All Actuaries Summit
New age, new actuary

11-13 June 2025, Sydney

Yes we GAN: Adapting generative models for predictive modelling with multivariate response

Prepared by Hugh Miller, Justin Sik Kwok Wong and Callum Sleigh

Presented to the Actuaries Institute
2025 All-Actuaries Summit
11-13 June 2025

This paper uses unit record data from Household, Income and Labour Dynamics in Australia Survey, HILDA. HILDA is conducted by the Australian Government Department of Social Services (DSS). The findings and views reported in this paper, however, are those of the author[s] and should not be attributed to the Australian Government, DSS, or any of DSS' contractors or partners. DOI: 10.26193/R4IN30.

Actuaries Institute
ABN 69 000 423 656
Level 2, 50 Carrington Street, Sydney NSW 2000
P +61 (0) 2 9239 6100 | actuaries.asn.au

Abstract

Much recent attention has been given to generative AI models, but most of this is devoted to 'human-centric' outputs such as text or image generation. Surprisingly little research has been focused on other types of generative tasks on structured data, such as prediction and simulation of multivariate responses. Our paper aims to rectify this.

Multivariate prediction and simulation is a common task with a wide variety of applications. For instance, actuarial microsimulation projects make predictions for people over multiple time periods, simulating a wide set of outcomes at each step. Such simulation is complex, since the value of one outcome will correlate with others at the same timepoint, meaning that intricate conditional sub-models are typically required.

We propose some novel forms of Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) designed to make accurate multivariate predictions that preserve these interdependencies across outcomes. A single AI model can then replace dozens of sub-models. Such a setup has the potential to greatly simplify modelling, as well as improve speed by leveraging modern AI computational hardware.

Results suggest good performance of these models, on both simulated and real-life datasets. Stability and speed were slightly better for the VAE model, and most accurate generation varied between the two models. Moreover, once the networks are fit, simulations can be generated very quickly; millions of rows per minute. We include a discussion of our experimentation with model structures and challenges encountered in constructing the models. Overall, the approach is promising, with simulated data that is statistically similar to real data in distribution, including capturing complex interactions between variables.

Keywords: Deep leaning, AI, multivariate prediction, generative adversarial networks, variational autoencoders, structured data

1 Introduction

1.1 Modern AI

To say that there is significant recent interest in AI models is an understatement. The paper by Vaswani et al. (2017) introducing transformer models into neural networks has generated over 170,000 citations within eight years. Models based on neural network designs have grown in complexity, data input, and training time. We use the term 'AI models' relatively loosely in this paper but intend it to mean any relatively complex network-based model.

Much of this attention (rightly) focuses on areas where computer models have traditionally done poorly, and AI models have taken rapid strides – such as audio, vision and natural language data. This progress covers rapid developments for both the discriminative applications (e.g. identifying the number of cars in an image) of AI models, as well as the generative applications (e.g. generating a picture of a traffic jam).

Relatively less attention has been given to tabular data applications – where data is structured in rows and variables. This is partly because existing tools for modelling tabular data are good (and are often better than neural network structures), but also because AI models tend to work best when dealing with a large number of variables with similar behaviours (e.g. a million pixels, all working on a RGB scale), rather than the heterogeneity we often see in tabular data. But there are a range of applications to tabular data where modern AI models offer strong possibilities and it is useful to understand both the potential and the pitfalls of these applications.

One area we explore is the situation of multivariate response, with interactions between response variables. By this we mean:

- There are at least two outcomes of interest that need modelling (for example, counting both the number of cars and the number of red cars in an image)
- There are dependencies between the two (for example, the number of red cars cannot be greater than the number of cars).

In standard predictive modelling, this would require sub-models to capture separate outputs and appropriate design to preserve the interactions. But modern generative methods offer the possibility of simultaneously generating multiple responses, with interdependence. An obvious example is modern image generators, where each pixel colour is an output. These are dependent on surrounding pixels, and they are generative (can produce multiple different pictures from similar prompts). Microsimulation models (introduced below) are one area with significant opportunity to benefit from more automated approaches that can leverage these properties.

Borisov et al. (2022) provides a recent survey on deep learning for tabular data including many references, including a range of open challenges – we do not attempt to replicate this broad perspective of existing work here. Our approaches perhaps most closely align with approaches to variable imputation, in that we effectively recast generative prediction as a type of vector completion. Relevant references include Yoon et al. (2018), Nazabal et al. (2020), Yoon et al. (2020), Telyatnikov and Scardapane (2023) and Sun et al. (2023).

1.2 Microsimulation models

Microsimulations are models that project at a very granular level – typically at the level of an individual person. We give some examples of 'actuarial' microsimulations to give a flavour:

1. **Australian welfare –** The Priority Investment Approach (PIA) model, managed by the Commonwealth Department of Social Services. This tracks the long-term welfare costs of Australians based on currently known factors such as demographics, benefit history and family structure.[1]
2. **Veterans affairs –** The Department of Veterans' Affairs Priority Investment Approach – Veterans, which tracks conditions and service needs for people receiving support from the DVA.[2] (for example, here).
3. **New Zealand adult population –** The New Zealand Ministry of Social Development manages a social outcomes model for adults. [3]
4. **NSW human services –** The NSW investment approach for human services, which projects outcomes and service use over 20 years for vulnerable young people and their families.[4]
5. **Disability support –** The National Disability Insurance Scheme (NDIS) has developed a microsimulation model to improve the accuracy of NDIS projections.[5]
6. **NZ child protection –** Oranga Tamariki built a Children's Wellbeing model, which tracks movements in wellbeing along with government service usage.[6]

We refer to these as 'actuarial', primarily because actuaries have been heavily involved in their construction. In practice these will strongly resemble microsimulation models built more broadly across economics, computer science and other domains. Terminology can also vary – sometimes microsimulations are referred to as 'agent-based models' or even 'digital twins' (Rasheed et al., 2020,

[1] https://www.dss.gov.au/australias-approach-social-investment

[2] https://www.dva.gov.au/newsroom/media-centre/departmental-media-releases/dva-wins-award-business-innovation

[3] A recent technical report is at https://www.msd.govt.nz/documents/about-msd-and-our-work/publications-resources/research/benefit-system/2023-social-outcomes-modelling-technical-report.pdf

[4] https://www.nsw.gov.au/community-services/investment-approach-for-human-services

[5] See Section 3.5 of the 2023-24 AFSR, https://www.ndis.gov.au/media/7358/download?attachment

[6] https://orangatamariki.govt.nz/assets/Uploads/About-us/Research/Research-seminars/February-2020/A-Health-Case-Study-using-the-Childrens-Wellbeing-Model.pdf

provides a broad perspective on progress and opportunities of digital twin models). To give a sense of this broader environment:

- Research teams at the Australian National University have built and maintained microsimulations related to tax and transfer policy over many years.[7]
- Agent-based models were used to model the transmission of COVID-19, and the impact of policy settings during the pandemic (Chang et al., 2020).
- Agent-based models are used for modelling systems such as court systems of hospital emergency departments (Moyaux et al., 2023).

Such models have a range of potential applications. They can be used for system management – providing a long-term view of trends and the impact of policy intervention. They can also be used for scenario modelling and testing the impact of changes in a complex system. They are also valuable for monitoring and evaluation – understanding the impact of different interventions for cohorts within the whole, and converting results to a longer-term impact. This versatility helps explain their popularity.

While such models are common, we comment:

- **Building and maintaining microsimulations can be expensive –** The process is time-consuming. For example, the PIA model cost $2m per year to update and run.[8] Much of the cost relates to the complexity associated with many model components. Separate models are required to model the evolution of benefit status, household status, payment levels, partial capacity to work status etc. These must be built and then validated to check they interact together.
- **Best practice is still evolving –** We have seen a range of approaches to build and validate models. Often these too are impeded by the time required (e.g. the cost of refitting on older data to run a proper back-test).
- **Construction is often quite bespoke –** This is natural given the variety of underpinning data and context for building the models. Such bespoke construction can limit transferability.
- **Models are typically CPU-bound, running on traditional statistical code –** For many projects this means long projection runs (often measured in tens of hours) to apply to the full population. Modern AI approaches recognise the value of GPU-based computation, when processes can be neatly split into pieces (such as running the projection on different cohorts).

Our current research seeks to overcome these limitations while retaining good predictive performance. This would bring clear benefits. Building models would be significantly cheaper, saving (often the taxpayer) money. It would also encourage more organisations to build and engage with such models, allowing more evidence-based policy.

1.3 Structure of this paper

The remainder of the paper is structured as follows:

- Section 2 covers the formal problem definition, including notation
- Section 3 describes our approach (model setup) and datasets
- Section 4 provides results
- Section 5 provides discussion
- The Appendix provides further tables and plots of results, to reduce the length of the paper's main sections.

[7] PolicyMod is a recent version of this, https://csrm.cass.anu.edu.au/research/publications/policymod-microsimulation-model-australian-tax-and-transfer-system-december

[8] https://www.tenders.gov.au/Cn/Show/c1a94502-3ecb-43d0-ad25-01aa900faad1

Our paper (particularly Section 3) assumes a basic knowledge of neural network construction (e.g. defining layers of hidden neurons) and how networks are traditionally trained (e.g. backpropagation). Many introductory discussions are available, such as Chapter 11 of Hastie et al. (2009). We aim to make sections 4 and 5 readable for those less interested in the detail of formulae and model construction.

2 Problem description

For convenience we will refer to our unit of modelling as a 'person'. Suppose our microsimulation has p 'dynamic' variables $X_{1,t}, X_{2,t}, \dots, X_{p,t}$ that are evolving non-deterministically for each person that we observe at time t. Our task is to estimate these variables at the next timestep $t+1$. Assume also we have 'static' variables $S_1, S_2, \dots, S_q$ that are constant for an individual (e.g. sex) or evolve deterministically (e.g. age). Later, we also use Z_j variables to donate random noise (in the case of GAN models) and latent variables (in the case of VAE models).

We can express our task as estimating the following distribution f:

$$f\left(X_{1,t+1}, X_{2,t+1}, \dots, X_{p,t+1} \middle| X_{1,t}, X_{2,t}, \dots, X_{p,t},\ S_1, S_2, \dots, S_q\right)$$

The key complicating factor is that these variables are not independent; for example if a person enters hospital, that changes their probability of also entering welfare. This non-independence creates a challenge.

The default approach for many existing microsimulation models is to treat this as a chained conditional probability problem. First estimate the distribution for $X_{1,t+1}$

$$f\left(X_{1,t+1} \middle| X_{1,t}, X_{2,t}, \dots, X_{p,t},\ S_1, S_2, \dots, S_q\right)$$

Then estimate $X_{2,t+1}$ conditional on $X_{1,t+1}$ (as well as the other values at time t):

$$f\left(X_{2,t+1} \middle| X_{1,t}, X_{2,t}, \dots, X_{p,t},\ S_1, S_2, \dots, S_q, X_{1,t+1}\right)$$

This can be continued, estimating each variable in sequence. The creates the long list of sub-models described in the previous section. This structure can be beneficial (a lot of care can go into the construction of components, and expertise and subject matter knowledge embedded in various parts of the design), but is labour intensive.

A second complicating factor is that we are genuinely interested in **distributions**, not just averages for our vector. Since we are simulating, we need the projection to take plausible different values. This is more challenging than most machine learning approaches that focus on a particular metric (such as the mean) without regard for distribution.

3 Approach

3.1 Generative Adversarial Networks

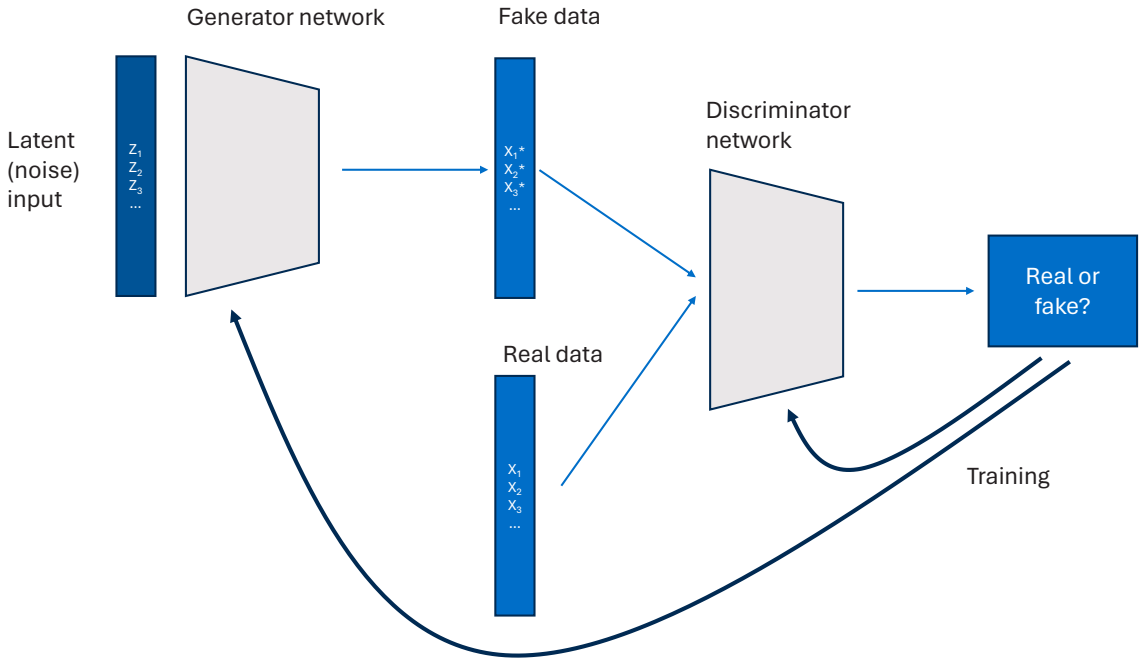
Generative Adversarial Networks (GANs) were developed in 2014 by Ian Goodfellow and colleagues (Goodfellow et al., 2020), with the original idea arising from a bar discussion.[9] They have proven a popular way to train networks with generative applications. The underlying aim is to generate plausible synthetic versions of real data – that is, a generative task. A GAN does this by training two networks (see Figure 3.1) that effectively compete with each other:

- The *generator* converts some random noise (often the same dimension as the target vector) into a fake data vector which is intended to look like real data.
- The *discriminator* network then accepts data (fake or real) and attempts to determine whether the data is real or synthetic.

