

BT YOUNG SCIENTIST AND TECHNOLOGY EXHIBITION 2024

Developing a Generative Adversarial Network to explore Cost of Living hotspots in Ireland.

Addison Carey



Project Reference Number: PR02536

Stand Number: 3110

Project Report Book

Contents

Abstract	3
Introduction	5
Explanations of Components and Terms	6
<i>Cost of Living & How its Calculated</i>	6
<i>Inflation</i>	8
<i>Artificial Intelligence</i>	10
<i>Generative Adversarial Networks</i>	12
My Code / Experimental Methods	17
<i>Iteration 1:</i>	19
<i>Iteration 2:</i>	28
<i>Iteration 3:</i>	35
Results	43
Conclusions	48
Improvements	50
Acknowledgments	51
Appendices – Entire Script	52
References	53
Comments.....	55

Abstract

The cost-of-living crisis in Ireland has been a major issue in recent years. The consequences include increases in house prices, rental prices, fuel costs, health care costs and daily essentials such as food and electricity. The cost of living in Ireland has had a significant impact on its residents and has placed enormous financial pressure on individuals and families.

During the lockdown I started learning about Artificial Intelligence (AI) and Generative Adversarial Networks (GANs). AI refers to the development of computer systems that can perform tasks that normally require human intelligence, such as understanding natural language, recognizing patterns and making decisions. GANs are a type of machine learning model consisting of two components, a generator and a discriminator, which compete in a training process. The generator creates synthetic data, and the discriminator evaluates whether it is real or fake. Over time, GANs offer the advantage of generating highly realistic and diverse data, making them valuable for tasks such as image synthesis, data augmentation, and anomaly detection. They are an approach to generative modeling using deep learning methods, such as convolutional neural networks (Brownlee 2019).

My GAN Project will involve creating a simulation and analysis of the cost of living in Ireland. It creates artificial cost of living data based on various factors. The goal is to develop a GAN that helps individuals, families and government departments visualize the relationships between the integrated factors that affect the cost of living, making it easier for them to manage their finances and make informed decisions. My project can help people find affordable housing and help government policy making. It empowers individuals to budget, and helps governments allocate support and plan for housing and transport, promoting well-being and economic

growth. It is beneficial to use a heatmap to display the data as it is valuable here to show geographic cost of living variations, which helps to quickly identify high and low-cost areas.

In **July 2023**, I started my project with a basic GAN, generating synthetic data based on limited information about factors that influence the cost of living. As progress was made, I improved data visualization using heatmaps and scatterplots. Currently, my project works with a small dataset. I plan to train it on a larger and more comprehensive data set and on different factors that affect the cost of living such as housing or food prices. This will enable the model to better analyse cost of living factors in Ireland and produce a more insightful visualization.

My Hypothesis is that a GAN developed to explore cost of living hotspots in Ireland can benefit individuals, families and government departments to manage finances, make informed decisions and help them to visualize the relationships between the integrated factors that affect the cost of living. The GAN will also be able to generate synthetic cost of living data that is indistinguishable from real life data.

Introduction

My interest in Computer Science and Coding goes back to early Primary school. I started using Scratch and Micro:Bit but soon transitioned onto HTML/CSS and JS. Python was my introduction to the power of computer code to manipulate data.

The Cost of Living Crisis in Ireland poses a significant challenge for numerous households, marked by inflation reaching unprecedented levels. Prices for various goods and services, such as food, energy, housing, and transportation, have surged, placing a considerable burden on budgets, especially for low-income families. There are several factors that have contributed to the Cost of Living Crisis including the Covid-19 Pandemic and the conflict in Ukraine. The Irish cost of living crisis is making it hard for people to meet basic needs and cut essential spending. Businesses are facing higher costs, and consumers are spending less, impacting the economy.

My project involves the creation of a GAN, a type of machine learning, to understand how various factors impact the cost of living in Ireland. GANs have two parts: a generator that makes data similar to real examples and a discriminator that tells if it's real or generated. They compete and improve each other, creating more realistic and high-quality data over time. A simple analogy for this could be to think of it as a contest between the two. The generator creates things, such as a painting mimicking a famous artist, while the discriminator acts as a critic or judge. The discriminator's job is to look at what the generator creates and tries to spot the real from fake. In this back-and-forth process, the generator improves, making things that are increasingly hard for the detective to identify as fakes. The competitive cycle pushes both the generator and the discriminator to enhance their skills continuously.

Explanations of Components and Terms

In this section of my report book, you will find explanations of each primary component of the Report Book. These Primary Components include Cost of Living, Inflation, Artificial Intelligence and Generative Adversarial Networks.

Cost of Living

Cost of living refers to the amount of money required to cover basic expenses such as housing, food, transportation, healthcare, and other essential goods and services in a particular location during a specified period. It is a crucial economic indicator and varies significantly between regions and countries. Several factors contribute to the cost of living in a specific area, including housing prices, utility costs, transportation expenses, healthcare costs, and the price of goods and services. The cost of living is often used to compare different locations or assess how changes in economic conditions, inflation, and other factors impact the affordability of life in a particular area. It plays a crucial role in personal financial planning, business operations, and government policy-making.

The Cost of Living crisis in Ireland is a significant issue that has been affecting the country for several years. Inflation has been particularly high in Ireland, reaching 9.2% in October 2022. The combined effects of pandemic supply chain disruptions, a rapid rebound in demand as economies re-opened after the pandemic, and war-related energy and food price shocks have all contributed to very high rates of inflation in Ireland (Lydon 2022). This has led to an increase in the cost of everyday essentials, such as groceries, housing, and energy. Lower income, older and rural households experienced relatively larger cost of living increases from higher inflation in recent months (Lydon 2022). The cost of living crisis has had a number of negative impacts on the country. It has made it more difficult for people to make ends meet, and has led to an increase in poverty and homelessness. It has also put a

strain on businesses, as they have to deal with rising costs for goods and services. In response to the crisis, the Irish government has implemented measures such as energy credits, social welfare increases, rent freezes, and affordable housing investments. However, these interventions have not fully mitigated the impact of inflation.

As we look towards the future, understanding the potential trajectory of living expenses is essential for planning our finances and making informed decisions. Several factors are expected to shape the cost of living in the years to come. Inflation is a primary driver of living expenses. If inflation continues to rise, it will lead to higher costs for consumers across various categories, including food, energy, housing, and transportation. Energy prices, play a significant role in determining the overall cost of living. If energy prices remain elevated, it will translate into higher costs for transportation, heating, and other energy-intensive activities, putting further strain on household budgets. Other factors, like housing prices and wage growth will also affect the cost of living in years to come.

Overall, the cost of living is a vital economic indicator affecting individuals and nations. In Ireland, challenges such as inflation and housing shortages contribute to the cost of living crisis. This crisis can lead to financial strain on households and broader economic impacts, prompting the need for government intervention and policy adjustments. Effectively addressing these challenges is essential for ensuring economic stability and the well-being of the population.

How it's Calculated

Many people confuse the Consumer price index (CPI) for the Cost-of-living index (COLI). The CPI is used to determine “*the change in the average level of prices (inclusive of all indirect taxes) paid for consumer goods and services by all private and institutional households in the country and by foreign tourists holidaying in Ireland*” (CSO 2024). One problem with using this as a barometer for a population is that it does not localise the experience of price increase or decrease on individuals. COLI allows for this finer granularization of data. (Disch 2023).

My initial algorithm for calculation the COLI was to get the average of the 5 factors.

$$\sum \frac{Factors}{n}$$

I quickly realized that this would not allow for a weighting to be applied. Applying a weighting would allow for different COLI factors to be applied based on location. This would allow for enhanced granularization of the data, which is currently missing from CSO and other available databases.

To create an algorithm that would accommodate the neural network central to my GAN, I first looked at how the individual components would interact.

$$\sum_n weight_n input_n + bias$$

```
costOL = np.sum(factors*weights axis=1) + np.random.normal(0, 0.1, len(factors))
```


I have tried to show an example below using real data to visualize how the algorithm is applied:

rent =1.4

education=2.4

food=3.2

X= [1.4, 2.4, 3.2]

Weights could be attached at each connect within a neural node:

$$W = \begin{bmatrix} 0.5 & 0.3 \\ 1.2 & 1.5 \\ -0.8 & 0.25 \end{bmatrix}$$

Biases attached to 2 nodes might be $bias = [0.5, -0.7]$

An activation function (α) applied to each node would be dependent of the modules, (TensorFlow) that one is using.

An internodal calculation would look like:

$$\alpha_1 [(0.5 \times 1.4) + (2.4 \times 1.2) + (3.2 \times -0.8) + 0.5]$$

a subsequent internodal calculation would look like:

$$\alpha_2 [(0.5 \times 1.4) + (2.4 \times 1.2) + (3.2 \times -0.8) + 0.5]$$

In this example the weights (W), biases, and activation function (α) work together transform the input vector (X) through the network layers and nodes to produce the final output, which is the COLI.

Inflation

Inflation is the rate at which the general level of prices for goods and services rises, leading to a decrease in purchasing power. When inflation occurs, each unit of currency buys fewer goods and services than it did before. Central banks and governments often aim to maintain a moderate level of inflation to encourage spending and investment while preventing deflation, where prices decrease over time. Inflation can be caused by various factors, including increased demand, production costs, and monetary policies. It is commonly expressed as an annual percentage, indicating the average price increase over a specific period.

Inflation can have several negative effects on individuals, businesses, and the overall economy, including:

- **Reduced Purchasing Power:** Inflation erodes the purchasing power of money, meaning that the same amount of money buys fewer goods and services over time. This can lead to a decline in the real standard of living for individuals and households.
- **Uncertainty and Planning Challenges:** High or unpredictable inflation rates can create uncertainty in the economy. Businesses may find it challenging to plan for the future, set prices, and make long-term investment decisions, which can hinder economic growth.
- **Interest Rate Pressures:** Central banks may raise interest rates to combat inflation. Higher interest rates can increase the cost of borrowing for businesses and individuals, potentially slowing down economic activity and impacting investments and consumer spending.
- **Distorted Price Signals:** Inflation can distort price signals in the market. Prices are essential for conveying information about supply and demand, and when prices rise due to inflation, it becomes more challenging to discern changes in underlying economic conditions.

Inflation in Ireland has been on the rise in 2023, reaching a high of 7.7% in March. This has been driven by a number of factors, including the war in Ukraine, which has caused energy prices to soar, and supply chain disruptions caused by the COVID-19 pandemic. The rising cost of living is putting a strain on household budgets, particularly for low-income households.

In conclusion, inflation is the gradual rise in prices that diminishes the purchasing power of money. In Ireland, the impact depends on global and domestic factors. Policymakers aim to manage inflation to ensure economic stability and the well-being of individuals and businesses through effective monetary policies.

Artificial Intelligence

Artificial Intelligence (AI) refers to the development of computer systems or machines that perform tasks that would typically require human intelligence. It creates intelligent systems that can perceive, learn, reason, and make decisions similar to how humans do. It involves developing algorithms and computer programs that learn from and make predictions or decisions based on data. These algorithms can be trained on large datasets, enabling them to recognise patterns and make predictions or decisions that are accurate and consistent.

At its core, AI is about enabling machines to mimic certain aspects of human intelligence, such as:

- Perception: AI systems perceive and interpret information from their environment. This includes tasks like image recognition, speech recognition, natural language processing, and understanding sensor inputs.

- **Learning:** AI learns from data and experiences to improve their performance. They can identify patterns, extract insights, and adapt their behaviour based on feedback.
- **Natural Language Processing:** AI understands and generates human language. This includes tasks like language translation, chatbots, sentiment analysis, and speech synthesis.
- **Robotics and Automation:** AI can be applied to robotics, enabling machines to perform physical tasks autonomously or assist humans in various domains. This includes industrial automation, autonomous vehicles, and robot assistants.

