PyData Global

Harnessing Generative Models for Synthetic Non-Life Insurance Data

Claudio Giorgio Giancaterino

09/12/2025

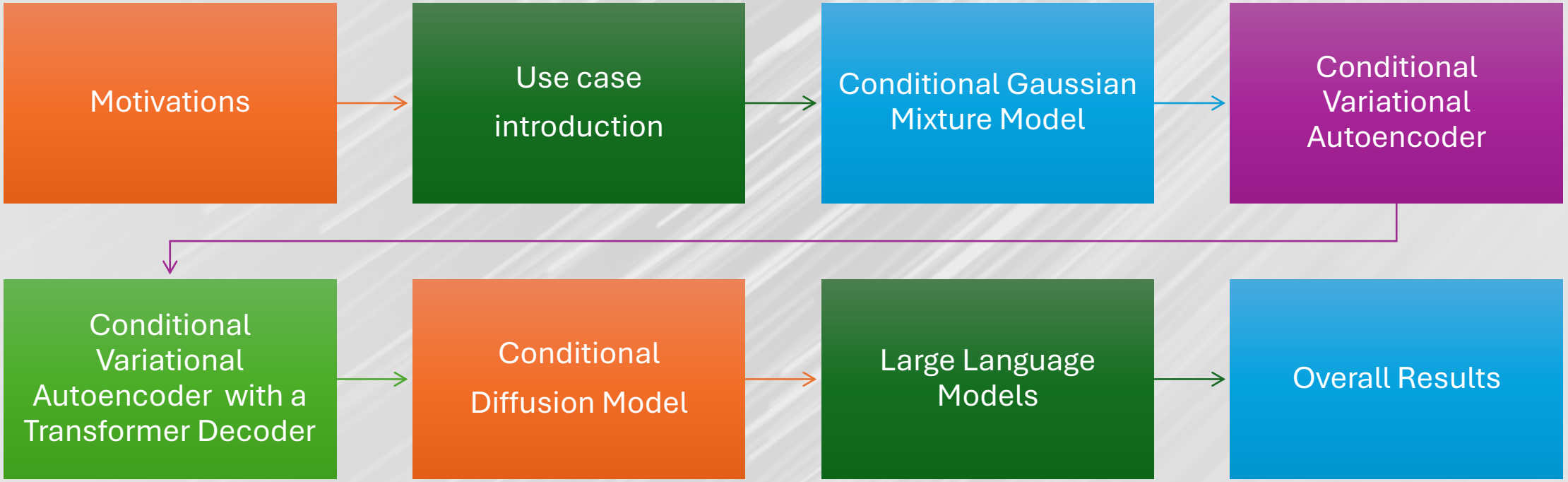
MySelf:

Actuary during the day &
AI Scientist in the free time

Reach Me:

MyLinks

Agenda


Motivations
Use case introduction
Conditional Gaussian Mixture Model
Conditional Variational Autoencoder
Conditional Variational Autoencoder with a Transformer Decoder
Conditional Diffusion Model
Large Language Models
Overall Results

Motivations

- Insurance Use Case
- Employ a wide range of Generative Models

Google Nano Banana

Data Scarcity in Insurance Research

The Problem: Access to realistic datasets is a major barrier to advancing actuarial research and developing insurance analytics tools.

- Most insurance data is confidential and proprietary
- Data masking and bureaucratic processes prevent disclosure
- Limited open resources lack diversity for robust modelling

The Solution: Develop simulated datasets using generative models

The Anatomy of Insurance Non-Life Risk Data: Statistical Components in the Pricing dataset

Risk Premium Components:

Frequency = Number of Claims / Number of Exposures

Severity = Losses / Number of Claims

Risk Premium = Frequency × Severity

The risk premium equals the product of expected claim frequency and expected cost per claim

Datasets employed

Datasets are retrieved from the "CASdatasets" R Package

Dataset 1: ausprivauto0405 - Automobile claim datasets in Australia
It is a data frame of 9 columns and 67,856 rows:
Exposure, VehValue, VehAge, VehBody, Gender, DrivAge, ClaimOcc, ClaimNb, ClaimAmount

Dataset 2: swmotorcycle - Swedish Motorcycle Insurance dataset
It is a data frame of 9 columns and 64,548 rows:
OwnerAge, Gender, Area, RiskClass, VehAge, BonusClass, Exposure, ClaimNb, ClaimAmount

Unlocking Data Quality: Leveraging Claim Occurrence for Stable Generative Modelling

What is ClaimOcc ?

It is a binary indicator which indicates occurrence of a claim

If ClaimOcc = 0 → ClaimNb = 0 and ClaimAmount = 0

If ClaimOcc = 1 → ClaimNb ≥ 1 and ClaimAmount > 0

Why use ClaimOcc?