[9] https://www.technologyreview.com/2018/02/21/145289/the-ganfather-the-man-whos-given-machines-the-gift-of-imagination/

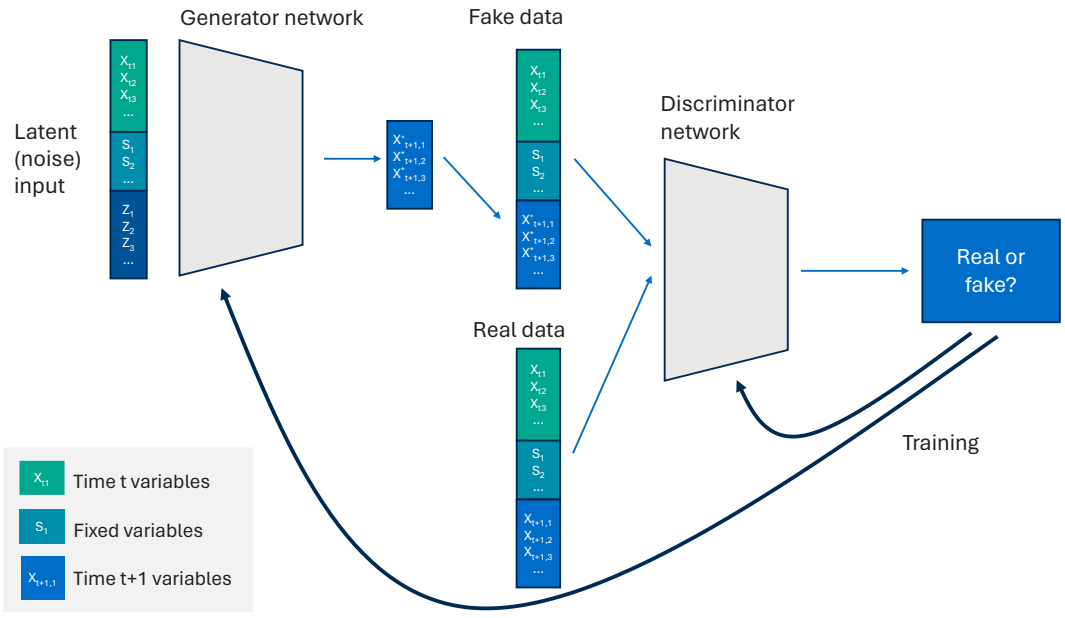
The performance of the discriminator model is then used to train the two networks; the generator attempts to trick the discriminator (lowering accuracy of the final binary choice), and discriminator attempts to improve the accuracy.

Figure 3.1 – Traditional structure of a GAN



We tailor this setup in a few ways to make it suitable for our microsimulation design, as shown in Figure 3.2.

Figure 3.2 – Modified GAN structure for a microsimulation setup



The main features of our revised structure are:

- The generator network accepts time t variables (dynamic and static) in addition to the noise variables Z_j. They thus inform the generation, but are not themselves generated

- We add the time t variables (dynamic and static) to the generated samples of $t+1$ variables. The discriminator can then judge the time $t+1$ variables generated with reference to the time t variables. In this way the discriminator can check the compatibility between time t and $t+1$ variables

The generator network, once trained, can be used for generation in the simulation. Each time step we can take the current information with respect to a person, plus some randomly generated noise, to generate plausible $t+1$ data.

For the formal loss functions, there are two loss functions applicable to the training based on cross-entropy (equivalent to traditional loss functions used elsewhere, such as logistic regression), with the 'response' corresponding to an indicator for whether the observation was real. For the **discriminator**, we first calculate the discriminator's probability of observations being real, p_{real}, for both real and generated (fake) data and calculate the loss to minimise as:

$$L_{disc} = -\left(\sum_{i,real\ obs} \log\left(p_{real}\right) + \sum_{i,fake\ obs} \log\left(1 - p_{real}\right) \right)$$

For the **generator**, we focus on the discriminator's predictions for the fake data and try to encourage high probability with the loss function:

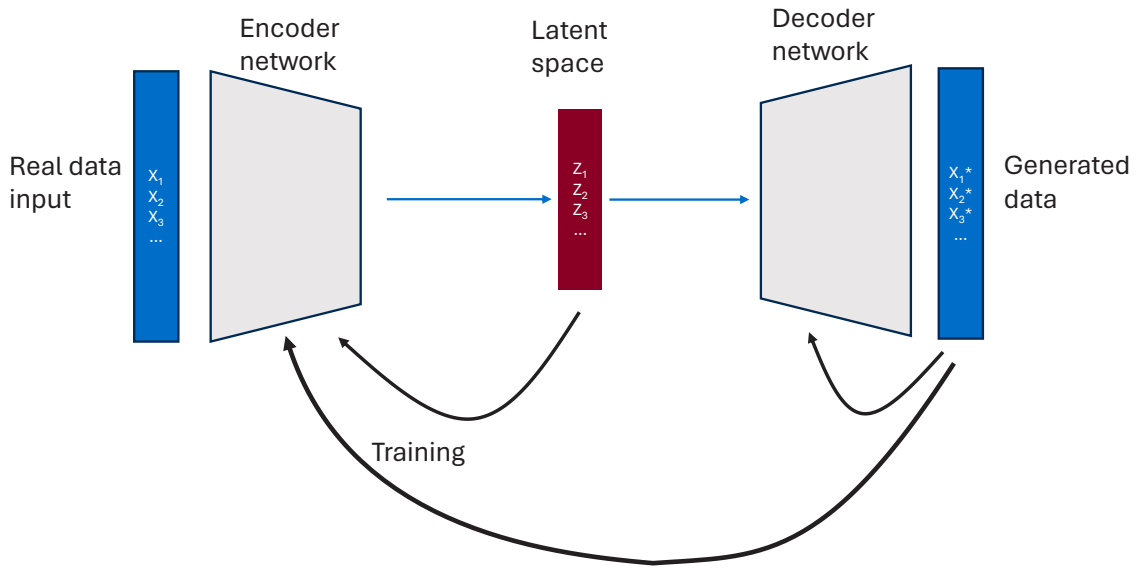
$$L_{gen} = -\left(\sum_{i,fake\ obs} \log\left(p_{real}\right) \right)$$

Both models can be improved through backwards propagation of the loss functions on batches of data, as is standard for neural network models.

3.2 VAEs

Variational Autoencoders (VAEs) represent another popular type of neural network structure, first proposed by Kingma & Welling (2013). As the name suggests, they are part of a broader family of autoencoder models that seek to understand data structures by encoding them into (usually lower-dimensional) spaces. Their standard design is shown in Figure 3.3.

Figure 3.3 – Traditional VAE structure



An encoder network converts input points into a latent (typically lower-dimensional) space. Importantly this space is a distribution (each Z_j is most commonly Gaussian described by its mean and standard

deviation μ_j and σ_j respectively) rather than specific numbers. The lower-dimensional space is intended to capture the essence of the data, and variable relationships. The decoder network converts a selection (randomly drawn selections from the latent space) from that space back into the data space, creating generated values. Training the networks relies on a combined loss function – examining both the difference (error) between real and generated data, as well as the distance of the latent space from a simple Gaussian distribution.

$$Loss = Error(x, x^*) + \beta D_{KL}(Z, \mathcal{N}(0,1))$$

The first term, the reconstruction error, seeks to make generated values similar to the input, whereas the second term is the Kullback–Leibler (KL) divergence between the distributions created for the latent space and the standard normal distribution. The KL term ensures a distributional smoothness, which skews the model away from overfitting. The β term is selected and controls the relative balance between the two components. The reconstruction error function for the first term is most commonly squared error. If f and g are the encoder and decoder functions respectively, this becomes (with P the number of dimensions of x_{ij}, N the number of observations being considered, and g_j the jth component of the decoder output vector):

$$Error(x, x^*) = Error\left(x, g(f(x))\right) = \frac{1}{NP}\sum_i \sum_j \left[x_{ij} - g_j(f(x_i))\right]^2$$

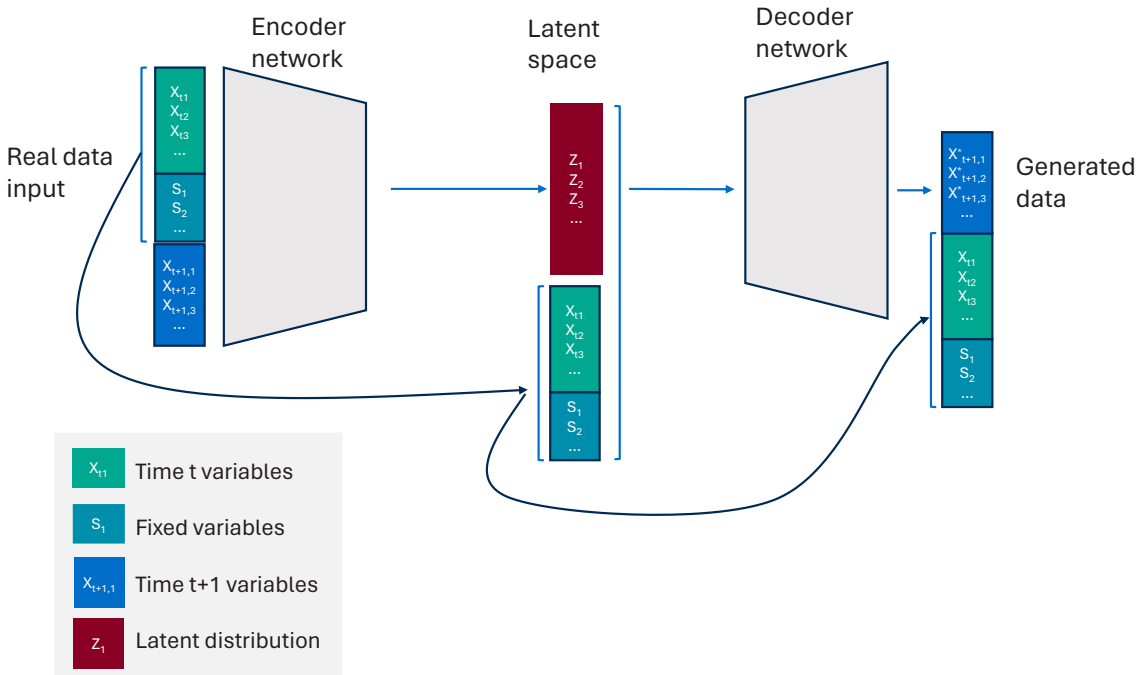
The KL divergence is calculated as:

$$-\frac{1}{2}\sum_j \left(1 + \log(\sigma_j^2) + \mu_j^2 + \sigma_j^2\right)$$

The underlying theory for VAEs, and the derivation of loss and fitting methods, rely on relatively deep Bayesian theory. Interested readers are referred to Kingma and Welling (2019) – our paper focuses on the modification and results of the VAE use.

As with GANs, it is possible to modify the model structure to make it amenable to our problem, illustrated in Figure 3.4. In this case the encoder network takes all the data (time t, time $t+1$ and fixed variables), but the latent space is intended to only capture the nature of the time $t+1$ variables. We then re-supply the time t and fixed variables as inputs to the decoder network, so that it seeks to generate new data conditional on these inputs. The final generated data again has a copy of the time t and fixed variables to complete the generation.

Figure 3.4 – Modified VAE structure for a microsimulation setup



3.3 Extensions and variations

A number of steps are required as part of the implementation of a GAN or VAE, covered below.

Standardisation of continuous variables

Continuous variables can work on very different scales, which can be a challenge to network models. Further, many continuous variables often have very skewed distributions with extreme outliers. This may be challenging to model as such outliers may have undue leverage (depending on the loss function used) and thus cause a poor fit. Extremely large values may also lead to the exploding or vanishing gradient problem, which can cause neural networks to collapse. Standardisation generally improves model fits and convergence.

We:

- Standardise all continuous variables by subtracting the mean and dividing by the standard deviation.
- Exclude a small number of outliers in our real-life dataset from the calculation of mean and standard deviation in our real-life example (but keep the outliers in the data – we just ignore them when choosing scaling parameters). This ensures that the bulk of the distribution is not too squashed around zero.

There are alternative approaches to transformation (for example, using cumulative density functions to convert continuous variables to well-behaved distributions), that we have not pursued in this paper.

Binary and categorical variables

Most neural network designs assume all variables are numeric.

General tabular datasets (and our real-life example) will have a variety of distributions, including categorical variables. For categorical variables we apply one-hot encoding (a variable with K categories are converted to K 0-1 variables, one for each level), as is often done for statistical modelling. We treat binary (or 'flag') variables as categorical variables (with two response categories), so these variables are also one-hot encoded, in this case into two 0-1 variables.

For the **GAN model**, we use the Gumbel-Softmax distribution (see Maddison et al., 2016 and Jang et al., 2016) on each group of one-hot encoded variables as a final layer of generation. This preserves the required differentiability and randomness to ensure that model optimisation (via backpropagation) is still possible. If π_k denote the outputs corresponding to each of the levels of the one-hot encoded variable (analogous to the linear predictor in multinomial regression), then our last layer converts these to probabilities:

$$p_k = \frac{\exp([\log(\pi_k) + g_k]/\tau)}{\sum_{j=1}^{K} \exp([\log(\pi_j) + g_j]/\tau)} \text{ for } k = 1, \dots, K$$

Here the g_k are samples from the Gumbel(0,1) distribution and τ is the 'temperature'; a higher value will tend to encourage a more even spread across categories. The final generation for the variable is sampled from these probabilities. For the GAN we also applied temperature annealing, where by the temperature τ starts at a higher level (more randomness) and is allowed to cool over the model training.

For the **VAE mode**l we still use the Gumbel-Softmax distribution but applied somewhat differently. For defining the reconstruction error with the main loss function, we use the derived probabilities directly (rather than sampling from them), but still sample for the eventual generation stage.

We also modify the loss function so that it uses cross-entropy for the 0-1 variables arising from categorical variables rather than squared error. If P_1 and P_2 are the number of continuous and one-hot encoded variables respectively, the reconstruction error term for the VAE becomes:

$$Error(x, x^*) = \frac{1}{NP_1}\sum_i \sum_{j\ cts} \left[x_{ij} - g_j\left(f(x_i)\right)\right]^2 + \frac{\gamma}{NP_2}\sum_i \sum_{j\ one-hot} -x_{ij}\log\left(p_{ij}\right)$$

The γ term is a tuning parameter that balances the relative weight of the categorical and continuous components of the model.

Mixed distributions

Our real-life dataset also contains mixed distribution variables, which are continuous but have point masses at certain levels. In our case these are typically income variables – if a person is not employed then their weekly wages variable is zero (a point mass), otherwise will follow a continuous distribution.

Again there are various options to handle (including making no adjustment and hoping the model will correctly reflect the shape, perhaps with some rounding). Our approach is to add an additional binary variable that captures when the mixed distribution variable is equal to the point mass, and modelling the continuous distribution afterward. Under this setup there is a final check applied in the generation step where if the point mass binary is one, the continuous portion is forced to the value of the point mass (and the corresponding flag forced to one).

For the VAE, this means that a mixed distribution is encoded in both the continuous and categorical portions of the reconstruction error term. This could potentially overweight the variable's role in the model. We have not tested options for ameliorate this; one possibility is to introduce indicators so only one of the categorical or continuous loss portions would be active at a time.

Interdependence between individuals

In many microsimulations data will often show interdependence between individuals. For instance, people in the same household may move location, or see changes in household income, at the same time. This can in theory be modelled by adding interdependence between data rows.

We consider this beyond the scope of this paper, and do not make any explicit adjustment for interdependence between individuals. However, the modified network structures could in principle be used to extend to these cases.

For example, consider the household income variable, which is effectively shared across household members. Household income can be added as a dynamic variable (along with potentially other household variables such as household size) and then two versions of the model fit:

- The first model is built as usual and applied to the first member of a household, setting household income in time $t + 1$.
- The second model is trained with household income in time $t + 1$ as a static (already known) variable. This is applied to other household members, effectively conditioning on household income.

While this requires more models, they would share most of the same model structure and could even share training parameters – so there would be some efficiencies.

Missing variables and imputation

After data cleaning and preparation, no missing variables exist in our simulated and real-life datasets used. However, we note that missing variables may be addressed by:

- **Formally recognising missingness:**
  - For categorical variables, missing levels may be treated as another category
  - For continuous variables, missing values may be treated as a mixed distribution. All missing values are encoded as a separate value (e.g. the mean) and producing a flag to capture when

this variable is missing. Then, this variable may be treated as a mixed distribution with a point mass at the encoded missing value

- **Variable imputation –** We discussed in Section 1.1 that our model design bears some resemblance to missing variable imputation (where our 'missing' values are variables at time $t+1$). Formal imputation models can be built to run first, or the main models can be designed to be tolerant to missing values in input variables.

