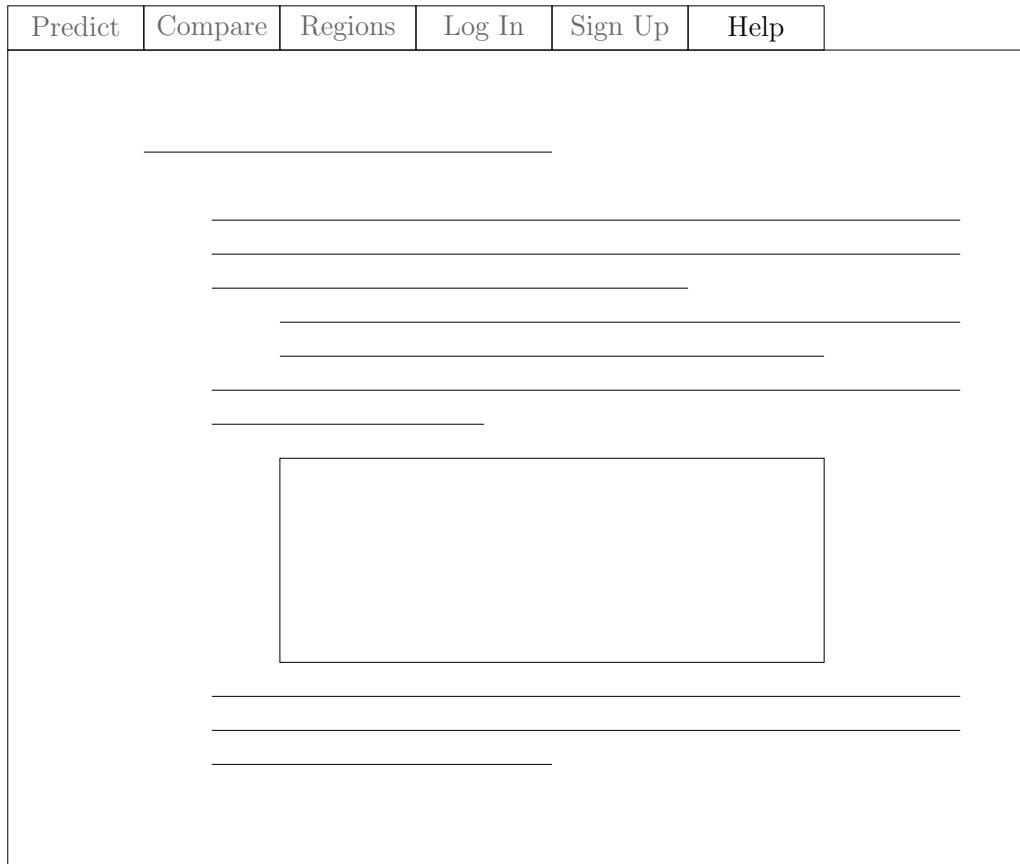
Below the button, I also added a warning, to let the user know that logging in will override the regions in the regions tab to be the regions held in the account. This is a pretty good usability feature, as it was not obvious before that it would function this way.

The sign up tab is very similar to the log in tab, in its design as well as the changes made to the structure (button and result moved to a different column).

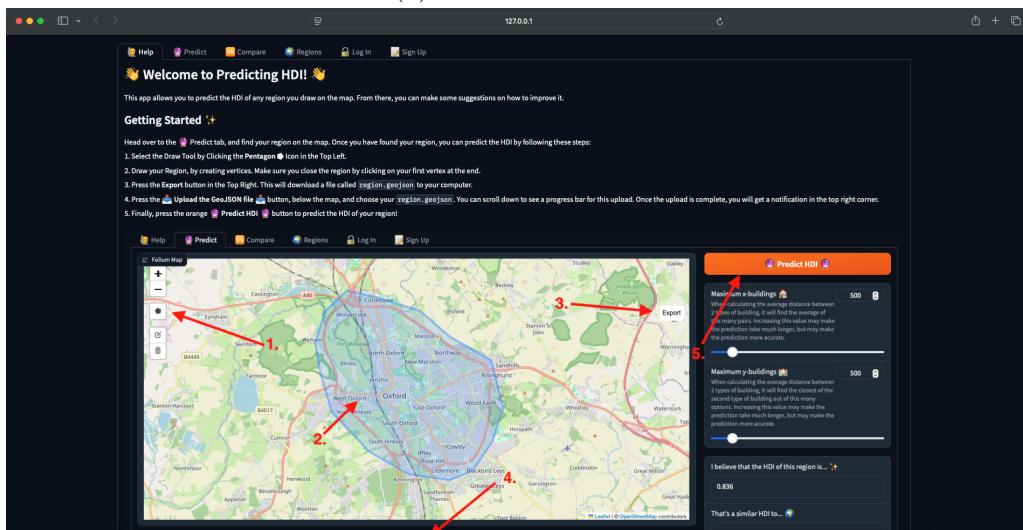
There is no unexpected behaviour here that warrants a warning, but I decided to include the password requirements on this page, so that the user wouldn't receive multiple error messages when attempting different passwords.

Overall, I believe that the log in and sign up tabs were a success, as they are very similar to the sketch. I believe the changes made actually make it more usable, where the changes made to the previous tabs were made because of how the backend functioned.

Help Tab



(a) Interface Sketch



(b) Result

Figure 6.7: Comparison to Interface Sketch, help tab (Figure 4.11)

Initially, the help tab was at the end, as shown in the interface sketch, but when I wrote it I decided to make it first, so that the user would immediately see it when they launch the app.

The sketch of the help tab is not very detailed, as I had not written the contents yet. The sketch includes lines of text, with some indented to represent (for example) ordered lists, as well as a large rectangle in place of an image. The final interface includes all of these elements, and so therefore I believe that this tab has been successful.

6.3.2 Addressing Poor Usability Features in Further Development

The Upload Button

This is the main usability feature which did not go as intended, and has decreased the usability of the app the most. In the design section, I described that I wanted the user to just be able to draw a region, and then press the predict button to predict the HDI of it, but unfortunately the process is more complicated, so much so that it requires 5 steps with a labelled diagram to complete it.

The issue stems from the fact that we cannot access the bounding coordinates of the region the user draws on the map, without the user pressing the “export” button on the map.

In order to fix this, I could attempt to make my own map, and not use a library. Currently, I am using a library called `folium`, which allows for the creation of these maps.

Moving away from `folium` would also allow me to use a newer version of `gradio`, as `folium` is only compatible with `gradio` up to a certain version (which I am using), which is not the newest version. Newer versions of `gradio` may have more features (such as UI elements) and may contain bug fixes.

Gradio Dataframes

One bug in `gradio` that causes the usability of the app to be less than expected is to do with the `gr.DataFrame`. You may have noticed that the dataframes have a vertical and horizontal scroll wheel on the left and bottom, even when they are not needed. This stands out from the rest of the interface and may be distracting.

Furthermore, when new rows are added to a dataframe (e.g. when the suggestions table is filled the number of rows goes from 1 to 8), this change is not automatically made. If the user wishes to see the new rows, they have to select the dataframe and scroll around. Scrolling will cause an update to happen, which will update the size of the dataframe, showing all new rows.

To fix this, `gradio` should be updated, which again requires the removal of `folium`. Alternatively, I could attempt to make my own table, by using `gr.Rows` and `gr.Columns`, but this may not work as well as a dataframe with no bugs.

Editing & Creating Regions in the Table

When asking the stakeholders on how usable they believed the app was, they said that creating new regions, or editing existing ones was also one of the least usable parts of the

app.

I believe that this is because the rules for how you could enter data were not clear, and so to fix this in further development, I could make it more clear, by specifying the data types allowed in each column.

6.4 Maintenance Issues

6.4.1 Maintainability

Overall, I believe the code is fairly maintainable, as all variables are appropriately named, and there are many comments describing the code where necessary. I have also made use of many sub-programs, to ensure a modular design, which further improves the maintainability.

Furthermore, I believe the use of `gradio` makes the code for the UI very maintainable, as its structuring makes it very intuitive to see how it will be structured when it is run. There is no need to consider parent and child elements, or containers, when you can specify if you want to place elements in a `gr.Row` (for example).

6.4.2 Documentation

There is no explicit documentation for this project, and so I could write one to improve the maintainability. It would specify what each function does, as well as the data types of the parameters and outputs, to ensure that future developers have no trouble understanding what a certain function does.

There does exist documentation on how to use it (the help tab), but no documentation on how to develop it.

6.4.3 Code Style Guidelines

I also did not use any code style guidelines, when writing code. Some of my variable names are in camel case, while others are in snake case, with no real system to distinguish between which variable should be in which case.

In order to make it more maintainable, I could specify an exact code style guideline, and change the case of many identifiers. For example, function names could be written in snake case and variable names could be written in camel case.

6.5 Limitations of the Solution

6.5.1 Data Limitations

As described in the Analysis Section (Section 1.5), there are a few limitations to do with the training data, the main one being the bias towards more developed countries, meaning that the network is often unaware of less developed countries. This, along with the fact that there may not have been enough training data, lead to the network not making completely accurate predictions.

I am also not taking into account the past HDIs of a given region, as OpenStreetMaps can only serve me the data as it is now, and not an arbitrary point in the past. As a result, the network hasn't explicitly learned about how HDI can change over time, and how that change happens. It only learned about why the HDI is what it is at the current moment.

6.5.2 Usability Challenges

As stated before, I believe that the usability of this app has been a partial success, and the main reason that it is not fully successful, is the fact that you have to export your region and then upload it. This prompted the creation of a user manual (the help tab), which includes step by step instructions and a labelled diagram on how to do this. This is unfortunate for the main function of the app, but is unfortunately necessary.

Other usability challenges include poor signposting for some features, as well as some unexpected behaviour to do with dataframes (as described in Section 6.3.2).

To overcome these limitations, the main thing that needs to be done is moving away from using the `folium` library, as it is causing minor dependency conflicts, as well as the fact that the user must export and import their regions. I could not find any more suitable alternatives, so I would be forced to create my own map.

6.5.3 Scalability & Technical Constraints

As this is built with `gradio`, this means that once deployed, the app can be run on any computer with a web browser. As the vast majority of operating systems often come with a web browser, this means that anyone on any device can use the app, including those on mobile devices.

There is also a queue system built into `gradio`, where if there are many users trying to use the app at the same time, the processes are added to a queue, and so therefore are processed in a First In First Out (FIFO) order. This is suitable as users who pressed the button first should be the first to receive a response.

6.6 Conclusion

Overall, I believe that the development of this app has been a success, as I have done what I originally set out to do, which is predict the HDI of a region, given only its geographical factors. While there are some limitations, such as the predictions not being 100% accurate, as well as some parts of the app which are not very usable, such as the export & upload process, I believe that these hindrances are not very major, and can be fixed in a future update. Therefore, I think that this app is now in a position where it is ready to be deployed, for everyone to interact with.

A. Final Code

A.1 Data Processing

This section will include all files in the `data_processing` folder, which were mostly only run once to create the dataset in Section 3.1.6.

A.1.1 `data_processing / calc_factors.py`

(This code was only used when calculating the factors for the training data. There is a more up-to-date version which is used directly by the app in Section A.3.2)

```

1 import numpy as np
2 import random
3 from get_osm_data import get_osm_data
4
5 EARTH_RADIUS = 6371 # (in km)
6
7 # converts angle in degrees to angle in radians
8 def deg2rad(angle):
9     return (np.pi/180)*angle
10
11 # finds the distance (in km) along the surface of the earth, between 2
12 # coordinates
12 def distBetween2Points(p1, p2):
13     La1 = deg2rad(p1[0])
14     Lo1 = deg2rad(p1[1])
15     La2 = deg2rad(p2[0])
16     Lo2 = deg2rad(p2[1])
17     return EARTH_RADIUS * np.arccos(np.sin(La1)*np.sin(La2) +
18         np.cos(La1)*np.cos(La2)*np.cos(Lo1-Lo2))
19
19 # finds the average distance between 2 types of objects, A(x,y)
20 def averageDistance(x_objects, y_objects, max_x_objects=2000):
21     if len(x_objects) == 0 or len(y_objects) == 0: # the average distance is
22         undefined
23         return None
24     if len(x_objects) > max_x_objects: # random sample
25         x_objects = random.sample(x_objects, max_x_objects)

```

```

25 # iterate over each x_object, and find the closest y_object
26 min_dists = []
27 for x_object in x_objects:
28     dist_to_ys = []
29     for y_object in y_objects:
30         dist_to_ys.append(distBetween2Points(x_object, y_object))
31     min_dists.append(min(dist_to_ys)) # the closest y_object
32 return np.mean(min_dists)

33

34 # calculates the area (in km^2) of a given region bound by a set of
35 # latitude/longitude coordinates
36 def calcArea(bounding_coords):
37     if bounding_coords[0] == bounding_coords[-1]: # if the first and last
38         # coordinates are the same, then delete the last one
39         bounding_coords.pop(-1)
40     # convert lat/lon pairs to 3d position vectors
41     bounding_vectors = []
42     for coord in bounding_coords:
43         La = deg2rad(coord[0])
44         Lo = deg2rad(coord[1])
45         bounding_vectors.append(np.matrix([
46             [np.cos(La)*np.cos(Lo)],
47             [np.cos(La)*np.sin(Lo)],
48             [np.sin(La)]
49         ]))
50     # iterate over each vertex and find its anticlockwise angle
51     anticlockwise_angles = []
52     for vertex_index, vertex in enumerate(bounding_vectors):
53         prev_vertex = bounding_vectors[vertex_index-1]
54         next_vertex = bounding_vectors[0 if vertex_index ==
55             len(bounding_vectors)-1 else vertex_index+1]
56         La = deg2rad(bounding_coords[vertex_index][0])
57         Lo = deg2rad(bounding_coords[vertex_index][1])
58         # define the rotation and translation matrices
59         rotation_matrix_z = np.matrix([
60             [np.cos(-Lo), -np.sin(-Lo), 0],
61             [np.sin(-Lo), np.cos(-Lo), 0],
62             [0, 0, 1]
63         ])
64         rotation_matrix_y = np.matrix([
65             [np.cos(La-np.pi/2), 0, np.sin(La-np.pi/2)],
66             [0, 1, 0],
67             [-np.sin(La-np.pi/2), 0, np.cos(La-np.pi/2)]
68         ])
69         translation_vector = np.matrix([
70             [0],
71             [0],
72             [0],
73             [0]
74         ])

