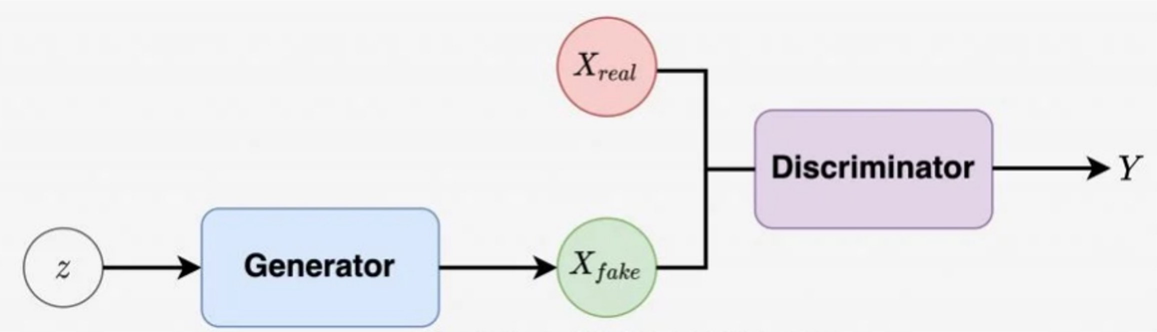
Generating Synthetic Tabular Data with GANs

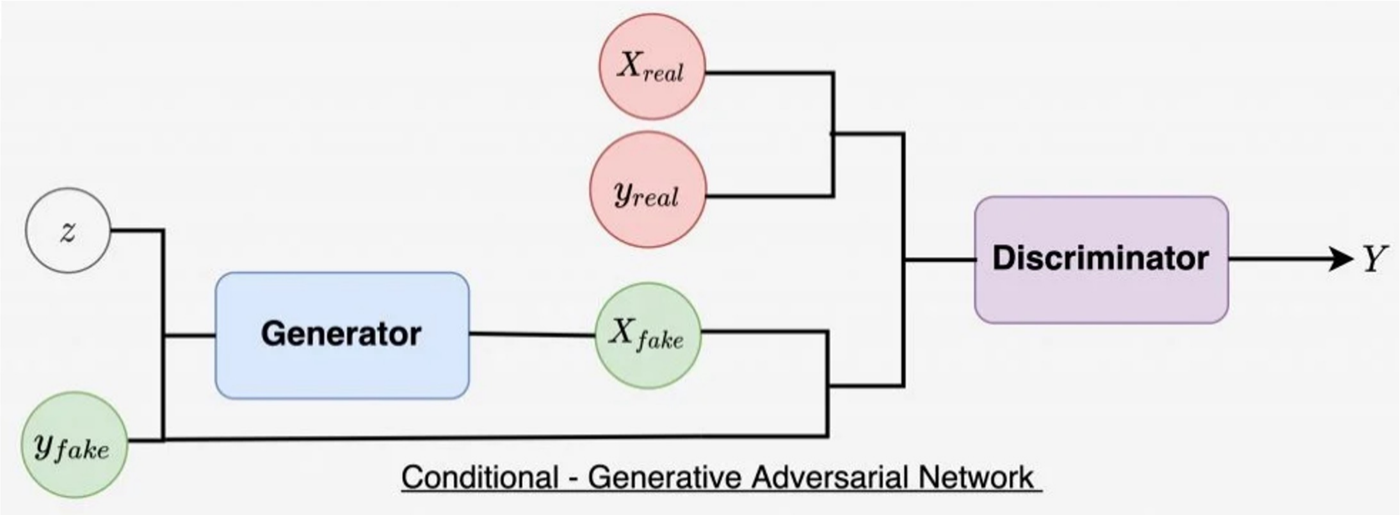
GANs vs cGAN



GANs

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

cGAN



Conditional - Generative Adversarial Network

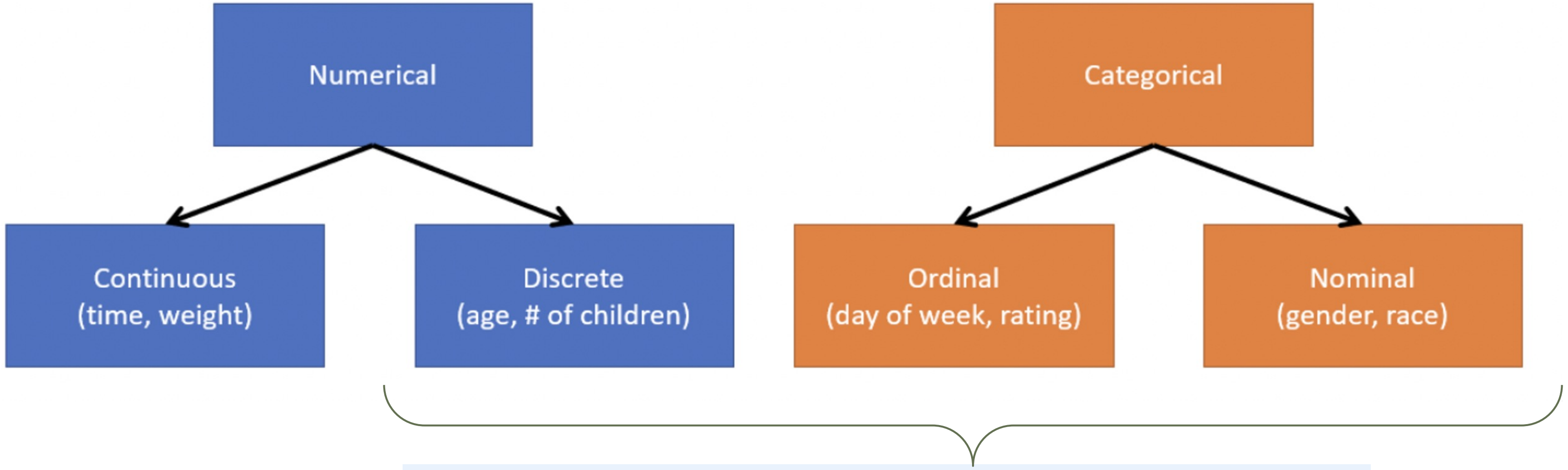
$$\mathop{\mathbb{E}}_{x\sim p_{data}(x)}[\log D(x,y)] + \mathop{\mathbb{E}}_{z\sim p_z(z)}[\log(1 - D(G(z,y),y))]$$