The first approach for continuous variables has the potential to bias distribution towards whatever point is selected for missing values; we suspect some form of imputation will produce better results.

Multiple time steps

Our model description focuses on estimating variables at time $t+1$ given status at time t. Generally this is then extended to longer timeframes iteratively.

There is an alternative approach to modelling $t+S$ variables from time t, provided the original longitudinal dataset has at least $S+1$ time steps. We can simply replace the $t+1$ variables in the model setup with $t+S$ and so skip to that time in one model pass. This is less useful in many cases (where the full time series is desired), but is a powerful way to test whether the iterative single steps are working well – we can generate variables at time $t+S$ both ways (via S single step and via one single step) and test whether the single step provides substantially better generation.

We test these variants and present results for $S=3$ in Section 4.

3.4 Assessing the performance of models

Some care is needing in diagnostics since we are testing multivariate distributional generation, rather than more standard mean prediction problems. We used a range of tools:

- Discriminator models comparing real and fake data – Our main diagnostic takes inspiration from the GAN itself. Using holdout data, we create a stacked dataset of generated and actual rows (variables being all the static, dynamic t and $t+1$ variables), with response variable whether the row is actual data. We then fit a predictive model to see how well it can separate the real and generated rows. For this, we used a gradient boosting machine (GBM) with trees A superior generation model will register as poorer predictive performance. We measure the overall fit of the GBM discriminator using the gains ratio[10]; therefore, a superior GAN or VAE will achieve a *lower* gains ratio for the GBM discriminator model.
- Other distributional checks We also assess some simple metrics, including:
  - One-way histograms, comparing generated distribution to actual
  - Two-way heatmaps, comparing generated distribution to actual
  - Overall goodness-of-fit statistics, such as the Kolmogorov-Smirnov statistic for continuous variables and total variation distance for categorical variables.[11]

3.5 Implementation details

Both the GAN and VAE were implemented in Python using PyTorch. Standard functions from scikit-learn were used to clean and pre-process the modelling data.

---

[10] A gains chart is a measure of how well high predicted probabilities align with positive responses – see for example https://www.listendata.com/2014/08/excel-template-gain-and-lift-charts.html. Our gains ratio measures the scaled area under the gains chart, with 1 being a perfect model (all positive responses identified without conflating with any negative responses) and 0 being a random model (no distinguishing between positive and negative responses).

[11] The sdmetrics python library, https://docs.sdv.dev/sdmetrics, contains a good range of such tests.

All models were trained on a workstation with the following specifications:

- **CPU:** intel i7-12700T
- **GPU:** NVIDIA T1000 8GB
- **RAM:** 128 GB

Models were trained on the GPU using CUDA. The Quadro T1000 card (released in 2021) is relatively entry-level by modern standards.

Some aspects of our diagnostics (the GBM models for testing model effectiveness) and other diagnostics were implemented in R.

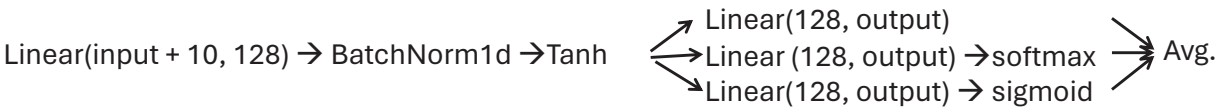
The results presented choose better performing models amongst a significant amount trial and error (even for final parameters, performance varied on differently seeded runs, particularly for the GAN. The examples related to the real-life dataset included dozens of runs with different parameters and seeds. Some observations from the experimentation are covered in the Section 5 discussion. Additionally, there is every chance that further improvements are possible with improvements to the model setup. In this sense, we do not claim that our results are optimal or final.

The presented GAN model structure is made up of:

- Generator structure: Linear(input, 128) → ReLU → Linear(128, 64) → ReLU → Linear(64, 32) → ReLU → Linear(32, output)
- Discriminator structure: Linear(input, 128) → LeakyReLU → Linear(128, 64) → LeakyReLU → Linear(64,1) → Sigmoid
- Parameters: training 200 epochs, 128 batch size, 0.0001 generator learning rate, a 0.001 discriminator learning rate, and τ annealing from 2 to 0.5.

The presented VAE model structure:

- 10 latent dimensions
- Encoder structure: Linear(input, 128) → BatchNorm1d → ReLU → Linear(128, 128) → BatchNorm1d → ReLU → Linear(128, 128) → BatchNorm1d → ReLU → Linear(128, output) → BatchNorm1d → Tanh → Linear(128, 10)
- Decoder structure has a slightly unusual last layer, where three different final layers are defined and then averaged.[12]



- Other parameters: 500 training epochs, 128 batch size, 0.01 learning rate, β=1, $\gamma = 0.02$, and τ=0.2

Both models used Adam optimizer and network weights initialized using normal Xavier initialization.

[12] There is no particular reason for this setup

4 Results

4.1 Small simulated example

4.1.1 Setup

We first demonstrate results on a simulated baby example, where the underlying data generation can be known and no explicit time dimension (instead we treat a subset of variables to be known – equivalent to fixed and time t information, and the others to be simulated as needed for the time $t+1$ variables)

Consider the setup of six-variables setup:

- $D_1, \dots, D_6$ are first generated from a multivariate normal distribution with means 0, standard deviation 1, and correlation matrix:

$$\begin{bmatrix} 1 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 1 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 1 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 1 \end{bmatrix}$$

- $X_1 = D_1 + 4$
- $X_2 = (D_2 + 2)^2 + 4$
- $X_3 = 4D_3$
- $X_4 = 4D_4$
- $X_5 = (X_1 - 5)^2 + X_2 - 2X_3 + (1.5D_5 + 1)^2$
- $X_6 = 0.5X_4 + 2\sin(X_5/4) + 1 + 0.7D_6$.

We assume X_1 to X_4 are the known variables and X_5 and X_6 the unknown. The unknown variables are both non-linear and highly dependent on the known variables and each other. There are also hard threshold – for instance the value of X_2 cannot be below 4, and X_5 cannot fall below the level of $X_2 - 2X_3$.

Because of the simple correlation structure in the simulated data, visual inspection of scatter plots is sufficient to get a good sense of how accurate the model has reproduced the data structure. In the plots displayed in Figure 4.1, we show the original data in blue, overlaid with data generated by the model in pink.

4.1.2 GAN results

Fitting the GAN for a dataset with 10,000 rows is very quick, and only took around a minute and half. Furthermore, generating predictions from the fitted model is effectively instantaneous. We did notice significant variation in the quality of the GAN model, depending on different weight initialisations for starting optimisation. This variation in quality occurred even using normal Xavier initialisation.

To see how the generator and discriminator are balancing in our finalised model, we look at the average discriminator prediction on 'real' data vs. 'fake' data created by the generator. On both these datasets, the discriminator gives an average prediction of ~0.5 probability the data is 'real'. This shows the discriminator is very effectively 'fooled' by the generator.

As displayed in Figure 4.1, the GAN model, picks up a lot of the data structure well. In particular:

- Non-linearities are captured well, as can be seen by the sinusoidal pattern in X_5 vs X_6.
- In general, the spread of generated values appears reasonable – enough noise appears to be added to the model. However, the model has slightly too little spread when looking at the plot of X_4 vs X_6.
- The generative model tends to respect the hard boundary conditions we imposed well. For example, looking at the hard cutoff at X_2= 4.

Figure 4.1 – Scatter plots: original data / generated data from GAN

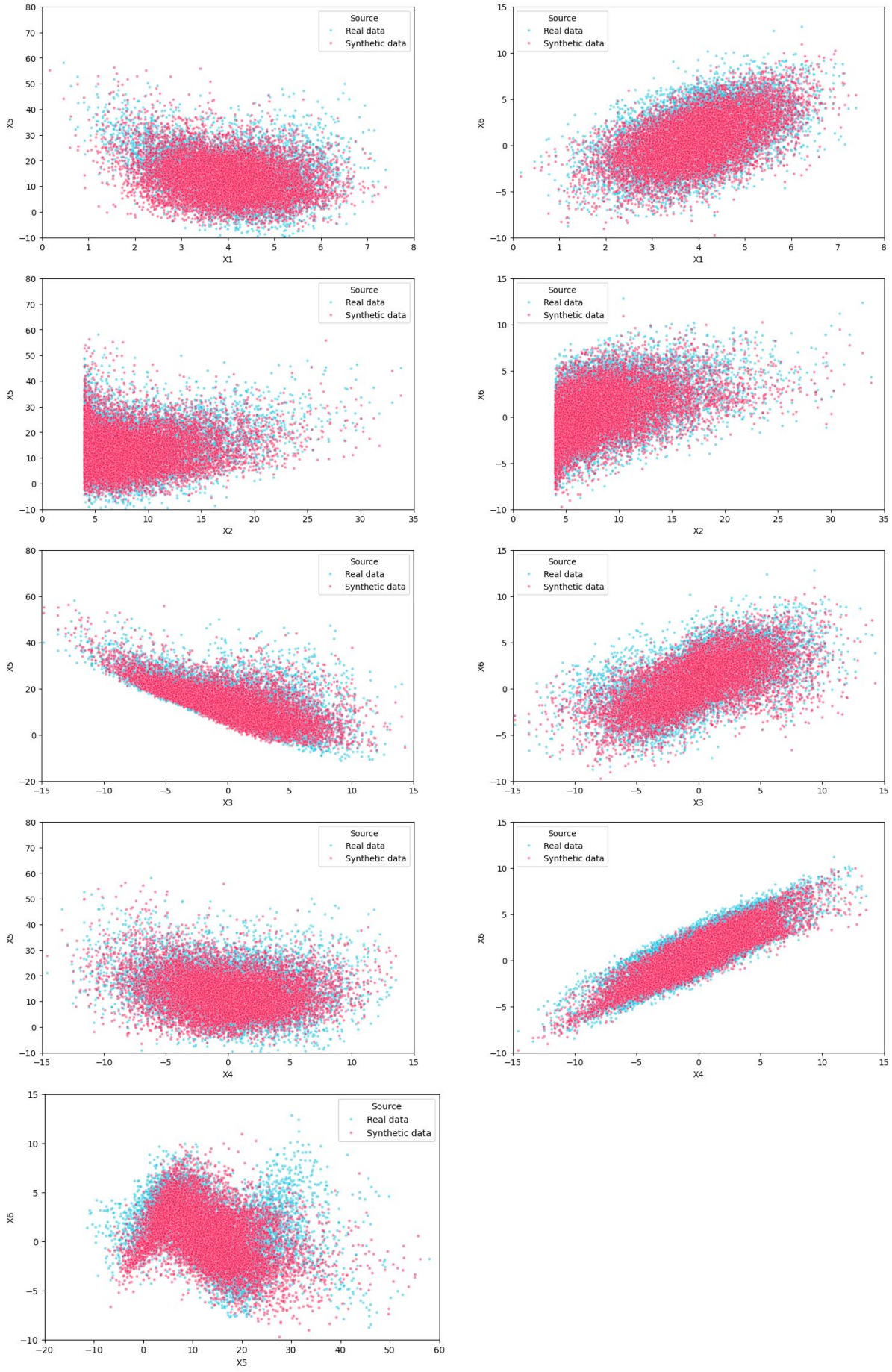


Table 4.1 compares univariate statistics for the two generated variables, which are generally reasonable. The slight biases in X_6 (lower spread) are visible.

Table 4.1 – Univariate statistics for real and generated data, GAN

	X_5		X_6	
Statistic	**Real**	**Generated**	**Real**	**Generated**
Mean	14.33	14.15	0.96	1.52
Mean abs. deviation	5.32	5.47	2.04	1.92
Std dev	8.42	8.72	2.92	2.76
Skewness	0.45	0.65	-0.02	0.05

4.1.3 VAE results

We get similar results for the VAE model – fitting the model takes about a minute, and sampling data from the fitted model is essentially instantaneous. The VAE architecture seemed more stable than the GAN architecture, in the sense that there was less variation in quality for different random weight initialisations.

The correlations captured by the VAE are similar to the GAN, and are displayed in Figure A.1 in Appendix A.1. Most of these are similar to the GAN, with the exceptions that the VAE is doing a worse job of capturing the full spread of some variables. This can be seen in the plot of X_4 vs X_6 where the simulated data has a narrower range of X_6 values, throughout.

Visual inspection makes it slightly hard to see if the non-linear relationship between X_5 and X_6 is captured by the VAE as well as the GAN. This can be confirmed by calculating the 'Correlation Similarity' metric[13], which is 0.97 for both models (to two decimal places).

Univariate statistics are compared in Table 4.2. Results are similar to the GAN, with slightly less bias in the mean and slightly less spread in X_5.

Table 4.2 – Univariate statistics for real and generated data, GAN

	X_5		X_6	
Statistic	**Real**	**Generated**	**Real**	**Generated**
Mean	14.33	13.68	0.96	1.20
Mean abs. deviation	5.32	4.86	2.04	1.91
Std dev	8.42	7.47	2.92	2.66
Skewness	0.45	0.57	-0.02	-0.15

4.2 Real-life data – HILDA longitudinal data

4.2.1 Introducing HILDA

The Household, Income and Labour Dynamics in Australia (HILDA) survey is a household-based study and is the leading longitudinal survey in the country. It captures data across a range of domains, including work, education and training, demographics, fertility and children, health and welfare usage. It is a longitudinal dataset, captured each year since 2001 by the Melbourne Institute.

HILDA release 23 (results up to 2023) was used for this project. We use survey responses from 2017 onwards for modelling. Given in a row of data we require both time t and $t+1$, this corresponds to five years of data and 76,400 rows of data (with another 18,200 rows of data withheld and used as a holdout

[13] For two synthetic variables S_1, S_2 and variables R_1, R_2 from the underlying real dataset, the Correlation Similarity metric is defined to be $1 - 0.5 * |Corr(S_1, S_2) - Corr(R_1, R_2)|$, where $Corr$ is Pearson correlation. A value of 1 indicates matching variable correlations.

dataset on our diagnostics). The survey data includes both *responding persons* (those members of a household who responded to the survey) and *enumerated persons* (members of a household who did not respond); we only use survey responses from *responding persons*.

HILDA is an appropriate dataset to use because it is:

- Longitudinal, meaning that the same respondent households answer the survey each year. Therefore, time *t+1* responses exist for modelling. Moreover, this allows for testing of model stability and goodness of fit over a longer period of time (e.g. can test how well the model performs over a period of three years)
- Contains variables similar to those typically used in microsimulation modelling, such as education, labour and welfare.
- The data is of good quality and has undergone an extensive cleaning and imputation process. For example, survey responses that are deemed implausible after intensive checking are removed. Respondents who are unable to provide an answer to a question (for example, the dollar amount of welfare benefits received in the year to date) have their responses imputed.[14]

We selected a subset of variables from HILDA that we thought would provide a challenging modelling task across variable types and interdependences – see table Table 4.3. Of the 24 variables, two were deterministic (age and sex), and the other 22 dynamic, of which six were continuous and the remainder categorical. After one-hot encoding this was 74 dynamic variables, and after attaching 74 variables relating to time $t + 1$ the final full vector contained (2+74+74=) 150 variables.