```

```

69      [-1]
70  ])
71  # rotate and translate the plane tangent to the vertex such that it is
72  #→ the xy-plane
73  prev_vertex = np.matmul(rotation_matrix_y, np.matmul(rotation_matrix_z,
74  #→ prev_vertex)) + translation_vector
75  vertex = np.matmul(rotation_matrix_y, np.matmul(rotation_matrix_z,
76  #→ vertex)) + translation_vector
77  next_vertex = np.matmul(rotation_matrix_y, np.matmul(rotation_matrix_z,
78  #→ next_vertex)) + translation_vector
79  # project onto the xy-plane
80  prev_vertex[2][0] = 0
81  next_vertex[2][0] = 0
82  # calculate anticlockwise angle between the vectors
83  prev_to_current = vertex - prev_vertex
84  current_to_next = next_vertex - vertex
85  anticlockwise_angle =
86  #→ np.arctan2((prev_to_current.item(1,0)*current_to_next.item(2,0) -
87  #→ prev_to_current.item(2,0)*current_to_next.item(1,0)) -
88  #→ (prev_to_current.item(2,0)*current_to_next.item(0,0) -
89  #→ prev_to_current.item(0,0)*current_to_next.item(2,0)) +
90  #→ (prev_to_current.item(0,0)*current_to_next.item(1,0) -
91  #→ prev_to_current.item(1,0)*current_to_next.item(0,0)),
92  #→ prev_to_current.item(0,0)*current_to_next.item(0,0) +
93  #→ prev_to_current.item(1,0)*current_to_next.item(1,0) +
94  #→ prev_to_current.item(2,0)*current_to_next.item(2,0))
95  anticlockwise_angles.append(anticlockwise_angle)
96  # find the interior angles from the anticlockwise angles
97  sum_anticlockwise_angles = round(sum(anticlockwise_angles), 5)
98  if sum_anticlockwise_angles == 0:
99  return None
100 if sum_anticlockwise_angles < 0:
    anticlockwise_angles = [-1*angle for angle in anticlockwise_angles]
interior_angles = [np.pi-angle for angle in anticlockwise_angles]
# return the area
num_edges = len(bounding_coords)
return (EARTH_RADIUS ** 2) * (sum(interior_angles) - np.pi*(num_edges-2))

# finds the density of a type of object, D(x)
def density(x_objects, area):
    return len(x_objects)/area

# finds all the factors for a given region
def getAllFactors(region):
    # retrieve all relevant osm data

```

```

101     object_types = ['house', 'school', 'hospital', 'pharmacy', 'restaurant',
102                     'place_of_worship', 'bank', 'slot_machines', 'fast_food', 'toilets',
103                     'police', 'university', 'library', 'post_box', 'vending_machine',
104                     'bench', 'tree']
105     all_objects = {}
106     for object_type in object_types:
107         print(f'getting {object_type} data...') # log message
108         all_objects[object_type] = get_osm_data(object_type, region)
109     # return a dictionary of all the factors
110     print('calculating area...') # log message
111     area = calcArea(region)
112     print('calculating factors...') # log message
113     return {
114         "A(House School)": averageDistance(all_objects['house'],
115                                             all_objects['school']),
116         "A(House Hospital)": averageDistance(all_objects['house'],
117                                             all_objects['hospital']),
118         "A(House Pharmacy)": averageDistance(all_objects['house'],
119                                             all_objects['pharmacy']),
120         "A(House Restaurant)": averageDistance(all_objects['house'],
121                                             all_objects['restaurant']),
122         "A(School Hospital)": averageDistance(all_objects['school'],
123                                             all_objects['hospital']),
124         "A(Police Hospital)": averageDistance(all_objects['police'],
125                                             all_objects['hospital']),
126         "A(House Place of Worship)": averageDistance(all_objects['house'],
127                                             all_objects['place_of_worship']),
128         "A(Bank Slot Machine)": averageDistance(all_objects['bank'],
129                                             all_objects['slot_machines']),
130         "A(Fast-Food Place Toilet)": averageDistance(all_objects['fast_food'],
131                                             all_objects['toilets']),
132         "A(House Police)": averageDistance(all_objects['house'],
133                                             all_objects['police']),
134         "A(University Library)": averageDistance(all_objects['university'],
135                                             all_objects['library']),
136         "A(House Library)": averageDistance(all_objects['house'],
137                                             all_objects['library']),
138         "D(School)": density(all_objects['school'], area),
139         "D(Hospital)": density(all_objects['hospital'], area),
140         "D(Pharmacy)": density(all_objects['pharmacy'], area),
141         "D(Police)": density(all_objects['police'], area),
142         "D(Library)": density(all_objects['library'], area),
143         "D(Toilet)": density(all_objects['toilets'], area),
144         "D(Restaurant)": density(all_objects['restaurant'], area),
145         "D(Place of Worship)": density(all_objects['place_of_worship'], area),
146         "D(Post Box)": density(all_objects['post_box'], area),
147         "D(Vending Machine)": density(all_objects['vending_machine'], area),

```

```

133     "D(Bench)": density(all_objects['bench'], area),
134     "D(Tree)": density(all_objects['tree'], area),
135 }
```

Code Snippet A.1: Final Code for data_processing / calc_factors.py

A.1.2 data_processing / compile_data.py

(This code was only ran once, in Section 3.1.6)

```

1 from calc_factors import getAllFactors
2 import json
3 import csv
4
5 print('reading files...') # log message
6 # read json file
7 with open('data/region_coords.json', 'r') as file:
8     regionData = json.load(file)
9
10 # read csv file
11 with open('data/hdi.csv', 'r') as file:
12     reader = csv.DictReader(file)
13     hdiData = []
14     for record in reader:
15         hdiData.append(record)
16
17 MOST_RECENT_REGION = 'AFGr108'
18 started = False
19 field_names = ['country', 'region', 'code', 'hdi', 'A(House School)', 'A(House
   ↳ Hospital)', 'A(House Pharmacy)', 'A(House Restaurant)', 'A(School Hospital)',
   ↳ 'A(Police Hospital)', 'A(House Place of Worship)', 'A(Bank Slot Machine)',
   ↳ 'A(Fast-Food Place Toilet)', 'A(House Police)', 'A(University Library)',
   ↳ 'A(House Library)', 'D(School)', 'D(Hospital)', 'D(Pharmacy)', 'D(Police)',
   ↳ 'D/Library)', 'D(Toilet)', 'D(Restaurant)', 'D(Place of Worship)', 'D(Post
   ↳ Box)', 'D(Vending Machine)', 'D(Bench)', 'D(Tree)']
20
21 # iterate over each region
22 for region_number, region in enumerate(regionData):
23     if not started:
24         if region['properties']['gdlcode'] == MOST_RECENT_REGION:
25             started = True
26             continue
27         print(f'Processing {region["properties"]["gdlcode"]}...') # log message
28         # find the main bit of the region (if it has multiple)
29         shapes = region['geometry']['coordinates']
30         if region['geometry']['type'] == 'Polygon':
31             mainShape = shapes[0]
```

```

32     else:
33         lengths = [len(polygon[0]) for polygon in shapes]
34         mainShape = shapes[lengths.index(max(lengths))][0]
35         mainShape = [coordinate[::-1] for coordinate in mainShape]
36         # find all the factors
37         allFactors = getAllFactors(mainShape)
38         # match it with the hdi
39         matchingHDI = [record for record in hdiData if record['code'] ==
40                         ↪ region['properties']['gdlcode']]
41         if len(matchingHDI) != 0:
42             newRecord = matchingHDI[0]
43             newRecord.update(allFactors)
44             print('writing to file...') # log message
45             with open('src/data/training_data.csv', 'a') as file:
46                 writer = csv.DictWriter(file, fieldnames=field_names)
47                 writer.writerow(newRecord)

```

Code Snippet A.2: Final Code for data_processing / compile_data.py

A.1.3 data_processing / get_osm_data.py

(This code is used by the main app, all other scripts in this folder were only ran once)

```

1 import requests
2
3 # makes a request to overpass turbo to get osm data
4 def get_osm_data(object_type, bounding_coords):
5     overpass_url = 'http://overpass-api.de/api/interpreter'
6     key_type = ('building' if object_type == 'house' else 'natural' if
7                 ↪ object_type == 'tree' else 'gambling' if object_type == 'slot_machines'
7                 ↪ else 'amenity') # get the right key type
8     while len(bounding_coords) >= 100000:
9         print(f'truncating region with {len(bounding_coords)} vertices...') # log
10            ↪ message
11         bounding_coords = [bounding_coord for n, bounding_coord in
12                           ↪ enumerate(bounding_coords) if n % 2 == 0]
13     bounding_coords = [(latitude, (longitude + 180) % 360 - 180) for latitude,
14                         ↪ longitude in bounding_coords]
15     region_poly = ' '.join([' '.join([str(latlong) for latlong in coord]) for
16                             ↪ coord in bounding_coords]) # converts list of tuples to correct format
17     ↪ for overpass turbo
18     overpass_query = [
19         '[out:json];',
20         '(',
21         f'    nwr["{key_type}"="{object_type}"]({poly}:{region_poly});',
22         ')',
23         'out center;'

```

```

18     ]
19     overpass_query = '\n'.join(overpass_query) # concatenates list items into a
20     ↵ multi-line string
21
22     response = requests.post(overpass_url, data=overpass_query)
23     print(response) # log message, shows the HTTP response code
24     data = response.json()
25
26     coords = []
27     for nwr in data['elements']:
28         if nwr['type'] == 'node':
29             coords.append((nwr['lat'], nwr['lon']))
30         else: # it is a way or relation
31             coords.append((nwr['center']['lat'], nwr['center']['lon']))
32
33     return coords

```

Code Snippet A.3: Final Code for data_processing / get_osm_data.py

A.1.4 data_processing / pmcc.py

(This code was only ran once, in Section 5.2.3)

```

1 import csv
2 import numpy as np
3 import seaborn
4
5 # load the training data
6 with open('data/training_data.csv', 'r') as file:
7     reader = csv.DictReader(file)
8     trainingData = []
9     for record in reader:
10         trainingData.append(record)
11
12 # iterate over every pair of factors
13 factors = list(trainingData[0].keys())[3:]
14 pmcc_grid = []
15 for factor1 in factors:
16     pmcc_row = []
17     for factor2 in factors:
18         print(f'calculating pmcc of {factor1} and {factor2}...') # log message
19         x_values = [record[factor1] for record in trainingData]
20         y_values = [record[factor2] for record in trainingData]
21         # remove pairs with empty values, iterating over the lists in reverse as
22         ↵ to not disturb the index numbers
23         for index in range(len(x_values)-1, -1, -1):
24             if x_values[index] == '' or y_values[index] == '':

```

```

24         x_values.pop(index)
25         y_values.pop(index)
26     # convert to float
27     x_values = [float(x) for x in x_values]
28     y_values = [float(y) for y in y_values]
29     # calculate pmcc and add it to grid
30     pmcc = sum([(x_values[index]-np.mean(x_values))*(y_values[index]
31             -np.mean(y_values)) for index in range(len(x_values))])/np.sqrt(
32             (sum([(x_values[index]-np.mean(x_values))**2 for index in
33             range(len(x_values))]))*(sum([(y_values[index]-np.mean(y_values))**2
34             for index in range(len(x_values))])))
35     pmcc_row.append(pmcc)
36     pmcc_grid.append(pmcc_row)

# generate the image
heatmap = seaborn.heatmap(pmcc_grid, xticklabels=factors, yticklabels=factors,
                           vmin=-1, vmax=1, center=0, cmap='Spectral')
heatmap.get_figure().savefig('data/pmcc.png', bbox_inches='tight', dpi=600)

```

Code Snippet A.4: Final Code for data_processing / pmcc.py

A.1.5 data_processing / process_hdi.py

(This code was only ran once, in Section 3.1.2)

```

1 import csv
2
3 #retrieve data
4 with open('data/subnationalHDI.csv', 'r') as file:
5     reader = csv.DictReader(file)
6     newcsv = []
7     for record in reader:
8         if record['year'] == '2022': # the most recent year
9             newRecord = {
10                 "country": record["country"],
11                 "region": record["region"],
12                 "code": record["GDLCODE"],
13                 "hdi": record["shdi"]
14             } # only the most useful information
15             newcsv.append(newRecord)
16             print(newRecord)
17
18 #write to new csv
19 with open('data/hdi.csv', 'w') as file:
20     field_names = ['country', 'region', 'code', 'hdi']
21     writer = csv.DictWriter(file, fieldnames=field_names)
22     writer.writeheader()

```

```

23     for record in newcsv:
24         writer.writerow(record)

```

Code Snippet A.5: Final Code for `data_processing / process_hdi.py`

A.1.6 `data_processing / process_shapefile.py`

(This code was only ran once, in Section 3.1.6, and has been adapted from this StackOverflow solution: [8])

```

1 import shapefile
2 import json
3
4 # read records of shapefile
5 reader = shapefile.Reader("data/shapefiles/GDL Shapefiles V6.3 large.shp")
6 fields = reader.fields[1:]
7 field_names = [field[0] for field in fields]
8 data = []
9 for shape_record in reader.shapeRecords():
10     print(shape_record.record) # log message
11     data.append({
12         "type": "Feature",
13         "geometry": shape_record.shape.__geo_interface__,
14         "properties": dict(zip(field_names, shape_record.record))
15     })
16
17 # write data to file
18 print('writing to file...') # log message
19 with open("data/region_coords.json", "w") as file:
20     json.dump(data, file)
21 print('done!') # log message

```

Code Snippet A.6: Final Code for `data_processing / process_shapefile.py`

A.2 Training the Neural Network

This section will include all files in the `network_training` folder, which were all only run a limited number of times to train and test the neural network, in Sections 3.2.5 & 3.2.6. The actual code for the neural network can be found in Section A.3.3.

For each set of files (`digits` or `hdi`), `load_x_data.py` was ran first, then `train_x.py`, and finally `test_x.py`.