Structural relationship: drives logical dependency between ClaimNb and ClaimAmount

Class imbalance: separates rare claim events from common no-claim cases

Model stability: improves synthetic data quality by conditioning

Role in Generative Models

CGMM: Fits separate Gaussian mixture models for each ClaimOcc value

CVAE & CTVAE: Concatenates ClaimOcc to the latent encoding for conditional generation

CDM: Uses ClaimOcc as a condition signal to guide the denoising process

Synthetic Data Generation Trials

Trial 1: 80% Training → 80% Generation
Baseline Quality Assessment
What is the baseline quality of synthetic insurance data when a generative model is trained on 80% of the original data? How well does it preserve statistical distributions and enable accurate GLM predictions?

Trial 2: 60% Training → 80% Generation
Data Augmentation with Limited Training
Can a generative model trained on only 60% of the data successfully generate synthetic samples to reach 80% dataset size while maintaining statistical fidelity and predictive performance?

Trial 3: 80% Gender-Masked → 80% Generation
Fairness-Aware Generation
Can a gender-masked generative model generate unbiased synthetic insurance data while maintaining predictive utility, and what is the fairness-accuracy trade-off?

Conditional Gaussian Mixture Model (CGMM)

What is it?
A Gaussian Mixture Model assumes that the data are generated by a mixture of K Gaussian distributions, each with unknown parameters and corresponding to a cluster. Every Gaussian has its own mean μ_k, covariance Σ_k, and mixing weight π_k.

Gaussian Components:
- **Mean (μ_k):** Center of each Gaussian component
- **Covariance (Σ_k):** Spread and orientation of the cluster
- **Mixing weights (π_k):** represent the contribution of each component to the overall mixture

How does it work?
The Expectation-Maximization (EM) algorithm iteratively optimizes model parameters
-**Initialization:** Start with initial guesses for the parameters (μ_k, Σ_k, π_k).
-E-Step: Calculate the probability that each data point belongs to each Gaussian component.
-M-Step: Update model parameters (μ_k, Σ_k, π_k) to maximize the likelihood of the data.
-Repeat until convergence
Once fitted to observations, GMM becomes a generative probabilistic model.

Switch to Conditional Gaussian Mixture Model
Use **ClaimOcc** as the binary conditioning variable (0 = no claim, 1 = claim).
CGMM fits separate models for claim/no-claim groups, preserving claim occurrence patterns.

Conditional Variational Autoencoder (CVAE)

What is it?
A Variational Autoencoder (VAE) is a type of neural network that learns to represent input data as a probability distribution in a lower-dimensional space and then decode it to generate similar input data.

Components:
- **Encoder (Inference Network):** maps input data into a latent space
- **Latent Space:** a compressed representation of the input data
- **Decoder (Generative Network):** reconstruct the original input from the latent space

How does it work?
-Input data is fed into the **encoder**, producing two vectors for each feature: the mean and standard deviation.
-A **latent vector** is sampled using the reparametrization trick.
-The **decoder** comes back to the input, taking a sample from the compressed representations. After training, the decoder can be used to generate new, similar data.
The model is trained to minimize:
-**Reconstruction Loss**
-**KL Divergence**

Switch to Conditional Variational Autoencoder
Use **ClaimOcc** as the binary conditioning input (0 = no claim, 1 = claim). The CVAE learns claim-dependent latent structure, enabling the decoder to generate synthetic samples consistent with claim occurrence.

Conditional Variational Autoencoder with a Transformer-based Decoder (CTVAE)

What is it?
It's a hybrid architecture that leverages:
-VAEs for learning a smooth, continuous latent space.
-Transformers for decoding sequences with long-range dependencies and attention mechanisms.

Components:
- **Encoder (Inference Network):** maps input data into a latent space
- **Latent Space:** a compressed representation of the input data
- **Transformer Decoder:** decodes the latent representation into a reconstructed input

How does it work?
-Input data is fed into the **encoder**, producing two vectors for each feature: the mean and standard deviation.
-A **latent vector** is sampled using the reparametrization trick.
-The **Transfomer decoder** comes back to the input, taking a sample from the compressed representations. It uses self-attention to generate output sequences.
The model is trained to minimize:
-**Reconstruction Loss**
-**KL Divergence**

Switch to Conditional Variational Autoencoder with a Transformer-based Decoder
Uses **ClaimOcc** as binary conditioning variable.
During training, it's concatenated with input features for encoding. During decoding, **ClaimOcc** is concatenated with the sampled latent vector, guiding the transformer decoder to generate realistic synthetic data consistent with the claim status

Conditional Diffusion Model (CDM)

What is it?
A Diffusion Model learns to generate data by simulating a gradual noising-and-denosing process.

Components:
- **Forward process (Diffusion):** gradually adds noise to data over time, turning it into pure noise.
- **Reverse Process (Denoising):** learns to remove noise step-by-step to recover the original data
- **Neural Network:** predicts the noise at each step (typically U-Net architecture)

How does it work?
The forward phase gradually corrupts clean data by adding noise via a Markov chain of steps, transforming the original data into Gaussian noise.
In the reverse phase, starting from pure noise, the model trains a neural network to predict and subtract noise at each step, and generate samples that match the data distribution.
The model is trained to minimize the difference between the actual noise and the predicted noise at each step.