[14] Detailed information on HILDA and the data construction are in the regular user manuals, with the 2023 release available at https://melbourneinstitute.unimelb.edu.au/__data/assets/pdf_file/0006/5166807/HILDA-User-Manual-Release-23.0.pdf

Table 4.3 – Variables used from HILDA for modelling

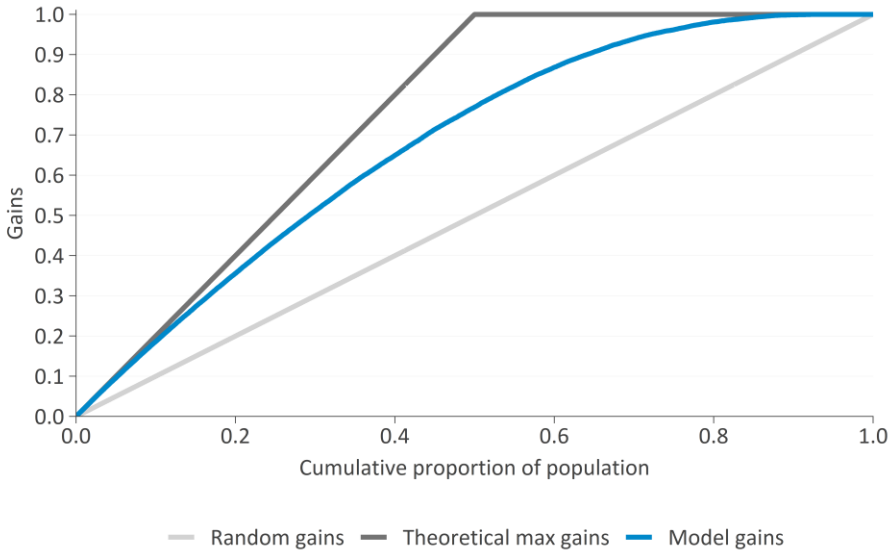
HILDA Variable ID	Short descriptor	Description
hhiage	Age	Age last birthday at date of interview
hgsex	Sex	Sex of respondent
mrcurr	Marital status	Current marital status
hhs3add	SEIFA decile (location)	SEIFA relative socioeconomic advantage/disadvantage decile 2021
hgndi	NDIS flag	NDIS-agreed support package
hglth	Disability flag	Long-term health condition, disability or impairment
cccinhh	Children flag	Respondent has children aged under 14 in the household
ccftb	FTB flag	Respondent receives family tax benefit
edhigh1	Education level	Highest level of education attained
bncap	Age pension flag	Respondent currently receiving aged pension
bncapui	Benefits amount	Value of current weekly public transfers excluding family tax benefit, imputed
bnccrp	Carer flag	Respondent currently receiving carer payment
bncdsp	DSP flag	Respondent currently receiving disability support pension
bncdvaa	DSP amount	Value of latest disability support pension payments
bncnws	Jobseeker flag	Respondent currently receiving Jobseeker
bncpar	Parent Payment amount	Value of latest weekly parenting payment
tifefp	Total annual income	Total regular gross income in the last financial year, including transfers/welfare
wscei	Weekly wages	Current weekly gross wages and salary across all jobs, imputed
wscmtoj	Multiple job flag	Respondent is currently working in more than one job
es	Employment status	Respondent's current employment status
esbrd	Labour force status	Respondent's current labour force status
jbcasab	Casual flag	Casual worker
jbcmocc	Change in job flag	Occupation changed since last interview
rtyr	Year retired	Year retired
gh1	Health rating	Self-assessed health (Excellent / Very good / Good / Fair / Poor)
losat	Life satisfaction	Respondent's life satisfaction (varying from 0 to 10, where 0 is not satisfied and 10 is most satisfied)

4.2.2 GAN results on HILDA – single time step

The GAN model achieves a reasonable (but not spectacular) level of performance on the HILDA dataset. The gains chart (Figure 4.2) shows a gains ratio of 0.711[15], which shows that the generator is routinely fooling the model, but there are still systematic differences detectable between the generated and actual data.

[15] Here a ratio of 0 means the discriminator model is randomly guessing between generated and real (so cannot see a difference), and 1 means the model can perfectly identify generated data

Figure 4.2 – Gains chart on holdout data for selected GAN model. Gains ratio is 0.711



Note: A gains ration of 0 the model is randomly guessing between generated and real (cannot detect fakes), and the blue line would sit at the 45 degree line. A ratio 1 means the model can perfectly identify generated data, and the blue line would sit at the theoretical max gains.

The GBM reports relative variable importance, which gives guidance as to which variables are being most used to distinguish between real and generated data. The top three are continuous variables: Gross income, weekly wages and life satisfaction. Many detected misfits are potentially interactions rather than one-way issues.

One-way comparisons generally look good. For **categorical variables**, Table 4.4 shows the Total Variation Distance statistic[16] (on hold-out data):

$$TVD = 1 - \frac{1}{2}\sum_{n=1}^{N} |P_n - Q_n|$$

Where P_n and Q_n are the proportions of real and simulated data from the GAN respectively, for a category n of a variable with N categories. Scores are uniformly high (although some that tend to be highest are also the easiest to predict as they change less over time). We have tended to observe when there are distributional differences it is more common for the lower-frequency classes to be under-represented. Two of the poorer performing categorical variables are shown in the Note: 1 represents perfectly matching distributions

Figure 4.3, that exhibit this pattern.

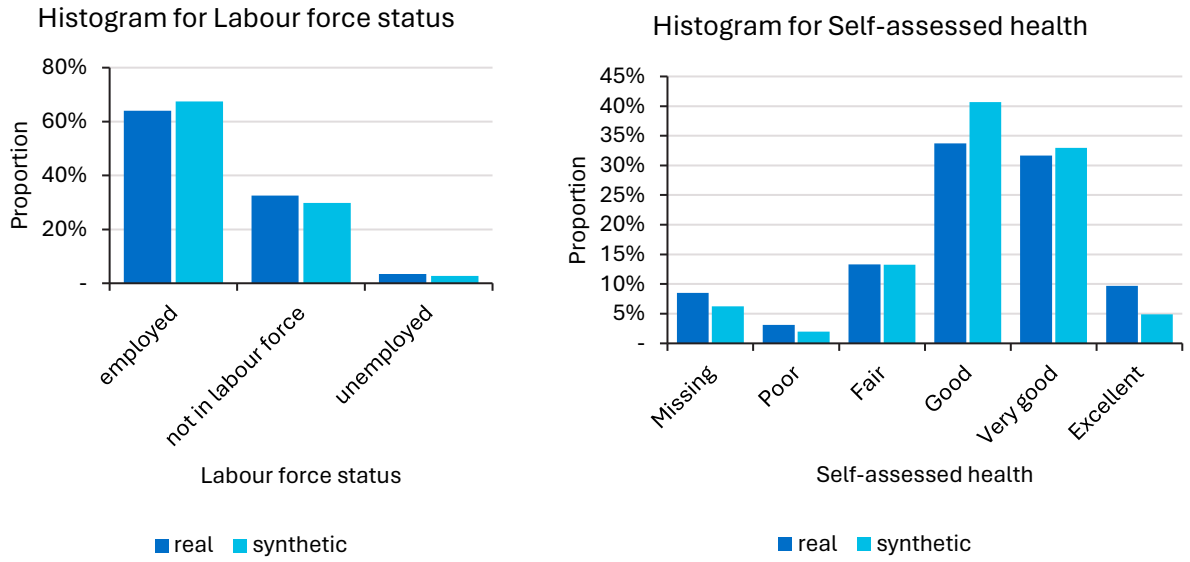
[16] The leading "1 −" for the TVD and KS scores are not standard, but are used by the sdmetrics library and we have retained. They have the effect of making 1 represent a 'good' distribution, rather than 0.

Table 4.4 – GAN Total variation distance goodness of fit statistic for categorical variables at time $t + 1$

Categorical variable	Total variation distance
Children flag	**0.987**
FTB flag	0.998
Education level	0.994
Age pension flag	0.993
Carer flag	0.999
DSP flag	0.999
Jobseeker flag	0.989
Multiple job flag	0.981
Employment status	0.961
Labour force status	0.968
Casual work flag	0.996
Change job since flag	0.998
Retired flag	0.999
Marital status	0.972
SEIFA decile	0.978
NDIS flag	0.998
Disability flag	0.888
Health rating	0.925

Note: 1 represents perfectly matching distributions

Figure 4.3 – GAN generated distributions for two categorical variables



For continuous variables, which tend to be harder to model, we also found that one-way distributional comparisons were mixed. Table 4.5 shows the Kolmogorov-Smirnov statistic[17] for continuous variables, defined as:

[17] See previous footnote

$$KS = 1 - \sup_x |F(x) - G(x)|$$

Where $F(x)$ is the empirical distribution function of the real data and $G(x)$ of the simulated data from the GAN. Values closer to 100% indicate a superior fit. The Transfers and DSP payment variables are artificially low due to their mixed distribution setup; for example the DSP fit has a good mean and point mass at zero, but poor tail shape for the small fraction of non-zero values.

Table 4.5 – Kolmogorov-Smirnov goodness of fit statistic for continuous variables, GAN model

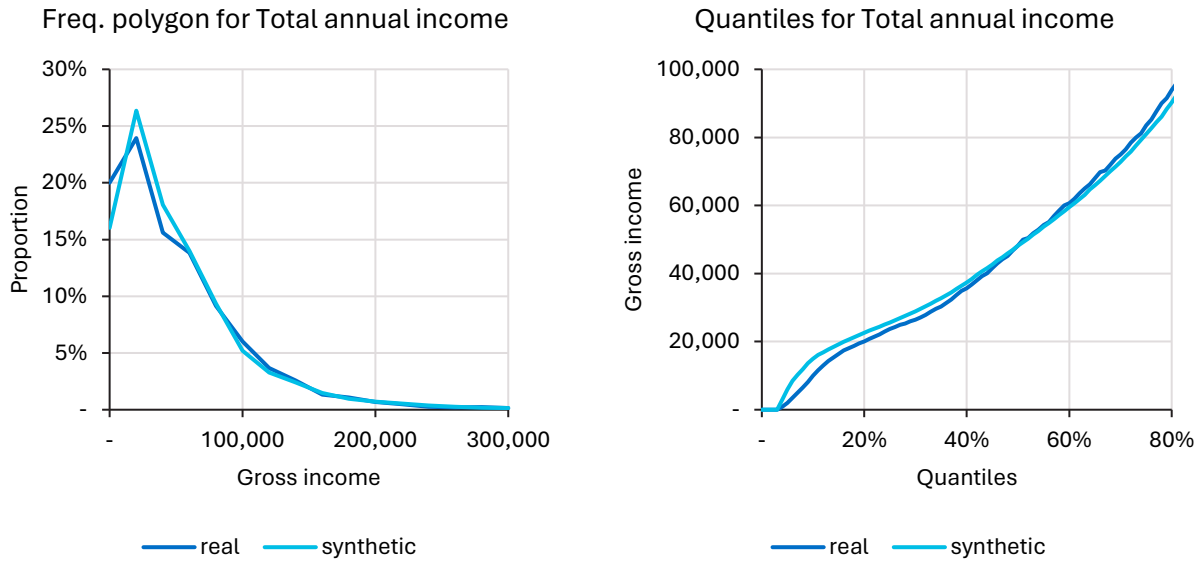
Continuous variable	Kolmogorov-Smirnov statistic
Benefits amount[(b)]	0.272
DSP amount[(b)]	0.004
Parenting amount[(b)]	0.983
Total annual income	0.957
Weekly wages[(b)]	0.591
Life satisfaction	0.862

(a) 1 represents perfectly matching distributions.

(b) KS scores strongly affected by the handling of the mixed/skewed distribution

Figure 4.4 shows the frequency polygon and the quantiles for the Total annual income variable. We regard the fit as good, given the inherent challenges in fitting continuous distributions.

Figure 4.4 – Frequency polygon and quantile plots for Total annual income

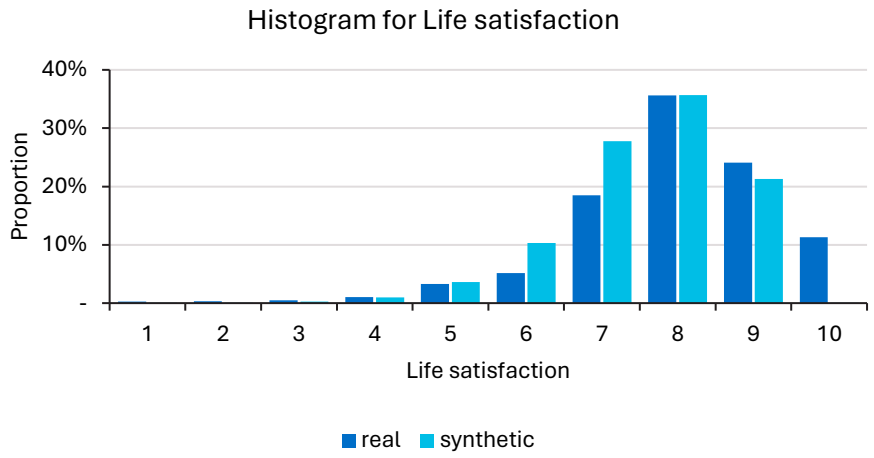


Note: Plots truncated at $300,000 / 80th percentile respectively for better visibility of important parts of the distribution

One interesting feature was a consistent failure to capture the top of the distribution for the life satisfaction variable (scored on a 1 to 10 scale) – the synthetic data rarely generated a 10, which is different to the actuals, as shown in Figure 4.5. We are unsure if the integer nature of the data, or its boundedness, contribute to the misfit.

There is a substantial misfit for the value of transfers excluding family tax benefit. The model appears to overpredict the proportion of the population who receive the family tax benefit, and of those who do receive it, it appears to overpredict the amount.

Figure 4.5 – Distribution of life satisfaction score, GAN model



Of particular interest is how interactions between variables are modelled. These are shown in a series of heat plots below. We observe interactions are generally captured very well, despite being very awkward two-way distributions to model:

- The relationship between age pension receipt and age (Figure 4.6)
- Weekly wages and labour force status (Figure 4.7), where those that are not in the labour force or unemployed should not have weekly wages, plus a plausible distribution for those that are employed
- Weekly wages and total gross income (Figure 4.8), which has both a loose correlation plus a distribution of total income for those with zero or low weekly wages (as people can have other income apart from wages). The generated correlation is perhaps a little tighter than actual, but the relationship is fairly well captured.
- Weekly wages and government transfers (Figure 4.9), where the 'one or the other' dynamic is well captured.