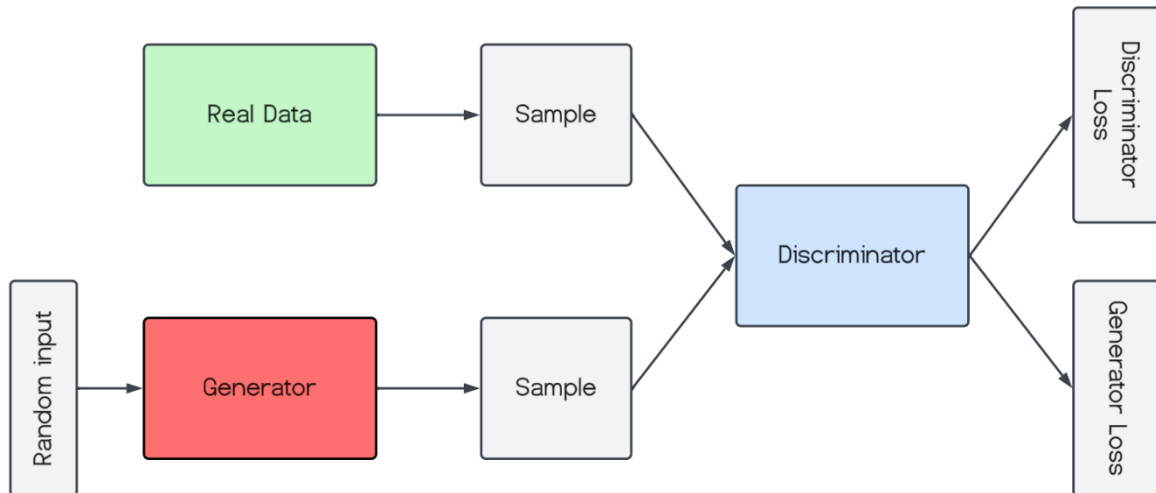
There are several types of AI, including rule-based systems, machine learning, and deep learning. Rule-based systems rely on a set of predefined rules to make decisions or predictions. Machine Learning algorithms use statistical models to learn from data and improve their performance over time. Deep learning is a subset of machine learning that uses artificial neural networks to learn from data and make decisions or predictions.

AI is used in a wide range of applications, from virtual assistants like Siri and Alexa to self-driving cars, medical diagnosis, fraud detection, and more. As AI continues to advance, it has the potential to revolutionise many aspects of our lives and transform industries across the board.

Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a class of AI models introduced by Ian Goodfellow and his colleagues in 2014. GANs consist of two neural networks, a generator, and a discriminator, engaged in a unique and adversarial training process.

- **Generator:** The generator's role is to create synthetic data, such as images, which is indistinguishable from real examples. It learns to generate content by mapping random noise into meaningful data, effectively creating new samples that resemble the training dataset.
- **Discriminator:** The discriminator acts as a detective, attempting to differentiate between real data from the training set and the generated data produced by the generator. It learns to classify input as either real or fake. The objective of the discriminator is to become adept at distinguishing between the two.



The generator's objective is to generate realistic data that is indistinguishable from real data. The goal of generative modeling is to autonomously identify patterns in input data, enabling the model to produce new examples that feasibly resemble the original dataset (Rahul 2023). The generator loss measures how well the generator is accomplishing this task. The generator loss is typically defined as the error between the generated samples and the real samples. The goal is to minimize this loss, encouraging the generator to produce samples that are increasingly similar to real data. The generator loss is often formulated as a minimax game where the

generator aims to minimize the probability that the discriminator correctly classifies generated samples as fake.

The discriminator's objective is to distinguish between real and generated data accurately. The discriminator loss measures how well the discriminator is performing this classification task. This loss is the error between the predictions made by the discriminator for real and generated samples. The goal for the discriminator is to maximize this loss, trying to correctly classify real and generated samples.

The training process involves a continuous loop of the generator creating fake data and the discriminator providing feedback. As the generator improves its ability to create realistic data, the discriminator adapts to better distinguish between real and generated examples. This adversarial process forces both networks to improve, and eventually the generator becomes so good that it can fool the discriminator into believing that its fake data is real.

GANs are based on a minimax game framework, where a generator G and a discriminator D are trained simultaneously. The GAN objective function is often expressed as:

$$\min_G \max_D V(D, G)$$

$$V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

x represents real data samples drawn from the true data distribution $p_{data}(x)$.

z represents random noise samples drawn from a latent distribution $p_{latent}(z)$.

$G(z)$ is the generated sample by the generator from the noise z .

$D(x)$ is the output of the discriminator indicating the probability that x is a real sample.

$\log D(x)$ encourages the discriminator to maximize the probability of correctly classifying real samples as real.

$\log(1 - D(G(z)))$ encourages the generator to minimize the probability of the discriminator correctly classifying generated samples as fake.

The training involves a dynamic interplay: the discriminator tries to get better at distinguishing real and generated data, while the generator tries to create data that is indistinguishable from real data.

The optimal state is reached when the generator produces data so realistic that the discriminator can't reliably tell if it's real or generated. This occurs when D is unable to distinguish between x and $G(z)$ effectively.

GANs provide a way to learn deep representations without extensively annotated training data. They achieve this by deriving backpropagation signals through a competitive process involving a pair of networks (Creswell 2018). They have found applications in various fields, including image and video generation, style transfer, data augmentation, and even text-to-image synthesis. Despite their effectiveness, training GANs can be challenging, and issues such as model collapse (where the generator produces limited types of samples) and training stability continue to be areas of research. The versatility and ability of GANs to generate diverse and realistic data make them a powerful tool in the field of artificial intelligence.

Some benefits of using GANs include:

- GANs surpass the constraints of traditional generative models, producing more realistic and creative data compared to models like variational autoencoders (VAEs). Their ability to learn from both the dataset and the

discriminator's feedback enables GANs to capture intricate patterns and relationships in the data.

- They are versatile tools suitable for various applications, including image and video generation, audio creation, and text generation, making them valuable for solving diverse problems.
- They enhance the performance of machine learning models like image classifiers and natural language processing (NLP) models by generating ample high-quality data for their training.
- They serve as a potent tool in research and development, aiding in the creation of novel data generation methods, enhancing machine learning model performance, and innovating new applications.

GANs are a rapidly developing field of research, and there are many new and exciting applications for this technology being developed all the time. GANs are a rapidly developing field of research, and there are many new and exciting applications for this technology being developed all the time. GANs have gained massive attraction in computer vision, feature representation, and more recently in Natural Language Processing (NLP) tasks (Wang 2017). I believe that they have the potential to revolutionise the way we generate data and solve problems in a variety of fields.

In summary, GANs represent a revolutionary advancement in artificial intelligence. Their unique architecture enables the creation of realistic and diverse data across various domains, impacting fields from image generation to model enhancement. Despite their versatility, challenges like training stability and model collapse persist, prompting ongoing research. As they evolve, their influence on research, development, and creative applications holds great promise for shaping the future of artificial intelligence.

My Code / Experimental Methods

In this section, you will find a breakdown of the algorithms used in my code to create the GAN model, and the HTML/CSS and JS used for the GUI.

I decided to create my project using the principles of User-centred design (UCD). This is a design approach that focuses on creating products, services, and experiences that are centred around the needs and desires of the people who will use them. To achieve this, UCD relies on a set of principles that guide the design process. These principles include:

- Empathy: Understanding the needs, goals, and motivations of the users and putting oneself in their shoes.
- Collaboration: Involving users in the design process and collaborating with them to co-create solutions.
- Iteration: Prototyping and testing ideas quickly and repeatedly to refine and improve the design.
- Inclusivity: Designing for the diverse needs and abilities of all potential users.
- Accessibility: Creating products and services that are accessible and usable by people with disabilities.
- User testing: Gathering feedback from users and using it to inform and improve the design.

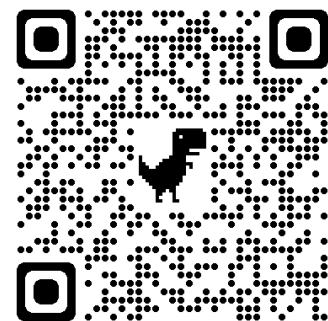
I decided to apply UCD using the Agile Framework. Agile methodology is a project management approach that enables flexibility and empowers practitioners to engage in rapid iteration. It is based on the Agile Manifesto (Fowler 2001), a set of values and principles for software development that prioritize the needs of the customer, the ability to respond to change, and the importance of delivering working software regularly. Agile methodologies, such as Scrum and Lean, are designed to

help teams deliver high-quality products in a fast-paced and dynamic environment by breaking down complex projects into smaller, more manageable chunks and continually reassessing and adjusting the project plan as needed. Agile teams are typically self-organizing and cross-functional and rely on regular communication and collaboration to achieve their goals.

While agile is best used in a team setting, it can be adapted to use by individual developers. I managed my Agile system using Kanban (Fig.1)



Kanban is a project management method that originated in Japan in the 1950s and is based on the principles of lean manufacturing. It is designed to help teams visualize and optimize their workflows by breaking down tasks into smaller, more manageable chunks and tracking their progress through a series of stages. Kanban uses a visual board to represent the distinct stages of a project, with cards representing individual tasks and columns representing the various stages of the workflow. The goal of Kanban is to increase efficiency and transparency by making it easier for team members to see where work is in the process, identify bottlenecks and inefficiencies, and adjust as needed. Kanban is often used in conjunction with other agile methodologies, such as Scrum, to help teams manage their work more effectively. (Corona 2013). Please scan here for my full Kanban journey.



Iteration 1:

Planning Phase

I started my project by creating a mind map (Fig. 2) of all the potential ideas that I could pursue through the lens of GANs. A quick check of the BT Young Scientist website (Competition 2023) showed several my ideas had already been tried by other competitors over the last few years so I looked towards a more topical issue.

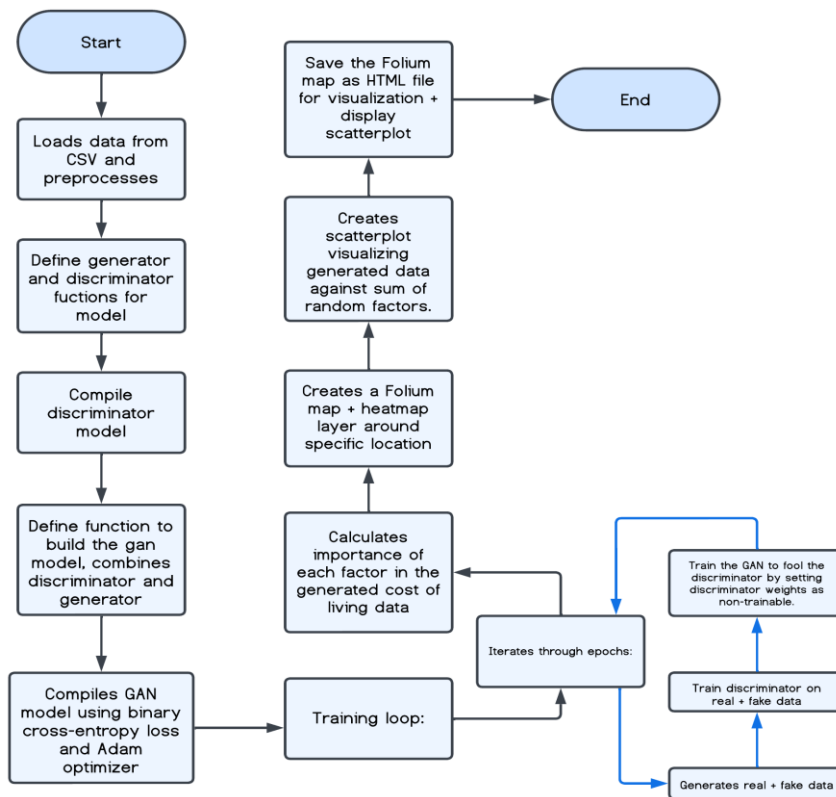


Fig.2

I settled (Fig 3.) on the Cost of Living Crisis and creating a GAN to generate cost of living hotspots. My Hypothesis is that a GAN developed to explore cost of living hotspots in Ireland can benefit individuals, families and government departments to manage finances, make informed decisions and help them to visualize the relationships between the integrated factors that affect the cost of living. Additionally, the generated data from the GAN will be indistinguishable from real data.



Design Phase



The basic algorithm seen above (Fig.4) for my first iteration was a simple GAN to analyse factors and produce a heatmap. I found that mapping out the data flow before I coded it, helped me to plan for any tricky coding elements such as where I would need to put a conditional statement or a function.

Development Phase – Model Development Steps

In this section of the Iteration 1 Development Phase, you will find the steps it took to develop and train the model and an explanation of the functions used in version 1. These steps and processes are crucial when developing any AI model.

1. Data Loading:

```
10 # Load and preprocess your spatial data (assuming it's in a CSV file)
11 df = pd.read_csv('your_spatial_data.csv')
```

Fig. 5

The `pd.read_csv` function from the Pandas library (Fig. 5) is used to read data from a CSV file (line 11). The file path is `'your_spatial_data.csv'`. This assumes that the data is stored in a CSV file. A CSV (Comma-Separated Values) file is a plain text file format used to store tabular data, with each line representing a row and fields separated by commas, enabling easy interchange of data between applications. The loaded data is stored in a Pandas DataFrame called `'df'`. A DataFrame is a two-dimensional, tabular data structure that allows for easy manipulation and analysis of data. The columns in the CSV file have information such as latitude, longitude, and a value for `'cost_of_living'`.