A.2.1 `network_training / load_digits_data.py`

(This code was only ran a limited number of times, in Section 3.2.5, and has been adapted from Micheal Nielsen's code repository [9])

```

1 import pickle
2 import gzip
3 import numpy as np
4
5 # loads the MNIST data set
6 def load_data():
7     with gzip.open('data/mnist.pkl.gz', 'rb') as file:
8         unpickled = pickle._Unpickler(file)
9         unpickled.encoding = 'latin1'
10        training_data, validation_data, test_data = unpickled.load()
11    return (training_data, validation_data, test_data)
12
13 # makes a vector for the output layer from the correct output label
14 def convert_to_vector(correct_output):
15     output_layer = np.zeros((10, 1))
16     output_layer[correct_output] = 1
17     return output_layer
18
19 training_data, _, test_data = load_data()
20 # convert training data into the right format
21 training_inputs = [np.reshape(input_data, (784, 1)) for input_data in
22     ↪ training_data[0]]
22 training_outputs = [convert_to_vector(output_data) for output_data in
23     ↪ training_data[1]]
23 training_data = [{[
24     'input': training_inputs[example],
25     'output': training_outputs[example]
26 } for example in range(len(training_inputs))}]
27 # convert testing data into the right format
28 test_inputs = [np.reshape(input_data, (784, 1)) for input_data in test_data[0]]
29 test_outputs = test_data[1]
30 test_data = [{[
31     'input': test_inputs[example],
32     'output': test_outputs[example]
33 } for example in range(len(test_inputs))}]
34
35 # save data to a file
36 with open(f'data/digits.pkl', 'wb') as file:
37     pickle.dump({'training': training_data, 'testing': test_data}, file)

```

Code Snippet A.7: Final Code for `network_training / load_digits_data.py`

A.2.2 network_training / load_hdi_data.py

(This code was only ran a limited number of times, in Section 3.2.6)

```

1 import pickle
2 import gzip

```

```

3 import numpy as np
4
5 # loads the MNIST data set
6 def load_data():
7     with gzip.open('data/mnist.pkl.gz', 'rb') as file:
8         unpickled = pickle._Unpickler(file)
9         unpickled.encoding = 'latin1'
10        training_data, validation_data, test_data = unpickled.load()
11    return (training_data, validation_data, test_data)
12
13 # makes a vector for the output layer from the correct output label
14 def convert_to_vector(correct_output):
15     output_layer = np.zeros((10, 1))
16     output_layer[correct_output] = 1
17     return output_layer
18
19 training_data, _, test_data = load_data()
20 # convert training data into the right format
21 training_inputs = [np.reshape(input_data, (784, 1)) for input_data in
22     ↪ training_data[0]]
22 training_outputs = [convert_to_vector(output_data) for output_data in
23     ↪ training_data[1]]
23 training_data = [{{
24     'input': training_inputs[example],
25     'output': training_outputs[example]
26 } for example in range(len(training_inputs))}]
27 # convert testing data into the right format
28 test_inputs = [np.reshape(input_data, (784, 1)) for input_data in test_data[0]]
29 test_outputs = test_data[1]
30 test_data = [{{
31     'input': test_inputs[example],
32     'output': test_outputs[example]
33 } for example in range(len(test_inputs))}]
34
35 # save data to a file
36 with open(f'data/digits.pkl', 'wb') as file:
37     pickle.dump({'training': training_data, 'testing': test_data}, file)

```

Code Snippet A.8: Final Code for network_training / load_hdi_data.py

A.2.3 network_training / test_digits.py

(This code was only ran a limited number of times, in Section 3.2.5)

```

1 import pickle
2
3 # load the test data from the file

```

```

4  with open('data/digits.pkl', 'rb') as file:
5      data = pickle.load(file)
6  testing_data = data['testing']
7
8  # import the class MultilayerPerceptron from the neural_network.py file
9  import os
10 import sys
11 current = os.path.dirname(os.path.realpath(__file__))
12 parent = os.path.dirname(current)
13 sys.path.append(parent)
14 from neural_network import MultilayerPerceptron
15
16 # iterate over each of the testing examples and see how many it gets right
17 network = MultilayerPerceptron([784, 16, 16, 10])
18 network.load_model('models/digits.pkl')
19 success = 0
20 for total_so_far, example in enumerate(testing_data):
21     prediction = network.predict(example['input'])
22     list_of_probabilities = [prediction.item(x,0) for x in range(10)]
23     if list_of_probabilities.index(max(list_of_probabilities)) ==
24         example['output']:
25         success += 1
26     print(f'after {total_so_far+1} examples, the success rate is
27         {(success/(total_so_far+1))*100}%')

```

Code Snippet A.9: Final Code for network_training / test_digits.py

A.2.4 network_training / test_hdi.py

(This code was only ran a limited number of times, in Section 3.2.6)

```

1  import pickle
2
3  # load the test data from the file
4  with open('data/hdi.pkl', 'rb') as file:
5      data = pickle.load(file)
6  testing_data = data['testing']
7
8  # import the class MultilayerPerceptron from the neural_network.py file
9  import os
10 import sys
11 current = os.path.dirname(os.path.realpath(__file__))
12 parent = os.path.dirname(current)
13 sys.path.append(parent)
14 from neural_network import MultilayerPerceptron
15
16 # iterate over each of the testing examples and see how many it gets right

```

```

17 network = MultilayerPerceptron([24, 18, 12, 6, 3, 1])
18 network.load_model('models/hdi-temp.pkl')
19 differences = []
20 for num, example in enumerate(testing_data):
21     correctAnswer = example["output"].item(0,0)
22     prediction = round(network.predict(example['input']).item(0,0), 3)
23     difference = round(abs(correctAnswer-prediction), 3)
24     print(f'{num}) correct output: {correctAnswer}, prediction: {prediction},
25           difference: {difference}')
25     differences.append(difference)
26
27 import numpy as np
28 print(f'average difference: {np.mean(differences)}')

```

Code Snippet A.10: Final Code for `network_training / test_hdi.py`

A.2.5 `network_training / train_digits.py`

(This code was only ran a limited number of times, in Section 3.2.5)

```

1 import pickle
2
3 # load the training data from the file
4 with open('data/digits.pkl', 'rb') as file:
5     data = pickle.load(file)
6 training_data = data['training']
7
8 # import the class MultilayerPerceptron from the neural_network.py file
9 import os
10 import sys
11 current = os.path.dirname(os.path.realpath(__file__))
12 parent = os.path.dirname(current)
13 sys.path.append(parent)
14 from neural_network import MultilayerPerceptron
15
16 # train the network
17 network = MultilayerPerceptron([784, 16, 16, 10])
18 network.train(training_data, 10, 30, 3)
19 network.save_model('digits')

```

Code Snippet A.11: Final Code for `network_training / train_digits.py`

A.2.6 `network_training / train_hdi.py`

(This code was only ran a limited number of times, in Section 3.2.6)

```

1 import pickle
2

```

```

3 # load the training data from the file
4 with open('data/hdi.pkl', 'rb') as file:
5     data = pickle.load(file)
6 training_data = data['training']
7
8 # import the class MultilayerPerceptron from the neural_network.py file
9 import os
10 import sys
11 current = os.path.dirname(os.path.realpath(__file__))
12 parent = os.path.dirname(current)
13 sys.path.append(parent)
14 from neural_network import MultilayerPerceptron
15
16 # train the network
17 network = MultilayerPerceptron([24, 18, 12, 6, 3, 1])
18 network.train(training_data, 5, 50, 0.2)
19 network.save_model('hdi-temp')

```

Code Snippet A.12: Final Code for `network_training / train_hdi.py`

A.3 The Main App

This section will include all other files in the source code directory. They are the files which are used by the app while it is running (as well as `data_processing / get_osm_data.py` in Section A.1.3)

A.3.1 `app.py`

(Note, on line 33, the password to access the database has been replaced with '`<db_password>`.)

```

1 import gradio as gr
2 from gradio_folium import Folium
3 import folium
4 from folium.plugins import Draw
5 import random
6 import json
7 from data_processing.get_osm_data import get_osm_data
8 from calc_factors import averageDistance, calcArea, density, distBetween2Points
9 import numpy as np
10 from neural_network import MultilayerPerceptron
11 import copy
12 from pymongo.mongo_client import MongoClient
13 from pymongo.server_api import ServerApi
14 import certifi
15 import bcrypt

```

```

16 import csv
17
18 # initialise neural network
19 network = MultilayerPerceptron([24, 18, 12, 6, 3, 1])
20 network.load_model('models/hdi.pkl')
21
22 # define global vars
23 currentRegion = []
24 all_objects = {}
25 allFactors = []
26 currentHDI = -1
27 allUserRegions = []
28 loggedInUsername = ''
29 averageDistanceFactors = [['house', 'school'], ['house', 'hospital'], ['house',
   ↵ 'pharmacy'], ['house', 'restaurant'], ['school', 'hospital'], ['police',
   ↵ 'hospital'], ['house', 'place_of_worship'], ['bank', 'slot_machines'],
   ↵ ['fast_food', 'toilets'], ['house', 'police'], ['university', 'library'],
   ↵ ['house', 'library']]
30 densityFactors = ['school', 'hospital', 'pharmacy', 'police', 'library',
   ↵ 'toilets', 'restaurant', 'place_of_worship', 'post_box', 'vending_machine',
   ↵ 'bench', 'tree']
31
32 # connect to mongoDB
33 mongoURI = "mongodb+srv://lukerhodri:<db_password>@predictinghdi.3nkqe.mongodb
   ↵ .net/?retryWrites=true&w=majority&appName=predictingHDI"
34 mongoClient = MongoClient(mongoURI, server_api=ServerApi('1'),
   ↵ tlsCAFile=certifi.where())
35 users = mongoClient['predictingHDI']['users']
36 # send a ping to confirm a successful connection
37 mongoClient.admin.command('ping')
38 print("Pinged your deployment. You successfully connected to MongoDB!")
39
40 # load the training data
41 with open('data/training_data.csv', 'r') as file:
42     reader = csv.DictReader(file)
43     trainingData = []
44     for record in reader:
45         trainingData.append(record)
46
47 # code to process the user uploading their geojson file
48 def uploadGeoJSON(filename, max_x_objects, max_y_objects, currentRegionName,
   ↵ progress=gr.Progress()):
49     global currentRegion
50     global all_objects
51     global allFactors
52     global allUserRegions
53     progress(0, desc='Starting...')
```

```

54     with open(filename, 'r') as file:
55         data = json.load(file)
56         currentRegion = [coordinate[::-1] for coordinate in
57             ↪ data['features'][0]['geometry']['coordinates'][0]]
58     shortFilename = filename[filename.rindex("/") + 1:]
59     if shortFilename[:shortFilename.rindex('.')] in [region['name'] for region in
60         ↪ allUserRegions] or shortFilename[:shortFilename.rindex('.')] == '':
61         gr.Warning('There is already a region with this name! Please rename the
62             ↪ file first.')
63     return 'There is already a region with this name! Please rename the file
64             ↪ first.', updateRegionsTable(), updateRegionsDropdown(),
65             ↪ currentRegionName
66     # retrieve all relevant osm data
67     object_types = ['house', 'school', 'hospital', 'pharmacy', 'restaurant',
68         ↪ 'place_of_worship', 'bank', 'slot_machines', 'fast_food', 'toilets',
69         ↪ 'police', 'university', 'library', 'post_box', 'vending_machine',
70         ↪ 'bench', 'tree']
71     all_objects = {}
72     for num, object_type in enumerate(object_types):
73         progress(0.3 * (num / len(object_types)), desc=f'Downloading {object_type}
74             ↪ data...')
75         print(f'getting {object_type} data...') # log message
76         all_objects[object_type] = get_osm_data(object_type, currentRegion)
77     # calculate average distances
78     allFactors = []
79     for num, averageDistanceFactor in enumerate(averageDistanceFactors):
80         progress(0.3 + 0.6 * (num / len(averageDistanceFactors)), desc=f'Calculating
81             ↪ Average Distance between {averageDistanceFactor[0]} and
82             ↪ {averageDistanceFactor[1]}...')
83         print(f'finding A({averageDistanceFactor[0]},'
84             ↪ {averageDistanceFactor[1]}...)') # log message
85         allFactors.append(averageDistance(all_objects[averageDistanceFactor[0]],
86             ↪ all_objects[averageDistanceFactor[1]], max_x_objects, max_y_objects))
87     # calculate area
88     progress(0.9, desc='Calculating Area...')
89     print('calculating area...') # log message
90     area = calcArea(currentRegion)
91     # calculate density factors
92     for num, densityFactor in enumerate(densityFactors):
93         progress(0.91 + 0.09 * (num / len(densityFactors)), desc=f'Calculating
94             ↪ {densityFactor} Density...')
95         print(f'finding D({densityFactor})') # log message
96         allFactors.append(density(all_objects[densityFactor], area))
97     allUserRegions.append({
98         'name': shortFilename[:shortFilename.rindex('.')],
99         'objects': all_objects,
100        'factors': allFactors
101    }

```

```

87 })
88 # save regions to account, if logged in
89 if loggedInUsername != '':
90     users.update_one({'username': loggedInUsername}, {'$set': {'regions':
91         ↪ allUserRegions}})
92 gr.Info(f'Successfully Uploaded {shortFilename}', 5)
93 return f'Successfully Uploaded {shortFilename}', updateRegionsTable(),
94     ↪ updateRegionsDropdown(), shortFilename[:shortFilename.rindex('.')]
95
96 # predict the HDI of the given region
97 def processPrediction():
98     global currentHDI
99     if allFactors == []:
100        gr.Warning('You have not uploaded a region!')
101        return 'You have not uploaded a region!', ''
102    # predict HDI
103    inputLayer = np.matrix([[100 if factor == None else float(factor)/10 if index
104        ↪ <= 11 else float(factor)] for index, factor in enumerate(allFactors)])
105    prediction = round(network.predict(inputLayer).item(0,0), 3)
106    currentHDI = prediction
107    # find similar region
108    diff = 0
109    possible_choices = []
110    while len(possible_choices) == 0:
111        # search below & above (round to remove floating point errors)
112        possible_choices += [region for region in trainingData if region['hdi']
113            ↪ == str(round(prediction-diff, 3))]
114        possible_choices += [region for region in trainingData if region['hdi']
115            ↪ == str(round(prediction+diff, 3))]
116        # increment difference (round to remove floating point errors)
117        diff = round(diff + 0.001, 3)
118    # choose one of the options at random
119    chosenRegionRecord = random.choice(possible_choices)
120    similar_region = f'{chosenRegionRecord["region"]},
121        ↪ {chosenRegionRecord["country"]}'
122    gr.Info('Successfully Predicted HDI!', 5)
123    return prediction, similar_region
124
125 # helper function to calculate mean of coordinates
126 def calcMeanOfCoords(coords):
127     return [np.mean([coord[0] for coord in coords]), np.mean([coord[1] for coord
128         ↪ in coords])]
129
130 # calculate the optimal place for a new building to go when suggesting
131 def calcOptimalPlaceFor(building, extraBuildings):
132     global all_objects
133     # construct all_objects_in_consideration

```

```

127     all_objects_in_consideration = {}
128     for key in all_objects.keys():
129         if key in extraBuildings.keys():
130             all_objects_in_consideration[key] = all_objects[key] +
131                 extraBuildings[key]
132         else:
133             all_objects_in_consideration[key] = all_objects[key]
134     # iterate over the factors
135     optimalPositions = []
136     zeroOfEverything = True
137     zeroOfThings = []
138     for averageDistanceFactor in averageDistanceFactors:
139         if averageDistanceFactor[1] == building:
140             # initialise variables
141             x_objects = all_objects_in_consideration[averageDistanceFactor[0]]
142             y_objects = all_objects_in_consideration[averageDistanceFactor[1]]
143             # update zero of everything flag
144             if len(x_objects) != 0:
145                 zeroOfEverything = False
146                 # calculate the shortest distance from each
147                 # {averageDistanceFactor[0]} to {averageDistanceFactor[1]}
148                 shortest_dists = []
149                 for x_object in x_objects:
150                     dists = []
151                     if len(y_objects) != 0:
152                         for y_object in y_objects:
153                             dists.append(distBetween2Points(x_object, y_object))
154                     else:
155                         dists.append(10000)
156                     if min(dists) == 0:
157                         shortest_dists.append(0.00001)
158                     else:
159                         shortest_dists.append(min(dists))
160                 # calculate the optimal position
161                 list_of_numpy_arrays = [np.array(x_object) for x_object in
162                     x_objects]
163                 optimalPosition = (sum([x_object / shortest_dists[index] for index,
164                     x_object in enumerate(list_of_numpy_arrays)] ,
165                     np.array([0, 0])))/(sum([1 / shortest_dist for shortest_dist in
166                     shortest_dists]))
167                 optimalPositions.append(optimalPosition)
168             else:
169                 zeroOfThings.append(averageDistanceFactor[0])
170     # return either something else to build, or the new position
171     if zeroOfEverything:
172         return random.choice(zeroOfThings)
173     return calcMeanOfCoords(optimalPositions)