Figure 4.6 – Receipt of age pension flag (dynamic) by age (deterministic). Legend shows number of observations. Real distribution shown left, generated data to the right

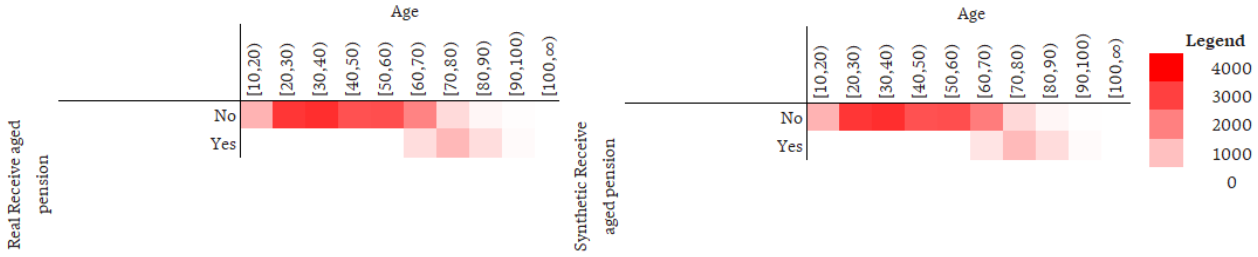


Figure 4.7 – Wages (dynamic) by labour force status (dynamic)

Figure 4.8 – Weekly wages (dynamic) and total gross income (including transfers/welfare, dynamic)

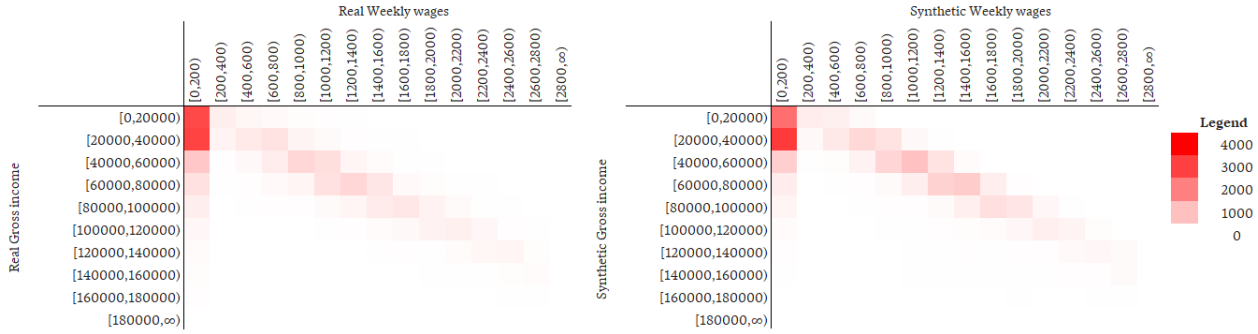
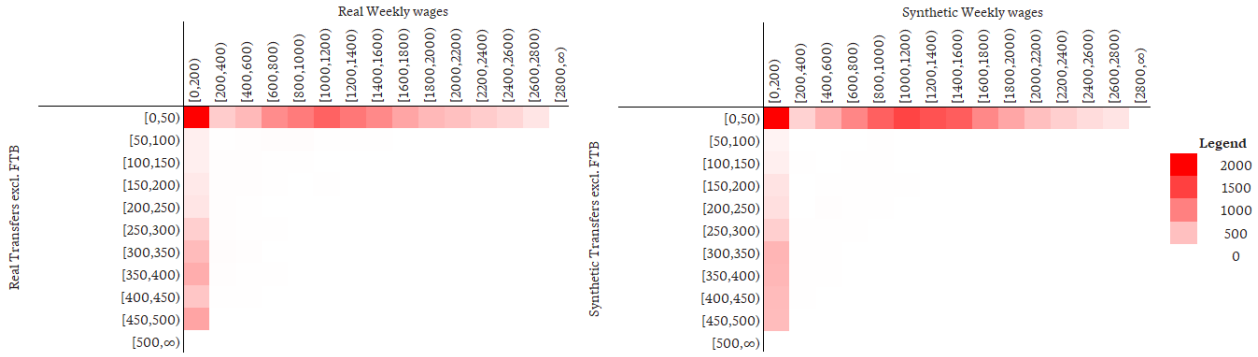


Figure 4.9 – Weekly wages (dynamic) and transfers (excluding family tax benefit, dynamic)



4.2.3 GAN results on HILDA – three-year time step

Chained GAN model

We created a version of HILDA data over the same time interval but with a three-year timestep. This resulted in a smaller dataset (time t ranged from 2017 to 2020) with 46,138 rows in the training data and 10,908 rows in the holdout. We can then compare:

- The performance of a three year 'chained' GAN (applying three times successively), using the model from section 4.2.2
- The performance of a three-year single step model, trained on the modified data.

We see a substantial deterioration in the overall performance of the GAN when chained. While some deterioration is expected (chaining three years is intrinsically harder than a single step), it appeared significant. The gains ratio rose from 0.711 (for one step ahead) to 0.927 – meaning that it was very easy for the discriminator to spot generated records. The most important variables in the GBM discriminator are weekly wages, total income and transfers, indicating that the difficulties modelling continuous variables are exacerbated when chained.

The misfit for this particular model on total annual income is evident in Figure 4.10 – the distribution has shifted right and amounts are too high. A similar shift is visible in weekly wages, although interestingly the two-way relationship between the two remains intact (see Figure 4.11), conditional on the shift.

Figure 4.10 – Frequency polygon and quantiles for total annual income, chained three-year GAN model

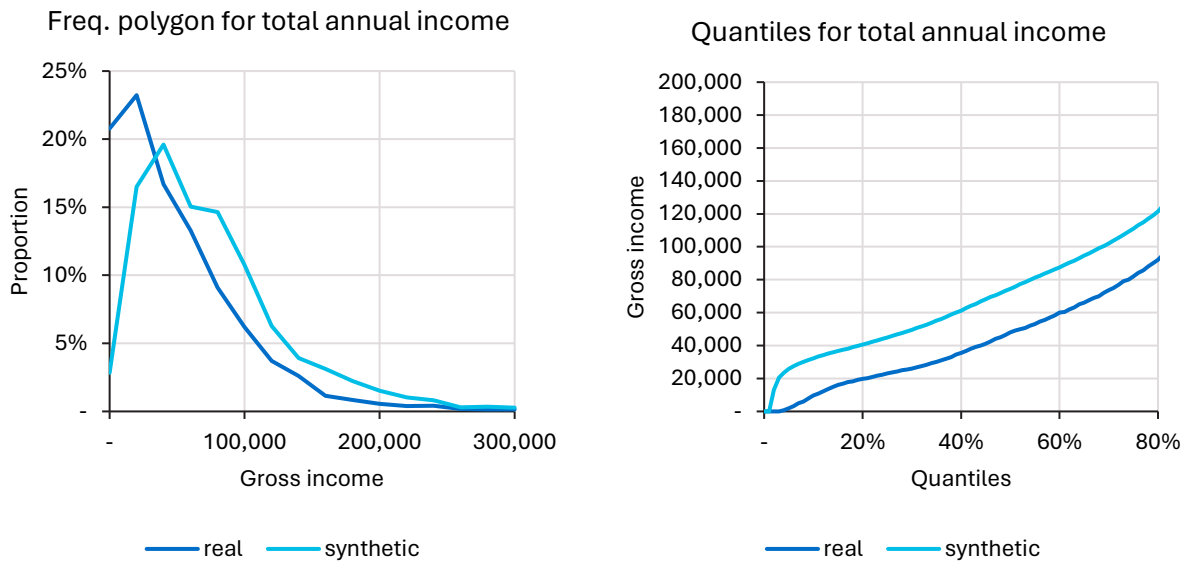
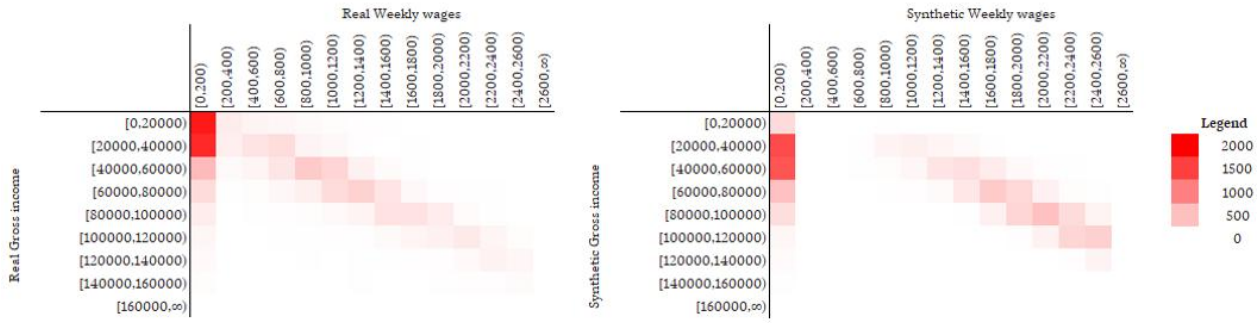


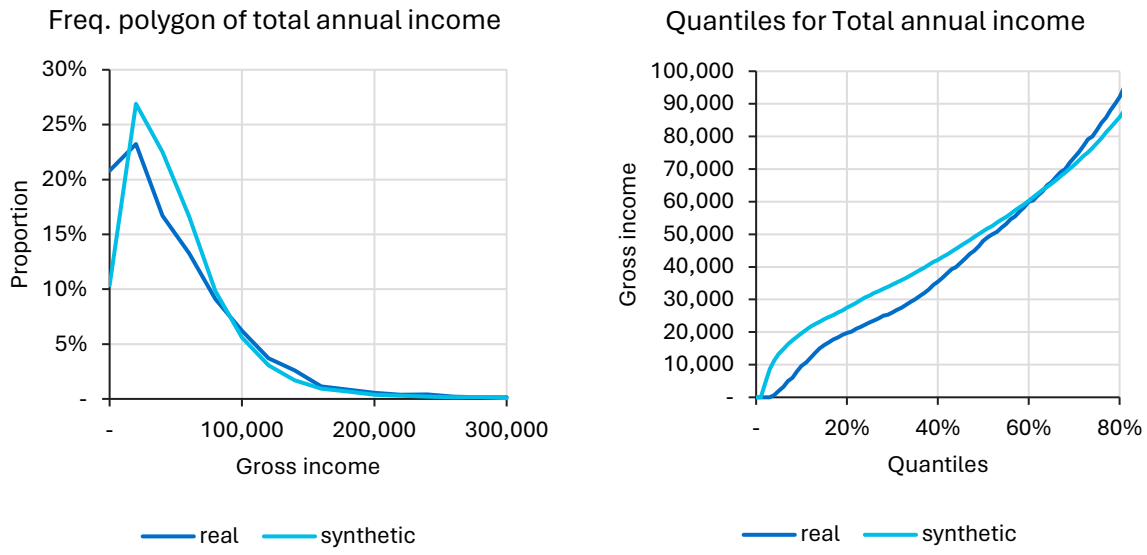
Figure 4.11 – Weekly wages and total gross income (including transfers/welfare) from chained three-year GAN



Single step three-year GAN

We also tested a model using data at time t to predict outcomes at time $t + 3$ otherwise keeping the model setup for training the same. The gains ratio was 0.84, which is materially better than the chained model but significantly worse than the one-step model. This deterioration we largely attribute to the vagaries of GAN fitting – see our discussion in Sections 5. Some intermediate-level misfit on gross annual income is shown in the figure below – while potentially passable in some contexts, combined with other misfits suggests significant gaps in the generation.

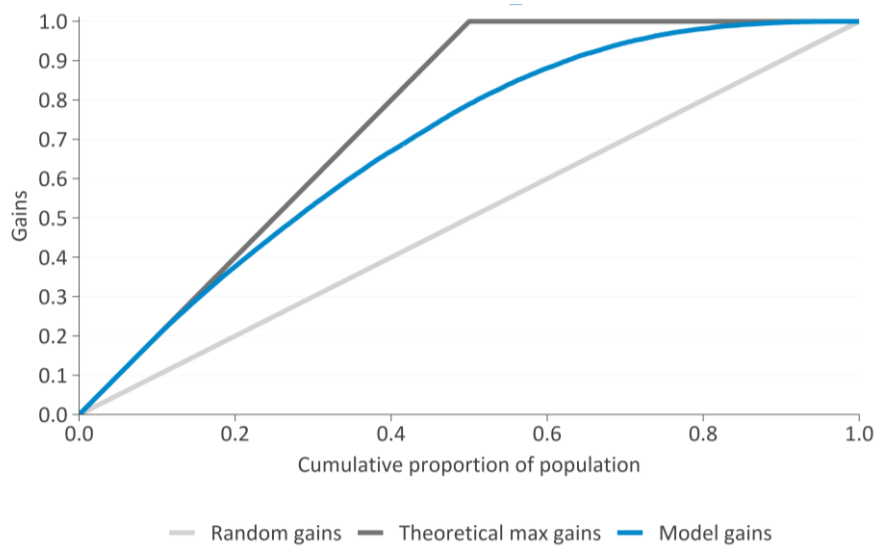
Figure 4.12 – Frequency polygon and quantiles for single-step three-year GAN



4.2.4 VAE results on HILDA – single time step

The gains ratio on the VAE is 0.76, which is in the vicinity (albeit somewhat worse than) the single step GAN model. Results are similarly encouraging, in that the high-performing GBM model is routinely fooled by the generations.

Figure 4.13 – Gains chart on holdout data for selected GAN model. Gains ratio is 0.756 (0=random, 1=perfect discrimination)



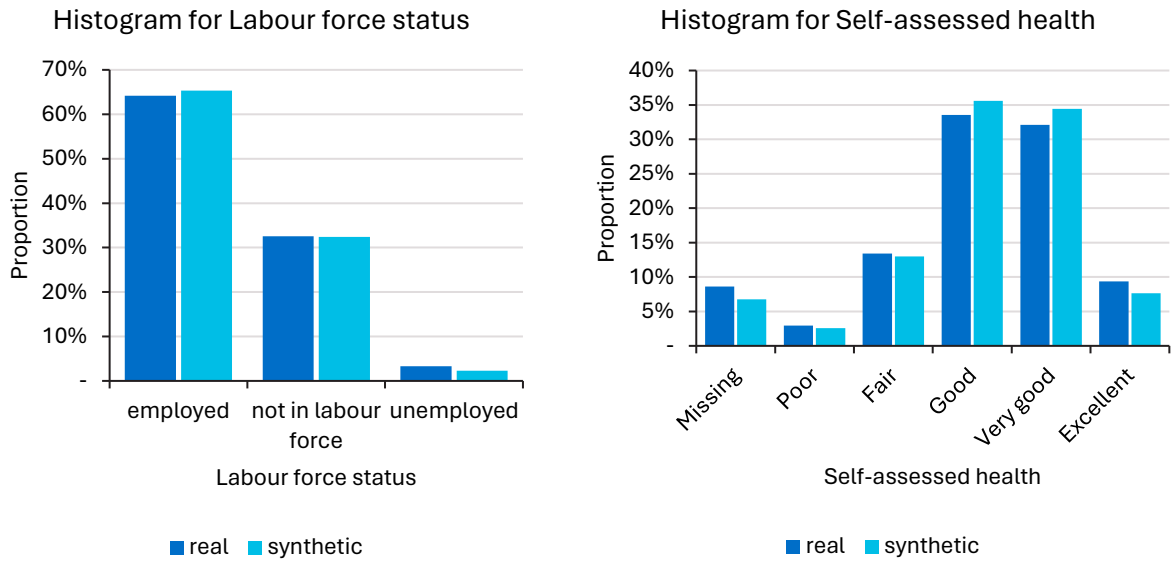
Categorical variables fits were generally good. Total variation distance scores were noticeably higher than the equivalent for GAN. The disability flag in particular is much improved.

Table 4.6 – VAE Total variation distance goodness of fit statistic for categorical variables at time $t + 1$

Categorical variable	Total variation distance
Children flag	0.992
FTB flag	0.999
Education level	0.987
Age pension flag	0.997
Carer flag	0.993
DSP flag	0.999
Jobseeker flag	0.994
Multiple job flag	0.996
Employment status	0.986
Labour force status	0.988
Casual work flag	0.990
Change job since flag	0.978
Retired flag	0.996
Marital status	0.992
SEIFA decile	0.996
NDIS flag	0.996
Disability flag	0.997
Health rating	0.959

The VAE achieves a similar fit to the GAN for categorical variables, with the one notable difference being an improvement in predicting the long-term condition flag. Two examples are shown in Figure 4.14.