Why synthetic data?

- **Data Privacy:** Synthetic data is a great way to ensure data privacy while being able to share microdata, allowing organizations to share sensitive and personal (synthetic) data without concerns with privacy regulations
- **Prototype Development**: Collecting and modeling tremendous amounts of real data is a complicated and tedious process. Generating synthetic data makes data available sooner. Besides that, it can help in faster iteration through the data collections development for ML initiatives
- **Edge-case Simulation**: It is often seen that the collected data do not contain every possible scenario which affects the model performance negatively. In such cases, we can include those rare scenarios by artificially generating them

Challenges

- **Mixed data types:** Numerical data, Categorical data (ordinal, low cardinality, etc.) , Text, Boolean



Numerical

Continuous
(time, weight)

Discrete
(age, # of children)

Categorical

Ordinal
(day of week, rating)

Nominal
(gender, race)

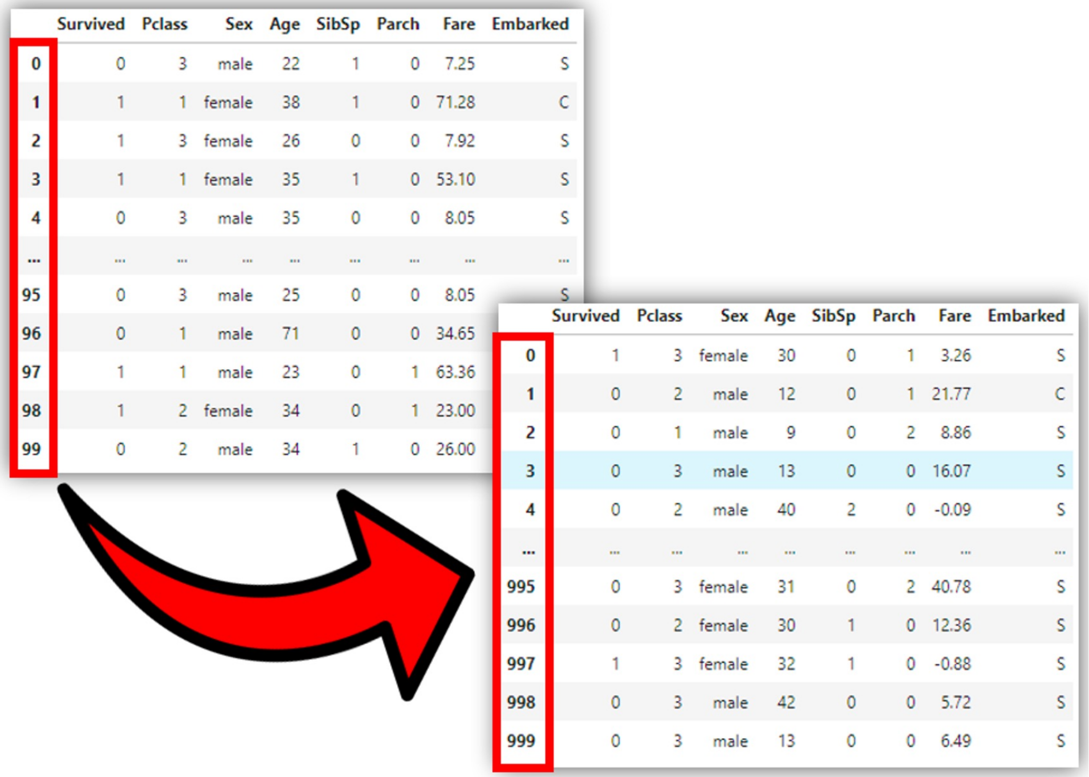
discrete data → represented as vectors {0,0,1,...}

- **Sparse data**
- **Unbalanced data**
- ...