2. Model Preparation:

```
20 def build_generator(latent_dim):
21     model = Sequential()
22     model.add(Dense(16, input_dim=latent_dim, activation='relu'))
23     model.add(Dense(1, activation='linear'))
24     return model
25
26
27 # Discriminator model
28 def build_discriminator():
29     model = Sequential()
30     model.add(Dense(16, input_dim=1, activation='relu'))
31     model.add(Dense(1, activation='sigmoid'))
32     return model
33
34 # Build and compile GAN model
35 def build_gan(generator, discriminator):
36     discriminator.trainable = False
37     gan_input = Input(shape=(latent_dim,))
38     generated_data = generator(gan_input)
39     gan_output = discriminator(generated_data)
40     gan = Model(gan_input, gan_output)
41     gan.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.001))
42     return gan
```

Fig. 6

```

88 # Hyperparameters
89 latent_dim = 10 # Latent dimension for generator
90 epochs = 10
91 batch_size = 32
92
93 # Build models
94 generator = build_generator(latent_dim)
95 discriminator = build_discriminator()
96
97 # Compile the discriminator model
98 discriminator.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])
99
100 # Build GAN model and compile it
101 gan = build_gan(generator, discriminator)

```

Fig. 7

Model preparation involves defining and configuring the components of the GAN. The *'build_generator'* function (Fig. 6) defines a generator model using the Sequential API from TensorFlow's Keras (line 20,21). The generator takes random noise as input and produces synthetic cost of living data. It consists of two fully connected layers, the first with 16 units and ReLU activation (line 22), and the second with 1 unit and linear activation (line 23). The *'build_discriminator'* function defines a discriminator model using the Sequential API (line 28,29). The discriminator is designed to classify whether the input data is real or generated. It consists of two fully connected layers, the first with 16 units and ReLU activation (line 30), and the second with 1 unit and sigmoid activation (line 31).

The discriminator model is then compiled using binary cross-entropy loss and the Adam optimizer (line 98). This prepares the discriminator for binary classification, where it learns to distinguish between real and generated cost of living data.

The *'build_gan'* function (Fig. 6) combines the previously defined generator and discriminator models into a GAN model (line 35). The discriminator is set as non-trainable during the training of the GAN to ensure that only the generator weights are updated (line 36). The model is then compiled with binary cross-entropy loss and the Adam optimizer (line 41). Hyperparameters such as the latent dimension (latent_dim), number of epochs, and batch size are specified to configure the training process (Fig. 7). The discriminator model is separately compiled using binary cross-entropy loss and the Adam optimizer. This step is necessary as the discriminator is trained independently during the adversarial training loop.

3. Model Creation and Model Fitting:

```
13 # Generate synthetic data
14 def generate_real_data(samples):
15     factors = np.random.rand(samples, 5) # 5 factors affecting cost of living
16     cost_of_living = np.sum(factors, axis=1) + np.random.normal(0, 0.1, samples)
17     return factors, cost_of_living
```

Fig. 8

The `'generate_real_data'` function (Fig. 8) is responsible for creating synthetic real data samples that mimic the factors affecting the cost of living. This function is used within the training loop of the GAN to generate both real and fake data for training the discriminator and the generator.

The function takes a single parameter (*samples*) representing the number of synthetic data samples to be generated (line 14). The factors are created as random values in a 2D NumPy array (*factors*), where each row corresponds to a single data sample, and there are 5 factors influencing the cost of living (line 15). The cost of living is computed based on the generated factors. The cost is determined by summing the values along the rows of the factors array and adding a small amount of random noise using `'np.random.normal'` (line 16). This introduces variability to the cost of living values. Two arrays are then returned, `'factors'` and `'cost_of_living'` (line 17).

```
93 # Build models
94 generator = build_generator(latent_dim)
95 discriminator = build_discriminator()
96
97 # Compile the discriminator model
98 discriminator.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])
99
100 # Build GAN model and compile it
101 gan = build_gan(generator, discriminator)
```

Fig. 9

The model creation and fitting occur within the training loop for the GAN. The generator model (Fig. 9), `'build_generator'`, creates a neural network model that takes random noise as input and generates synthetic cost of living data (line 94). The discriminator model, `'build_discriminator'`, creates a neural network model that classifies whether input data is real or generated (line 95). The build and compile

GAN Model, *'build_gan'*, combines the generator and discriminator into a GAN model (line 101).

```

43 # Training loop
44 def train_gan(generator, discriminator, gan, real_data, epochs, batch_size):
45     for epoch in range(epochs):
46         for _ in range(batch_size):
47             real_factors, real_cost_of_living = real_data
48             real_labels = np.ones(batch_size) # All real data is labeled as 1
49             fake_cost_of_living = generator.predict(np.random.rand(batch_size, latent_dim))
50             fake_labels = np.zeros(batch_size)
51
52             d_loss_real = discriminator.train_on_batch(real_cost_of_living, real_labels)
53             d_loss_fake = discriminator.train_on_batch(fake_cost_of_living, fake_labels)
54
55             d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
56
57             fake_labels_gan = np.ones(batch_size)
58             g_loss = gan.train_on_batch(np.random.rand(batch_size, latent_dim), fake_labels_gan)
59
60             print(f"Epoch {epoch+1}, D Loss: {d_loss}, G Loss: {g_loss}")

```

```

103 # Training loop
104 real_data = generate_real_data(batch_size)
105 train_gan(generator, discriminator, gan, real_data, epochs, batch_size)

```

Fig. 10

The *'train_gan'* function (Fig. 10) implements the training loop for the GAN (line 44). The following steps occur within this function:

Generate Real and Fake Data:

- *'real_factors', 'real_cost_of_living = real_data'*: Real data is sampled from the actual dataset.
- *'fake_cost_of_living=generator.predict(np.random.rand(batch_size, latent_dim))'*: Synthetic data is generated by the generator.

Discriminator Training:

- Trains the discriminator on real and fake data separately using the *'train_on_batch'* method.
- Computes the discriminator loss, *'d_loss'*, as a combination of losses from real and fake data.

Generator Training:

- Trains the model to generate data that fools the discriminator (i.e., make the discriminator predict generated data as real).
- Computes the generator loss, *'g_loss'*, based on its output.

Epoch Iteration:

- Repeats the above steps for the specified number of iterations (controlled by *'batch_size'*) within each epoch.
- Prints the discriminator and generator losses for each epoch executed.

```
62 # Calculate factor importance
63 def calculate_factor_importance(generator, latent_dim):
64     # Initialize an array to store the effects of each factor
65     factor_effects = np.zeros(latent_dim)
66
67     # Generate random noise
68     noise = np.random.rand(1000, latent_dim)
69
70     # Generate synthetic cost of living values
71     generated_cost_of_living = generator.predict(noise)
72
73     # Calculate the effect of each factor by subtracting the mean effect
74     mean_effect = np.mean(generated_cost_of_living)
75
76     # Calculate the effect of each factor individually
77     for i in range(latent_dim):
78         factor_noise = noise.copy()
79         factor_noise[:, i] = 0 # Set one factor to zero while keeping others constant
80         factor_effect = np.mean(generator.predict(factor_noise)) - generated_cost_of_living
81         factor_effects[i] = factor_effect
82
83     # Print the factor effects
84     print("Factor Effects:")
85     for i, effect in enumerate(factor_effects):
86         print(f"Factor {i+1}: {effect}")
```

Fig. 11

The *'calculate_factor_importance'* function (Fig. 11) is used to analyze the importance of each factor in the generated cost of living data. This function provides insights into how individual factors contribute to the variation in the synthetic cost of living values generated by the GAN.

The function creates an array, *'factor_effects'*, to store the effects of each factor (line 65). Random noise is then generated to create synthetic data using the generator (line 71). The function further calculates the effect of each factor by subtracting the mean effect from the generated cost of living data (line 74). The effect of each factor is calculated individually by setting one factor to zero while keeping others constant (line 79). The results are printed at the end to show the factor effects.

4. Model Prediction, Evaluation and Visualisation:

```
97 # Compile the discriminator model
98 discriminator.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])
```

Fig. 12

During the training loop, the discriminator model is trained on both real and fake data (Fig. 12). The accuracy of the discriminator is evaluated using binary cross-entropy loss and the Adam optimizer during its compilation (line 98).

```
49 fake_cost_of_living = generator.predict(np.random.rand(batch_size, latent_dim))
```

The generator is used to predict synthetic cost of living data based on random noise (Fig. 12). This is done during the training loop to generate fake data for training the GAN (line 49).

```
114 m = folium.Map(location=[53.350140, -6.266155], zoom_start=12)
115
116 # Add a heatmap layer using your cost of living data
117 heat_data = [[row['latitude'], row['longitude'], row['cost_of_living']] for index, row in df.iterrows()]
118 folium.plugins.HeatMap(heat_data).add_to(m)
119
120 # Display the map
121 m.save('cost_of_living_heatmap.html')
```

Fig. 13

A Folium map (Fig. 13) is created and centered around a specific location (line 114). In the first version of the code, the location is set within the parameters in the code. A heatmap layer is added to the map using the cost of living data from the original dataset (line 117,118).

```
122 # Visualize generated data
123 plt.scatter(np.sum(np.random.rand(num_samples, 5), axis=1), generated_cost_of_living, label='Generated Data')
124 plt.xlabel('Sum of Factors')
125 plt.ylabel('Cost of Living')
126 plt.legend()
127 plt.show()
```

Fig. 14

Matplotlib is used to create a scatter plot (Fig. 14) to visualize the generated cost of living data against the sum of random factors (line 123).

Using these visualisation steps helped in assessing the quality of the generated data and how well it aligns with the original dataset. The heatmap provides a simple geographical representation of the cost of living, while the scatter plot shows the relationship between the generated cost of living and the sum of random factors.

Testing Phase

The testing in iteration one was especially important as this was the first version of my GAN model. I had to test various parts of the code throughout development. Testing the factor importance, generator, discriminator and GAN functions was very important. I had to make sure that the functions were working correctly at given inputs and the factor importance was functioning correctly.

Experimenting with the hyperparameters was another example of testing in iteration one. There are various parameters in the code like the number of epochs, batch size, and latent dimensions. I found that the more epochs run, the better the GAN gets over time.

I also had to make sure that the scatterplot and folium maps were being generated each time the code was run. I made sure that the folium map was saving as a HTML file each time the code was finished running. Although the model in this iteration was trained on a very small, simple dataset, testing each function and each section of the code thoroughly was very important to make sure that is I used a larger dataset that the code would run smoothly without hassle.

Iteration 2:

Planning Phase

My plan for iteration 2 was to keep working on the GAN model. I decided that I wanted to add a feature in the code where the user can enter a number of various factors (e.g. rent, fuel, education) and the GAN will search for the columns with those given factors in the dataset. The model will then analyse those factors and create synthetic data based on whatever factors the user enters. I also decided to add in an input so the user can enter a location, and the GAN will create the heatmap based on that given location and its relevant cost of living factors.

As well as adding those inputs, I planned on adjusting the code to work with these new functions and update any old ones to work with these inputs. (Fig. 15)

Additionally, I wanted to try create a basic user GUI to work with the factor and location inputs. This would allow users to enter the factors and location outside of the regular Thonny/VSC interface.

Design Phase

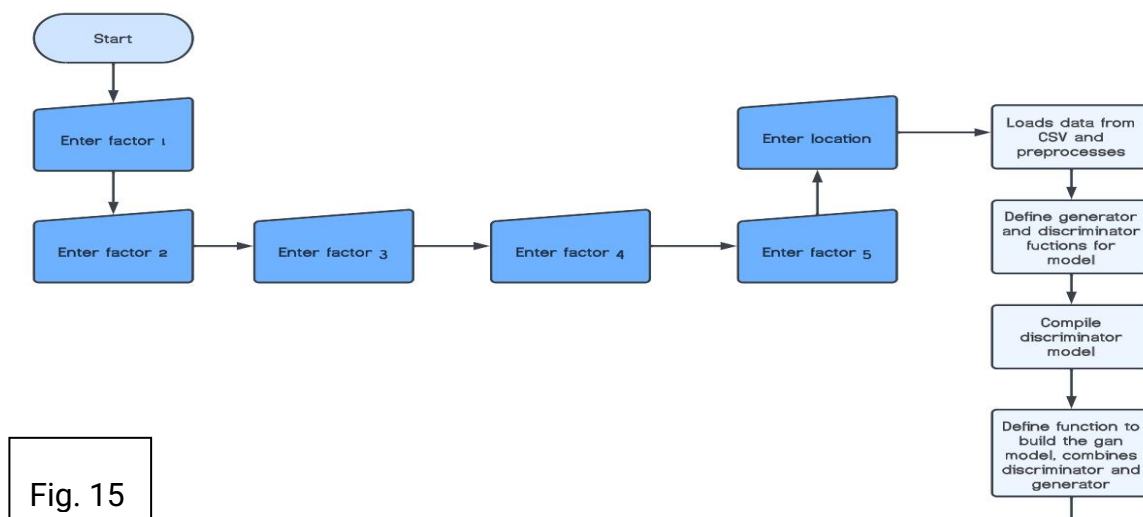


Fig. 15

During this iteration I focused on the function for the location and factor inputs. I needed to be sure that when coding I understood how the algorithm (Fig.10)

would interact with the GAN and the other functions already in the code. These factors would be put into a dictionary once they were found in the dataset. A Python dictionary is a mutable, unordered collection of key-value pairs, where each key must be unique, and the keys and values can be of various data types.