Figure 4.14 – VAE generated distributions for two categorical variables



The KS-scores remain hard to judge due to the mixed-distribution effects affecting the scores. Total annual income distribution score appears slightly worse than the GAN, and life satisfaction improved.

Table 4.7 – Kolmogorov-Smirnov goodness of fit statistic for continuous variables, VAE model

Categorical variable	Kolmogorov-Smirnov statistic
Benefits amount[(b)]	0.274
DSP amount[(b)]	0.999
Parenting amount[(b)]	0.019
Total annual income	0.940
Weekly wages[(b)]	0.982
Life satisfaction	0.897

(a) 1 represents perfectly matching distributions.

(b) KS scores strongly affected by the handling of the mixed/skewed distribution

Figure 4.15 shows the frequency polygon and the quantiles for the total annual income variable. The fit captures key aspects of the shape – with the main visible defect underpredicting around zero and overprediction around $20,000. The equivalent plots for Weekly wages (Figure 4.16) shows the successful capturing of the multimodal pattern of the variable. Figure 4.17 shows underprediction of high life satisfaction, similar to the GAN.[18]

[18] We do not think this is a universal judgement by AI models on human happiness.

Figure 4.15 – Histogram of gross income from VAE (truncated at $300,000)

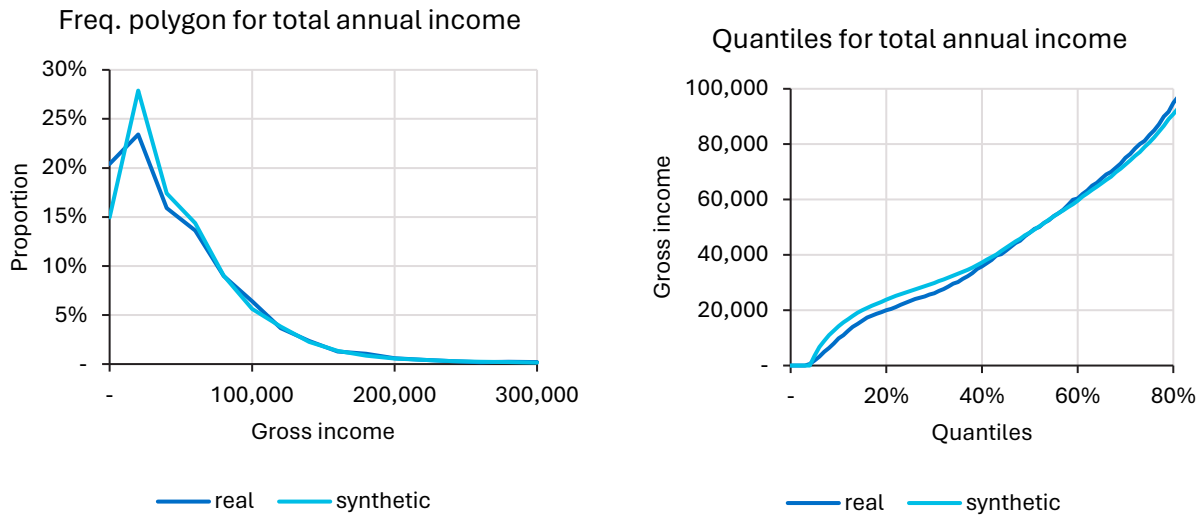


Figure 4.16 – Histogram of weekly wages from VAE

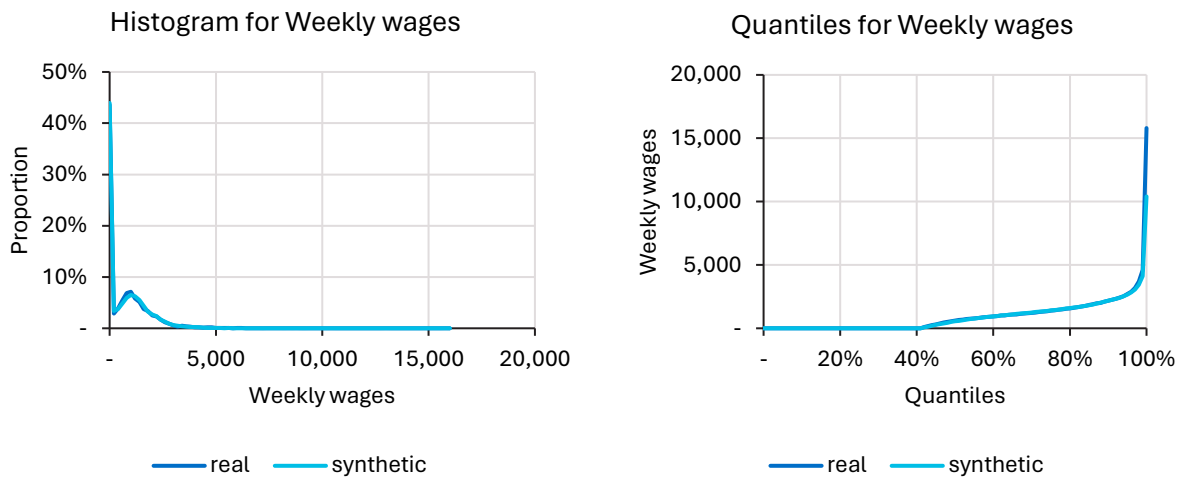
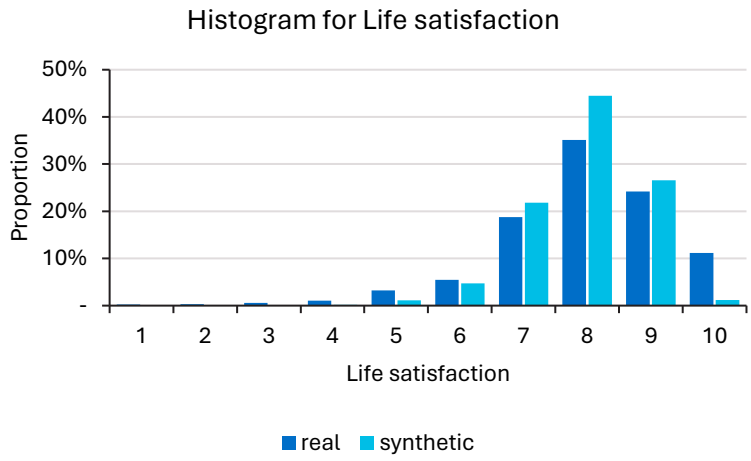


Figure 4.17 – Distribution of life satisfaction score, VAE model



The VAE does a good job at modelling the relationship between different payment variables, with close alignment between the real and synthetic heatmaps. Selected heatmaps are provided in the appendix –

the generally appear comparable to those of the GAN, demonstrating that complex interdependencies between variables are effectively embedded in the model.

4.2.5 VAE results on HILDA – three-year time step

Interestingly (and unlike the GAN), the three-year predictions the VAE appear very promising. The gains ratio for the chained model (applying the one-year model successively three times) is 0.78, very close to the one-year performance. The performance of the single step three-year model is also 0.78, suggesting very little deterioration related to the chaining. Gains charts for these models are included in the appendix.

Chained VAE model

Figure 4.18 shows the distribution for total annual income for the chained model after three years – while the underprediction of low incomes is a little more pronounced, the shape for the remaining curve is very faithful. The weekly wage univariate distribution (Figure 4.19) is very accurate. The complex two-way relationship remains intact (Figure 4.20).

Figure 4.18 – Frequency polygon and quantiles for total annual income chained three-year VAE model

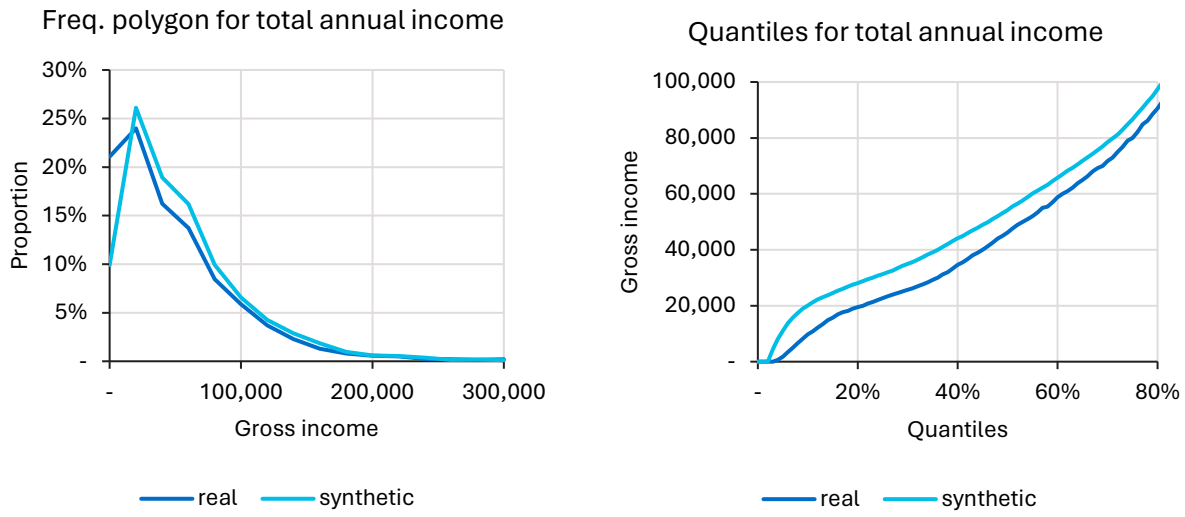


Figure 4.19 – Histogram of weekly wages for real and simulated data from chained three-year VAE

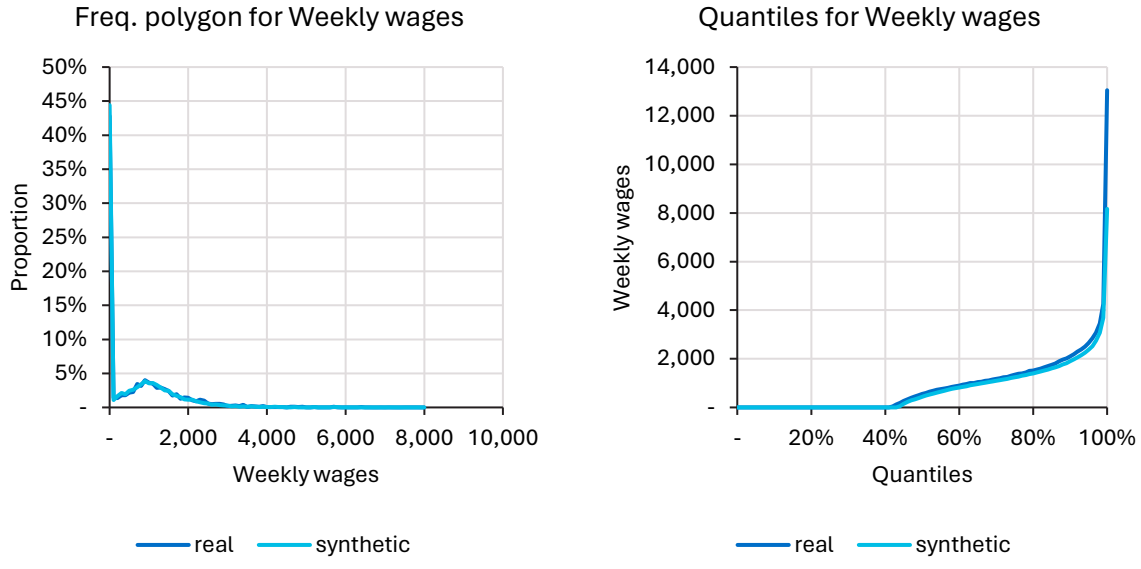
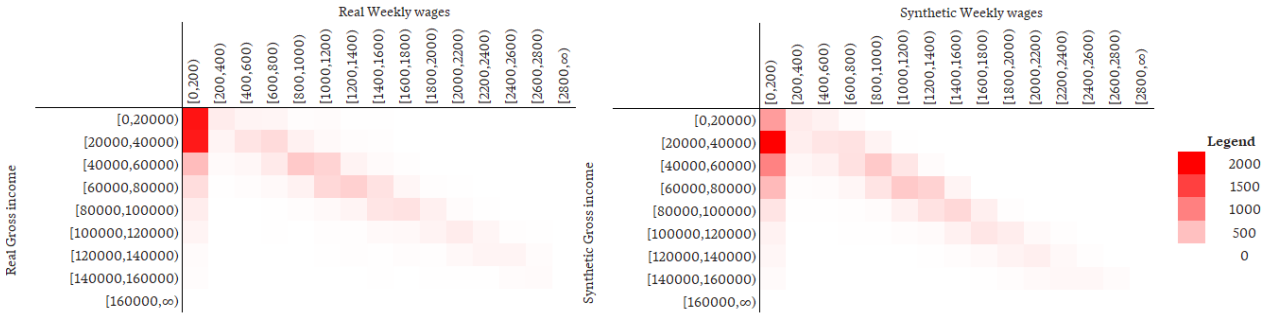


Figure 4.20 – Weekly wages and total gross income (including transfers/welfare) from chained three-year VAE



Stepped three-year VAE

Like the GAN, we also tested a model using data at time t to predict outcomes at time $t + 3$. Model performance looked quite similar to the one-step model across a broad range of variables – some additional plots are included in the Appendix.

Overall, the chained and stepped three-year VAEs offer far superior performance to the chained and stepped three-year GANs, suggesting superior model stability. This most likely would extend to longer time windows too.

4.3 Training time and computational considerations

We have tracked training and generation times across a range of models, using the computational setup described in Section 3.5. Significant acceleration is undoubtably possible with higher-end setups (more powerful GPUs or multiple cards).

When training we found:

- Both core (one time step) models took about two hours to train about 1000 epochs (although the models required fewer epochs of training). This scales linearly with the number of epochs, and the number of rows in the dataset, as might be expected. This may have implications for very large datasets, although we would expect such datasets would require fewer epochs for convergence. Example timings are provided in the appendix.
- Model fit time was only marginally affected by the number of generator layers for both the GAN and VAE. This speaks to the relative efficiency of backpropagation, and suggests the core backpropagation is not the main bottleneck in our fitting. Example timings are provided in the appendix.

For projection, Table 4.8 show the amount of time taken to score the GAN and VAE on datasets of differing numbers of rows. The VAE was three times faster to generate – most likely due to fewer layers in the reconstruction network (compared to the GAN generator network).

Table 4.8 – GAN projection time for differing number of rows

Number of records generated	GAN scoring time (seconds)	VAE scoring time (seconds)
10,000	0.2	0.02
100,000	0.9	0.3
1,000,000	8.6	4.7
10,000,000	135.7	40.9

While timings for other microsimulations are not public, we regard the speeds above as close to best-in-class; and this is without distribution across multiple cards and time spent optimising. Much of the speed comes from how the PyTorch package can effectively leverage the parallelism of GPUs.

Note that 10 million rows in 40 seconds may still mean long simulation times for larger setups. A simulation across 27 million Australians for 50 time steps, repeated 10 times, would still correspond to 15 hours of projection time. A large GPU network such as 8×Tesla V100 cards (as available in Amazon's p3.16xlarge instance) would be about 40 times more powerful than our current setup, reducing this to about 20 minutes.