Generating tabular data with GANs

- Problem statement:

  A tabular dataset *T* can be said to contain *Nd* discrete columns and *Nc* continuous columns. The goal of tabular data generation is to train a generator *G* to learn to generate a synthetic dataset *Tsynth* from *T.*



	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22	1	0	7.25	S
1	1	1	female	38	1	0	71.28	C
2	1	3	female	26	0	0	7.92	S
3	1	1	female	35	1	0	53.10	S
4	0	3	male	35	0	0	8.05	S
...	...	...	...	...	...	...	...	...
95	0	3	male	25	0	0	8.05	S
96	0	1	male	71	0	0	34.65	
97	1	1	male	23	0	1	63.36	
98	1	2	female	34	0	1	23.00	
99	0	2	male	34	1	0	26.00	

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	3	female	30	0	1	3.26	S
1	0	2	male	12	0	1	21.77	C
2	0	1	male	9	0	2	8.86	S
3	0	3	male	13	0	0	16.07	S
4	0	2	male	40	2	0	-0.09	S
...	...	...	...	...	...	...	...	...
995	0	3	female	31	0	2	40.78	S
996	0	2	female	30	1	0	12.36	S
997	1	3	female	32	1	0	-0.88	S
998	0	3	male	42	0	0	5.72	S
999	0	3	male	13	0	0	6.49	S

How to deal with tabular data

- Rows are treated as data samples
  - **One row** is **one data sample**
- **GANs** (unsupervised learning) used to **randomly** create samples (data is generated randomly)
- **cGANs** (supervised learning) used to create samples by **selecting a given category**
  - Categorical columns are used as the label

How to deal with tabular data

- Every columns should be defined as a **numerical (float) value**
  - Many machine learning algorithms perform better or converge faster when features are on a relatively similar scale and/or close to normally distributed.
  - It is important to find the right distribution/transformation that fits the data.

- **Scale:** changing the range of the values. The shape of the distribution doesn't change. The range is often set at 0 to 1.
- **Standardize:** changing the values so that the distribution's standard deviation equals 1. Scaling is often implied.
- **Normalize:** either of the above things (and more!)

How to deal with tabular data

- **sklearn.preprocessing** → Preprocessing
  - **StandardScaler** assumes your data is normally distributed within each feature and will scale them such that the distribution is now centred around 0, with a standard deviation of 1.

    $$\frac{x_i - mean(x)}{stdev(x)}$$

  - **MinMaxScaler** essentially shrinks the range such that the range is now between 0 and 1 (or -1 to 1 if there are negative values).

    $$\frac{x_i - min(x)}{max(x) - min(x)}$$

    - it is sensitive to outliers
  - **RobustScaler** uses a similar method to the MinMaxScaler but it instead uses the interquartile range

    $$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$

    - It is robust to outliers.
  - **Normalizer** scales each value by dividing each value by its magnitude in n-dimensional space for n number of features.

    $$\frac{x_i}{\sqrt{x_i^2 + y_i^2 + z_i^2}}$$

How to deal with tabular data

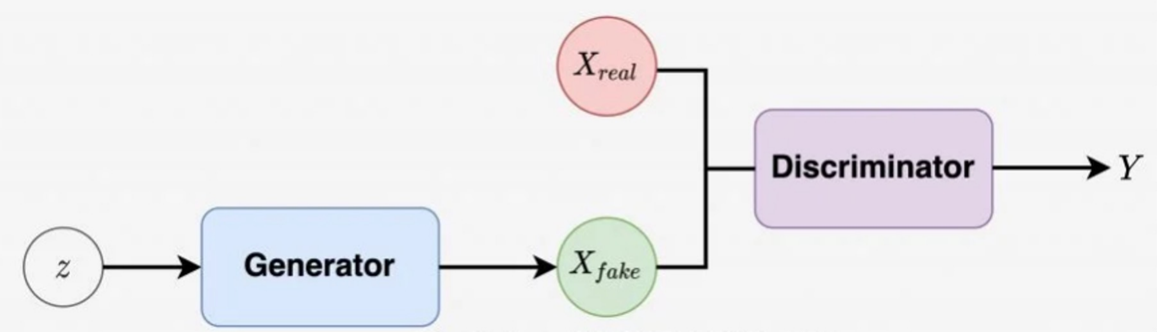
- **sklearn.preprocessing** → Preprocessing
  - **OneHotEncoder** convertes categorical variables are converted into a numerical representation.
    - Categorizing every category in a discrete variable into its own dimension.
    - It does not assume ordinal relationship

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
1	0	0	0	0	0	0
0	1	0	0	0	0	0
…	…	…	…	…	…	…

Monday = [1, 0, 0, 0, 0, 0, 0]