Switch to Conditional Diffusion Model
Uses **ClaimOcc** (claim occurrence) as a binary conditioning variable to control the generation process, ensuring synthetic samples maintain logical consistency with the claim/no-claim structure

Large Language Model (LLM)

What is it?
Language Models can be defined as a probability distribution over sequences of words.
Given a vocabulary of words, a Language Model assigns a probability to each sequence of words, and the purpose is to predict the next word in the sequence.
Large Language Models like BERT and GPT are trained on a vast amount of text data, to learn the structure, meaning, and usage of language.

Components:
- **Transformer architecture:** the backbone of most LLMs. It's an encoder-decoder network based on the self-attention mechanism to weigh the importance of different words in a sentence, allowing the model to understand context more effectively.

How does it work?
-Tokenization: Convert words to numbers
-Token Embeddings: convert a token into a vector
-Positional Embeddings: calculate the position of each token in the sequence of tokens
-Self-Attention: assign weights to every word relative to every other word
-Feed-Forward Network: predicts the next word by assigning scores for every word
-Softmax Layer: convert scores into probabilities
-De-Tokeinzation: convert token into word

How is it employed in synthetic data generation?
By a detailed prompt in the ChatGPT platform with the distribution of data, describing some relationships, and providing a few record examples.

Validation by Consistency records

```python
# Find inconsistencies
inconsistent_records = new_samples_df[
    ~(((new_samples_df["ClaimNb"] == 0) & (new_samples_df["ClaimOcc"] == 0) & (new_samples_df["ClaimAmount"] == 0)) |
      ((new_samples_df["ClaimNb"] > 0) & (new_samples_df["ClaimOcc"] > 0) & (new_samples_df["ClaimAmount"] > 0)))
]

print(f"Number of inconsistent records: {len(inconsistent_records)}")
print(inconsistent_records.head())  # Show a few inconsistent rows
```

```
Number of inconsistent records: 1
       Exposure  VehValue  VehAge  VehBody  Gender  DrivAge  ClaimNb  \
49759  0.736911  5.434924       3        6       1        6      1.0

       ClaimAmount  ClaimOcc
49759          0.0         1
```

The consistency test validates the **logical relationships** between three insurance claim variables in the synthetic data:

-ClaimNb (Number of claims)
-ClaimOcc (Claim occurrences)
-ClaimAmount (Claim amount)

No Claims: All three = 0 (no claims occurred)
Claims Exist: All three > 0 (claims occurred with positive values)