I also planned out what inputs and features I wanted to include in the basic GUI. I figured the GUI would need to have the following features:

- Input spaces for each factor
- An input space for the location
- An appropriate submit button

As well as those features, once the code is done running, the scatterplot and heatmap would need to be displayed somewhere within the GUI.

Development Phase

```
12 variables = {}
13 variable_names = ["Variable1", "Variable2", "Variable3", "Variable4", "Variable5"]
14
15 # Loop to take input for each variable
16 for name in variable_names:
17     value = (input(f"Enter a numeric value for {name}: ")) # Convert input to float
18     variables[name] = value
19
```

Fig. 16

Here is where I added in the first input to enter the factors (Fig. 16). A dictionary, *'variables'*, is created as an empty dictionary (line 12). Then, it's populated by iterating through a list of variable names, *'variable_names'*, (line 13). For each name, the code prompts the user to input a numeric value (line 17), converts the input to a string, and assigns the value to the corresponding key in the *'variables'* dictionary (line 18). This results in a dictionary where each variable name is associated with a numeric value entered by the user.

```

23 Search_town = input(" Please enter a town: ")
24 def generate_realD_from_csv(data):
25     # Select the columns corresponding to the 5 factors
26
27     factors = data[[variables["Variable1"], variables["Variable2"], variables["Variable3"], variables["Variable4"], variables["Variable5"]]].values
28
29     # Calculate the cost of living
30
31     costOL = np.sum(factors, axis=1) + np.random.normal(0, 0.1, len(factors))
32
33     return factors, costOL
34
35
36 # Check if the town exists in the DataFrame
37 if Search_town in data['town'].values:
38     # Find the row corresponding to the town
39     row = data[data['town'] == Search_town].iloc[0]
40
41     # Extract longitude and latitude
42     Long1 = row['longitude']
43     Lat1 = row['latitude']
44
45     factors, costOL = generate_realD_from_csv(data)
46     Long1=Long1
47     Lat1=Lat1

```

Fig. 17

The section of the code above (Fig. 17) is where the user enters the location, and the underneath is the function taking the factors from the CSV file.

The user is prompted to input the name of a town or location (line 23). The town name entered is stored in the variable '*Search_town*'. A function named '*generate_realD_from_csv*' is then defined (line 24). This function takes the DataFrame, '*data*', as an argument. The function selects specific columns corresponding to the 5 factors identified by the variable names from the input DataFrame. These columns are stored in the '*factors*' variable (line 27). A cost of living, '*costOL*', is calculated as an example (line 31). It's computed as the sum of the selected factors along each row, with some random noise added.

The code checks if the town entered by the user, '*Search_town*', exists in the '*town*' column of the DataFrame (line 37). If it's found in the DataFrame, it proceeds to extract the corresponding row (line 39). Once the town is found, the code extracts the longitude ('*Long1*') and latitude ('*Lat1*') values from the row corresponding to the relevant town (line 42, 43). The function '*generate_realD_from_csv*' is then called with the entire DataFrame. The resulting factors and cost of living are stored in the variables '*factors*' and '*costOL*'. The longitude, '*Long1*', and latitude, '*Lat1*', values are assigned from the row corresponding to the entered town.

```

138 def calculate_factor_importance_and_plot(generator, latentDim, data):
139     factorNames = ['rent', 'fuel', 'energy', 'food', 'education']
140     factorImportance = []
141
142     for i in range(len(factorNames)):
143         # Create a baseline input (noise)
144         baselineNoise = np.random.rand(1000, latentDim)
145
146         # Create a modified input with the current factor set to its mean value
147         modifiedNoise = baselineNoise.copy()
148         modifiedNoise[:, i] = np.mean(data[factorNames[i]])
149
150         # Generate output with baseline and modified input
151         baselineOutput = generator.predict(baselineNoise).flatten()
152         modifiedOutput = generator.predict(modifiedNoise).flatten()
153
154         # Calculate the importance as the mean absolute difference
155         importance = np.mean(np.abs(baselineOutput - modifiedOutput))
156         factorImportance.append(importance)
157
158     return factorImportance
159
160 factorImportance = calculate_factor_importance_and_plot(generator, latentDim, data)
161
162
163

```

Fig. 18

This function assesses the importance of each factor in influencing the output of a generator model in the GAN. It takes in three parameters: *'generator'*, *'latentDim'* and *'data'* (line 138). For each factor in *'data'*, baseline input noise is generated (line 144), a modified input noise is created by setting the current factor to its mean value from the real dataset (line 147), an output is generated using both the baseline and modified input noises (line 151, 152), and the factor importance is calculated as the mean absolute difference between the outputs (line 155, 156). The function returns a list, *'factorImportance'*, containing the calculated importance values for each factor (line 159). Then, the function is called with the generator model, *'generator'*, latent dimension, *'latentDim'*, and the dataset *'data'* (line 162). The resulting factor importance values are stored in *'factorImportance'*.

```

194 # First plot (bar graph for factor importance)
195 axs[0].bar(['rent', 'fuel', 'energy', 'food', 'education'], factorImportance, color='skyblue')
196 axs[0].set_xlabel('Factors')
197 axs[0].set_ylabel('Importance')
198 axs[0].set_title('Importance of Each Factor on Cost of Living')

```

Fig. 19

Here, I decided to add in a bar graph showing the importance of the factors. This code creates a bar graph to display the importance of factors influencing the cost of living. The graph is plotted on the first subplot *'axs[0]'*, where the x-axis

represents factors, the y-axis represents their importance. When training the GAN on a larger dataset, this graph will show which factors affect the cost of living the most and least in the given location.

1	town	longitude	latitude	cost_of_living		rent	food	energy	fuel	education
2	Tokyo	139	35	0.12		0.12	12.3	0.5	2.9	2.6
3	Jakarta	106	6.19	0.11		0.12	12.3	0.5	2.9	2.6
4	Delhi	77.1	28	0.15		0.12	12.3	0.5	2.9	2.6
5	Guangzhou	113	23	0.15		0.12	12.3	0.5	2.9	2.6
6	Mumbai	72	19	0.2		0.12	12.3	0.5	2.9	2.6
7	Manila	120	14	0.14		0.12	12.3	0.5	2.9	2.6
8	Shanghai	121	31	0.56		0.12	12.3	0.5	2.9	2.6
9	Sao Paulo	46	23	0.4		0.12	12.3	0.5	2.9	2.6
10	Seoul	126	37	0.11		0.12	12.3	0.5	2.9	2.6

Fig. 20

Above (Fig. 20), is a section of the testing dataset I used in this version of the code. As seen, there are a few columns. Mentioned above, the program calls some of these columns once the user has entered a town. For testing purposes, I added random locations around the world and rough longitudes and latitudes corresponding. For example, if the user enters ‘Tokyo’, the program will find Tokyo in the town column and the GAN will run based on the five factors in the Tokyo row. If the town entered is not in the dataset then the program will not run as there would be no data to train on. The cost of living and other factors are also not real for testing purposes.


Once this code was working well, I decided to focus on trying to build a basic GUI for the user to interact with. A GUI (Graphical User Interface) is a particularly vital component in accessing Python programs. The text-based nature of my code does not lend itself to users who are not experienced in using IDE’s like Thonny or VSC.

My initial program had a series of input code lines that populated a dictionary, an example of this is below (Fig. x):

```
Enter the following factors (rent,food,education, energy and fuel) value for Variable1: rent
Enter the following factors (rent,food,education, energy and fuel) value for Variable2: food
Enter the following factors (rent,food,education, energy and fuel) value for Variable3: fuel
Enter the following factors (rent,food,education, energy and fuel) value for Variable4: education
Enter the following factors (rent,food,education, energy and fuel) value for Variable5: energy
Please enter location, e.g Cairo: Cairo
```

Fig.21

I had developed a skillset in the use of HTML/CSS and JavaScript from my 2023 BT Young Scientist Project, but this was not helpful when dealing with a purely Python-based project. I was watching some Youtubers who have had success with Tkinter (NeuralNine 2022). The appeal of Tkinter is the straightforward syntax rules which are aligned with the Python syntax. It gave me the ability to create text boxes, buttons, and other so-called widgets. Here is an example of one of my initial iterations of the GUI using Tkinter (Fig. 22).

 Data Input

Rent	<input type="text"/>
Fuel	<input type="text"/>
Energy	<input type="text"/>
Education	<input type="text"/>
Food	<input type="text"/>
Town	<input type="text"/>
<input type="button" value="Submit"/>	

Fig. 22

This GUI allowed me to input the five factors and the location into the dictionary I had written in the code and by pressing the submit button, enabling the program to run. Issues began when I tried to tie in the Matplotlib bar graph and the

HTML heatmap into the Tkinter GUI. There seemed to be an issue with the Tkinter module finding the matplotlib graph when it was created at the end of the process. I was also unable to embed the HTML heat map for the same reason.

Testing phase

Testing in iteration 2 was important as I had added new functions and features to the code. Like iteration 1, I tested each function and ensured they were giving the correct outputs. I also experimented with changing epoch and batch sizes to see what the GAN would output. The new functions worked well and the feature for inputting the factors and location worked great.

Below is an image while running the code (Fig. 23).

```
4/4 [=====] - 0s 0s/step
4/4 [=====] - 0s 0s/step
4/4 [=====] - 0s 0s/step
4/4 [=====] - 0s 0s/step
4/4 [=====] - 0s 0s/step
4/4 [=====] - 0s 5ms/step
4/4 [=====] - 0s 0s/step
4/4 [=====] - 0s 0s/step
4/4 [=====] - 0s 0s/step
4/4 [=====] - 0s 5ms/step
4/4 [=====] - 0s 0s/step
4/4 [=====] - 0s 0s/step
4/4 [=====] - 0s 0s/step
4/4 [=====] - 0s 5ms/step
4/4 [=====] - 0s 0s/step
Epoch 5, Discriminator Loss: [0.77102968 0.48828125], Generator Loss: 0.8263909816741943
4/4 [=====] - 0s 0s/step
4/4 [=====] - 0s 0s/step
4/4 [=====] - 0s 0s/step
```

Fig. 23

Like said previously, I was having issues with the Tkinter GUI. I tried to fix it a few different ways, but nothing seemed to solve what problem I was having. I knew that if a GUI wouldn't be possible in this case that the other code was working well, but I figured that a different approach later on might be the solution to creating a working GUI.