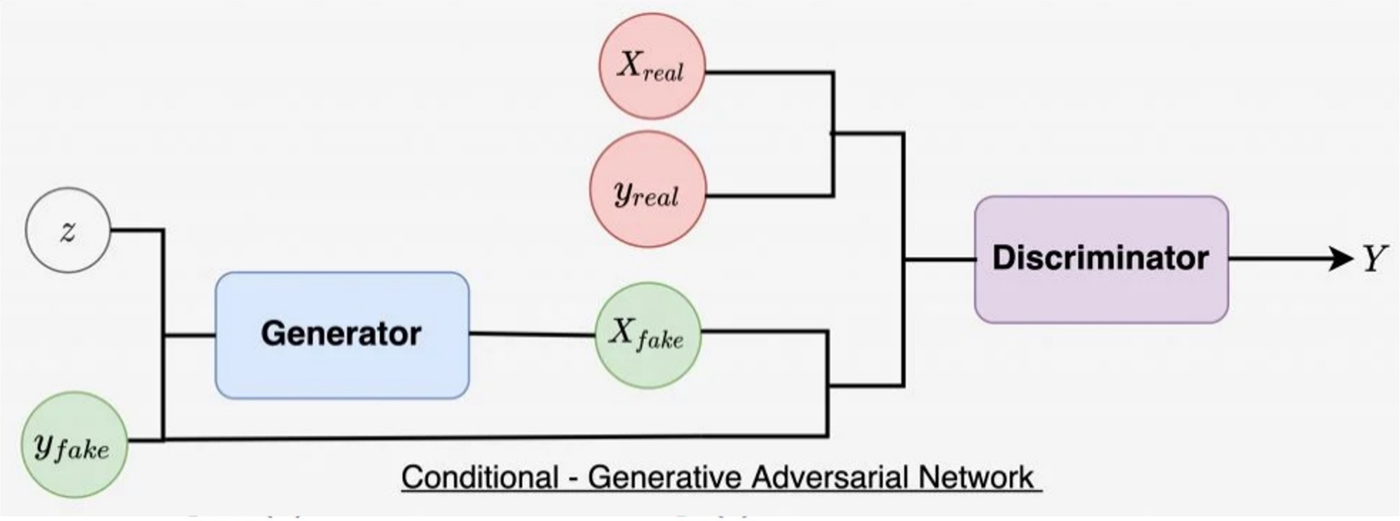
How to deal with tabular data

- Deal with the problem as we have already seen



GANs

cGAN



Conditional - Generative Adversarial Network

How to deal with tabular data

- Every columns should be defined as a **numerical (float) value**
  - It is important to find the right distribution/transformation that fits the data
- **sklearn.preprocessing** → Preprocessing and Normalization
  - **MinMaxScaler**
  - **Normalizer**
  - **OneHotEncoder**
  - **PowerTransformer**

CT-GAN

- To achieve the task of tabular data generation, one could train a vanilla GAN, however, there are **two adaptations** that CTGANs proposes that attempt to tackle two issues with GANs when applied to tabular data.
  - A representative normalization of continuous data
  - A fair sampling of discrete data

CT-GAN. Normalization of discrete data

- **Discrete data** is easy to represent → one-hot encoded.
  - One-hot encoding is simply the process of categorizing every category in a discrete variable into its own dimension.
  - For the weekdays (Monday, Tuesday, …., Sunday) instead of having a vector containing the day of the week, after one-hot encoding, we have 7 columns, one for each day of the week, with binary indications of class membership.

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
1	0	0	0	0	0	0
0	1	0	0	0	0	0
…	…	…	…	…	…	…

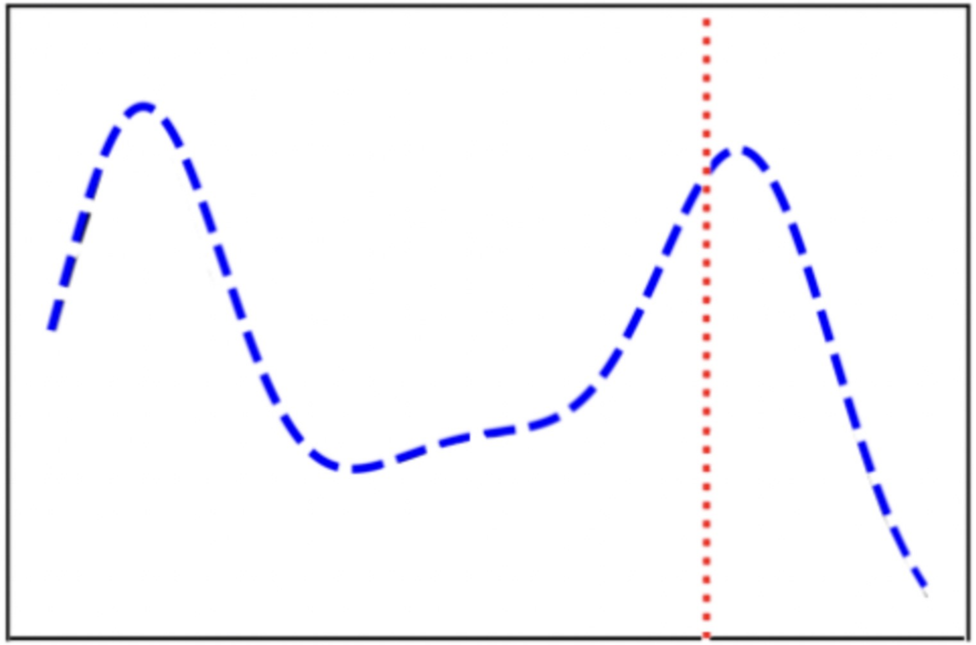
Monday = [1, 0, 0, 0, 0, 0, 0]

CT-GAN. Normalization of continuous data

- **Continuous data** is NOT SO easy to represent.
  - It is difficult to express all the information carried by the continuous variable..

We have a continuous variable like the one above (distribution in blue) and we want to represent our sample (in red).

- How can we normalize to be able to use the data in a ML model?
- Can we represent the distribution with a Normal distribution?