```

```

168
169 # make suggestions from a prediction
170 def makeSuggestions(max_x_objects, max_y_objects, num_new_buildings,
171     → progress=gr.Progress()):
172     global all_objects
173     if currentHDI == -1:
174         gr.Warning('You have not yet predicted an HDI! Please predict one
175         → first.')
176         return [['You have not yet predicted an HDI!', None, None, None]]
177     if all_objects == {}:
178         gr.Warning('This region does not have any associated buildings! Please
179         → choose a different region')
180         return [['This region does not have any associated buildings!', 'Please
181         → choose a different region', None, None]]
182     suggestions = ['school', 'hospital', 'place_of_worship', 'police',
183     → 'restaurant', 'slot_machines', 'library', 'pharmacy']
184     returnTable = []
185     for num, suggestion in enumerate(suggestions):
186         extraBuildings = {suggestion: []}
187         progress(num/len(suggestions), desc=f'Suggesting building a
188         → {suggestion}... ')
189         # find optimal place
190         positions = []
191         for buildingNum in range(num_new_buildings):
192             # check for other suggestion
193             if len(positions) > 0:
194                 if type(positions[0]) == type(''):
195                     continue
196             # calc actual position
197             progress((num/len(suggestions))+(buildingNum*(1/8)*(1/(
198                 → num_new_buildings+3))), desc=f'Suggesting building a {suggestion}
199                 → (finding optimal position,
200                 → {buildingNum+1}/{num_new_buildings})... ')
201             position = calcOptimalPlaceFor(suggestion, extraBuildings)
202             positions.append(position)
203             extraBuildings[suggestion].append(position)
204             if type(positions[0]) != type(''):
205                 # calculate average distances
206                 progress((num/len(suggestions))+((num_new_buildings)*(1/8)*(1/(
207                     → num_new_buildings+3))), desc=f'Suggesting building a {suggestion}
208                     → (calculating new average distances)... ')
209             allFactors = []
210             for averageDistanceFactor in averageDistanceFactors:
211                 # ensure that we include the new building
212                 if suggestion == averageDistanceFactor[0]:

```

```

202         allFactors.append(averageDistance(all_objects[
203             ↵     averageDistanceFactor[0]],
204             ↵     all_objects[averageDistanceFactor[1]], max_x_objects,
205             ↵     max_y_objects, must_include_x=positions))
206     elif suggestion == averageDistanceFactor[1]:
207         allFactors.append(averageDistance(all_objects[
208             ↵     averageDistanceFactor[0]],
209             ↵     all_objects[averageDistanceFactor[1]], max_x_objects,
210             ↵     max_y_objects, must_include_y=positions))
211     else:
212         allFactors.append(averageDistance(all_objects[
213             ↵     averageDistanceFactor[0]],
214             ↵     all_objects[averageDistanceFactor[1]], max_x_objects,
215             ↵     max_y_objects))
216     # calculate density factors
217     progress((num/len(suggestions))+((num_new_buildings+1)*(1/8)*(1/(
218         ↵     num_new_buildings+3))), desc=f'Suggesting building a {suggestion}
219             ↵     (calculating new densities)...')
220     area = calcArea(currentRegion)
221     for densityFactor in densityFactors:
222         # ensure that we include the new building
223         if suggestion == densityFactor:
224             allFactors.append(density(all_objects[densityFactor] +
225                 ↵     [position], area))
226         else:
227             allFactors.append(density(all_objects[densityFactor], area))
228     # predict HDI
229     progress((num/len(suggestions))+((num_new_buildings+2)*(1/8)*(1/(
230         ↵     num_new_buildings+3))), desc=f'Suggesting building a {suggestion}
231             ↵     (predicting new HDI)...')
232     inputLayer = np.matrix([[100 if factor == None else float(factor)/10
233         ↵     if index <= 11 else float(factor)] for index, factor in
234         ↵     enumerate(allFactors)])
235     prediction = round(network.predict(inputLayer).item(0,0), 3)
236     returnTable.append([suggestion, ', '.join([f'{round(position[0],
237         ↵     7)}°N, {round(position[1], 7)}°E' for position in positions]),
238         ↵     str(prediction), str(round(prediction-currentHDI, 3))])
239     else:
240         returnTable.append([suggestion, f'Build a new {positions[0]} first!',
241             ↵     '--', '--'])
242     gr.Info('Successfully Made Suggestions!', 5)
243     return returnTable
244
245
246     # place a pin on the map when user clicks the suggestion
247     def selectSuggestionsTable(evt: gr.SelectData):
248         cellData = evt.value
249         newMap = copy.deepcopy(foliumMap)

```

```

230     if 'o' in cellData: # it is a coordinate
231         splitByCommas = cellData.split(',')
232         coordinates = []
233         for halfIndex in range(int(len(splitByCommas)/2)):
234             coordinates.append([float(splitByCommas[2*halfIndex][1:-2]),
235                                 float(splitByCommas[2*halfIndex+1][-3])])
236         for coordinate in coordinates:
237             folium.Marker(location=coordinate).add_to(newMap)
238         newMap.location=random.choice(coordinates)
239
240     # user selects an item from the dropdown menu to compare to
241     def compareRegions(regionName):
242         # confirm the user has predicted
243         if allFactors == []:
244             gr.Warning('Please predict a region first!')
245             return [['Please predict a region first!', None, None]]
246         # undo the concatenation
247         split = regionName.split(',')
248         country = split[-1]
249         region = ', '.join(split[:-1])
250         # find the region record & get lists of factors
251         regionRecord = [record for record in trainingData if record['country'] ==
252                         country and record['region'] == region][0]
253         yourFactors = ['undefined' if factor == None else round(float(factor), 5) for
254                         factor in allFactors]
255         selectedFactors = ['undefined' if factor == '' else round(float(factor), 5)
256                         for factor in list(regionRecord.values())[4:]]
257         # construct the return list
258         factorNames = ['Average Distance from House to nearest School (km)', 'Average
259                         Distance from House to nearest Hospital (km)', 'Average Distance from
260                         House to nearest Pharmacy (km)', 'Average Distance from House to nearest
261                         Restaurant (km)', 'Average Distance from School to nearest Hospital
262                         (km)', 'Average Distance from Police Station to nearest Hospital
263                         (km)', 'Average Distance from House to nearest Place of Worship
264                         (km)', 'Average Distance from Bank to nearest Slot Machine (km)', 'Average
265                         Distance from Fast-Food Place to nearest Toilet (km)', 'Average Distance
266                         from House to nearest Police Station (km)', 'Average Distance from
267                         University to nearest Library (km)', 'Average Distance from House to
268                         nearest Library (km)', 'Number of Schools per km2', 'Number of Hospitals
269                         per km2', 'Number of Pharmcies per km2', 'Number of Police Stations per
270                         km2', 'Number of Libraries per km2', 'Number of Toilets per km2', 'Number of
271                         Restaurants per km2', 'Number of Places of Worship per km2', 'Number of
272                         Post Boxes per km2', 'Number of Vending Machines per km2', 'Number of
273                         Benches per km2', 'Number of Trees per km2']
274         returnList = [['HDI', currentHDI, regionRecord['hdi']]]
275         for num, factor in enumerate(factorNames):

```

```

258         returnList.append([factor, yourFactors[num], selectedFactors[num]])
259     return returnList
260
261 # creates a 2d list to put in the regions table, from allUserRegions
262 def updateRegionsTable():
263     table = []
264     for region in allUserRegions:
265         table.append([region['name']] + region['factors'])
266     # if allUserRegions is empty, return an empty table
267     if table == []:
268         table.append([''*25])
269     return table
270
271 # updates regions dropdown, from allUserRegions
272 def updateRegionsDropdown():
273     names = [region['name'] for region in allUserRegions]
274     return gr.update(choices=names)
275
276 # updates the allUserRegions list, when the regions table is updated
277 def updateAllUserRegions(table, currentRegionName):
278     global allUserRegions
279     # store the index of the current region
280     if currentRegionName != None:
281         currentRegionIndex = [region['name'] for region in
282                             allUserRegions].index(currentRegionName)
283     # validate the input - names of regions
284     regionNames = [row[0] for row in table]
285     if '' in regionNames or len(set(regionNames)) != len(regionNames):
286         gr.Warning('Please ensure every region has a unique name!')
287         return 'Please ensure every region has a unique name!',
288         updateRegionsDropdown(), currentRegionName
289     # validate the input - type check
290     for rowIndex, row in enumerate(table):
291         for cellIndex, cell in enumerate(row):
292             if cellIndex != 0 and (cellIndex > 12 or cell != ''):
293                 try:
294                     table[rowIndex][cellIndex] = float(cell)
295                 except ValueError:
296                     gr.Warning(f'Please Enter the Data Correctly! Found an error
297                     at cell (row {rowIndex}, col {cellIndex})')
298                     return f'Please Enter the Data Correctly! Found an error at
299                     cell (row {rowIndex}, col {cellIndex}),
299                     updateRegionsDropdown(), currentRegionName
300
301     # update the allUserRegions list
302     initialLength = len(allUserRegions)
303     for index, row in enumerate(table):
304         factors = [None if value == '' else value for value in row[1:]]

```

```

300     # if the user has changed the factors of an existing list, the associated
301     # objects should be removed
302     if index < initialLength:
303         if [float(factor) if factor != '' else None for factor in row[1:]] != 
304             allUserRegions[index]['factors']:
305                 allUserRegions[index] = {'name': row[0], 'objects': {}, 
306                                         'factors': factors}
307             else:
308                 allUserRegions[index]['name'] = row[0]
309             # if the user has added a new region, it should have no associated
310             # objects
311             else:
312                 allUserRegions.append({'name': row[0], 'objects': {}, 'factors': 
313                                         factors})
314             # save regions to account, if logged in
315             if loggedInUsername != '':
316                 users.update_one({'username': loggedInUsername}, {'$set': {'regions': 
317                                         allUserRegions}})
318             # update the name of the selected region
319             if currentRegionName == None:
320                 newRegionName = None
321             else:
322                 newRegionName = allUserRegions[currentRegionIndex]['name']
323             gr.info('Successfully Updated Regions', 5)
324             return 'Successfully Updated Regions', updateRegionsDropdown(), newRegionName
325
326             # updates the current region, when the user switches it with the dropdown on the
327             # regions tab
328             def updateCurrentRegion(dropdownValue):
329                 global all_objects
330                 global allFactors
331                 global currentHDI
332                 for region in allUserRegions:
333                     if region['name'] == dropdownValue:
334                         all_objects = region['objects']
335                         allFactors = region['factors']
336                         # predict HDI - to update it internally
337                         inputLayer = np.matrix([[100 if factor == None else float(factor)/10
338                                         if index <= 11 else float(factor)] for index, factor in 
339                                         enumerate(allFactors)])
340                         prediction = round(network.predict(inputLayer).item(0,0), 3)
341                         currentHDI = prediction
342
343             # validate sign up & add account to mongoDB
344             def signUp(username, password, password2):
345                 global loggedInUsername
346                 # validate no empty fields

```

```

338     if username == '' or password == '' or password2 == '':
339         gr.Warning('A field has been left empty! Please fill in all fields.')
340         return 'Please fill in all fields!', updateRegionsTable(),
341             ↵ updateRegionsDropdown()
342     # validate username - does not already exist
343     if users.count_documents({'username': username}) != 0:
344         gr.Warning('This username is already in use! Please choose another
345             ↵ username.')
346         return 'This username is already in use! Please choose another
347             ↵ username.', updateRegionsTable(), updateRegionsDropdown()
348     # validate password - double entry validation
349     if password != password2:
350         gr.Warning('The passwords do not match! Please ensure that you have
351             ↵ entered the correct password.')
352         return 'The passwords do not match! Please ensure that you have entered
353             ↵ the correct password.', updateRegionsTable(), updateRegionsDropdown()
354     # validate password - at least 8 characters long
355     if len(password) < 8:
356         gr.Warning('Your password must have at least 8 characters!')
357         return 'Your password must have at least 8 characters!',
358             ↵ updateRegionsTable(), updateRegionsDropdown()
359     # validate password - at least 1 digit
360     digits = '0123456789'
361     hasDigit = False
362     for digit in digits:
363         if digit in password:
364             hasDigit = True
365     if not hasDigit:
366         gr.Warning('Your password must include at least 1 digit!')
367         return 'Your password must include at least 1 digit!',
368             ↵ updateRegionsTable(), updateRegionsDropdown()
369     # validate password - at least 1 special character
370     specialChars = '!#"${&\`()*+,.-/:;=>?@[\\]^_`{|}~'
371     hasSpecialChar = False
372     for specialChar in specialChars:
373         if specialChar in password:
374             hasSpecialChar = True
375     if not hasSpecialChar:
376         gr.Warning('Your password must include at least 1 special character!')
377         return f'Your password must include at least 1 special character!
378             ↵ ({specialChars})', updateRegionsTable(), updateRegionsDropdown()
379     # salt & hash password
380     salt = bcrypt.gensalt()
381     hashed_password = bcrypt.hashpw(password.encode('utf-8'), salt)
382     # add to mongoDB
383     user = {
384         'username': username,

```

```

377     'hashedPassword': hashed_password,
378     'regions': allUserRegions
379 }
380 users.insert_one(user)
381 loggedInUsername = username
382 gr.Info('Successfully created & logged into an account', 5)
383 return f'Successfully created & logged into an account, with username:
384     {username}!', updateRegionsTable(), updateRegionsDropdown()

385 # validate & log in the user
386 def logIn(username, password):
387     global loggedInUsername
388     global allUserRegions
389     # validate no empty fields
390     if username == '' or password == '':
391         gr.Warning('A field has been left empty! Please fill in all fields.')
392         return 'Please fill in all fields!', updateRegionsTable(),
393             updateRegionsDropdown()
394     # check username exists
395     if users.count_documents({'username': username}) == 0:
396         gr.Warning('This username does not exist! Please ensure that you have
397             entered the correct username.')
398         return 'This username does not exist! Please ensure that you have entered
399             the correct username.', updateRegionsTable(), updateRegionsDropdown()
400     # check password
401     user = users.find_one({'username': username})
402     if not bcrypt.checkpw(password.encode('utf-8'), user['hashedPassword']): #
403         # the hashed passwords do not match
404         gr.Warning('Incorrect Password! Please try again.')
405         return 'Incorrect Password! Please try again.', updateRegionsTable(),
406             updateRegionsDropdown()
407     # grant access to the account
408     loggedInUsername = username
409     allUserRegions = user['regions']
410     gr.Info(f'Successfully logged in as {username}!', 5)
411     return f'Successfully logged in as {username}!', updateRegionsTable(),
412         updateRegionsDropdown()

413 # define some starting locations
414 locations = [[51.5360, -0.1196], [40.8093, -73.9678], [34.0069, -118.2304],
415     [25.7617, -80.1918], [48.8488, 2.3470], [41.8840, 12.4790], [52.5078,
416     13.4003], [37.9964, 23.7293], [59.3265, 18.0680], [40.4172, -3.6867],
417     [41.3940, 2.1606], [38.7253, -9.1403], [55.7476, 37.6209], [31.2230,
418     121.4860], [39.8958, 116.4000], [37.5467, 126.9901], [35.6777, 139.7685],
419     [1.3294, 103.8081], [-33.8782, 151.1754], [30.0465, 31.2246], [6.5047,
420     3.3732], [-1.2873, 36.8198], [-33.9425, 18.5065], [19.3916, -99.1279],
421     [-23.5727, -46.6207], [36.1458, -115.1832], [38.8939, -77.0372], [49.2501,
422     -123.1164], [43.6888, -79.4190]]