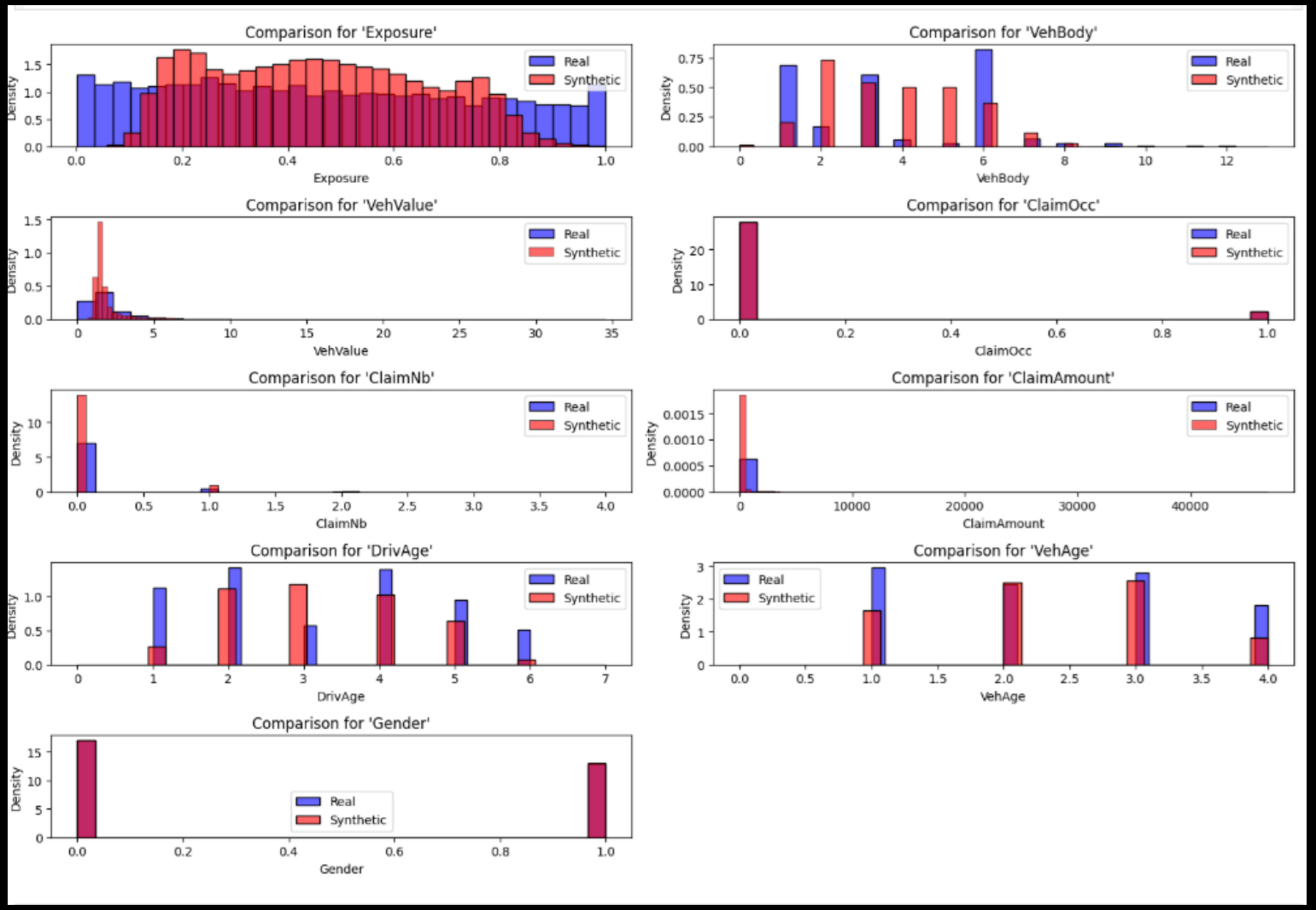
Validation by Kolmogorov-Smirnov Test

```
]:  # Kolmogorov-Smirnov test
    for column in X_train.columns:
        original = X_train[column].values
        generated = new_samples_df[column].values
        statistic, p_value = ks_2samp(original, generated)
        print(f"KS Test for {column}: Statistic={statistic}, P-value={p_value}")
```

```
KS Test for Exposure: Statistic=0.13863465866466618, P-value=0.0
KS Test for VehValue: Statistic=0.2830457614403601, P-value=0.0
KS Test for VehAge: Statistic=0.07691297824456114, P-value=1.3001617749392413e-137
KS Test for VehBody: Statistic=0.20864591147786948, P-value=0.0
KS Test for Gender: Statistic=0.001444111027756878, P-value=0.9999999970492804
KS Test for DrivAge: Statistic=0.12738184546136533, P-value=0.0
KS Test for ClaimOcc: Statistic=0.0, P-value=1.0
KS Test for ClaimNb: Statistic=0.0017254313578394243, P-value=0.9999982405003918
KS Test for ClaimAmount: Statistic=0.016185296324081055, P-value=1.6976741222647111e-06
```

The Kolmogorov-Smirnov (KS) test compares the distributions of original and synthetic data for each feature. A **high p-value (>0.05)** indicates that the two distributions are statistically similar, while a **low p-value (<0.05)** suggests significant differences.