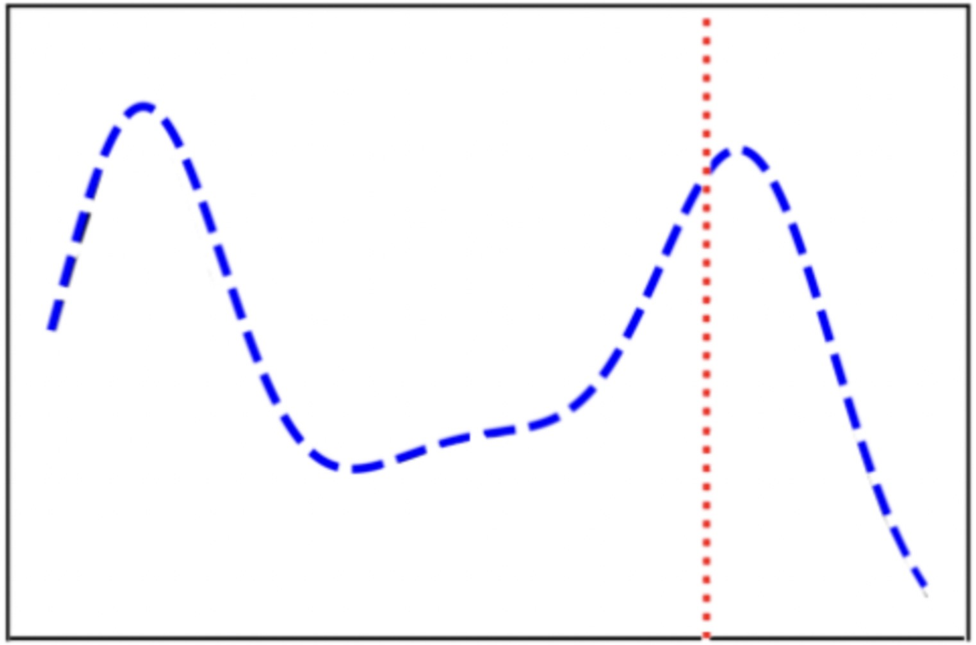


CT-GAN. Normalization of continuous data

- **Continuous data** is NOT SO easy to represent.
  - It is difficult to express all the information carried by the continuous variable..

The distribution is quite complex, it has multiple modes.

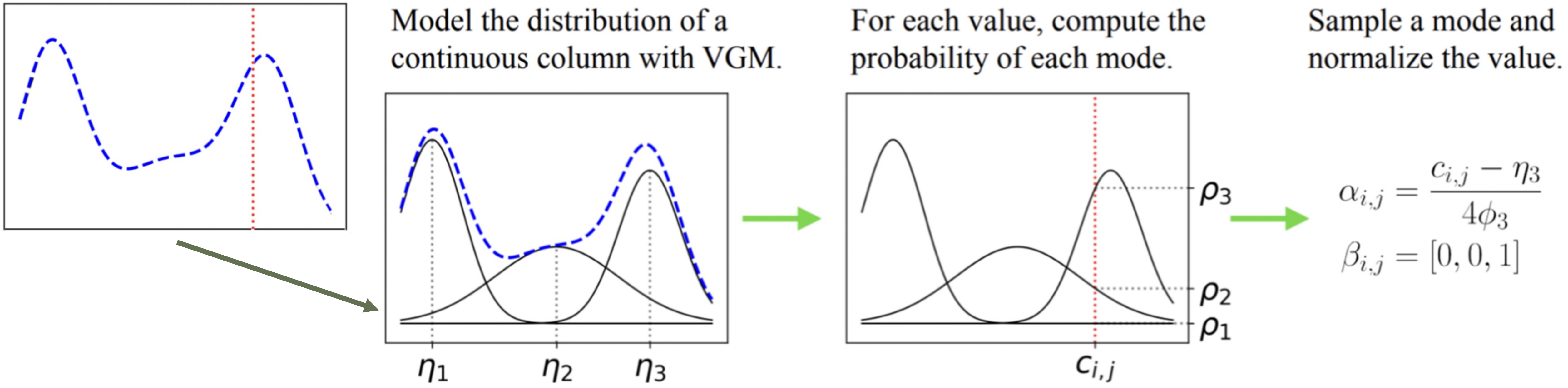
Therefore by simply giving the model the value of the continuous variable at our sample, we may lose some information, such as what mode the sample belongs to, and its importance within that mode.



- Solution: **mode-specific normalization**

CT-GAN. Normalization of continuous data

- **Mode-specific normalization** works by first fitting a VGM (variational Gaussian mixture model) to each continuous variable.
  - A gaussian mixture model simply tries to find the best k Gaussians to represent the data through expectation maximization.



Model the distribution of a continuous column with VGM.

For each value, compute the probability of each mode.

Sample a mode and normalize the value.

$$\alpha_{i,j} = \frac{c_{i,j} - \eta_3}{4\phi_3}$$
$$\beta_{i,j} = [0, 0, 1]$$

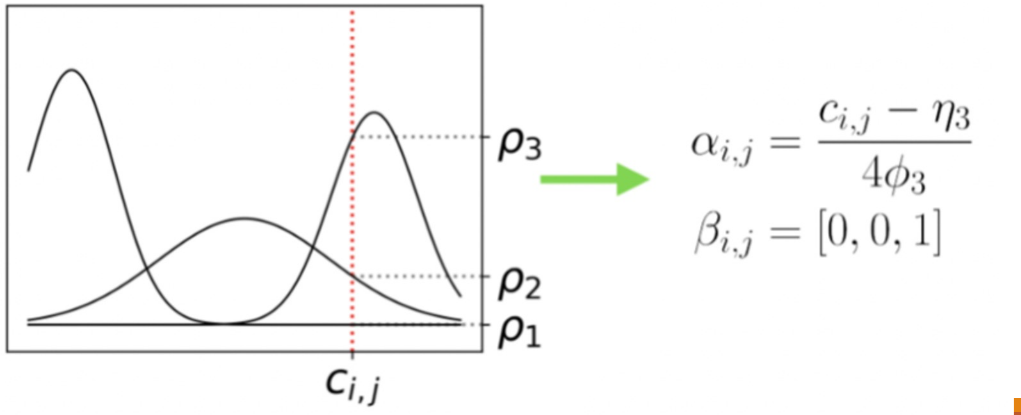
CT-GAN. Normalization of continuous data