```

```
410
411 # initialise the map for drawing on
412 foliumMap = folium.Map(location=random.choice(locations))
413 draw = Draw(
414     export=True,
415     filename='region.geojson',
416     position='topleft',
417     draw_options={
418         'polyline': False,
419         'circlemarker': False,
420         'polygon': {'allowIntersection': False},
421         'marker': False,
422         'circle': False,
423         'rectangle': False,
424     },
425     edit_options={'poly': {'allowIntersection': False}}
426 )
427 draw.add_to(foliumMap)
428
429 # define markdown in help tab
430 markdown = """
431 # \U0001F44B Welcome to Predicting HDI! \U0001F44B
432
433 This app allows you to predict the HDI of any region you draw on the map. From
434 → there, you can make some suggestions on how to improve it.
435 ## Getting Started \U000002728
436
437 Head over to the \U0001F52E Predict tab, and find your region on the map. Once
438 → you have found your region, you can predict the HDI by following these steps:
439 1. Select the Draw Tool by Clicking the **Pentagon** \U000002B53 Icon in the Top
440 → Left.
441 2. Draw your Region, by creating vertices. Make sure you close the region by
442 → clicking on your first vertex at the end.
443 3. Press the **Export** button in the Top Right. This will download a file called
444 → `region.geojson` to your computer.
445 4. Press the \U0001F4E4 **Upload the GeoJSON file** \U0001F4E4 button, below the
446 → map, and choose your `region.geojson`. You can scroll down to see a progress
447 → bar for this upload. Once the upload is complete, you will get a notification
→ in the top right corner.
448 5. Finally, press the orange \U0001F52E **Predict HDI** \U0001F52E button to
→ predict the HDI of your region!
449
450 
451
452 ## More Detail on The \U0001F52E Predict Tab
```

448
449 ### Help! It's Taking Too Long! \U0001F553
450
451 **If** you drew a very large region, it is likely that Uploading it took a **very
 ↳ long** time \U0001F553. This is because, to predict the HDI, we use data
 ↳ about the positions of buildings, such as houses, schools, hospitals and
 ↳ police stations. This data must be downloaded, so if you chose a region with
 ↳ a lot of these things, then it will take quite a while to download it all.
452
453 **To** predict the HDI from these positions, we calculate different different
 ↳ factors, such as the average distance from a house to the nearest school, or
 ↳ nearest hospital, which can also take a while. To decrease the time this may
 ↳ take, you can change the values of the **Maximum x-buildings** \U0001F3E0 and
 ↳ **Maximum y-buildings** \U0001F3EB sliders.
454
455 **Changing** **Maximum x-buildings** \U0001F3E0 will decrease the number of
 ↳ x-buildings we consider. For example, in the case of average distance from
 ↳ house to school, this slider corresponds to the number of houses we consider.
 ↳ Similarly, changing **Maximum y-buildings** \U0001F3EB corresponds to
 ↳ changing the number of schools we consider for each house (in this example).
 ↳ However, this will cause the calculated values for these factors to be less
 ↳ accurate, and therefore may cause the predicted HDI to be less accurate.
456
457 ### Help! It Just Crashed! \U00000274C
458
459 **If** the progress bar suddenly turned into an **ERROR** when uploading a region, it
 ↳ is likely that you just timed out from the OverpassTurbo servers, which is
 ↳ what we are using to get the location of all these buildings. This is totally
 ↳ normal, and so if this happens, just try to upload it again.
460
461 **If** the issue persists, it is likely something is wrong with your region, so try a
 ↳ different one
462
463 ### Suggestions \U0001F4A1
464
465 **Once** you have predicted the HDI of a region, you can make some suggestions. Do
 ↳ this, by simply pressing the \U0001F914 **Make Suggestions** \U0001F914
 ↳ button. The suggestions will be in the format: "Build (a number of) new
 ↳ buildings, in these locations". You can change this value, to anything
 ↳ between 1 and 10, by moving the slider, labelled **Number of New Buildings**
 ↳ \U0001F3D9.
466
467 **This** may also take some time, As we must calculate the optimal position to place
 ↳ these new buildings, and the re-calculate the factors. Once it is finished,
 ↳ the Suggestions \U0001F4A1 Table will be populated with 8 different
 ↳ suggestions, on how to improve the HDI of your region. To view the locations
 ↳ of the suggestion on the map, simply select the cell that contains the
 ↳ coordinates. The map will update to have pins.

```
468
469 
470
471 If your region is quite empty, you may have got some suggestions that did not
    ↵ return any locations, or a new HDI, but instead a prompt to build some other
    ↵ buildings first. This is because we did not have sufficient data to calculate
    ↵ where a new building should go, and so unfortunately, there is no real
    ↵ suggestion.
472
473 ### What Makes a Good Region? \U0001F914
474
475 This brings me onto how to select a region, which will yield good results.
476
477 - **Don't make your region too large.** Larger regions will take longer to
    ↵ download, and longer to calculate the factors of. You will therefore want to
    ↵ decrease the **Maximum x-buildings** \U0001F3E0 and **Maximum y-buildings**
    ↵ \U0001F3EB sliders, leading to a less accurate prediction. You will then also
    ↵ not get very good suggestions, as building 5 schools in a large metropolitan
    ↵ area isn't going to do anything.
478 - **Ensure your region doesn't self-intersect.** Your region should be a simple
    ↵ 2D shape.
479 - **Use Small Towns.** I believe this works best with small towns, as the
    ↵ suggestions will often increase the HDI by a more significant amount.
    ↵ Downloading the Region and Calculating the Factors also won't take too long.
480 - **Don't Make Empty Regions.** Predicting the HDI of an empty field or the ocean
    ↵ is not going to yield any useful results.
481
482 ### Drawing Regions \U0001F58B
483
484 If you make a mistake while drawing your region, you can press the "Delete last
    ↵ point" button, next to the Pentagon \U00002B53 icon. If you wish to delete
    ↵ your region, press the Bin \U0001F5D1 icon in the top left.
485
486 You can zoom in and out of the map, by using the scroll wheel, or by using the +
    ↵ and - buttons in the top right.
487
488 If you draw multiple regions before pressing Export, we will only consider the
    ↵ first one you drew. If you wish to predict the HDI of 2 disconnected regions,
    ↵ you can get around this by connecting them with a very thin line.
489
490 ## The \U0001F19A Compare Tab
491
492 ### Comparing your Region \U0001F4CA
493
494 You can use the \U0001F19A Compare tab to compare your region to an existing
    ↵ sub-national region. Sub-national regions are 1 level below countries, so for
    ↵ example, the sub-national regions of the United States are the 50 states.
```

495
496 To do this, simply select a region from the dropdown menu at the top of the
 → screen. To search for a specific region, you can also type into this dropdown
 → menu. When you select one, the table below will populate with the HDI of your
 → region and the selected region, as well as each of the factors that
 → contribute to it.

497
498
499
500 \U0001F4C4 **Note:** You must predict the HDI of a region first, before you do
 → this.

501
502 ### Correlation Heatmap \U0001F4C8
503
504 At the bottom of this tab, you will also find a graph, showing the correlation
 → between each factor, and HDI. You can use this to find out which factors
 → contribute most towards the HDI, as well as which ones contribute towards
 → each other.

505
506 Each cell has a colour, which corresponds to a number between -1 and 1, as shown
 → by the key on the right. A value of 1 means that the 2 factors are very
 → positively correlated, a value of -1 means that the 2 factors are very
 → negatively correlated, and a value of 0 means that the 2 factors are not
 → correlated. All values in between are as you would expect (for example, a
 → value of -0.3 would mean that the 2 factors are slightly negatively
 → correlated).

507
508 ## The \U0001F30D Regions Tab
509
510 In the \U0001F30D Regions tab, you can switch between different regions that you
 → have predicted the HDI of. To do this, simply select one from the dropdown
 → menu at the top of the screen.

511
512 This tab also allows you to manually edit the factors of your regions, rename
 → your regions, and add new ones. You can edit any cell in the table, so long
 → as they follow these rules:

513
514 - The values in the columns for A(... ...) must either be a number or empty
515 - The values in the columns for D(...) must be a number
516 - The values in the name column must be unique

517
518 (A(x y) means "Average distance from x to nearest y", and D(x) means "Number of x
 → per unit area")

519
520 Once you have made your changes, you can press the \U0001F504 **Submit Table
 → Changes** \U0001F504 button, to make your changes. If any of the rules above
 → have not been met, the changes will not go through, and you will be told
 → which cell is causing an error.

```
521
522 \U00002757 **WARNING:** You will no longer be able to make suggestions from
  ↵ regions that you edit in the table. \U00002757 This is because the factors no
  ↵ longer reflect the positions of the buildings in the region, and so they are
  ↵ no longer accurate. We therefore cannot make any suggestions, as there are no
  ↵ buildings associated with the region.
523
524 ## Account Management \U0001F510
525
526 You can also create an account, to save your regions.
527
528 ### Creating an Account \U0001F4DD
529
530 To create an account, simply head to the \U0001F4DD Sign Up tab. Here, you get to
  ↵ choose a username and password. Your username must be unique, and your
  ↵ password must:
531
532 - Be at least 8 characters long
533 - Contain at least 1 digit
534 - Contain at least 1 special character. The characters that count as special
  ↵ characters are: ! " # $ % & \' ( ) * + , - . / : ; < = > ? @ [ \\ ] ^ _ ` { |
  ↵ }
535
536 When you create an account, you will automatically be logged in. Any changes you
  ↵ make while logged in will be saved to your account.
537
538 ### Logging Into an Account \U0001F513
539
540 To log into your account, head to the \U0001F513 Log In tab, and enter your
  ↵ username and password.
541
542 \U00002757 **WARNING:** Logging in will override the regions in the \U0001F30D
  ↵ Regions Tab, and replace them with the ones in the account you log into.
  ↵ \U00002757
543 !!!
544
545 # structure of the UI
546 with gr.Blocks() as app:
547     with gr.Tab(label='\U0001F64B Help'):
548         gr.Markdown(markdown)
549     with gr.Tab(label='\U0001F52E Predict'):
550         with gr.Row():
551             with gr.Column(scale=3):
552                 map = Folium(value=foliumMap)
553                 uploadButton = gr.UploadButton(label='\U0001F4E4 Upload the
  ↵ GeoJSON file \U0001F4E4', file_count='single')
554             with gr.Column(scale=1):
```

```

555         predictHDIbutton = gr.Button(value='\\U0001F52E Predict HDI
556             ↪  \\U0001F52E', variant='primary')
557         with gr.Group():
558             max_x_objectsSlider = gr.Slider(minimum=10, maximum=5000,
559                 ↪  value=500, label='Maximum x-buildings \\U0001F3E0',
560                 ↪  info='When calculating the average distance between 2
561                 ↪  types of building, it will find the average of this many
562                 ↪  pairs. Increasing this value may make the prediction take
563                 ↪  much longer, but may make the prediction more accurate.')
564             max_y_objectsSlider = gr.Slider(minimum=10, maximum=5000,
565                 ↪  value=500, label='Maximum y-buildings \\U0001F3EB',
566                 ↪  info='When calculating the average distance between 2
567                 ↪  types of building, it will find the closest of the second
568                 ↪  type of building out of this many options. Increasing
569                 ↪  this value may make the prediction take much longer, but
570                 ↪  may make the prediction more accurate.')
571             with gr.Group():
572                 HDIprediction = gr.Textbox(label='I believe that the HDI of
573                     ↪  this region is... \\U00002728', value='',
574                     ↪  interactive=False)
575                 similarHDI = gr.Textbox(label='That\\'s a similar HDI to...
576                     ↪  \\U0001F30D', value='', interactive=False)
577             with gr.Row():
578                 makeSuggestionsButton = gr.Button(value='\\U0001F914 Make Suggestions
579                     ↪  \\U0001F914')
580                 newBuildingsSlider = gr.Slider(minimum=1, maximum=10, value=5,
581                     ↪  step=1, label='Number of New Buildings \\U0001F3D9',
582                     ↪  interactive=True)
583                 suggestionsTable = gr.DataFrame(label='Suggestions \\U0001F4A1',
584                     ↪  headers=['New Building', 'Coordinates', 'New HDI', 'Change in HDI'],
585                     ↪  type='array', interactive=False)
586                 log = gr.Textbox(label='Log Messages \\U0001F4C4', value='',
587                     ↪  interactive=False)
588             with gr.Tab(label='\\U0001F19A Compare'):
589                 compareDropdown = gr.Dropdown(choices=[f'{region["region"]}',

590                     ↪  {region["country"]} for region in trainingData], label='Select a
591                     ↪  Region \\U0001F30D')
592                 compareTable = gr.DataFrame(label='Comparison \\U0001F4CA',
593                     ↪  headers=['Factor', 'Your Region', 'Selected Region'], type='array',
594                     ↪  interactive=False)
595                 pmccImage = gr.Image('data/pmcc.png', label='Correlation between Factors
596                     ↪  \\U0001F4C8', height=600, interactive=False)
597             with gr.Tab(label='\\U0001F30D Regions'):
598                 regionsDropdown = gr.Dropdown(choices=[], label='Current Region
599                     ↪  \\U0001F30D', interactive=True)
600                 regionsTable = gr.DataFrame(label='Your Regions \\U0001F304',
601                     ↪  headers=([['Name']] + list(trainingData[0].keys())[4:]), type='array',
602                     ↪  col_count=(25,'fixed'), interactive=True)

```

```

574     submitEditsButton = gr.Button(value='\\U0001F504 Submit Table Changes
575     ↵ \\U0001F504', variant='primary')
576     gr.Markdown('\\U00002757 **WARNING:** You will no longer be able to make
577     ↵ suggestions from regions that you edit here. \\U00002757')
578     submitEditsResult = gr.Textbox(label='Result \\U0001F4C4',
579     ↵ interactive=False)
580     with gr.Tab(label='\\U0001F513 Log In'):
581         with gr.Row():
582             with gr.Column():
583                 logInUsername = gr.Textbox(label='Username \\U0001F50F',
584                 ↵ placeholder='Enter your username...')
585                 logInPassword = gr.Textbox(label='Password \\U0001F511',
586                 ↵ placeholder='Enter your password...', type='password')
587                 with gr.Column():
588                     logInButton = gr.Button(value='\\U0001F4F2 Log In \\U0001F4F2',
589                     ↵ variant='primary')
590                     gr.Markdown('\\U00002757 **WARNING:** Logging in will override the
591                     ↵ regions in the \\U0001F30D Regions Tab, and replace them with
592                     ↵ the ones in the account you log into. \\U00002757')
593                     logInResult = gr.Textbox(label='Result \\U0001F4C4',
594                     ↵ interactive=False)
595         with gr.Tab(label='\\U0001F4DD Sign Up'):
596             with gr.Row():
597                 with gr.Column():
598                     signUpUsername = gr.Textbox(label='Username \\U0001F50F',
599                     ↵ placeholder='Enter your username...')
600                     signUpPassword = gr.Textbox(label='Password \\U0001F511',
601                     ↵ placeholder='Enter your password...', type='password')
602                     signUpPassword2 = gr.Textbox(label='Re-Enter Password
603                     ↵ \\U0001F510', placeholder='Re-Enter your password...', type='password')
604                     gr.Markdown('''
605                     Your password must:
606                     - Be at least 8 characters
607                     - Contain at least 1 digit
608                     - Contain at least 1 special character
609                     ''')
610                     with gr.Column():
611                         signUpButton = gr.Button(value='\\U0001F680 Create an Account
612                         ↵ \\U0001F680', variant='primary')
613                         signUpResult = gr.Textbox(label='Result \\U0001F4C4',
614                         ↵ interactive=False)
615
616 # functionality
617 uploadButton.upload(uploadGeoJSON, inputs=[uploadButton, max_x_objectsSlider,
618 ↵ max_y_objectsSlider, regionsDropdown], outputs=[log, regionsTable,
619 ↵ regionsDropdown, regionsDropdown])