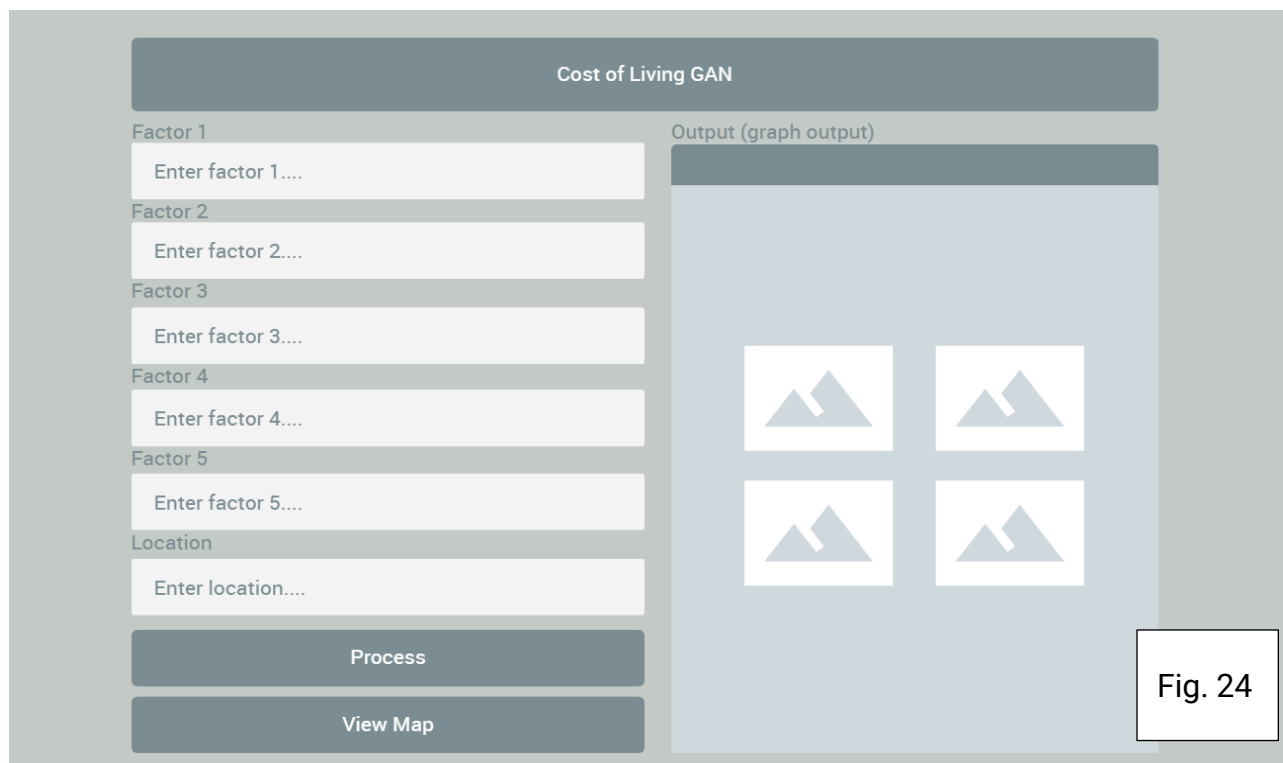
Iteration 3:

Planning phase

For the last iteration, I needed to focus on completing a few main things. I needed to think of another method to create a GUI and develop it. Although it was not the main focus of my project, I decided to add a bit of CSS to make the GUI look cleaner and more organised. Adding extra CSS or HTML wouldn't be too difficult as my Young Scientist project last year involved a lot of CSS/HTML.

I also planned to train my GAN on a real, large dataset. The program works well on a small dataset so using a larger dataset shouldn't cause too many problems. Additionally, If there is any parts of the code that need to be modified then I would complete that in this iteration.

Design phase



The screenshot displays a web application titled "Cost of Living GAN". On the left side, there are five input fields labeled "Factor 1" through "Factor 5", each with a placeholder text "Enter factor 1...." and a corresponding "Process" button below them. Below these is a "Location" input field with a placeholder "Enter location...." and a "View Map" button. On the right side, there is a section titled "Output (graph output)" which contains four placeholder images, each represented by a triangle icon. The entire interface is set against a light gray background.

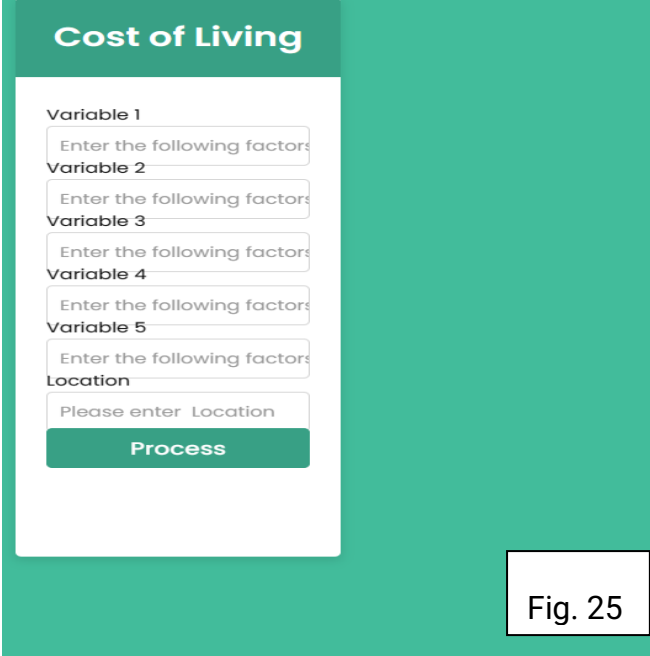
Fig. 24

I designed a simple wireframe of what I wanted the GUI to look like (Fig. 24) if I could achieve that level of customisation. I wasn't entirely sure if I would have the time or if I would be able to get something like that after having problems with the Tkinter GUI. If I ended up being able to get a GUI that looks similar, I would additionally change things like the colour and font styles. Again, this was not a main component of my project.

Development phase

After some thinking I disregarded the Tkinter GUI approach due to time constraints and focused on creating a simple Flask based GUI interface. I had initially thought about using a Django based GUI, but my teacher, concerned for the time necessary to become competent in Django, suggested that I try an online course in beginner web applications using Flask. I used this tutorial to get me started (Loeber 2021).

Using Flask allowed me to use my HTML/CSS/JavaScript coding experience once I was able to deploy the Flask module. I found a simple calculator python program that incorporated Flask in GitHub (Ahmed 2022). Through Flask, I was able to recreate the form I had developed from using Tkinter (Fig. 25).



Cost of Living

Variable 1
Enter the following factors

Variable 2
Enter the following factors

Variable 3
Enter the following factors

Variable 4
Enter the following factors

Variable 5
Enter the following factors

Location
Please enter Location

Process

Fig. 25

Flask also allowed me to incorporate the matplotlib bar graph (Fig. 26).

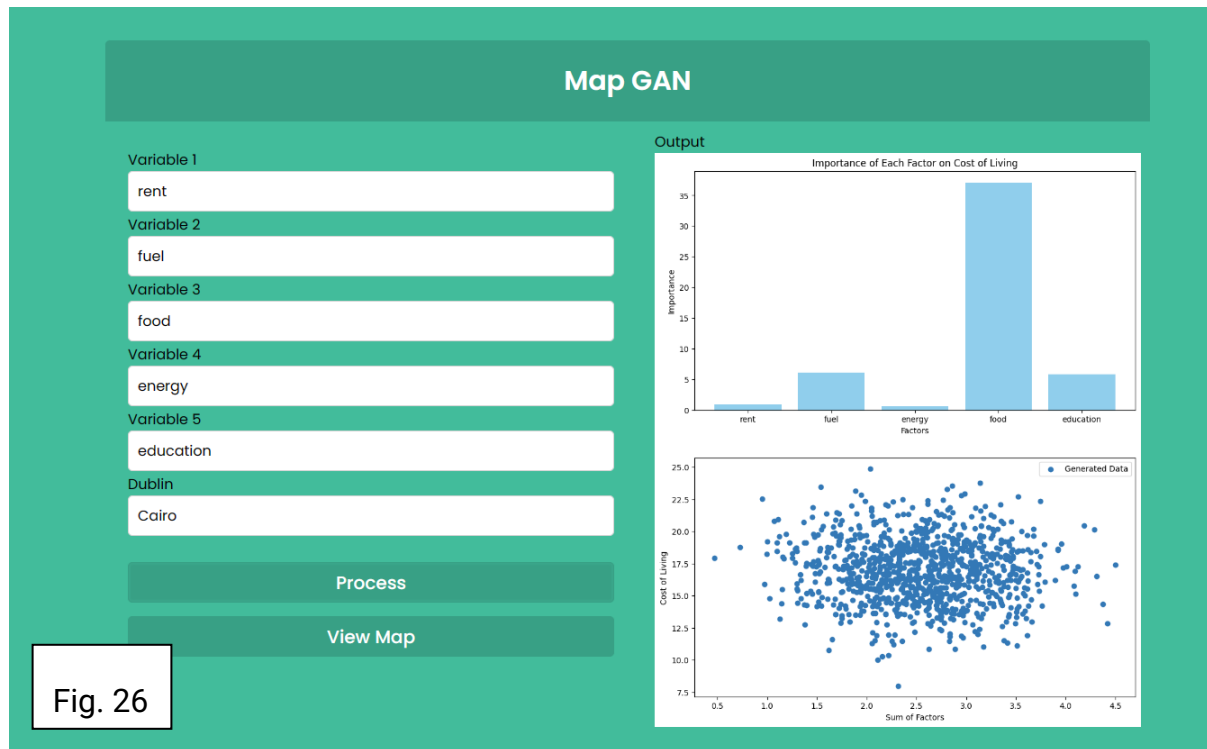


Fig. 26

Because the HTML heatmap is created at the same time as the matplotlib I was unable to get the heatmap to load onto the same page as the Bar and Scatter

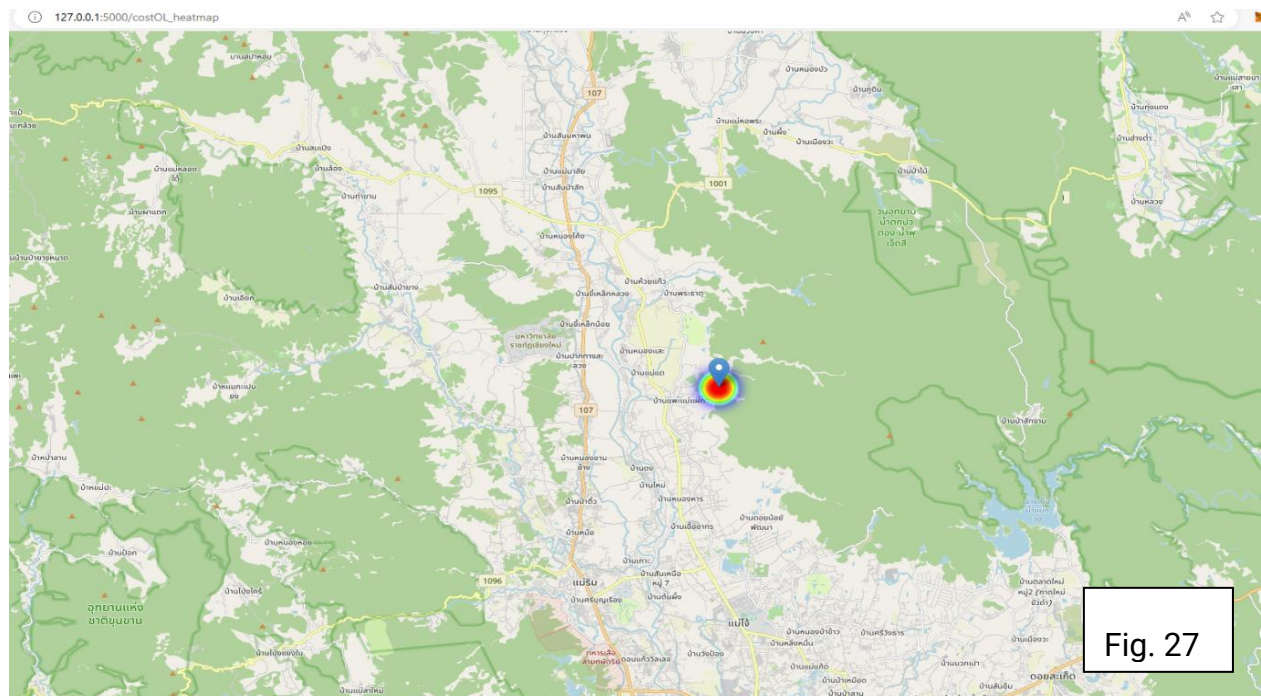


Fig. 27

plots. Instead, I created a simple ‘View Map’ button linked to a ‘get’ action in JavaScript which loads it on its own unique page (Fig. 27).

```
110     function get_map(){  
111     window.open(  
112         './costOL_heatmap',  
113         '_blank' // <- This is what makes it open in a new window.  
114     );  
115 }
```

Fig. 28

Once the heatmap has been created and saved, you can open it in a new tab using the ‘View Map’ Button. Above is the JavaScript code which opens it in a new window (Fig. 28).

Once the GUI was built and working with the Python code, I focused on the final few bits with the GAN. I needed to find and train the GAN on a realistic dataset and adjust the code in any ways that might fit more appropriately with the dataset.

Central Statistics Office (CSO) & Data Cleaning

I started by finding a dataset of urban areas in Ireland. My initial plan was to find a list of 18444 small census areas but the files I tried to access did not come with a key to convert the unique identification number into a longitude and latitude. The available converter (<https://gnss.osi.ie/new-converter/>) is a blunt tool and does not allow for the GEOID attached to the small census area to be converted.

I settled on the new urban Geography developed by CSO, Tailte Éireann and the Department of Housing. “They were generated using an automated process based on Prime2 building counts and constraints on distances between buildings.” (CSO, 2022). The definition that they use for an urban area is at least 100 buildings within 65m with building groups of 100 or more to be within 500m of each other.