Mode-specific normalization

- Once we have found the k Gaussian distributions that best model our continuous variable, we can evaluate the sample at each of the Gaussians.
  - From there, we can decide what distribution the sample belongs to (this is represented by β). Finally, we can represent the value of the sample within its distribution (how important that sample is in its gaussian) using the α term.

- In the example, the VGM finds 3 Gaussians to represent the distribution of the continuous variable (k=3). The sample c (in red) is encoded as a β vector {0,0,1}, and an α vector using the equation shown above.

- And that's it, to solve the normalization problem
  → we give it α and β.



$$\alpha_{i,j} = \frac{c_{i,j} - \eta_3}{4\phi_3}$$

$$\beta_{i,j} = [0, 0, 1]$$

CT-GAN. Fair sampling of discrete data

- When training the generator of a GAN, the input noise is drawn from a prior distribution (Z). Sampling in this way for discrete variables may miss information about their distribution. It would be useful for the model to somehow include information from the discrete variables as input, and for it to learn to map that input accordingly to the desired output.
- The solution the paper proposes consists of three key elements:
  - **conditional vector**,
  - **generator loss**,
  - and **training-by-sampling**.

CT-GAN. Fair sampling of discrete data

- **Conditional vector**: The same idea than cGAN
  - Some information about the desired discrete variables must be contained on the input aside from the random noise.

- The conditional vector allows us to force the generator to generate a sample from a chosen category. The conditional vector contains all the discrete columns, one-hot encoded, where all the values are set to zero except for one category in one discrete column (the condition we want the generated sample to fulfil). The condition is chosen through training-by-sampling.

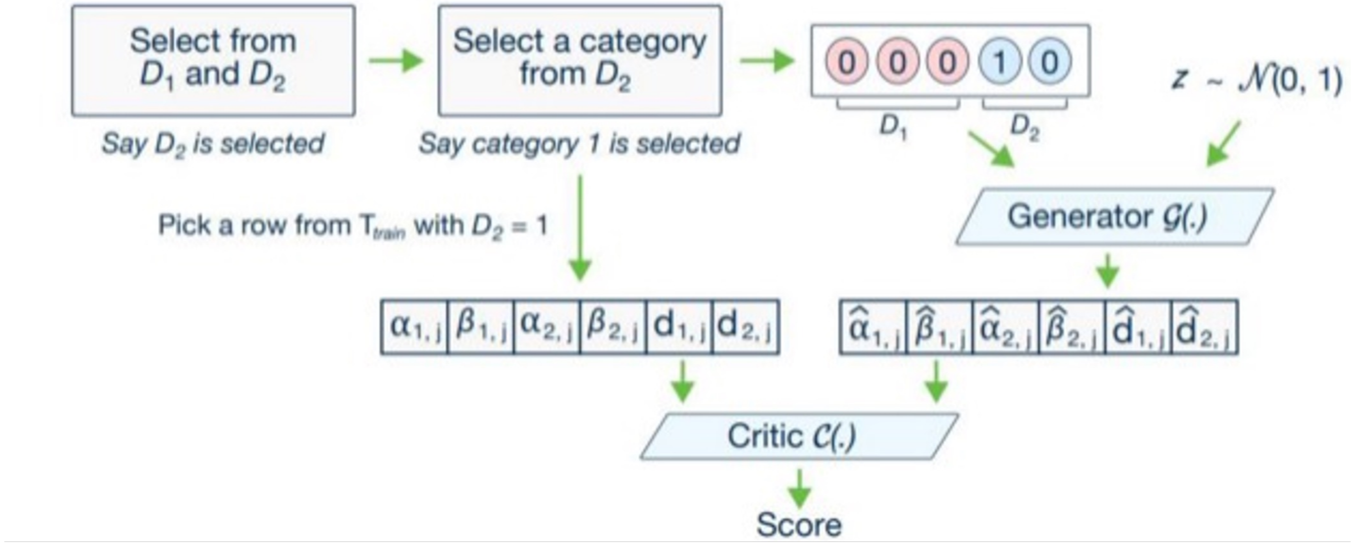
CT-GAN. Fair sampling of discrete data

- **Training-by-sampling** allows sampling the conditions to generate the conditional vectors such that the distributions generated by the generator match the distributions of the discrete variables in the training data.