4.4 Comparing GAN and VAE training experience

Even beyond distribution mismatches, we found the numerous challenges in training the GAN model, with many training runs failing to converge to a credible model. Key issues were:

- **Mode collapse –** this occurs when the generator network only produces a few (or only one) type of output (e.g. not generating over all possible categories of a variable), regardless of the input noise. The generator can find it hard to break out of these mode collapses, even if it is being exploited by the discriminator network.
- **Diminished gradient –** this occurs when the discriminator is too good at distinguishing real from fake data – it learns too quickly relative to the generator evolution. This causes a collapse in the loss function, limiting the useful signal is passed back to the generator and the generator is not able to improve. To help mitigate this problem we altered the optimisation algorithm so that the generator updates twice in each optimisation step (albeit at the lower learning rate), for each single update of the discriminator.
- **Sensitivity to initial conditions –** In general, the quality of the GAN model seemed highly dependent on the initial random weights. Running the optimisation algorithm with small random changes in initial weights seemed to lead to very different quality fitted models. Using the 'Normal Xavier' weight initialisation algorithm in PyTorch helped a great deal with this.

In contrast, the VAE proved much more consistent in training – mode collapses were infrequent and finding sensible parameter choices (such as the β to balance reconstruction error and KL-difference, and γ to balance continuous and categorical loss functions) were not too onerous.

As a bonus, VAE training time was lower (fewer epochs required) and generation was faster (simpler generator network), although these are obviously sensitive to the design.

5 Discussion

Overall feasibility of using AI models for microsimulation

We regard our results in this paper as a qualified success:

- At a basic level, the GAN and VAE models were successfully able to generate high-quality synthetic records on both the simulated dataset and the larger, more complex real-life dataset. The most common challenges of these tabular data applications are:
  - Different types of variables on different scales, including some quite complex distributions
  - Interdependencies between variables
  - Chaining together multiple predictions (for the VAE model).

  These have been successfully handled – the approach is well past the proof-of-concept stage.
- Overall, the GBM discrimination was still relatively good at distinguishing between the generated and actual projected values on our real-life dataset. Some of this performance is not fully understood – one- and two-way plots are generally good, so the effects being exploited (whether artefacts from

the network models or uncaptured structures in the real data) may be quite complex. Further worth using SHAP plots or similar would provide some insight into this. Conversely, we deliberately chose a challenging set of variables with many interdependencies, increasing the likelihood of such as result.

Ultimately, we have no like-for-like comparison of discrimination performance with existing microsimulation models, so cannot judge whether the overall performance achieved in this paper is sufficiently good.

- The power of GPU-based computing for AI models is also evident, with good model fitting and generation speeds without significant time optimising.

Comparison between the GAN and VAE

While the intuition behind the GAN is appealing, the VAE proved to be a far better model structure; convergence was more consistent (and significantly fewer collapsed models) and models were stable enough to perform multiple time steps. Given the long-term nature of some microsimulation work, such stability is at a premium.

While capturing the full richness of continuous variables were a challenge for the VAE (as it was for the GAN), the superior performance on chained results suggests that, in our example at least), the VAE has better captured some of the deeper structure for evolution over time.

Broader learnings

Some of our observations are applicable to microsimulation models more broadly:

- **Improved diagnostics –** From our experience, much of the validation of microsimulation models relies on one-way and two-way summaries. This de-emphasises the relationships between variables. The idea of a discriminator model is therefore a useful addition to current practice, regardless of whether or not the underlying model is an AI model. Variable importance and tools such as SHAP values and partial dependence charts can then point to areas of model weakness efficiently. Similarly, the KS scores appear useful one-number diagnostics to track distributions across models.
- **Care in variable selection –** Continuous variables (and mixed distribution variables) proved to be the more challenging variables in our real-life dataset modelling. This is consistent with our experience more broadly – categorical and transition models tend to be easier to build and chain, and are pervasive in existing approaches. This means basic model design remains important – which variables to include to avoid unnecessary complexity. This includes considering the best way to handle mixed distribution variables.

  Relatedly, care is needed when choosing a larger number of highly correlated or interdependent variables. In some cases there may be parameterisations that are easier for models to handle; for example, modelling income components separately (rather than also modelling total income) reduces interdependence.
- **Multistep modelling** – Understanding the relative performance of a chained prediction versus a single leap (in our case over three years) is a natural way to understand the effectiveness of the chaining and identifying deterioration.

Areas for further research

There are numerous areas for potential extension and development.

- **Generalisability to other datasets –** Our approach has significant generalisability (for example, in the way we can handle both categorical, continuous and mixed distribution variables), but we have not applied the approach to other human services datasets. It would be interesting to see whether similar model structures (e.g. number of layers and nodes) remained suitable, or if these need to be highly tailored to the data context.

- **Scaling training and generation –** We do not explore more powerful hardware setups, which may be warranted for larger datasets. Scaling generation is straightforward, but finding the most efficient ways to scale learning in distributed environments may require experimentation.
- **Improving handling of continuous variables** – The different distributions of continuous variables present challenges. It may be that standardised transformations (e.g. converting to standard normal via CDF-based transformations) is an effective way to sidestep these issues.
- **Alternative AI model structures –** We selected GANs and VAEs as common models that could be readily tailored to our specific use case. The world of potential model structures is large. As examples of other untested alternatives:
  - Wasserstein GAN – Instead of a generator attempting to "fool" a discriminator, we replace the discriminator with a critic model. This critic model scores the quality of real vs. fake samples by measuring the Wasserstein distance. The critic thus aims to maximise the difference between the scores of the real and fake samples. The Wasserstein GAN tends to be less prone to mode collapse and vanishing gradients. This allows for improved stability and ease of training.
  - Hierarchical VAE – while a regular VAE has one layer of latent variables, a hierarchical VAE adds a hierarchical, multi-layered structure to the latent variables. The encoder model breaks input data down into these layers, while the decoder reconstructs the input by sampling from each layer from top to bottom. In principle this allows the model to prioritise core effects, with secondary variation then applied sequentially.
- **Developing tools for finer-grained control of the model –** Our model effectively takes the data at face value and attempts to reproduce it. In practice, projects may need to be slightly different to the recent history. For instance, time trends can be recognised, or patterns due to recent policy changes may need to be adjusted for, or macroeconomic forecasts allowed for. This can be easier in traditional microsimulation setups where specific models can be tweaked to reflect these factors. Doing this for AI models is less direct, but there are options:
  - **Including time factors or macroeconomic variables in the model –** including these factors allow the model to recognise some of the time trends. Judicious setting of these variables at the generation stage will then provide some control,
  - **Reweighting the training data –** If certain outcomes need to be modified by a known amount, the data can be weighted to adjust the average outcome rate by the necessary amount (e.g. skewing the modelling data towards higher employment to reflect expected improvements). This would become more complex if multiple outcomes needed to be adjusted simultaneously.

References

- Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., & Kasneci, G. (2022). Deep neural networks and tabular data: A survey. IEEE transactions on neural networks and learning systems.
- Chang, S. L., Harding, N., Zachreson, C., Cliff, O. M., & Prokopenko, M. (2020). Modelling transmission and control of the COVID-19 pandemic in Australia. *Nature communications*, *11*(1), 5710.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139-144.
- Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2, pp. 1-758). New York: springer.
- Jang, E., Gu, S., & Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.
- Kingma, D. P., & Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends in Machine Learning*, *12*(4), 307-392.
- Kingma, D. P., & Welling, M. (2013, December). Auto-encoding variational bayes.

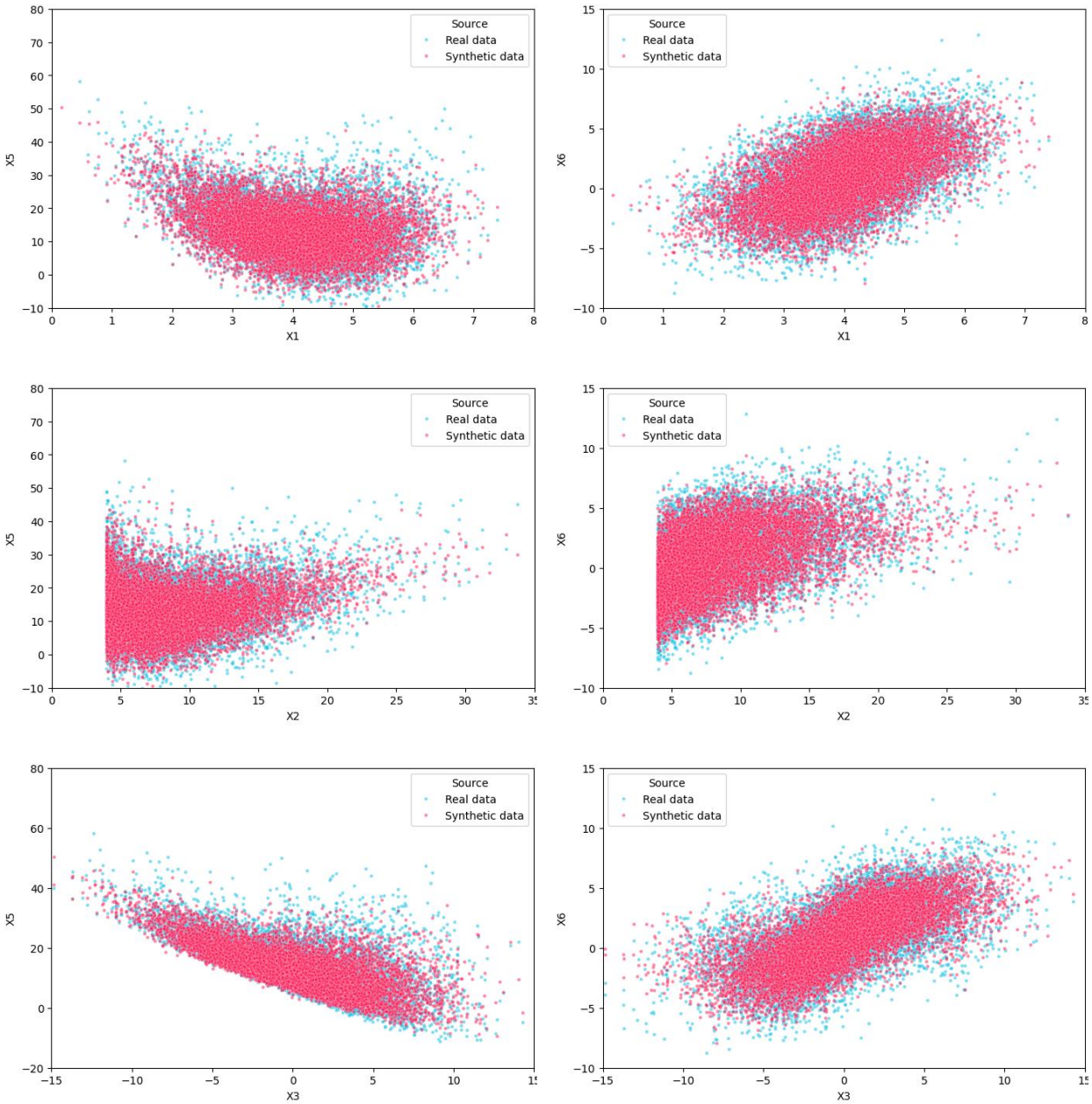
- Maddison, C. J., Mnih, A., & Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.
- Moyaux, T., Liu, Y., Bouleux, G., & Cheutet, V. (2023). An agent-based architecture of the digital twin for an emergency department. *Sustainability*, 15(4), 3412
- Nazabal, A., Olmos, P. M., Ghahramani, Z., & Valera, I. (2020). Handling incomplete heterogeneous data using vaes. *Pattern Recognition*, *107*, 107501.
- Rasheed, A., San, O., & Kvamsdal, T. (2020). Digital twin: Values, challenges and enablers from a modeling perspective. *IEEE access*, *8*, 21980-22012.
- Sun, Y., Li, J., Xu, Y., Zhang, T., & Wang, X. (2023). Deep learning versus conventional methods for missing data imputation: A review and comparative study. *Expert Systems with Applications*, *227*, 120201.
- Telyatnikov, L., & Scardapane, S. (2023, April). Egg-gae: scalable graph neural networks for tabular data imputation. In *International conference on artificial intelligence and statistics* (pp. 2661-2676). PMLR.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.
- Yoon, J., Jordon, J., & Schaar, M. (2018, July). Gain: Missing data imputation using generative adversarial nets. In *International conference on machine learning* (pp. 5689-5698). PMLR.
- Yoon, J., Zhang, Y., Jordon, J., & Van der Schaar, M. (2020). Vime: Extending the success of self-and semi-supervised learning to tabular domain. *Advances in neural information processing systems*, *33*, 11033-11043.

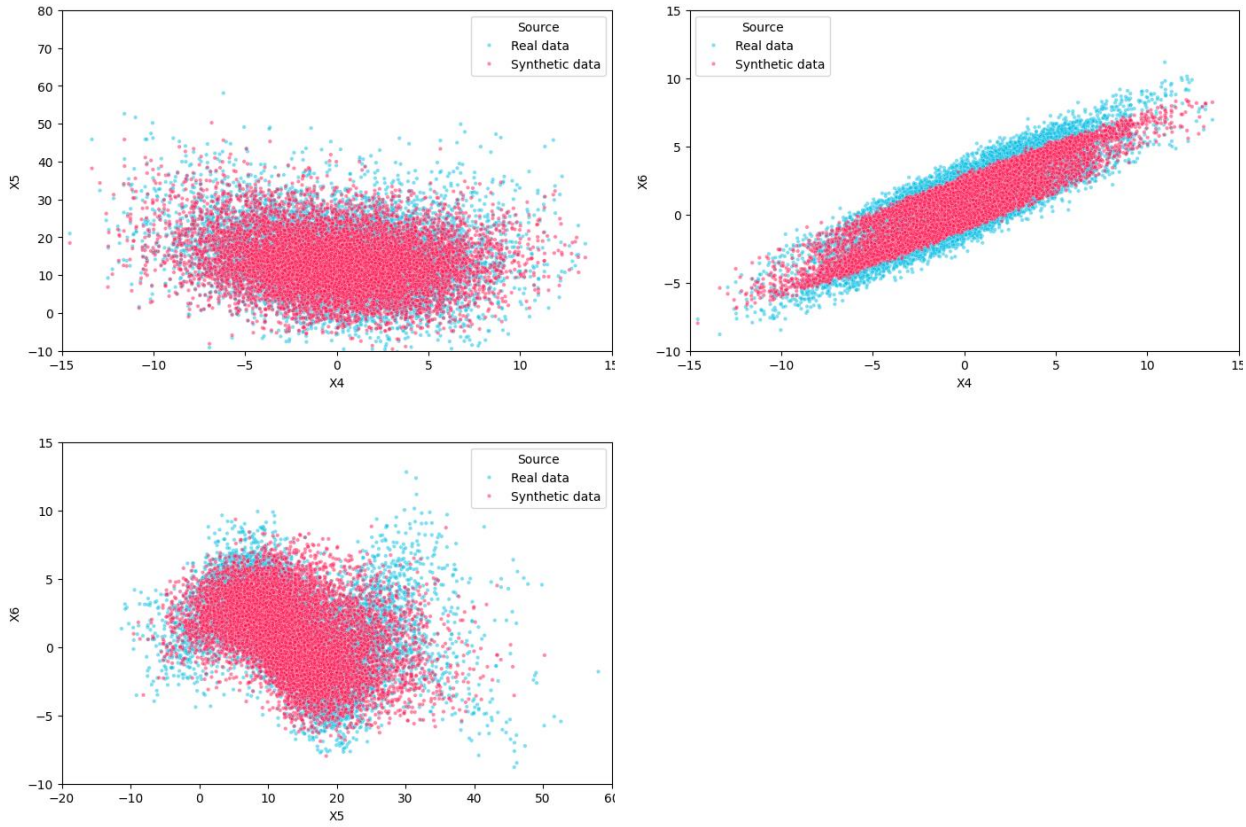
Appendix A Additional results

A.1 Additional result plots – simulated example

The plots below are real and synthetic data arising from the VAE, equivalent to the plots presented for the GAN in Figure 4.1.