‘Developing a Generative Adversarial Network to explore Cost of Living hotspots in Ireland.’

OBJECTID	URBAN_A	URBAN_A	URBAN_A	COUNTY	Centroid_x	Centroid_y	SHAPE_L	SHAPE_A
1	0138fb4f-2	27295	Bearna	Galway	522593.2	723182	18629.68	3989630
2	0139a442-	10019	Ardee	Louth	696152.2	790695.3	10477.58	4747904
3	020a2786-	28125	Ballinamc	Leitrim	613071.6	811532.5	11491.71	1205382
4	0261d090-	11408	Ratoath	Meath	701909.1	751650.8	11723.46	3067988
5	0265ada6-	16544	O'Briensb	Clare	566175	666640.3	7603.992	556121.6
6	026936ae-	18775	Inishann	Cork	554631.9	557441.9	10848.29	602251
7	026f406e-4	32058	Swanlinb	Cavan	619422.4	826996.9	4652.568	383567.6
8	02adfd39-	7658	Castlecon	Kilkenny	653108	672358.6	8434.483	1097055
9	03985ab2-	7666	Ballyragge	Kilkenny	644927.4	670925.1	7729.148	747545.3
10	03a8151e-	33068	Bridge En	Donegal	639591.4	921496	4254.902	307704.4
11	03e074bb-	10004	Termonfex	Louth	714101.6	780372	9316.231	879422.4
12	04070383-	13200	Athlone	Westmeat	604311.8	742043.2	37714.28	17302213
13	04456694-	18769	Ballinspit	Cork	558854.1	545913.9	8368.851	299071.9
14	04579a70-	8166	Emo	Laois	652546.2	705236.2	8397.787	319186
15	04928248-	18773	Crossbar	Cork	555556.6	561305.1	5133.541	287372.4
16	0598e968-	18742	Fermoy	Cork	580811.2	598586.7	17035.29	3412116
17	05a4a02c-	6436	Celbridge	Kildare	696859.1	733151.8	17532.65	5197524
18	0617f0f8-8	6463	Milltown	Kildare	676688.4	717461.8	10068.43	391171
19	0654abe6-	28137	Tullaghan	Leitrim	578763.9	858025.9	5678.251	532039.2
20	06d3a035-	23711	Golden	Tipperary	601225.1	638754.1	8076.896	312793.7
21	06ea8799-	4359	Garristow	Dublin	707174.8	758484	7216.351	576388.4

Fig. 29

As with many of the more modern CSV files (Fig. 29), there isn't usually a readily accessible latitude/longitude that corresponds to the area. Instead, they used a Centroid_x and Centroid_y. These correspond to the Irish Transverse Mercator (ITM) system used in surveying. This system breaks up Ireland into a unique grid as you can see from the diagram below (Fig. 30).

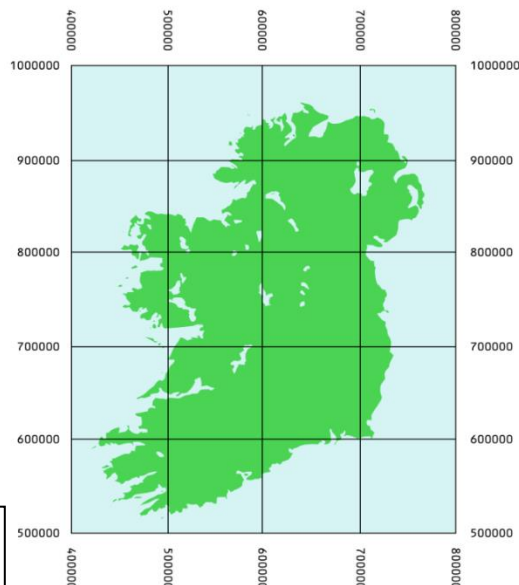


Fig. 30

In this system, Ireland is broken up in Northings and Eastings. This required me to write a program to convert ITM into a more traditional longitude and latitude. My program takes the Northing and Easting values and passes them through a python module called pyproj which is a powerful cartographic and co-ordinates transformation library. As you can see from the diagram below it passes the new converted values into a DataFrame which

writes them into the original files as new columns.


```
import pandas as pd
from pyproj import Proj, transform

df = pd.read_csv('coirish.csv')

itm = Proj(init='epsg:2157') # Irish Transverse Mercator
wgs84 = Proj(init='epsg:4326') # WGS84

def convert_itm_to_wgs84(easting, northing):
    longitude, latitude = transform(itm, wgs84, easting, northing)
    return pd.Series([latitude, longitude])

df[['Latitude', 'Longitude']] = df.apply(lambda row: convert_itm_to_wgs84(row['Easting'], row['Northing']), axis=1)

df.to_csv('converted_coordinates.csv', index=False)
```

Fig. 31

The CSO website has a lot of the data I needed in CSV form. After combining all the data into one csv sheet, I was able to use a simple algorithm to standardise the variables.

$$N(\text{variable}) = \frac{\max(\text{variable}) - \min(\text{variable})}{\text{variable} - \min(\text{variable})}$$

These are the 5 algorithms that the GAN uses to create the index values for the 5 variables (the acronyms are explained below) .

$$\text{Education} = w_{\text{SEP}} \times N(\text{SEP}) + w_{\text{PD}} \times N(\text{PD}) + w_{\text{PAD}} \times N(\text{PAD})$$

$$\text{Transport} = w_{\text{PD}} \times N(\text{PD}) + w_{\text{FC}} \times N(\text{FC}) + w_{\text{DUC}} \times N(\text{DUC})$$

$$\text{Housing} = w_{\text{PD}} \times N(\text{PD}) + w_{\text{IR}} \times N(\text{IR}) + w_{\text{RR}} \times N(\text{RR}) + w_{\text{NHUP}} \times N(\text{NHUP})$$

$$\text{Food} = w_{\text{EC}} \times N(\text{EC}) + w_{\text{CPI}} \times N(\text{CPI}) + w_{\text{AP}} \times N(\text{AP})$$

$$\text{Energy} = w_{\text{DUC}} \times N(\text{DTC}) + w_{\text{PD}} \times N(\text{PD}) + w_{\text{EP}} \times N(\text{EP})$$

SEP (Socio-economic Profile) - This is an index function I derived from CSO F7301, F7308.

PD (Population density) - List of the 867 largest towns densities from CSO F1013, F5134

PAD (Population age demographic) - The rate of birth increases in each town F5134, VSA102

FC (Fuel cost) - Rate of Transport fuel increase CPM03, CPA02

DUC (Distance from urban centre) - I wrote a python program that calculates using co-ordinate geometry, the distance from every town to its nearest large urban centre.

IR and RR (Interest rate and rental rate) - annualised rent a mortgage increases per annum since 2011

NHUP (New housing unit production) - BHQ16

EC (Energy cost) - CPM03, CPA02

CPI (Consumer Price index) - CPM03, CPA02

AP (Agricultural production) - AHMO4

A quick python program generates the 5 index values once all the data is in the same CSV file (Fig. 32). I found it way easier to cut and paste the columns I needed from the multiple CSV files I downloaded from the CSO website, rather than writing a program as the data columns are not all homogenous. I took data from 52 different CSV files to create my final dataset.

```
import pandas as pd

def normalize(value, min_value, max_value):
    return (value - min_value) / (max_value - min_value) if max_value > min_value else 0

def calculate_index(row, factors, weights):#Caluclates the index AC
    normalized_values = [normalize(row[factor], df[factor].min(), df[factor].max()) for factor in factors]
    return sum(weight * value for weight, value in zip(weights, normalized_values))

df = pd.read_csv('bigfile.csv')

indices_info = { #Reads in all the values and stick in Dictionary AC
    'Transport': ([ 'PD', 'FC', 'DUC'], [0.3, 0.4, 0.3]),
    'Energy': ([ 'DTC', 'PD', 'EP'], [0.3, 0.3, 0.4]),
    'Food': ([ 'EC', 'CPI', 'AP'], [0.3, 0.4, 0.3]),
    'Housing': ([ 'PD', 'IR', 'RR', 'NHUP'], [0.25, 0.25, 0.25, 0.25]),
    'Education': ([ 'SEP', 'PD', 'PAD'], [0.5, 0.3, 0.2])
}

for index_name, (factors, weights) in indices_info.items():
    df[index_name] = df.apply(calculate_index, axis=1, factors=factors, weights=weights)

df.to_csv('bigfile.csv', index=False)
```

Fig. 32

Once all of the data columns I needed were in the same CSV file, then I change the program to read in this CSV file instead of the old one. The new dataset contains over 800 different towns in Ireland and they each have their own factors. (Fig. 33)