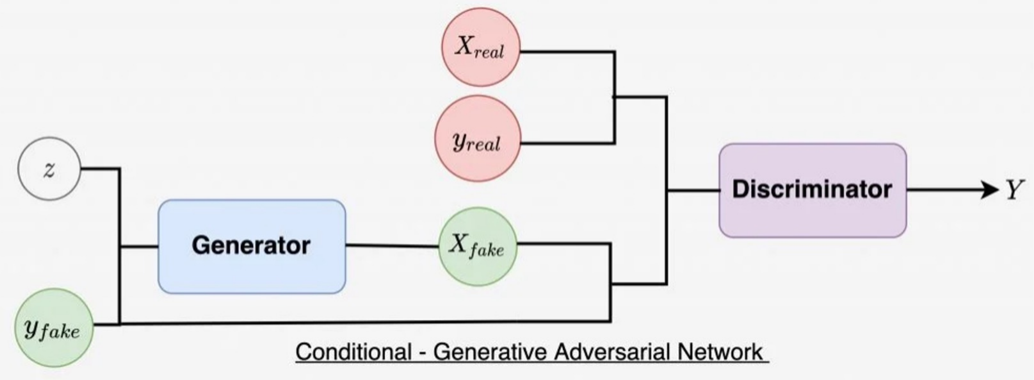
- Training-by-sampling is done as follows:
  - First, a random discrete column is selected
  - From this discrete column, a category is selected based on a probability mass function constructed from the frequencies of occurrence of each category in that discrete column.
  - The condition is transformed to the conditional vector and used as input to the generator

CT-GAN. Fair sampling of discrete data

- The **generator loss** is adapted to force the generator to generate a sample with this condition. They do this by adding the cross-entropy between the conditional vector and the generated sample to the loss term. This forces the produced samples to abide by the condition → The same than cGAN



Select from D_1 and D_2
Say D_2 is selected
Select a category from D_2
Say category 1 is selected
0 0 0 1 0
D_1
D_2
$z \sim \mathcal{N}(0, 1)$
Pick a row from T_{train} with $D_2 = 1$
Generator $\mathcal{G}(.)$
$\alpha_{1,j}$ $\beta_{1,j}$ $\alpha_{2,j}$ $\beta_{2,j}$ $d_{1,j}$ $d_{2,j}$
$\hat{\alpha}_{1,j}$ $\hat{\beta}_{1,j}$ $\hat{\alpha}_{2,j}$ $\hat{\beta}_{2,j}$ $\hat{d}_{1,j}$ $\hat{d}_{2,j}$
Critic $\mathcal{C}(.)$
Score

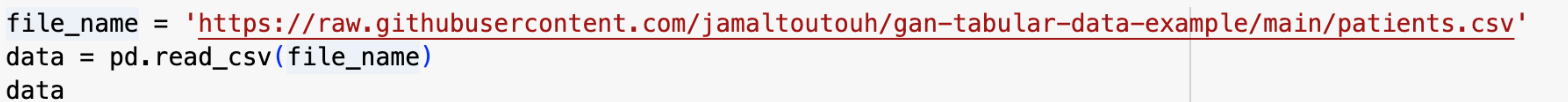


Conditional - Generative Adversarial Network

CT-GAN. Experiments with code

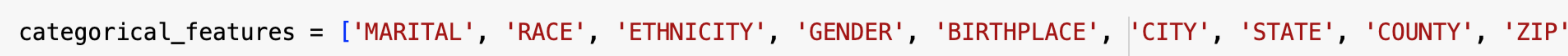
- Training a CT-GAN using CTGAN Python library from The Synthetic Data Vault

1. Get full tabular dataset



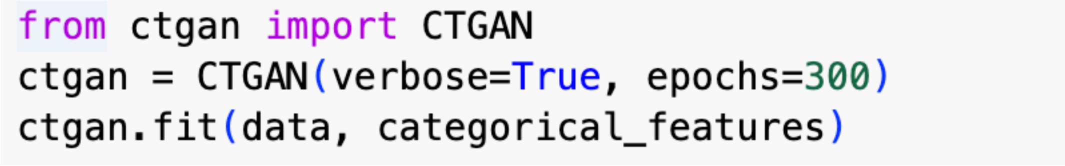
```
file_name = 'https://raw.githubusercontent.com/jamaltoutouh/gan-tabular-data-example/main/patients.csv'
data = pd.read_csv(file_name)
data
```

2. Get categorical features (columns)



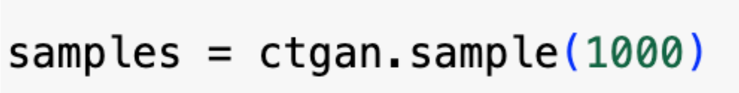
```
categorical_features = ['MARITAL', 'RACE', 'ETHNICITY', 'GENDER', 'BIRTHPLACE', 'CITY', 'STATE', 'COUNTY', 'ZIP'
```

3. Train the model



```
from ctgan import CTGAN
ctgan = CTGAN(verbose=True, epochs=300)
ctgan.fit(data, categorical_features)
```