```

```

604 predictHDIbutton.click(processPrediction, inputs=[], outputs=[HDIprediction,
   ↵ similarHDI])
605 makeSuggestionsButton.click(makeSuggestions, inputs=[max_x_objectsSlider,
   ↵ max_y_objectsSlider, newBuildingsSlider], outputs=[suggestionsTable])
606 suggestionsTable.select(selectSuggestionsTable, inputs=[], outputs=[map])
607 compareDropdown.input(compareRegions, inputs=[compareDropdown],
   ↵ outputs=[compareTable])
608 regionsDropdown.change(updateCurrentRegion, inputs=[regionsDropdown],
   ↵ outputs[])
609 submitEditsButton.click(updateAllUserRegions, inputs=[regionsTable,
   ↵ regionsDropdown], outputs=[submitEditsResult, regionsDropdown,
   ↵ regionsDropdown])
610 signUpButton.click(signUp, inputs=[signUpUsername, signUpPassword,
   ↵ signUpPassword2], outputs=[signUpResult, regionsTable, regionsDropdown])
611 logInButton.click(logIn, inputs=[logInUsername, logInPassword],
   ↵ outputs=[logInResult, regionsTable, regionsDropdown])
612
613 # launch the UI
614 app.launch(allowed_paths=["/"])

```

Code Snippet A.13: Final Code for app.py

A.3.2 calc_factors.py

(This is the code for the up-to-date version of calc_factors.py, which is used by the app)

```

1 import numpy as np
2 import random
3
4 EARTH_RADIUS = 6371 # (in km)
5
6 # converts angle in degrees to angle in radians
7 def deg2rad(angle):
8     return (np.pi/180)*angle
9
10 # finds the distance (in km) along the surface of the earth, between 2
11 # coordinates
12 def distBetween2Points(p1, p2):
13     La1 = deg2rad(p1[0])
14     Lo1 = deg2rad(p1[1])
15     La2 = deg2rad(p2[0])
16     Lo2 = deg2rad(p2[1])
17     return EARTH_RADIUS * np.arccos(np.sin(La1)*np.sin(La2) +
18         np.cos(La1)*np.cos(La2)*np.cos(Lo1-Lo2))
19
20 # finds the average distance between 2 types of objects, A(x,y)
21 def averageDistance(x_objects, y_objects, max_x_objects=2000, max_y_objects=2000,
   ↵ must_include_x=None, must_include_y=None):

```

```

20     if len(x_objects) == 0 or len(y_objects) == 0: # the average distance is
21         ↵ undefined
22         return None
23     if len(x_objects) > max_x_objects: # random sample
24         x_objects = random.sample(x_objects, max_x_objects)
25     if must_include_x != None: # include the new building when making suggestions
26         x_objects += must_include_x
27     # iterate over each x_object, and find the closest y_object
28     min_dists = []
29     for x_object in x_objects:
30         dist_to_ys = []
31         if len(y_objects) > max_y_objects:
32             y_objects_in_consideration = random.sample(y_objects, max_y_objects)
33         else:
34             y_objects_in_consideration = y_objects
35         if must_include_y != None: # include the new building when making
36             ↵ suggestions
37             y_objects_in_consideration += must_include_y
38         for y_object in y_objects_in_consideration:
39             dist_to_ys.append(distBetween2Points(x_object, y_object))
40         min_dists.append(min(dist_to_ys)) # the closest y_object
41     return np.mean(min_dists)

42 # calculates the area (in km^2) of a given region bound by a set of
43 # → latitude/longitude coordinates
44 def calcArea(bounding_coords):
45     if bounding_coords[0] == bounding_coords[-1]: # if the first and last
46         ↵ coordinates are the same, then delete the last one
47         bounding_coords.pop(-1)
48     # convert lat/lon pairs to 3d position vectors
49     bounding_vectors = []
50     for coord in bounding_coords:
51         La = deg2rad(coord[0])
52         Lo = deg2rad(coord[1])
53         bounding_vectors.append(np.matrix([
54             [np.cos(La)*np.cos(Lo)],
55             [np.cos(La)*np.sin(Lo)],
56             [np.sin(La)]
57         ]))
58     # iterate over each vertex and find its anticlockwise angle
59     anticlockwise_angles = []
60     for vertex_index, vertex in enumerate(bounding_vectors):
61         prev_vertex = bounding_vectors[vertex_index-1]
62         next_vertex = bounding_vectors[0] if vertex_index ==
63             ↵ len(bounding_vectors)-1 else vertex_index+1
64         La = deg2rad(bounding_coords[vertex_index][0])
65         Lo = deg2rad(bounding_coords[vertex_index][1])

```

```

62     # define the rotation and translation matrices
63     rotation_matrix_z = np.matrix([
64         [np.cos(-Lo), -np.sin(-Lo), 0],
65         [np.sin(-Lo), np.cos(-Lo), 0],
66         [0, 0, 1]
67     ])
68     rotation_matrix_y = np.matrix([
69         [np.cos(La-np.pi/2), 0, np.sin(La-np.pi/2)],
70         [0, 1, 0],
71         [-np.sin(La-np.pi/2), 0, np.cos(La-np.pi/2)]
72     ])
73     translation_vector = np.matrix([
74         [0],
75         [0],
76         [-1]
77     ])
78     # rotate and translate the plane tangent to the vertex such that it is
79     # the xy-plane
80     prev_vertex = np.matmul(rotation_matrix_y, np.matmul(rotation_matrix_z,
81     # prev_vertex)) + translation_vector
82     vertex = np.matmul(rotation_matrix_y, np.matmul(rotation_matrix_z,
83     # vertex)) + translation_vector
84     next_vertex = np.matmul(rotation_matrix_y, np.matmul(rotation_matrix_z,
85     # next_vertex)) + translation_vector
86     # project onto the xy-plane
87     prev_vertex[2][0] = 0
88     next_vertex[2][0] = 0
89     # calculate anticlockwise angle between the vectors
90     prev_to_current = vertex - prev_vertex
91     current_to_next = next_vertex - vertex
92     anticlockwise_angle =
93         np.arctan2((prev_to_current.item(1,0)*current_to_next.item(2,0) -
94         prev_to_current.item(2,0)*current_to_next.item(1,0)) -
95         (prev_to_current.item(2,0)*current_to_next.item(0,0) -
96         prev_to_current.item(0,0)*current_to_next.item(2,0)) +
97         (prev_to_current.item(0,0)*current_to_next.item(1,0) -
98         prev_to_current.item(1,0)*current_to_next.item(0,0)),
99         prev_to_current.item(0,0)*current_to_next.item(0,0) +
100         prev_to_current.item(1,0)*current_to_next.item(0,0) +
101         prev_to_current.item(0,0)*current_to_next.item(1,0) +
102         prev_to_current.item(2,0)*current_to_next.item(2,0))
103     anticlockwise_angles.append(anticlockwise_angle)
104     # find the interior angles from the anticlockwise angles
105     sum_anticlockwise_angles = round(sum(anticlockwise_angles), 5)
106     if sum_anticlockwise_angles == 0:
107         return None
108     if sum_anticlockwise_angles < 0:
109         anticlockwise_angles = [-1*angle for angle in anticlockwise_angles]

```

```

96     interior_angles = [np.pi-angle for angle in anticlockwise_angles]
97     # return the area
98     num_edges = len(bounding_coords)
99     return (EARTH_RADIUS ** 2) * (sum(interior_angles) - np.pi*(num_edges-2))
100
101 # finds the density of a type of object, D(x)
102 def density(x_objects, area):
103     return len(x_objects)/area

```

Code Snippet A.14: Final Code for calc_factors.py

A.3.3 neural_network.py

```

1 import numpy as np
2 import copy
3 import pickle
4 import random
5
6 class MultilayerPerceptron(object):
7     # constructor method
8     def __init__(self, layer_sizes):
9         # initialise layer sizes constant
10        self.layer_sizes = layer_sizes
11        self.L = len(layer_sizes)-1
12        # initialise matrices for the different quantities
13        self.activations = [np.matrix(np.zeros(shape=(layer_size,1))) for
14            layer_size in layer_sizes]
15        self.weights =
16            [np.matrix(np.random.randn(layer_sizes[index+1],layer_size)) for
17            index, layer_size in enumerate(layer_sizes[:-1])]
18        self.biases = [np.matrix(np.random.randn(layer_size,1)) for layer_size in
19            layer_sizes]
20        self.z = [np.matrix(np.zeros(shape=(layer_size,1))) for layer_size in
21            layer_sizes]
22        self.errors = [np.matrix(np.zeros(shape=(layer_size,1))) for layer_size in
23            in layer_sizes]
24        # initialise lists for training states
25        self.training_activations = []
26        self.training_z = []
27        self.training_errors = []
28
29        # carry out the feedforward algorithm
30        def feedforward(self):
31            for layer in range(0, self.L):
32                self.z[layer+1] = np.matmul(self.weights[layer],
33                    self.activations[layer]) + self.biases[layer+1]
34                self.activations[layer+1] = sigmoid(self.z[layer+1])

```

```

28
29     # carry out the backpropogation algorithm
30     def backpropogate(self, examples, learning_rate):
31         # reset the training lists
32         self.training_activations = []
33         self.training_z = []
34         self.training_errors = []
35         # iterate over each example
36         for example in examples:
37             # reset the activations, z-values and errors
38             self.activations = [np.matrix(np.zeros(shape=(layer_size,1))) for
39                                 ↳ layer_size in self.layer_sizes]
39             self.z = [np.matrix(np.zeros(shape=(layer_size,1))) for layer_size in
40                                 ↳ self.layer_sizes]
40             self.errors = [np.matrix(np.zeros(shape=(layer_size,1))) for
41                                 ↳ layer_size in self.layer_sizes]
41             # make a feedforward pass
42             self.activations[0] = example['input']
43             self.feedforward()
44             # calculate the error in the output layer
45             self.errors[self.L] = np.multiply(
46                 self.activations[self.L] - example['output'],
47                 sigmoid_prime(self.z[self.L]))
48             )
49             # backpropogate the error to previous layers
50             for layer in range(self.L, 1, -1):
51                 self.errors[layer-1] = np.multiply(
52                     np.matmul(
53                         self.weights[layer-1].T,
54                         self.errors[layer]
55                     ),
56                     sigmoid_prime(self.z[layer-1])
57                 )
58             # save the activations, z-values and errors
59             self.training_activations.append(copy.deepcopy(self.activations))
60             self.training_z.append(copy.deepcopy(self.z))
61             self.training_errors.append(copy.deepcopy(self.errors))
62             # use gradient descent to update the weights and biases
63             for layer in range(self.L, 0, -1):
64                 self.weights[layer-1] -= (learning_rate/len(examples)) * sum(
65                     [np.matmul(
66                         self.training_errors[example][layer],
67                         self.training_activations[example][layer-1].T
68                     ) for example in range(len(examples))],
69                     np.matrix(np.zeros(self.weights[layer-1].shape)) # (extra
70                                 ↳ argument in sum function to add matrices instead of numbers)
70                 )

```

```

71     self.biases[layer] -= (learning_rate/len(examples)) * sum(
72         [self.training_errors[example][layer] for example in
73          range(len(examples))],
74         np.matrix(np.zeros(self.biases[layer].shape)) # (extra argument
75          → in sum function to add matrices instead of numbers)
76     )
77
78     # run the feedforward algorithm on a set of input data and return the result
79     def predict(self, input_data):
80         self.activations[0] = input_data
81         self.feedforward()
82         return self.activations[self.L]
83
84     # carry out stochastic gradient descent over a number of epochs to train a
85     # → model
86     def train(self, examples, mini_batch_size, num_epochs, learning_rate):
87         for epoch in range(1, num_epochs+1):
88             print(f'training epoch {epoch}/{num_epochs}...') # log message
89             # randomly split into mini batches
90             random.shuffle(examples)
91             mini_batches = [examples[(batch_number *
92               → mini_batch_size):((batch_number+1) * mini_batch_size)] for
93               → batch_number in range(int(np.ceil(len(examples) /
94               → mini_batch_size)))]
95             # backpropogate for each mini batch
96             for num, mini_batch in enumerate(mini_batches):
97                 print(f' mini batch {num}/{len(mini_batches)}...') # log message
98                 self.backpropogate(mini_batch, learning_rate)
99
100            # saves weights and biases to an external file in the models folder
101            def save_model(self, filename):
102                parameters = {
103                    'weights': self.weights,
104                    'biases': self.biases,
105                    'layer_sizes': self.layer_sizes
106                }
107                with open(f'models/{filename}.pkl', 'wb') as file:
108                    pickle.dump(parameters, file)
109
110            # loads a model from a file path
111            def load_model(self, file_path):
112                with open(file_path, 'rb') as file:
113                    parameters = pickle.load(file)
114                    if self.layer_sizes != parameters['layer_sizes']:
115                        raise Exception('layer sizes do not match!')
116                    self.weights = parameters['weights']
117                    self.biases = parameters['biases']

```

```
112  
113 # activation function  
114 def sigmoid(x):  
115     return 1/(1+np.exp(-x))  
116  
117 # derivative of the activation function  
118 def sigmoid_prime(x):  
119     return np.multiply(sigmoid(x), 1-sigmoid(x))
```

Code Snippet A.15: Final Code for `neural_network.py`

List of Figures

1.1	Jupyter Notebook Format (Not Suitable)	10
1.2	HDI Scatter Plot By Country	10
1.3	HDI Factor Heatmap	11
1.4	HDI Factor Pie Chart Heatmap	12
1.5	Social, Economic & Environmental Factors Across Years (Not Suitable)	12
1.6	Random Forest Classification (Not Suitable)	13
1.7	HDI Ranking Chart	13
1.8	Hovering Over HDI Ranking Chart	14
1.9	HDI Over Time Waterfall Chart (Not Suitable)	14
1.10	Score for each end-feature	15
1.11	Score for each density factor	17
1.12	Score for each distance factor	18
1.13	Map of all shelters from OpenStreetMaps	19
2.1	Structure Diagram	23
2.2	Abstracted Model of the Earth	25
2.3	P_1 and P_2	26
2.4	Line segment AB	26
2.5	Line segment P_1P_2	27
2.6	Finding b	28
2.7	Angle θ between P_1 and P_2	28
2.8	Arc between P_1 and P_2	29
2.9	$A(x, y)$ Flowchart	31
2.10	Cutting into Triangles	32
2.11	Triangle Edges Extended	33
2.12	Area of a Spherical Lune	33
2.13	Considering Vectors	35
2.14	Projecting the Other Points onto the xy -plane	36
2.15	$D(x)$ Flowchart	38
2.16	Overall Data Collection & Processing Algorithm Flowchart	40
2.17	Neural Network Structure	44
2.18	Neural Network Weights	45
2.19	Calculating $a_1^{(1)}$ Example	46
2.20	Graph of $y = \sigma(x)$	47
2.21	Graph of $y = \sigma(x + 2)$	48