town	COUNTY	longitude	latitude	cost_of_living	Housing	Food	Energy	Transport	Education	PD	Population	FC	DUC	EC	CPI	AP	NHUP	IR	RR	SEP	PAD
Ballinabre	Carlow	-6.9842	52.7849	2.82595	2.74333	1.42045	2.45071	2.61525	4.9	1371.3	557	3.04	0.0791	16.03	3.2	7.1	0.59574	2.7	9.8	3.60239	1.61203
Ballon	Carlow	-6.7727	52.7397	1.92426	2.73777	1.42045	2.44534	2.60973	0.408	1353.8	801	3.04	0.17845	16.03	3.2	7.1	0.14894	2.7	9.8	1.91521	1.97533
Borris	Carlow	-6.9248	52.6035	3.05057	2.45331	1.42045	2.16869	2.32237	6.888	696.9	702	3.04	0.23189	16.03	3.2	7.1	0.44681	2.7	9.8	3.33139	1.8434
Carlow	Carlow	-6.9233	52.8354	2.73759	2.97324	1.42045	2.67651	2.84577	3.772	2333.9	27351	3.04	0	16.03	3.2	7.1	16.4255	2.7	9.8	5.16017	5.45532
Clonegall	Carlow	-6.6477	52.6955	2.25909	2.66045	1.42045	2.36975	2.53187	2.31292	1130.9	249	3.04	0.30909	16.03	3.2	7.1	0.04255	2.7	9.8	2.04652	0.80692
Fennagh	Carlow	-6.8477	52.7141	1.9311	2.81361	1.42045	2.5198	2.68605	0.2156	1614.6	479	3.04	0.14297	16.03	3.2	7.1	0.02128	2.7	9.8	0.86962	1.46117
Hacketts	Carlow	-6.5579	52.8639	2.63195	2.83535	1.42045	2.54126	2.70799	3.6547	1698.1	653	3.04	0.36657	16.03	3.2	7.1	0.29787	2.7	9.8	3.26685	1.77105
Kernanstown	Carlow	-6.8726	52.8249	1.949	2.50877	1.42045	2.22212	2.37848	1.2152	793.7	269	3.04	0.05183	16.03	3.2	7.1	0	2.7	9.8	3.1256	0.88418
Kildavin	Carlow	-6.6848	52.6841	0.7025	2.66245	1.42045	2.3717	2.53388	-5.476	1136.2	207	3.04	0.28248	16.03	3.2	7.1	0.02128	2.7	9.8	3.2569	0.62219
Leighlinbri	Carlow	-6.9796	52.7357	2.28724	2.75797	1.42045	2.4651	2.63002	2.16267	1418.8	959	3.04	0.11455	16.03	3.2	7.1	0.51064	2.7	9.8	3.19503	2.15536
Muinebea	Carlow	-6.9626	52.7016	2.65547	2.58506	1.42045	2.29618	2.45568	4.52	948.6	2945	3.04	0.13948	16.03	3.2	7.1	0.12766	2.7	9.8	2.73827	3.91202
Myshall	Carlow	-6.7795	52.6881	2.27008	2.56878	1.42045	2.28039	2.43926	2.6415	913.2	292	3.04	0.20596	16.03	3.2	7.1	0.19149	2.7	9.8	2.66456	0.96622
Palatine	Carlow	-6.8625	52.8608	2.06937	2.79217	1.42045	2.49869	2.66445	0.97108	1536.2	276	3.04	0.0659	16.03	3.2	7.1	0.04255	2.7	9.8	1.80263	0.90987
Rathoe	Carlow	-6.797	52.7869	2.57938	2.719	1.42045	2.42693	2.5908	3.7397	1296	296	3.04	0.13534	16.03	3.2	7.1	0.02128	2.7	9.8	2.01334	0.97983
Rathvilly	Carlow	-6.6978	52.881	2.61318	2.65656	1.42045	2.36587	2.52787	4.09517	1120.6	1074	3.04	0.23008	16.03	3.2	7.1	0.53191	2.7	9.8	3.38756	2.26861
Tinryland	Carlow	-6.8982	52.7998	2.57225	2.46716	1.42045	2.18192	2.33632	4.45538	720	338	3.04	0.04359	16.03	3.2	7.1	0	2.7	9.8	3.4569	1.11252
Tullow	Carlow	-6.7356	52.8033	2.26382	2.76047	1.42045	2.46751	2.63249	2.03818	1426.8	5138	3.04	0.19043	16.03	3.2	7.1	1.68085	2.7	9.8	3.68914	3.83389
Arvagh	Cavan	-7.5826	53.923	2.26391	2.63969	1.42045	2.34898	2.51041	2.4	1076.3	419	3.04	0.23975	16.03	3.2	7.1	0.04255	2.7	10.775	2.04108	1.32734
Bailieboro	Cavan	-6.968	53.9118	3.93831	2.69726	1.42045	2.4053	2.56853	10.6	1230.7	2974	3.04	0.39444	16.03	3.2	7.1	0.82979	2.7	10.775	4.03442	3.28713
Ballingh	Cavan	-7.408	53.9319	3.0634	2.63893	1.42045	2.34811	2.50952	6.4	1074.3	1012	3.04	0.08279	16.03	3.2	7.1	0.44681	2.7	10.775	3.48743	2.20915
Ballyconn	Cavan	-7.5776	54.1178	2.68135	2.75325	1.42045	2.46013	2.62492	4.148	1402	1422	3.04	0.25561	16.03	3.2	7.1	0.95745	2.7	10.775	3.7457	2.54929
Ballyhaise	Cavan	-7.3205	54.0428	1.87876	2.75295	1.42045	2.4598	2.62459	0.136	1401.2	748	3.04	0.05881	16.03	3.2	7.1	0.04255	2.7	10.775	0.90897	1.90687
Ballyjames	Cavan	-7.2018	53.8635	2.40764	2.80134	1.42045	2.50739	2.67335	2.63568	1567.9	2917	3.04	0.20052	16.03	3.2	7.1	0.6383	2.7	10.775	3.42123	3.26778
Belturbet	Cavan	-7.4443	54.1004	3.06058	2.80016	1.42045	2.50618	2.67211	5.904	1563.5	1610	3.04	0.1394	16.03	3.2	7.1	1.12766	2.7	10.775	4.01742	3.68888
Blacklion	Cavan	-7.8738	54.2909	2.70561	2.59015	1.42045	2.30089	2.46054	4.756	958.7	175	3.04	0.59863	16.03	3.2	7.1	0.14894	2.7	10.775	2.83192	0.45426
Butlersbri	Cavan	-7.3775	54.0456	3.43522	2.57634	1.42045	2.28707	2.44622	8.446	928.2	271	3.04	0.05646	16.03	3.2	7.1	0.38298	2.7	10.775	3.47747	0.89159
Cavan	Cavan	-7.3538	53.9944	1.94089	2.5855	1.42045	2.29556	2.45505	0.94792	947.4	11741	3.04	0	16.03	3.2	7.1	4.34043	2.7	10.775	3.59084	4.30407
Cootehill	Cavan	-7.0776	54.0735	3.51084	2.56591	1.42045	2.27706	2.43579	8.855	905.8	1856	3.04	0.28725	16.03	3.2	7.1	0.51064	2.7	10.775	3.61233	3.21888
Killeshand	Cavan	-7.5297	54.0144	1.77964	2.55239	1.42045	2.26389	2.42209	0.2394	877.7	248	3.04	0.17705	16.03	3.2	7.1	0.06383	2.7	10.775	1.12749	1.00357
Kilnaleck	Cavan	-7.3198	53.8615	2.56883	2.58727	1.42045	2.2978	2.45736	4.08126	952.3	481	3.04	0.13708	16.03	3.2	7.1	0.04255	2.7	10.775	1.12749	1.00357
Kingscourt	Cavan	-6.8098	53.9049	2.19702	2.82154	1.42045	2.5274	2.69381	1.52188	1643.2	2955	3.04	0.55128	16.03	3.2	7.1	0.91489	2.7	10.775	3.59084	4.30407
Loch Gow	Cavan	-7.533	53.8733	2.48488	2.37197	1.42045	2.08983	2.23892	4.30325	574.5	168	3.04	0.21633	16.03	3.2	7.1	0.02128	2.7	10.775	1.12749	1.00357
Mullagh	Cavan	-6.9446	53.8113	2.70213	2.92929	1.42045	2.63356	2.80209	3.72525	2109.7	1651	3.04	0.44825	16.03	3.2	7.1	1.93617	2.7	10.775	3.59084	4.30407
Shercock	Cavan	-6.8955	53.9946	2.55969	2.74571	1.42045	2.45288	2.61746	3.56197	1377.8	574	3.04	0.45828	16.03	3.2	7.1	0.10638	2.7	10.775	3.59084	4.30407
Swanlinbar	Cavan	-7.7074	54.1016	2.76545	2.68760	1.42045	2.49570	2.56903	3.10823	1732.1	777	3.04	0.44054	16.03	3.2	7.1	0	2.7	10.775	3.59084	4.30407

Fig. 33

Testing phase

Once I had the real dataset complete, I did my last testing of the model. On the first try there were a few errors. I went back and checked the dataset for any cells with strange inputs. One of the problems seemed to be caused by the fadas on some letters in a few towns. When it's converted into a DataFrame in the program, I think the fadas caused some problems in the program. I went back and changed any cells that might've caused problems.

After a few tries, the CSV file finally worked with the code. It worked very well like previously when I used a smaller fake dataset. I can enter a town in Ireland and once it's found in the dataset, the GAN will start its iteration process.

Results and graphs produced using the final version can be view in the results section. The link to the GitHub Repository with all of the code is located in the Appendices.

Results

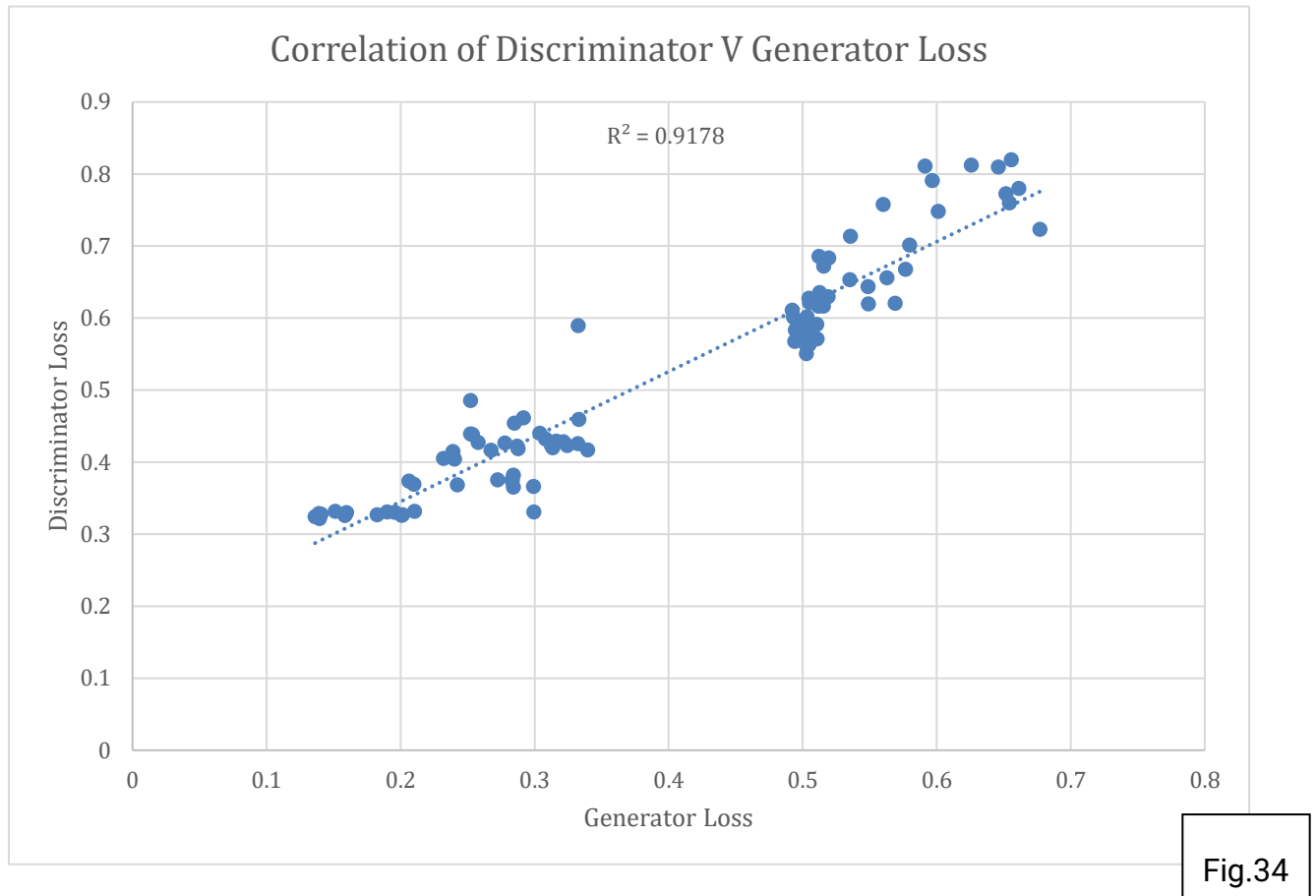


Fig.34

The correlation co-efficient between the Discriminator loss and the Generator loss is a very high $r = 0.958043$. This indicates that both the discriminator and the generator are matching each other. It should be noted from the raw data that the generator outperforms the discriminator as indeed it should do in a GAN. On the very few high Epoch runs I performed where the *epoch*=1000, the pace at which the loss in discrimination decreases as the GAN takes longer to detect false data.

One interest anomaly occurs at ~50 epochs when using *batch_size* >100. There is a pronounced decrease in the loss of both the discriminator and the generator. One hypothesis is that as the generator significantly improves its ability

to generate data that strongly mimics real world data it could lead to a sudden drop in both losses.

Using the large data base which I have constructed from the CSO website I was able to generate Cost-of-Living comparisons on the 5 key economic indicators of Housing, Transport, Education, Food and Energy.

Here you can see an example for a house in a new build in Dublin 6 (Fig. 35)

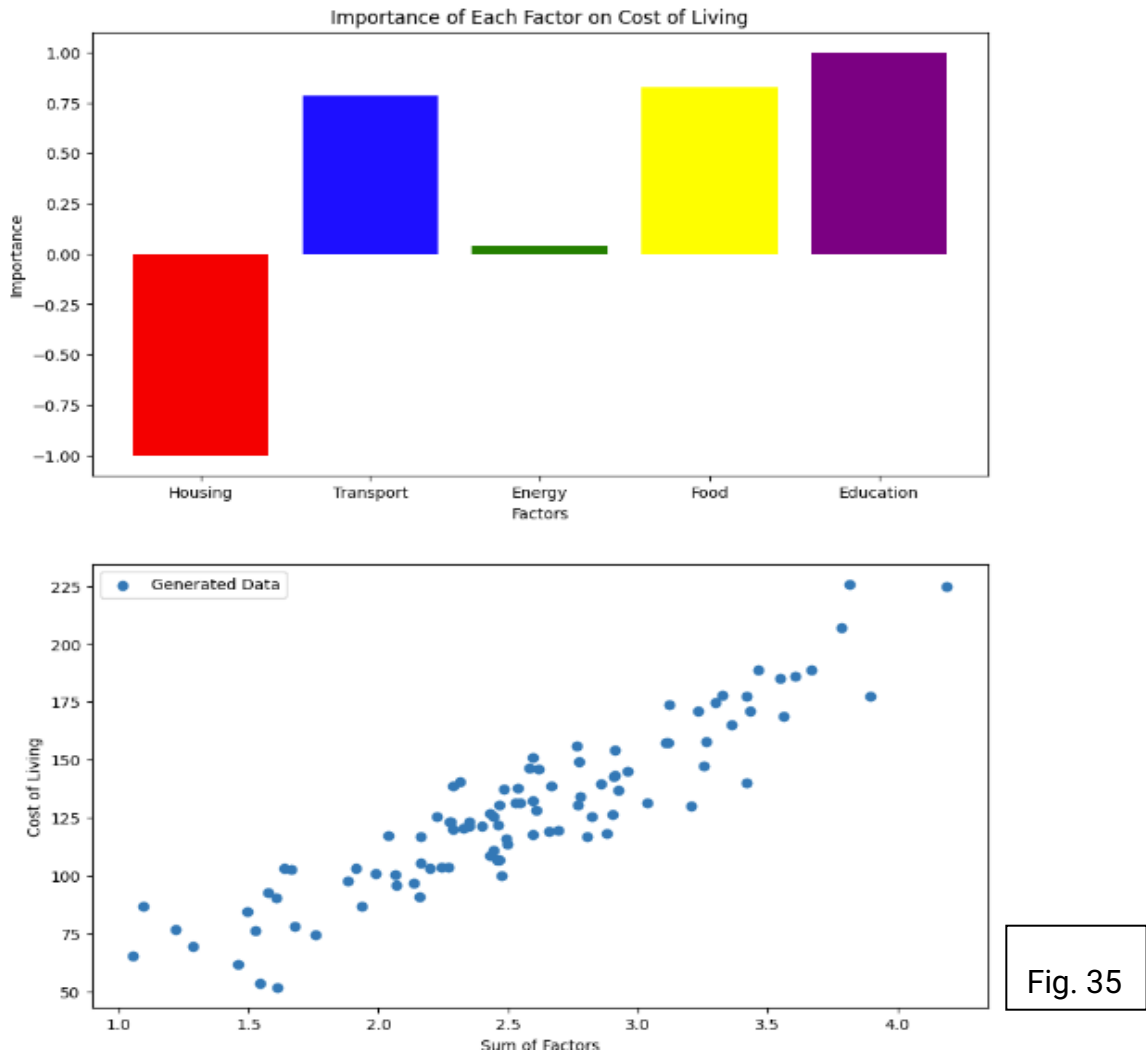


Fig. 35

As you can see in the bar chart, housing is not a major issue but Transport, food and especially Education. The scatterplot shows the sum of factors against the cost of living. According to the graph, this means the higher the sum of all factors, the higher the cost of living, which is accurate in the real world.