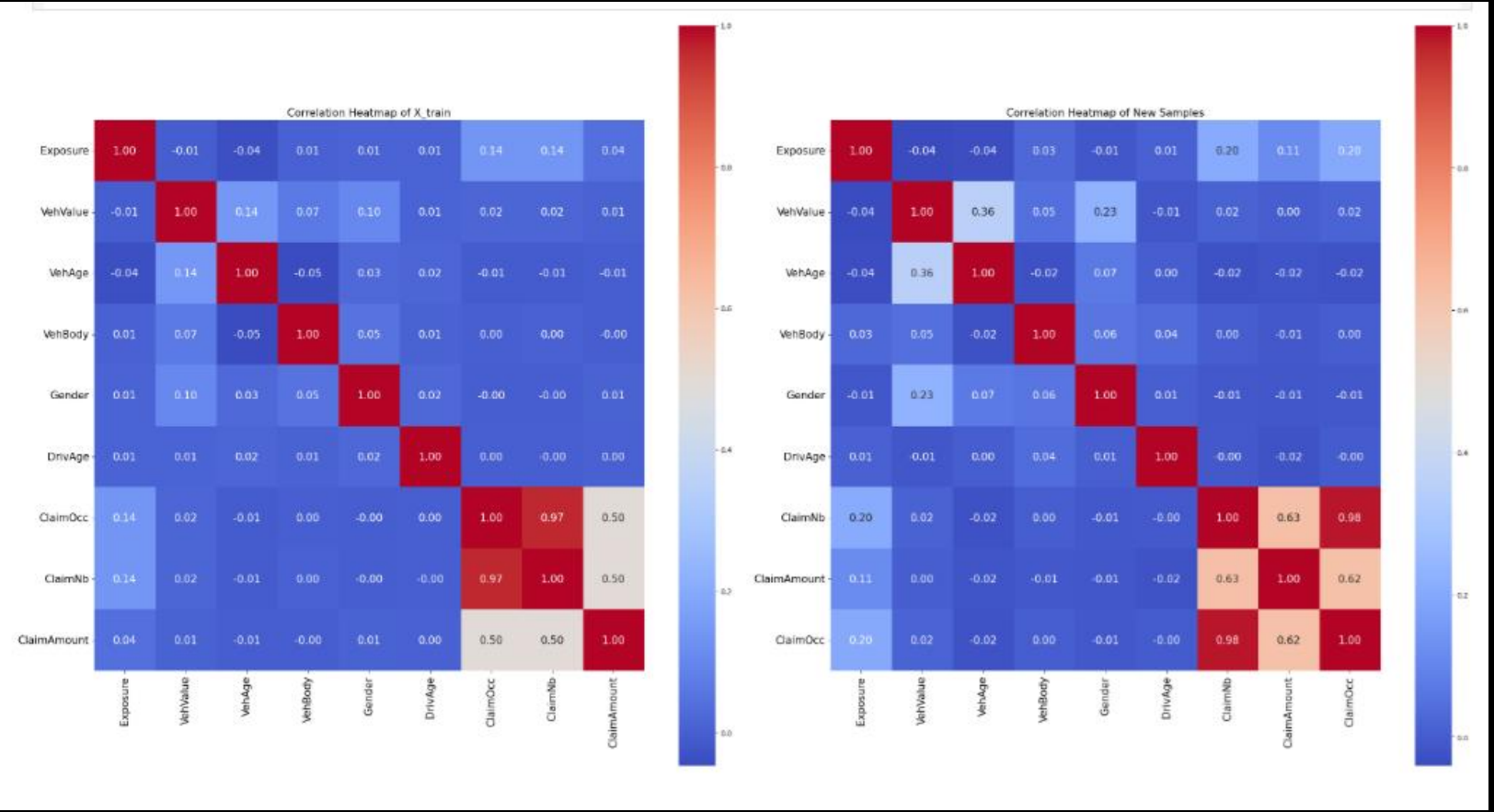
Validation by Data Visualization



Univariate Analysis

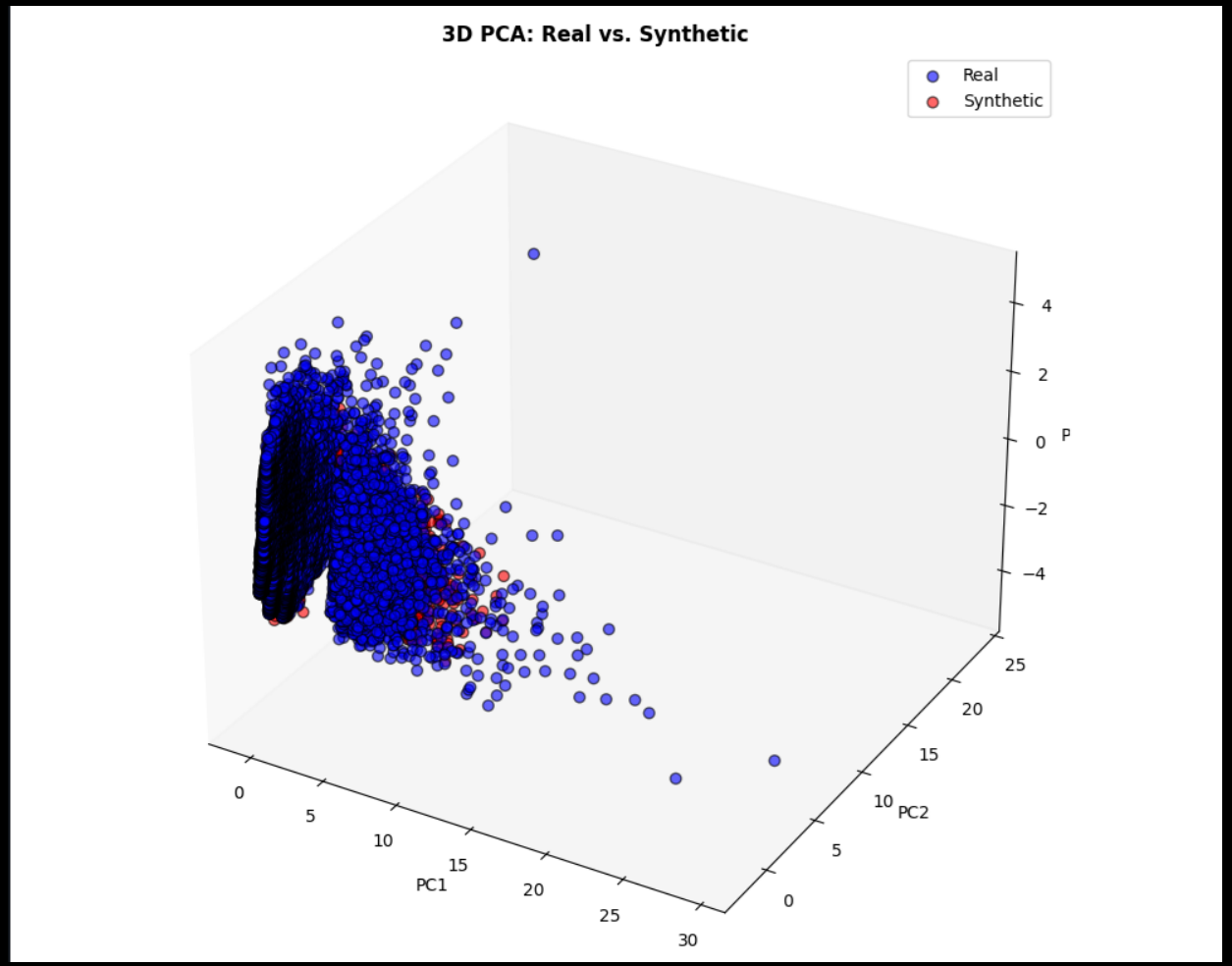
Displays the alignment of the univariate statistical properties for each feature across the real and synthetic data