2.22 Calculating an entire layer of neurons	49
2.23 Feedforward Algorithm Flowchart	51
2.24 Gradient Descent Example	52
2.25 $z_i^{(l-1)}$ Affects all neurons in layer l , which all affect C	55
2.26 Backpropogation Algorithm Flowchart	58
2.27 Shuffling into Mini Batches	59
2.28 Full Training Algorithm Flowchart	60
2.29 Multilayer Perceptron Class Diagram	61
2.30 Technical Interface Sketch	62
2.31 Interface Style Examples	64
2.32 Finding an Optimal Place for New Building Flowchart	67
2.33 Overall User Predicting HDI Algorithm Flowchart	69
 3.1 Full dataset containing unnecessary data	77
3.2 The new records	78
3.3 Dataset of only useful HDI information	79
3.4 Output of Example Overpass Turbo Query	82
3.5 Output of a <code>list</code> of Latitude/Longitude pairs	83
3.6 $A(x, y)$ 20 times, with <code>max_x_objects</code> set to 500	87
3.7 $A(x, y)$ 20 times, with <code>max_x_objects</code> set to 2000	88
3.8 Area of Oxford, with an Invalid Self-Intersection	95
3.9 Area of Oxford, with a Valid Self-Intersection	95
3.10 $D(\text{School})$ for Oxford	96
3.11 Log messages for Collecting data for all objects	98
3.12 Example all factors <code>dict</code> for Oxford	101
3.13 Each record of the shapefile	103
3.14 Sample of the resulting <code>.json</code> file	103
3.15 HTTP 414 Error	108
3.16 Overpass Turbo's POST request	109
3.17 All HTTP Responses now 200: <code>OK</code>	110
3.18 <code>list[list[list[coordinate]]]</code> debug print statement	111
3.19 Longitude Error Message	113
3.20 Final Training Data	115
3.21 Graph of the Standard Normal Distribution	119
3.22 Printing activations, weights and biases	120
3.23 Backpropogation Error	130
3.24 Examples from the MNIST Data Set	132
3.25 <code>ModuleNotFoundError</code>	135
3.26 Output for the first 7 training examples	139
3.27 Output for the last 7 training examples	139
3.28 Improved Output for the last 7 training examples	139
3.29 Some HDI Predictions	145
3.30 Improved HDI Predictions	147
3.31 Launching the Gradio App	150
3.32 Empty Gradio UI	150

3.33 Map Added to GUI	152
3.34 Drawing a Region	153
3.35 Map Starting at a Random City	154
3.36 Upload Button Added	156
3.37 Log Textbox Added	157
3.38 Uploading GeoJSON Files	159
3.39 New Structure of the GUI	161
3.40 Progress Bar	166
3.41 Successful Prediction from a Region	166
3.42 <code>max_x_objects</code> and <code>max_y_objects</code> Sliders	170
3.43 Prediction with lower <code>max_x_objects</code> and <code>max_y_objects</code> values	171
3.44 Suggestions Table	172
3.45 Invalid Table Format	179
3.46 Suggestions Table Successfully Generated	180
3.47 Error Message Shown in Table	180
3.48 Pin Shown on Map for Suggestion	182
3.49 Output for when we set <code>share=True</code>	184
 4.1 Structure Diagram (2 nd prototype)	186
4.2 Hashing Examples	187
4.3 Salting Examples	188
4.4 Creating an Account Flowchart	189
4.5 Logging In Flowchart	190
4.6 Technical Interface Sketch (2 nd prototype, main page)	193
4.7 Technical Interface Sketch (2 nd prototype, compare page)	194
4.8 Technical Interface Sketch (2 nd prototype, regions page)	195
4.9 Technical Interface Sketch (2 nd prototype, log in page)	196
4.10 Technical Interface Sketch (2 nd prototype, sign up page)	197
4.11 Technical Interface Sketch (2 nd prototype, help page)	198
4.12 PMCC Examples	201
4.13 Analysis of how much Each Factor Affects the Others Flowchart	202
4.14 Finding Similar Regions Flowchart	203
4.15 Finding a Better Optimal Place for New Building Flowchart	205
 5.1 Predict Tab	219
5.2 Log In & Sign Up Tabs	219
5.3 Placeholder Markdown	220
5.4 Complete Sign Up Tab	221
5.5 Censorship of the user's password	222
5.6 Complete Log In Tab	223
5.7 Creating the MongoDB	224
5.8 Adding a Database User (me)	224
5.9 Creating a Database & Collection (table)	225
5.10 SSL Certificate Error	226
5.11 Successful Connection to MongoDB	226

5.12 Creating a Temporary, Fake Account	228
5.13 Validating Username Input	229
5.14 Validating Password Input	232
5.15 Hashed Password	234
5.16 New Account in MongoDB	235
5.17 Validating Login	238
5.18 Compare Tab Interface	242
5.19 Example Options for Dropdown	243
5.20 All Dropdown Options	245
5.21 Output when the User has not Predicted Yet	247
5.22 Full Comparison between User's and Selected Regions	250
5.23 Initial Image of the Correlation Heatmap	254
5.24 Final Image of the Correlation Heatmap	255
5.25 Correlation Heatmap in Gradio	256
5.26 Similar Region Returned to Textbox	259
5.27 Comparison to Bangkok	259
5.28 Regions Tab	260
5.29 Progress Bar in the Log Messages	263
5.30 Region added to the Table	265
5.31 Updating the Dropdown Options	266
5.32 Dropdown Remains Interactive	267
5.33 Dropdown Value Updated	269
5.34 Submit Changes Button in Regions Tab	270
5.35 Validating Regions Table Input	273
5.36 <code>allUserRegions</code> & Dropdown Updated from Table	275
5.37 Cannot Upload Region with the Same Name	278
5.38 Region Added to MongoDB	280
5.39 Region with Objects Added to MongoDB	281
5.40 Predict Tab with New Slider	282
5.41 No Associated Buildings Error Message	287
5.42 Output for a Region with Many Buildings	290
5.43 Output for a Region with No Buildings	290
5.44 Output for Placing Pins on the Map	292
5.45 Screenshot of an Errored Output	293
5.46 Screenshot of the Corrected Output (<code>def makeSuggestions</code>)	294
5.47 Screenshot of the Corrected Output (<code>def logIn</code>)	296
5.48 HDI is not updating when Comparing	297
5.49 HDI is now updating when Comparing	298
5.50 Info Popup Examples	299
5.51 Emojis in Tabs	305
5.52 Emojis in the Predict Tab	307
5.53 Emojis in the Compare Tab	308
5.54 Emojis in the Regions Tab	309
5.55 Emojis in the Log In Tab	310
5.56 Emojis in the Sign Up Tab	311

5.57 Restructuring the Log In & Sign Up Tabs	313
5.58 Adding Warnings	315
5.59 Markdown Test Initial Attempt	316
5.60 Markdown Test Successul	318
5.61 Help Tab	318
6.1 Scores for Usability Tests	327
6.2 Comparison to Interface Sketch, predict tab (Figure 4.6)	333
6.3 Comparison to Interface Sketch, compare tab (Figure 4.7)	335
6.4 Comparison to Interface Sketch, regions tab (Figure 4.8)	336
6.5 Comparison to Interface Sketch, log in tab (Figure 4.9)	338
6.6 Comparison to Interface Sketch, sign up tab (Figure 4.10)	339
6.7 Comparison to Interface Sketch, help tab (Figure 4.11)	341

List of Code Snippets

3.1 Initialising csv reader	77
3.2 Refining the HDI Data	78
3.3 Writing back to csv	79
3.4 Overpass Turbo GET Request	80
3.5 Overpass Turbo Query	81
3.6 Getting a List of Latitude/Longitude Pairs	83
3.7 Finding the Distance between any 2 points	84
3.8 $A(x, y)$ Validation	85
3.9 Calculating $A(x, y)$	86
3.10 Converting list of bounding coordinates into Column Vectors	89
3.11 Coverting from Degrees to Radians	90
3.12 Defining Key Variables in calculating area	91
3.13 Rotating the plane tangent to each vertex	92
3.14 Finding the anticlockwise angles	93
3.15 Calculating the Area	94
3.16 Rounding the sum of anticlockwise angles	95
3.17 Calculating $D(x)$	96
3.18 Retrieving All OSM Data	97
3.19 Calculating All Average Distance Factors	99
3.20 Calculating All Density Factors	100
3.21 Processing the Shapefile	102
3.22 Reading the Region & HDI files	104
3.23 Iterating over each region	105
3.24 Finding the Main Area	106
3.25 Writing training data to csv	107
3.26 Using a POST request instead of a GET request	109
3.27 Correctly finding the Main Area	112
3.28 Specifying the most recent region	113
3.29 Finding the right Longitude	114
3.30 Truncating Detailed Regions	114
3.31 Defining the Constructor Method	118
3.32 Initialising the Matrices	118
3.33 Initialising the Training Lists	120
3.34 Defining the Sigmoid Function	121

3.35 Defining the derivative of the Sigmoid Function	122
3.36 Saving weights and biases	123
3.37 Loading Models	123
3.38 Ensuring Layer Sizes are the Same	124
3.39 Implementing the Feedforward Algorithm	125
3.40 Making a Prediction	125
3.41 Backpropogation Algorithm Structure	126
3.42 Resetting the Lists	127
3.43 Calculating the error in the output layer	127
3.44 Backpropogating the Error	128
3.45 Updating the weights and biases	129
3.46 Fixing the σ' function	130
3.47 Implementing Stochastic Gradient Descent	131
3.48 Loading in the MNIST data set	132
3.49 Converting the MNIST data to the right format	133
3.50 Saving the MNIST data to a file	134
3.51 Loading the MNIST Training Data	135
3.52 Importing from a Parent Directory	135
3.53 Training a Neural Network on the MNIST Data Set	136
3.54 Adding another log message	136
3.55 Loading the MNIST Testing Data	137
3.56 Testing the Neural Network on the MNIST Data Set	138
3.57 Improving the Training Parameters	139
3.58 Loading the HDI Data	140
3.59 Splitting into Testing and Training Data	141
3.60 Training the Neural Network on HDI	142
3.61 Preparing to Test on the HDI	143
3.62 Testing the Neural Network on HDI	144
3.63 Dividing some factors by 10	145
3.64 Adjusting the Training Parameters	146
3.65 Initial Gradio Framework	149
3.66 Initialising the Map	151
3.67 Adding the Map to the GUI	151
3.68 Setting the Initial Location to a Random City	154
3.69 Reading the Map for the Region	155
3.70 Adding a Log Textbox	156
3.71 Adding Functionality to the Upload Button	158
3.72 Placing the Predict Button	160
3.73 Processing the Prediction	161
3.74 Retrieving All Relevant OSM Data	162
3.75 Calculating $A(x, y)$ Factors	163
3.76 Calculating $D(x)$ Factors	164
3.77 Predicting the HDI	165
3.78 Adding Functionality to the Prediction Button	165
3.79 Implementing a Maximum for y -objects	167

3.80 Passing in the <code>max_y_objects</code>	168
3.81 Adding Sliders	169
3.82 Updating Functionality for Predicting HDI Button	170
3.83 Adding a Suggestions Table	172
3.84 Defining the Suggestions Function	173
3.85 Finding the mean of coordinates	174
3.86 Finding the Optimal Place for a New Building	174
3.87 Updating the Average Distance Function	175
3.88 Calculating the New Average Distances	176
3.89 Calculating the New Densities	177
3.90 Predicting the New HDI	177
3.91 Storing the Current HDI Globally & Returning a Table	178
3.92 Adding Functionality to the Suggestions Button	179
3.93 Creating a Readable Format for the Coordinates	179
3.94 Validating the <code>def makeSuggestions</code> function	180
3.95 Placing a Pin on the Map	181
3.96 Adding Functionality to the Suggestions Table	182
 5.1 Embedding the current UI into a Tab	218
5.2 Adding the Log In & Sign Up Tabs	219
5.3 Adding the Content of the Sign Up Tab	220
5.4 Adding the Content of the Log In Tab	223
5.5 Connecting to the MongoDB	225
5.6 Fixing the SSL Certificate Error	226
5.7 Adding the <code>def signUp</code> function	227
5.8 Ensuring no Empty Fields	227
5.9 Validating the Username	228
5.10 Validating the Password – Double Entry Validation	230
5.11 Validating the Password – Length	230
5.12 Validating the Password – Digits	231
5.13 Validating the Password – Special Characters	231
5.14 Salting & Hashing the Password	234
5.15 Adding the User to MongoDB	235
5.16 Adding the <code>def logIn</code> function	236
5.17 Ensuring No Empty Fields in Logging In	236
5.18 Validating the Username’s Existence	237
5.19 Checking the Password is Correct	237
5.20 Granting Access to the Account	237
5.21 Adding the Compare Tab	242
5.22 Importing the Training Data	244
5.23 Adding the Regions to the Dropdown	244
5.24 Making <code>allFactors</code> a <code>global</code> variable	246
5.25 Adding the Functionality to the Compare Tab	246
5.26 Undoing the Concatenation	247
5.27 Getting Lists of Factors	248

5.28 Returning the 2D List of Factors	249
5.29 Reading the Training Data	250
5.30 Iterating over All the Factors Twice	251
5.31 Ensuring Complete Pairs	252
5.32 Calculating the PMCC	253
5.33 Generating & Exporting the Figure	254
5.34 Improving the Quality of the Figure	255
5.35 Adding the Image to the Gradio Interface	256
5.36 Iterating over Different Possible HDIs	257
5.37 Choosing a Region to Display	258
5.38 Returning to the Textbox	258
5.39 Adding the Regions Tab	260
5.40 Changes to Accommodate the Transition	262
5.41 Storing New Regions when they Upload	264
5.42 Creating a 2D List for the Regions Table	264
5.43 Updating the Regions Table	265
5.44 Updating the Regions Dropdown	266
5.45 Ensuring the Dropdown is Interactive	267
5.46 Updating the Value of the Regions Dropdown	268
5.47 Adding a Button to Submit Edits to the Table	269
5.48 Adding the Submit Button Functionality	270
5.49 Saving the Index of the Current Region	271
5.50 Ensuring Unique Region Names	271
5.51 Type Checking Every Other Cell	272
5.52 Updating <code>allUserRegions</code>	274
5.53 Updating the Name of the Selected Region	275
5.54 Updating the Current Region	276
5.55 Ensuring the User doesn't Upload a Duplicate Region Name	277
5.56 Adding a Global Variable for Logging In	278
5.57 Logging Into a Newly Created Account	279
5.58 Adding Regions to the Account	279
5.59 Reading Regions from the Account	280
5.60 Handling Empty Region Lists	280
5.61 Correctly Identifying when Table Rows have Changed	281
5.62 Adding a Slider for Number of New Buildings	282
5.63 Finding All Objects in Consideration	283
5.64 Iterating over each Factor	284
5.65 Calculating the Shortest Distances & Optimal Position	285
5.66 Returning the Optimal Position	286
5.67 New <code>def makeSuggestions</code> function	287
5.68 Finding more than 1 Optimal Position	288
5.69 Returning a Different Suggestion	289
5.70 Updating the <code>must_include</code> functionality in <code>def averageDistance</code>	289
5.71 New Return Format for Suggestions Table	290
5.72 Placing Multiple Pins on the Map	291