As my data is continuous data, I picked a non-parametric test. I did this because I was not assuming any underlying distribution pattern and from several studies have shown that this statistical testing is frequently used to access accuracy and fidelity of GAN vs real world data (Valle, 2018).

The Kolmogorov-Smirnov (KS) Test was the test I chose because it is often a good choice when comparing real inflation-style continuous data and the GAN-generated data. It works by comparing the cumulative distributions of the two datasets to determine if they could be drawn from the same distribution.

To start I wrote a simple python program using SciPy (Fig. 36), which is an incredibly powerful Python library that can access multiple statistical packages.

```
import scipy.stats as stats
import pandas as pd
file_name='bigfile.csv'
TR='TransportReal' #Names of the columns I will compare AC
TG= 'TransportGAN'
data=pd.read_csv(file_name,usecols=[TR,TG])
# This is where I perform the KS Test AC
ks_statistic, p_value = stats.ks_2samp(data[TR], data[TG])#Using the stats package to compare AC

print(f"KS Statistic: {ks_statistic}")
print(f"P-Value: {p_value}")

if p_value < 0.05:#A simple condition to tell me the results
    print("There is a significant difference between the distributions.")
else:
    print("No significant difference between the distributions.")
```

Fig. 36

On the next page you will find tables of results after completing the KS test on different numbers of epochs.

Transport			
Epoch	KS Value	p-value	Interpretation
1	0.19	0.00146	Low
3	0.18	0.003067	Low
5	0.16	0.0119	low
10	0.124	0.097	High
20	0.101	0.2699	High
25	0.023	0.284219	High
50	0.095	0.3274	High
100	0.0876	0.4267	High
1000	0.0543	0.92967	High

Education			
Epoch	KS Value	p-value	Interpretation
1	0.15675	0.001219	Low
3	0.1485	0.002561	Low
5	0.132	0.009937	low
10	0.1023	0.080995	High
20	0.083325	0.225367	High
25	0.018975	0.237323	High
50	0.078375	0.273379	High
100	0.07227	0.356295	High
1000	0.044798	0.776274	High

Energy			
Epoch	KS Value	p-value	Interpretation
1	0.16208	0.001261	Low
3	0.153549	0.002648	Low
5	0.136488	0.010274	low
10	0.105778	0.083749	High
20	0.086158	0.233029	High
25	0.01962	0.245392	High
50	0.08104	0.282674	High
100	0.074727	0.368409	High
1000	0.046321	0.802668	High

Food			
Epoch	KS Value	p-value	Interpretation
1	0.201843	0.001551	Low
3	0.191219	0.003258	Low
5	0.169973	0.012642	low
10	0.131729	0.103046	High
20	0.107295	0.286723	High
25	0.024434	0.301934	High
50	0.100921	0.347807	High
100	0.09306	0.453296	High
1000	0.057684	0.987616	High

Housing			
Epoch	KS Value	p-value	Interpretation
1	0.198664	0.001527	Low
3	0.188208	0.003207	Low
5	0.167296	0.012443	low
10	0.129654	0.101423	High
20	0.105606	0.282207	High
25	0.024049	0.297179	High
50	0.099332	0.342329	High
100	0.091595	0.446158	High
1000	0.056776	0.972063	High

My Null Hypothesis is that the GAN data follows the same distribution and patterns as the real data from my five factors and I should reject the Null Hypothesis or fail to accept the alternate Hypothesis if the p values associated with the KS value is below 0.05. From my data I can see a big jump after just 10 epochs or cycles and another after 100 epochs with ties into the jump I saw when I plotted the loss values in my correlation relationship data.

Overall, I believe that I have achieved a lot of what I set myself to in the beginning. I have built a GAN that accurately generates synthetic Irish cost of living data based on multiple economic indicators that affect the cost of living, and produces a heatmap based on the specified location. Additionally, I build a basic GUI for a user to interreact with. The KS test proves that after a certain number of epochs, the GAN generates cost of living data that is practically indistinguishable from real data.

Conclusions

Using GANs to generate cost of living data based on certain factors can offer many advantages:

1. **Addressing Imbalances:** In real-world datasets, there might be imbalances in the distribution of factors influencing cost of living. GANs can generate synthetic data to balance the representation of different factors, ensuring a more comprehensive understanding of the cost of living landscape.
2. **Data Augmentation:** GANs can be used to augment existing datasets by generating additional samples. This is especially useful when working with small datasets, enhancing the diversity and size of the available data for more robust model training.
3. **Scenario Testing:** Cost of living data generated by GANs can be used for scenario testing and planning. This allows policymakers, researchers, or businesses to simulate the impact of different factors on cost of living without relying solely on historical or observed data.
4. **Machine Learning Model Training:** GANs can generate realistic synthetic data that aids in the training of machine learning models. Models trained on a combination of real and synthetic data may generalize better to diverse real-world scenarios.
5. **Flexibility and Customization:** GANs provide flexibility in generating data based on specific factors. If there are particular factors (e.g., rent, fuel, education) influencing cost of living, the GAN can be trained to understand and replicate their impact, allowing for customisation based on the chosen factors.

Additionally, using a GAN to generate cost of living data can provide benefits to families and businesses. Individuals can use synthetic cost of living data to simulate and plan for different financial scenarios. This helps in budgeting,

understanding the impact of various expenses, and making informed decisions about where to live based on cost considerations. Policymakers and government officials can leverage synthetic cost of living data to assess the potential impact of policy changes on residents. This includes evaluating the consequences of tax adjustments, subsidy programs, or changes in public services.

While there are many advantages to using GANs for this, there are some disadvantages:

1. **Difficulty in Validation:** It can be challenging to validate the quality and accuracy of the generated data. Without careful evaluation, the data may not faithfully represent the true distribution of factors influencing living costs.
2. **Uncertainty in Factor Relationships:** GANs may not accurately capture the complex relationships between different factors influencing the cost of living. The generated data might not align with the true interactions and dependencies present in real-world scenarios.
3. **Model Collapse:** Some GANs may suffer from model collapse, where the generator produces limited variations of data, leading to a lack of diversity in the synthetic dataset. This can result in a biased representation of the cost of living landscape.
4. **Computational Intensity:** Training and generating data using GANs can be computationally intensive, requiring substantial computing resources and time. This can be a limitation, especially for organisations with limited computational infrastructure.

My Improvements

There are several issues that, given time, would make my model possibly more accurate.

There are a few features I wanted to include in my project but due to short timing, I wasn't able to fully complete those features. On the heatmap, I wanted to create a program so the heat colour would correspond with the most and least factors that affect the area. There were a few problems with that code which I couldn't figure out in time. The code I attempted to use for this feature is added in quotations on the last version of my project.

With using the Flask GUI, which means the interface is run as a local host on my computer. This is not necessarily the best option as I cannot easily get people to beta test my project unless they run the whole program on their device, which is not efficient. Beta testing is really important when creating any GUI or front end.

The GAN currently takes in 5 main factors. Factors like housing and transport are very broad terms. If I put this into further development, it might be good to train the model on more specific factors rather than just 5 broad ones.

With a lot more time and development, my project could improve significantly. GANs learn over time, and I believe that this project could be developed a lot more in the future.

Acknowledgments

I would like to acknowledge the help I received from:

My parents for always encouraging me in my exploration of Computer Science. They have always supported me and make sure I have everything I need.

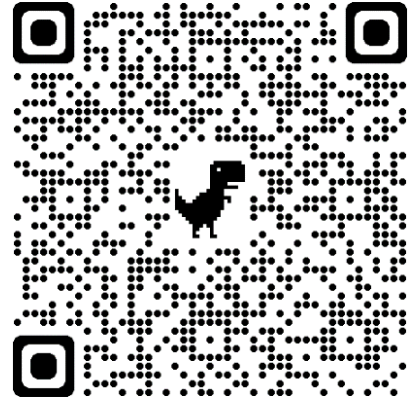
My teacher, Zita Murphy, for the help and support she has given me towards my projects.

My school for letting me apply to the BT Young Scientist Competition. I hope I get to represent the school many more times in the future.

The many YouTube content creators for posting free tutorials on how the GAN architecture works and how to create them.

Appendices - My Entire Script

Here you will find all my code for my GAN model. I decided that this section is the best place to include the code as it would become quite tedious and repetitive to comment on every line of code in my early sections. All of the final code and code from early iterations can be viewed on my GitHub Repository, as typing the code here would take up too many pages. The entire project directory and code can be viewed here with the QR code.



References

- Ahmed, Muhammad Nouman. 2022. *Calculator-Using-Flask*.
<https://github.com/Muhammad-Nouman-Ahmed/Calculator-using-Flask>.
- Brownlee, Jason. 2019 . *Machine Learning Mastery*. "What Are Generative Adversarial Networks (GANs)?" *Machine Learning Mastery*. 19 July . Accessed January 7, 2024. <https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/>.
- Competition, BT Young Scientist. 2023. "BT Young Scientist Past Projects." May.
<https://btyoungscientist.com/past-projects/>.
- Corona, Erika, and Filippo Eros Pani. 2013. ""A review of lean-kanban approaches in the software development."." *WSEAS transactions on information science and applications* 10,. 1-13.
- Creswell, Antonia, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. 2018. ""Generative adversarial networks: An overview."." *IEEE signal processing magazine* 35, no. 1: 53-65.
- CSO. 2024. *Consumer Price index*.
[https://www.cso.ie/en/methods/surveybackgroundnotes/consumerpriceindex/#:~:text=Definition,Consumer%20Price%20Index%20\(CPI\)](https://www.cso.ie/en/methods/surveybackgroundnotes/consumerpriceindex/#:~:text=Definition,Consumer%20Price%20Index%20(CPI)).
- Disch, Wendy, and Rachel Slaymaker. 2023. "Housing affordability: Ireland in a cross-country context." *Economic and Social Research Institute Research Series*, 164.
- Fowler, Martin, and Jim Highsmith. 2001. ""The agile manifesto."." *Software development* 9 8: 28-35.
- Loeber, Patrick. 2021. *Python Flask Beginner Tutorial - Todo App - Crash Course*.
<https://youtu.be/yKHJsLUENI0?si=1QD3M1uN5Mg9-T7v>.

- Lydon, Reamonn. 2022. "Cost of Living Symposium, Inflation and Monetary Policy – What Next?" *Journal of the Statistical and Social Inquiry Society of Ireland* 33.
- . 2022. "Household characteristics, Irish Inflation and the Cost of Living." *Central Bank of Ireland*. February. Accessed January 7, 2024.
<https://www.rte.ie/documents/news/2022/02/centralbank.pdf>.
- NeuralNine. 2022. *Tkinter Beginner Course - Python GUI Development*.
<https://www.youtube.com/watch?v=ibf5cx221hk>.
- Rahul, Roy. 2023. *GeeksForGeeks*. 23 November. Accessed January 7, 2024.
<https://www.geeksforgeeks.org/generative-adversarial-network-gan/>.
- Wang, Su. 2017. "Generative Adversarial Networks A Gentle Introduction." *Research Gate*. April. Accessed January 7, 2024.
https://www.researchgate.net/publication/316382604_Generative_Adversarial_Networks_GAN_A_Gentle_Introduction_UPDATED.

Comments