Validation by Data Visualization



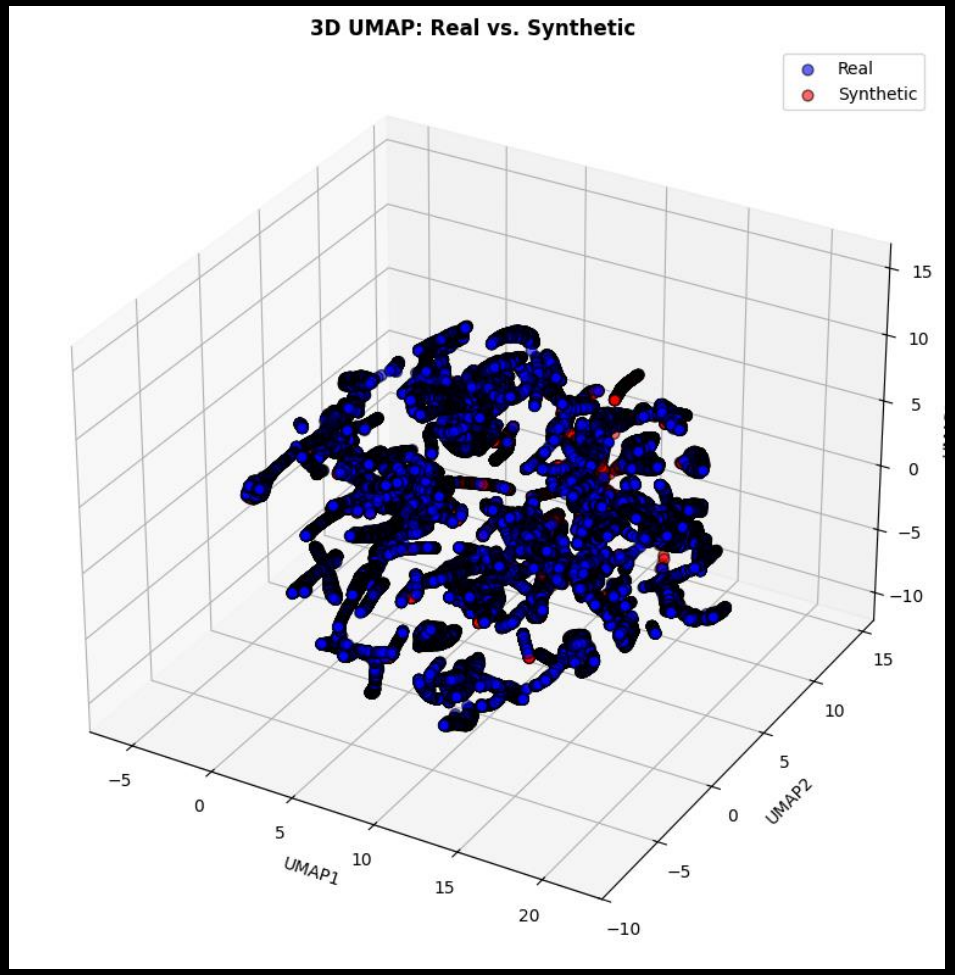
Correlation Matrix
Assesses the structural fidelity of the synthetic data by comparing the correlation matrix to the original data