5.73 Fixing the Error for <code>def makeSuggestions</code>	293
5.74 Fixing the Error for <code>def signUp</code>	295
5.75 Fixing the Error for <code>def logIn</code>	296
5.76 Adding Current Regions to the Account	297
5.77 Updating <code>currentHDI</code>	298
5.78 Info Popups in <code>def processPrediction</code>	299
5.79 Ensuring <code>def processPrediction</code> works with Logging In	300
5.80 Info Popups in <code>def uploadGeoJSON</code>	301
5.81 Info Popups in <code>def makeSuggestions</code>	301
5.82 Info Popups in <code>def compareRegions</code>	301
5.83 Info Popups in <code>def updateAllUserRegions</code>	302
5.84 Info Popups in <code>def signUp</code>	303
5.85 Info Popups in <code>def logIn</code>	304
5.86 Emojis in the Tabs	305
5.87 Emojis in the Predict Tab	306
5.88 Emojis in the Compare Tab	307
5.89 Emojis in the Regions Tab	308
5.90 Emojis in the Log In Tab	309
5.91 Emojis in the Sign Up Tab	310
5.92 Restructuring the Log In & Sign Up Tabs	312
5.93 Warning Messages	314
5.94 Adding the Help Tab	316
5.95 Allowing for Images in Markdown	317
A.1 Final Code for <code>data_processing / calc_factors.py</code>	349
A.2 Final Code for <code>data_processing / compile_data.py</code>	350
A.3 Final Code for <code>data_processing / get_osm_data.py</code>	351
A.4 Final Code for <code>data_processing / pmcc.py</code>	352
A.5 Final Code for <code>data_processing / process_hdi.py</code>	353
A.6 Final Code for <code>data_processing / process_shapefile.py</code>	353
A.7 Final Code for <code>network_training / load_digits_data.py</code>	354
A.8 Final Code for <code>network_training / load_hdi_data.py</code>	355
A.9 Final Code for <code>network_training / test_digits.py</code>	356
A.10 Final Code for <code>network_training / test_hdi.py</code>	357
A.11 Final Code for <code>network_training / train_digits.py</code>	357
A.12 Final Code for <code>network_training / train_hdi.py</code>	358
A.13 Final Code for <code>app.py</code>	377
A.14 Final Code for <code>calc_factors.py</code>	380
A.15 Final Code for <code>neural_network.py</code>	383

List of Tables

1.1 Stakeholders	9
1.2 Essential Features	19
1.3 Success Criteria	22
2.1 List of Key Variables in $A(x, y)$	41
2.2 List of Key Variables in $D(x)$	42
2.3 List of Other Key Variables in Data Collection & Processing	43
2.4 List of Key Variables in Finding Optimal Place for New Building	70
2.5 List of Other Key Variables in User Interface	70
2.6 Testing Data for Milestone 1	73
2.7 Testing Data for Milestone 2	73
2.8 Testing Data for Milestone 3	74
3.1 Success Criteria for Milestone 1	75
3.2 Testing Data for Milestone 1	76
3.3 Testing Table for Code Snippets 3.1 & 3.2	78
3.4 Testing Table for Code Snippet 3.3	79
3.5 T_1 Testing Table (for Code Snippets 3.4 & 3.5)	82
3.6 Testing Table for Code Snippet 3.6	83
3.7 Testing Table for Code Snippet 3.7	84
3.8 T_2 Testing Table (for Code Snippet 3.7)	85
3.9 T_3 Testing Table (for Code Snippet 3.8)	86
3.10 Testing Table for Code Snippet 3.9	87
3.11 Testing Table for Code Snippet 3.10	89
3.12 Testing Table for Code Snippet 3.11	90
3.13 Testing Table for Code Snippets 3.12 & 3.13	92
3.14 Testing Table for Code Snippet 3.14	93
3.15 T_4 Testing Table (for Code Snippet 3.15)	94
3.16 Testing Table for Code Snippet 3.16	95
3.17 Testing Table for Code Snippet 3.17	96
3.18 Testing Table for Code Snippet 3.18	97
3.19 Testing Table for Code Snippets 3.19 & 3.20	100
3.20 Testing Table for Code Snippet 3.21	103
3.21 Testing Table for Code Snippet 3.22	104
3.22 Testing Table for Code Snippet 3.23	105

3.23 Testing Table for Code Snippet 3.24	106
3.24 Testing Table for Code Snippet 3.25	108
3.25 Testing Table for Code Snippet 3.26	109
3.26 Testing Table for Code Snippet 3.27	112
3.27 Testing Table for Code Snippet 3.28	113
3.28 Testing Table for Code Snippet 3.29	114
3.29 Testing Table for Code Snippet 3.30	114
3.30 Passed Testing Data for Milestone 1	116
3.31 Achieved Success Criteria for Milestone 1	116
3.32 Success Criteria for Milestone 2	117
3.33 Testing Data for Milestone 2	117
3.34 Testing Table for Code Snippets 3.31, 3.32 & 3.33	121
3.35 Testing Table for Code Snippets 3.34 & 3.35	122
3.36 Testing Table for Code Snippets 3.36 & 3.37	124
3.37 Testing Table for Code Snippet 3.38	124
3.38 Testing Table for Code Snippets 3.39 & 3.40	126
3.39 Testing Table for Code Snippets 3.41, 3.42, 3.43, 3.44 & 3.45	130
3.40 Testing Table for Code Snippet 3.46	130
3.41 Testing Table for Code Snippet 3.47	131
3.42 Testing Table for Code Snippets 3.48, 3.49 & 3.50	134
3.43 Testing Table for Code Snippets 3.51 & 3.52	136
3.44 Testing Table for Code Snippets 3.53 & 3.54	137
3.45 Testing Table for Code Snippet 3.55	137
3.46 Testing Table for Code Snippet 3.56	138
3.47 T_5 Testing Table (for Code Snippet 3.57)	140
3.48 Testing Table for Code Snippets 3.58 & 3.59	142
3.49 Testing Table for Code Snippet 3.60	143
3.50 Testing Table for Code Snippets 3.61 & 3.62	144
3.51 Testing Table for Code Snippet 3.63	146
3.52 Testing Table for Code Snippet 3.64	146
3.53 Passed Testing Data for Milestone 2	147
3.54 Achieved Success Criteria for Milestone 2	148
3.55 Success Criteria for Milestone 3	148
3.56 Testing Data for Milestone 3	149
3.57 Testing Table for Code Snippet 3.65	150
3.58 Testing Table for Code Snippets 3.66 & 3.67	153
3.59 Testing Table for Code Snippet 3.68	155
3.60 Testing Table for Code Snippet 3.69	156
3.61 Testing Table for Code Snippet 3.70	157
3.62 Testing Table for Code Snippet 3.71	159
3.63 Testing Table for Code Snippet 3.72	161
3.64 Testing Table for Code Snippets 3.73 & 3.74	163
3.65 Testing Table for Code Snippets 3.75 & 3.76	164
3.66 T_6 Testing Table (for Code Snippets 3.77 & 3.78)	167
3.67 Testing Table for Code Snippet 3.79	168

3.68	Testing Table for Code Snippets 3.80, 3.81 & 3.82	171
3.69	Testing Table for Code Snippet 3.83	173
3.70	Testing Table for Code Snippet 3.84	173
3.71	T_7 Testing Table (for Code Snippets 3.85 & 3.86)	175
3.72	Testing Table for Code Snippets 3.87 & 3.88	176
3.73	Testing Table for Code Snippet 3.89	177
3.74	T_8 Testing Table (for Code Snippet 3.90)	178
3.75	Testing Table for Code Snippets 3.91 & 3.92	179
3.76	Testing Table for Code Snippet 3.93	180
3.77	Testing Table for Code Snippet 3.94	181
3.78	Testing Table for Code Snippets 3.95 & 3.96	182
3.79	Passed Testing Data for Milestone 3	183
3.80	Achieved Success Criteria for Milestone 3	183
4.1	List of Key Variables in Creating an Account	191
4.2	List of Key Variables in Logging In	192
4.3	List of Other Key Variables in Authentication System	192
4.4	List of Key Variables in Calculating PMCC	206
4.5	List of Key Variables in Finding Similar Regions	207
4.6	List of Key Variables in Finding an Optimal Place for a New Building	207
4.7	List of Other Key Variables in User Interface	208
4.8	Testing Data for Milestone 4	209
4.9	Testing Data for Milestone 5	210
4.10	Final Testing Data for Function	212
4.11	Final Testing Data for Robustness	214
4.12	Final Testing Data for Usability	216
5.1	Success Criteria for Milestone 4	217
5.2	Testing Data for Milestone 4	218
5.3	Testing Table for Code Snippets 5.1 & 5.2	220
5.4	Testing Table for Code Snippet 5.3	222
5.5	Testing Table for Code Snippet 5.4	223
5.6	Testing Table for Code Snippet 5.5	226
5.7	Testing Table for Code Snippet 5.6	227
5.8	Testing Table for Code Snippet 5.7	227
5.9	T_9 Testing Table (for Code Snippets 5.8 & 5.9)	230
5.10	T_{10} Testing Table (for Code Snippets 5.10, 5.11, 5.12 & 5.13)	233
5.11	T_{11} Testing Table (for Code Snippet 5.14)	234
5.12	Testing Table for Code Snippet 5.15	235
5.13	Testing Table for Code Snippet 5.16	236
5.14	T_{12} Testing Table (for Code Snippets 5.17, 5.18, 5.19 & 5.20)	238
5.15	Passed Testing Data for Milestone 4	239
5.16	Achieved Success Criteria for Milestone 4	239
5.17	Success Criteria for Milestone 5	240
5.18	Testing Data for Milestone 5	241

5.19	Testing Table for Code Snippet 5.21	243
5.20	Testing Table for Code Snippets 5.22 & 5.23	245
5.21	Testing Table for Code Snippets 5.24 & 5.25	247
5.22	Testing Table for Code Snippets 5.26, 5.27 & 5.28	250
5.23	Testing Table for Code Snippets 5.29 & 5.30	251
5.24	T_{13} Testing Table (for Code Snippet 5.31)	252
5.25	Testing Table for Code Snippet 5.32	253
5.26	Testing Table for Code Snippet 5.33	255
5.27	Testing Table for Code Snippet 5.34	256
5.28	Testing Table for Code Snippet 5.35	257
5.29	T_{14} Testing Table (for Code Snippets 5.36 & 5.37)	258
5.30	Testing Table for Code Snippet 5.38	259
5.31	Testing Table for Code Snippet 5.39	261
5.32	Testing Table for Code Snippet 5.40	263
5.33	Testing Table for Code Snippets 5.41, 5.42 & 5.43	265
5.34	Testing Table for Code Snippet 5.44	267
5.35	Testing Table for Code Snippet 5.45	268
5.36	Testing Table for Code Snippet 5.46	269
5.37	Testing Table for Code Snippet 5.47	270
5.38	T_{15} Testing Table (for Code Snippets 5.48, 5.49, 5.50 & 5.51)	274
5.39	Testing Table for Code Snippets 5.52 & 5.53	276
5.40	Testing Table for Code Snippet 5.54	277
5.41	Testing Table for Code Snippet 5.55	278
5.42	Testing Table for Code Snippets 5.56 & 5.57	279
5.43	Testing Table for Code Snippets 5.58, 5.59 & 5.60	280
5.44	Testing Table for Code Snippet 5.61	281
5.45	Testing Table for Code Snippet 5.62	283
5.46	T_{16} Testing Table (for Code Snippets 5.63, 5.64, 5.65 & 5.66)	286
5.47	Testing Table for Code Snippet 5.67	288
5.48	Testing Table for Code Snippets 5.68, 5.69, 5.70 & 5.71	291
5.49	Testing Table for Code Snippet 5.72	292
5.50	Testing Table for Code Snippet 5.73	294
5.51	Testing Table for Code Snippets 5.74 & 5.75	297
5.52	Testing Table for Code Snippet 5.76	297
5.53	Testing Table for Code Snippet 5.77	298
5.54	Testing Table for Code Snippet 5.78	300
5.55	Testing Table for Code Snippet 5.79	300
5.56	Testing Table for Code Snippets 5.80, 5.81, 5.82, 5.83, 5.84 & 5.85	304
5.57	Testing Table for Code Snippets 5.86, 5.87, 5.88, 5.89, 5.90 & 5.91	311
5.58	Testing Table for Code Snippet 5.92	313
5.59	Testing Table for Code Snippet 5.93	315
5.60	Testing Table for Code Snippet 5.94	317
5.61	Testing Table for Code Snippet 5.95	318
5.62	Passed Testing Data for Milestone 5	319
5.63	Achieved Success Criteria for Milestone 5	320

6.1	Testing for Function to Inform Evaluation	323
6.2	Testing for Robustness to Inform Evaluation	325
6.3	Testing for Usability to Inform Evaluation	328
6.4	Met (✓), Partially Met (Partial ✓) or Unmet (✗) Success Criteria	330

Bibliography

- [1] sandeeppondru. human-development-index. <https://github.com/sandeepponduru/human-development-index>, 2023.
- [2] julieanneco. predictinghdi. <https://github.com/julieanneco/predictingHDI>, 2020.
- [3] Human Development Reports. country-insights. <https://hdr.undp.org/data-center/country-insights#/ranks>, 2022.
- [4] Global Data Lab. Subnational hdi version archive. <https://globaldatalab.org/shdi/archive/>, 2024.
- [5] Global Data Lab. Gdlcode shapefiles. <https://globaldatalab.org/mygdl/downloads/shapefiles/>, 2024.
- [6] Grant Sanderson. Neural networks – chapters 1-4. https://www.youtube.com/playlist?list=PLZHQB0WTQDNU6R1_67000Dx_ZCJB-3pi, 2017.
- [7] Micheal Nielsen. Neural networks and deep learning – chapter 2. <http://neuralnetworksanddeeplearning.com/chap2.html>, 2019.
- [8] alexisdevarennes. Shapefile into geojson conversion python 3. <https://stackoverflow.com/questions/43119040/shapefile-into-geojson-conversion-python-3>, 2020.
- [9] Micheal Nielsen. neural-networks-and-deep-learning. <https://github.com/unexploredtest/neural-networks-and-deep-learning>, 2022.
- [10] GeeksforGeeks. Python - import from parent directory. <https://www.geeksforgeeks.org/python-import-from-parent-directory/>, 2024.
- [11] MaxwellN. Pymongo [ssl: Certificate_verify_failed] certificate verify failed: unable to get local issuer certificate. <https://stackoverflow.com/questions/68123923/pymongo-ssl-certificate-verify-failed-certificate-verify-failed-unable-to-ge>, 2021.
- [12] EmojiDB. Emojidb beta. <https://emojidb.org>, 2024.
- [13] millermuttu. Adding image in markdown not working, gives broken image in the browser. <https://github.com/gradio-app/gradio/issues/1997>, 2022.