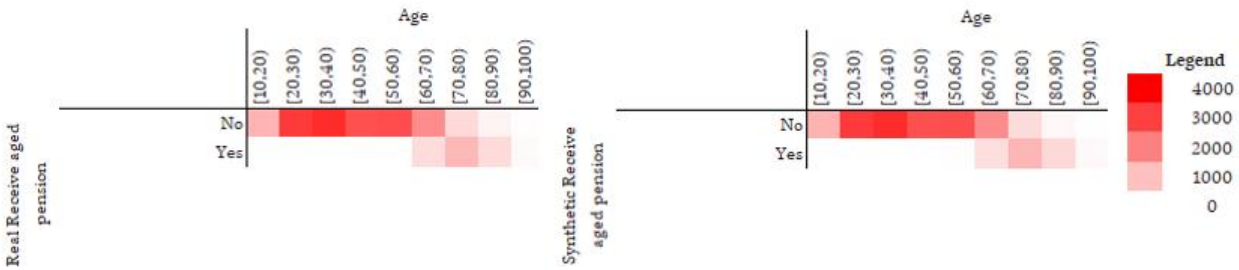
Figure A.1 – Scatter plots: original data / generated data from VAE





A.2 Additional result plots – HILDA VAE fit (single time step)

Figure A.2 – Receipt of age pension by age



Like the GAN, the VAE is able to capture the usage of the age pension from the (60, 70] age bracket upwards.

Figure A.3 – Wages by labour force status

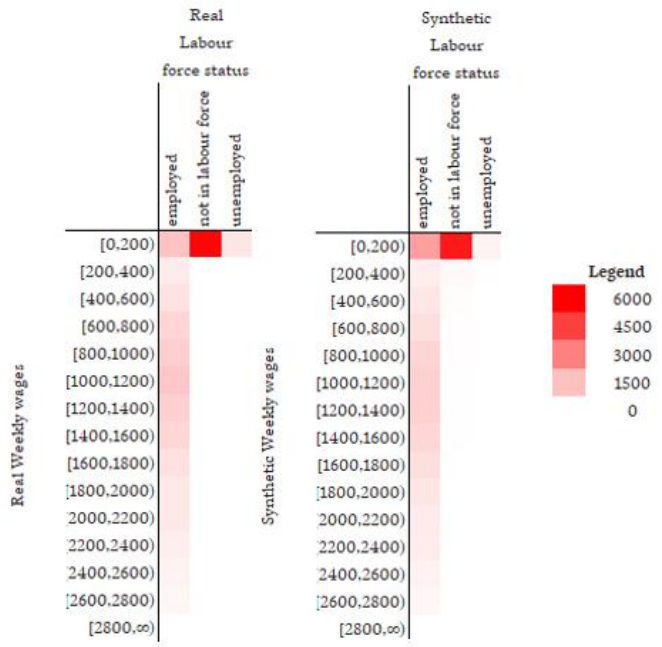


Figure A.4 – Weekly wages and total gross income (including transfers/welfare)

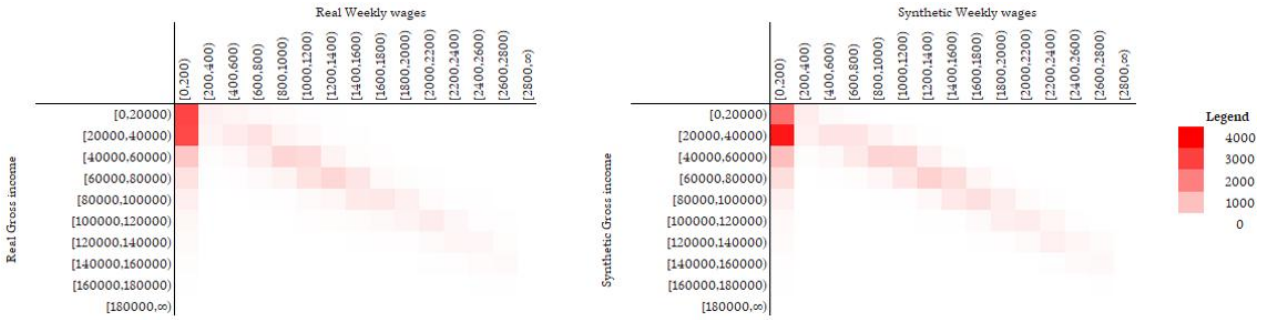


Figure A.5 – Weekly wages and transfers (excluding family tax benefit)

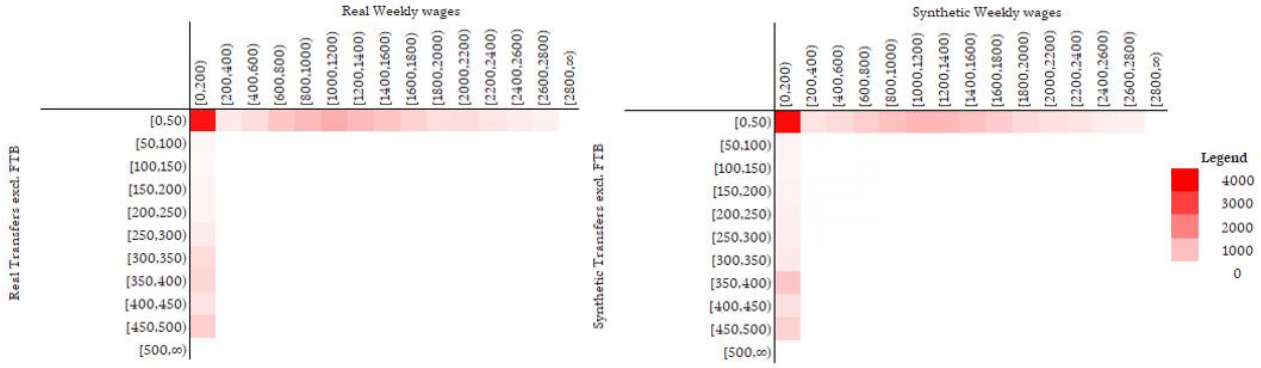


Figure A.6 – Gains chart for the chained three-year VAE model (gains ratio 0.782)

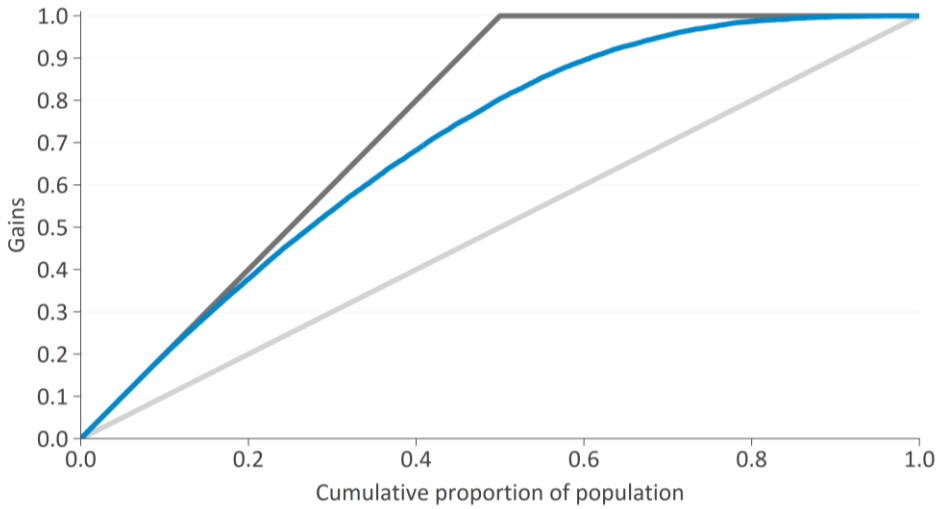


Figure A.7 – Gains chart for single step three-year VAE model. Gains ratio is 0.778

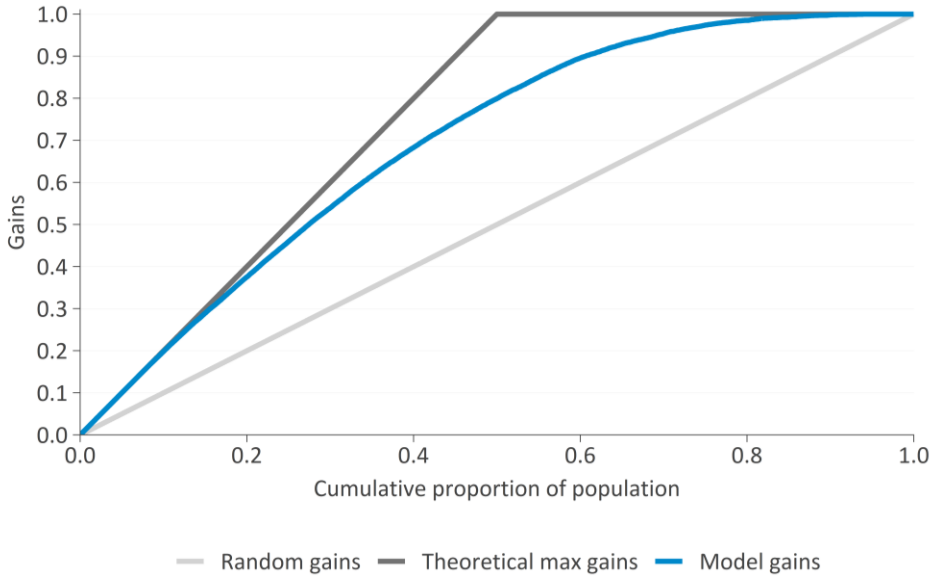


Figure A.8 – Frequency polygon and quantiles for total annual income, single-step three-year VAE model

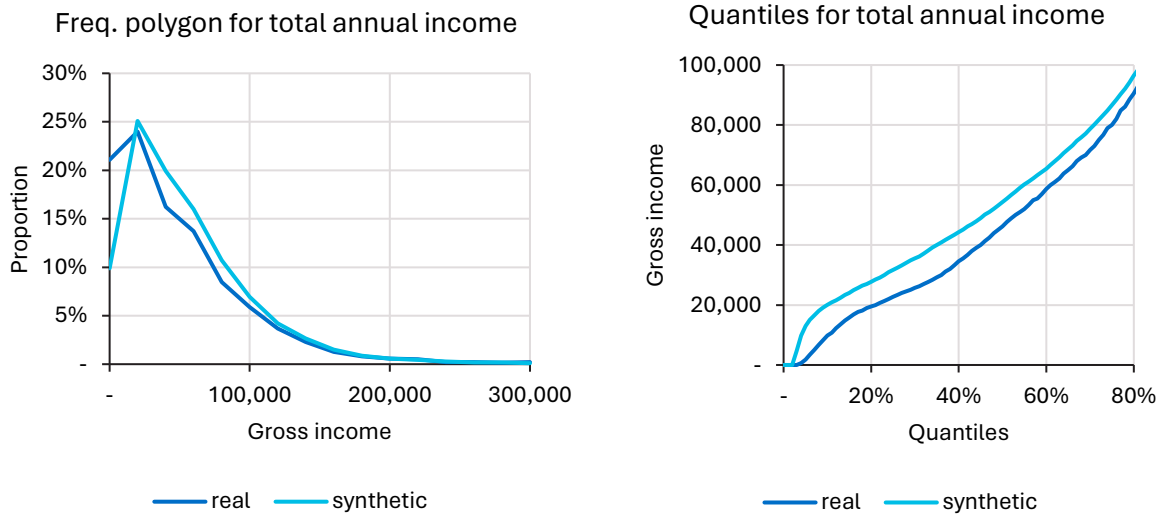
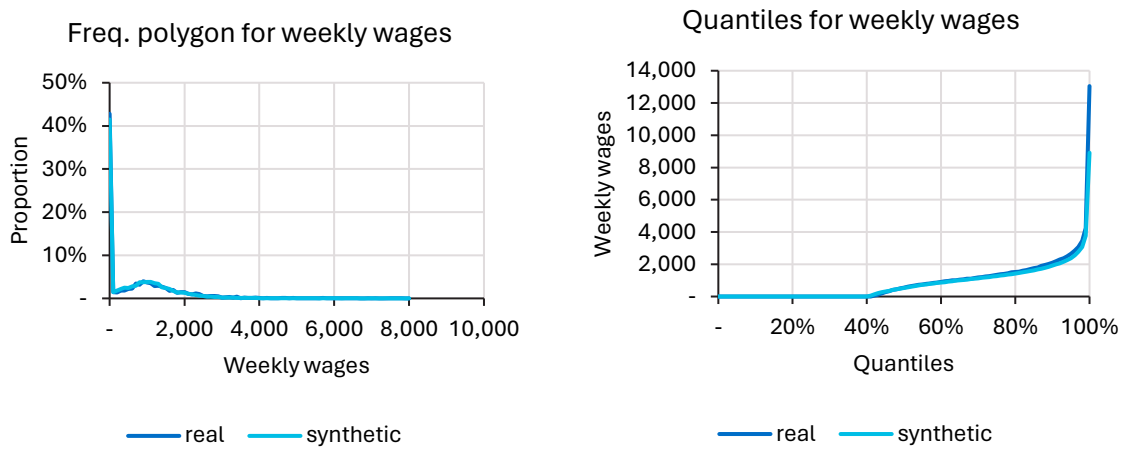


Figure A.9 – Frequency polygon and quantiles for weekly wages, single-step three-year VAE model



A.3 Additional timings

Training time by number of Epochs

Table A.1 –Model training time for different numbers of epochs

Number of training epochs	GAN training time (minutes)	VAE training time (minutes)
200	28.3	22.9
500	71.7	69.1
1000	143.7	138.5

Both the GAN and VAE take a substantial amount of time to train, despite the relatively small size of the dataset used. However, this may be sped up using improved hardware.

Training time by number of layers

We measured the time taken to train models with differing numbers of layers (in the GAN generator and in the VAE encoder networks respectively). This is shown in **Error! Reference source not found.** and REF _Ref196142893 \h **Error! Reference source not found.**. In general, we find there are small increases in training time when the number of layers is increased.

Table A.2 – GAN training time for differing numbers of generator layers

Number of generator layers	Training time (minutes)
3 (linear, ReLU, linear)	56.4
5 (linear, ReLU, linear, ReLU, linear)	57.2
7 (linear, ReLU, linear, ReLU, linear, ReLU, linear)	57.5

Table A.3 – VAE training time for differing numbers of encoder layers

Number of generator layers	Training time (minutes)
6 (linear, BatchNorm1d, ReLU, linear, BatchNorm1d, Tanh)	16.4
9 (linear, BatchNorm1d, ReLU, linear, BatchNorm1d, ReLU, linear, BatchNorm1d, Tanh)	16.6
12 (linear, BatchNorm1d, ReLU, linear, BatchNorm1d, ReLU, linear, BatchNorm1d, ReLU, linear, BatchNorm1d, Tanh)	17.1