Validation by Data Visualization



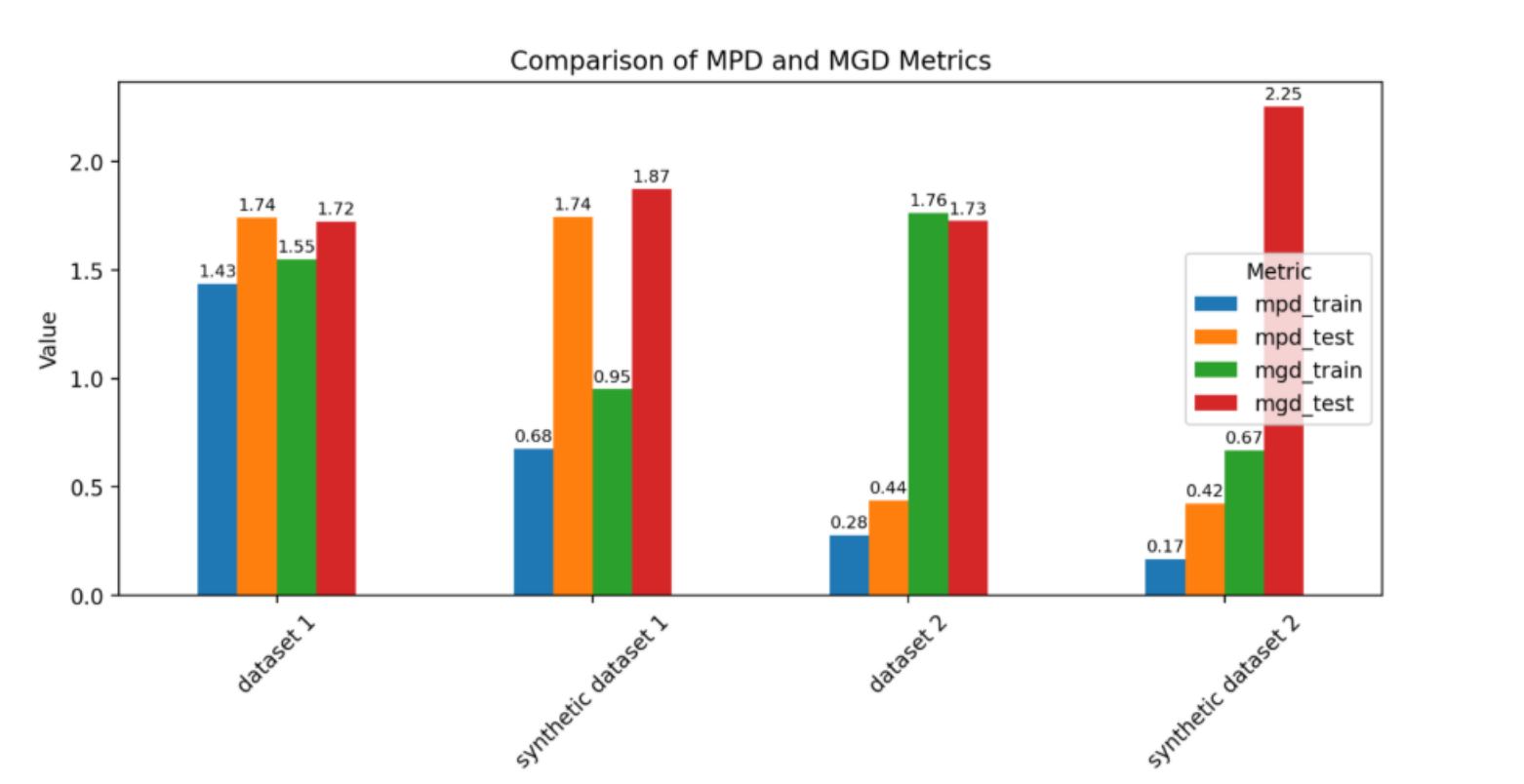
3D PCA
Compares the explained variance captured by the principal components of the synthetic data against the real data

Validation by Data Visualization



3D UMAP
Evaluates the degree to which the synthetic data preserves the local and global topological structure of the original data

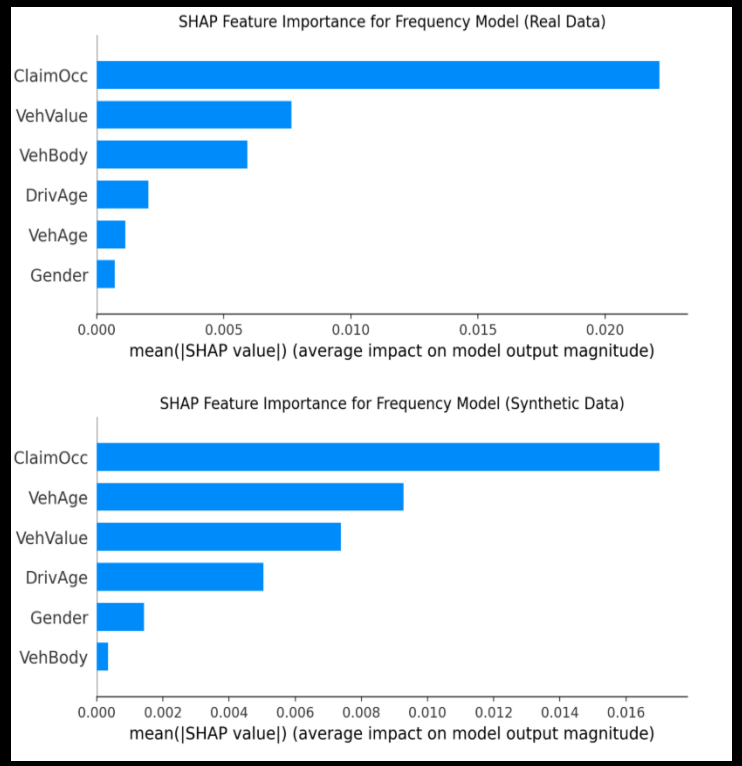
Validation by Predictive Modelling



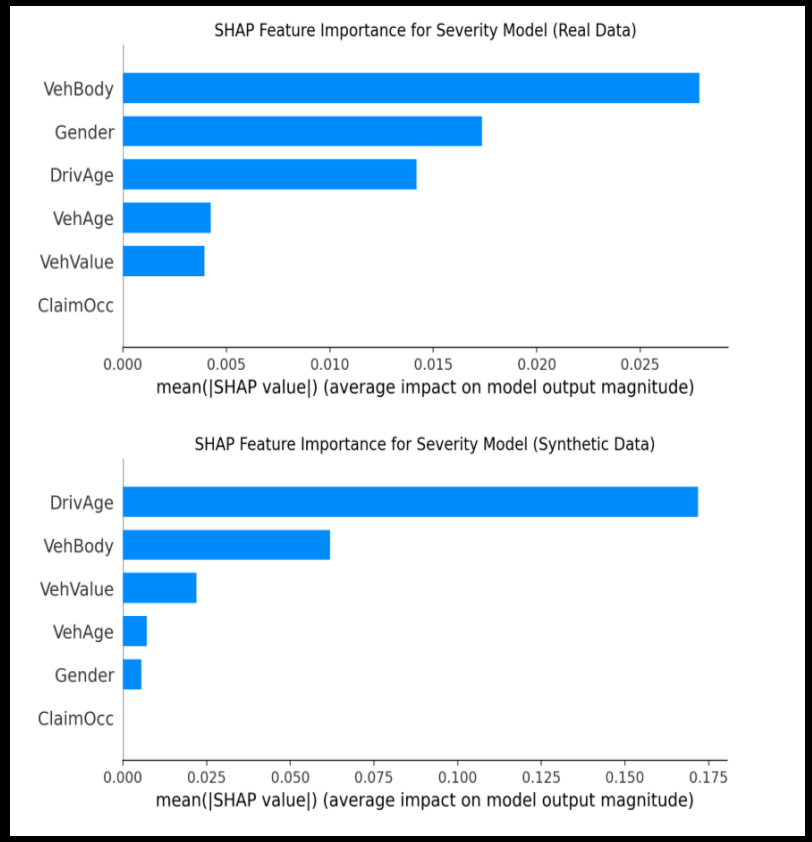
Frequency and Severity prediction

GLM on both synthetic and real data to evaluate performance using deviance metrics.

Validation by Feature Importance



SHAP Feature Importance
Shows whether drivers of frequency and severity predictions remain consistent across datasets.



Overall Results

Focusing on the performance of predictive modelling and on preserving the statistical properties of the Kolmogorov-Smirnov test and the correlation matrix from trial 1, I liked the Conditional Gaussian Mixture Model, the Large Language Model, and the Conditional Diffusion Model. The two Conditional Variational Autoencoders produce overfitting in predictive modelling. In subsequent trials, all models degrade in performance, but in some cases, they remain acceptable. What the results in the demo web app.

Conclusions

What I've learned

- ✅ Conditional Gaussian Mixture Models are competitive as generative models for tabular data. Good compromise between ease of use and effectiveness.
- ✅ Large Language Models are promising for synthetic data generation due to their performance and ease of use. Complex model implementation isn't required, but detailed prompting and deep data analysis are needed to write a good prompt.
- ✅ Conditional Diffusion Model is competitive in performance, but it requires computational and coding effort.

References

-Jan Goodfellow and Yoshua Bengio and Aaron Courville, 2016, *Deep Learning*, MIT Press.
-Mario V. Wuthrich, Ronald Richman, Benjamin Avanzi, Mathias Lindholm, Michael Mayer, Jürg Schelldorfer, Salvatore Scognamiglio, 2025, *AI Tools for Actuaries*, SSRN.
-David Foster, 2023, *Generative Deep Learning, 2nd Edition*, O'Reilly.
-Jake VanderPlas, 2016, *Python Data Science Handbook*, O'Reilly.
-Jamotton, Charlotte; Hainaut, Donatien, 2023, *Variational autoencoder for synthetic insurance data*, ISBA.
-Harshvardhan GM, Mahendra Kumar Gourisaria, Manjusha Pandey, Siddharth Swarup Rautaray, 2020, *A comprehensive survey and analysis of generative models in machine learning*, ScienceDirect.
https://github.com/claudio1975/Generative_Modelling
https://huggingface.co/spaces/towardsinnovationlab/Generative_Models_4_Insurance_Data
https://medium.com/@c.giancaterino/stop-waiting-for-data-how-generative-models-are-reshaping-insurance-analytics-ec102a2e5177

?

?

?

?

?

?

?

Thank you

Keep in touch:

- Linkedin
- Newsletter
- Medium
- Website