4. Produce samples



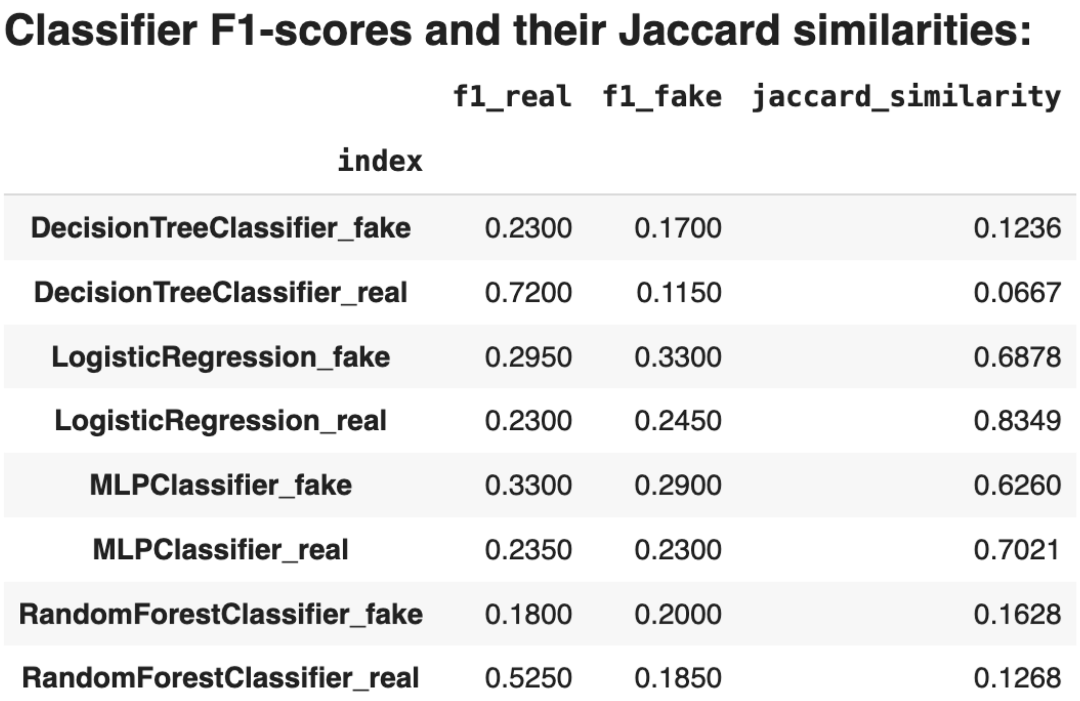
```
samples = ctgan.sample(1000)
```

CT-GAN. Experiments with code

- Evaluating the produced simples using TableEvaluator Python library
  - https://baukebrenninkmeijer.github.io/table-evaluator/
- It provides a series of metrics and visualizations to compare the distribution of real data and synthetic data
  - F1-scores and Jaccard similarity
  - Absolute log mean and standard deviation of numeric data
  - Cummulative sums
  - Distributions per feture
  - Difference between real and fave correlations among features
  - First two PCA components

CT-GAN. Experiments with code

- Evaluating the produced simples using TableEvaluator Python library
  - https://baukebrenninkmeijer.github.io/table-evaluator/
- It provides a series of metrics and visualizations to compare the distribution of real data and synthetic data
  - F1-scores and Jaccard similarity
    - Given a specific column

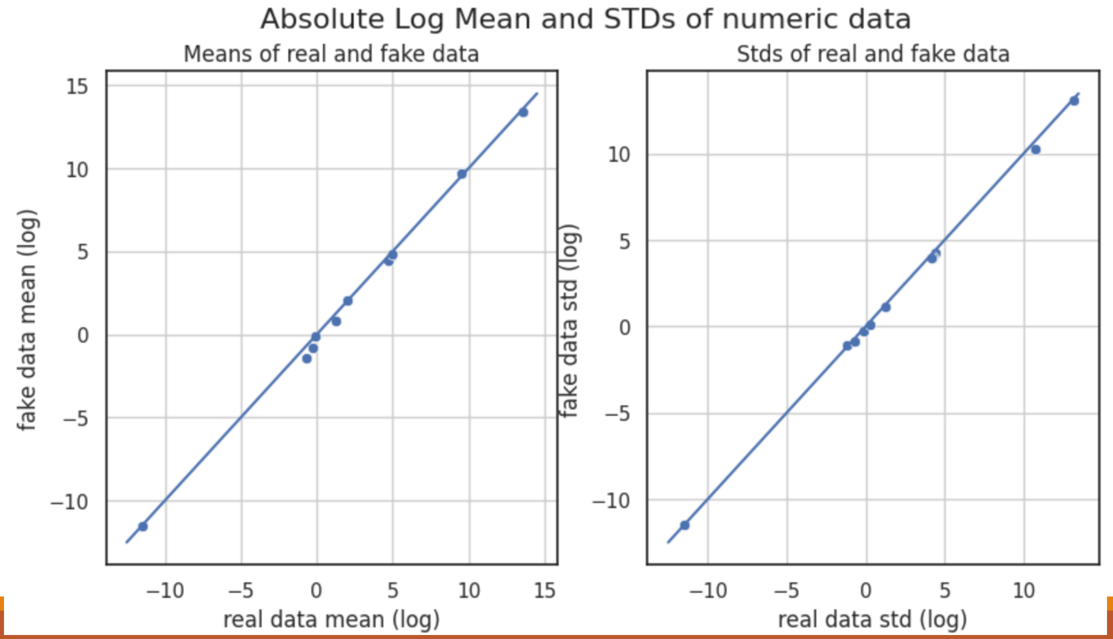


Classifier F1-scores and their Jaccard similarities:

	f1_real	f1_fake	jaccard_similarity
index			
DecisionTreeClassifier_fake	0.2300	0.1700	0.1236
DecisionTreeClassifier_real	0.7200	0.1150	0.0667
LogisticRegression_fake	0.2950	0.3300	0.6878
LogisticRegression_real	0.2300	0.2450	0.8349
MLPClassifier_fake	0.3300	0.2900	0.6260
MLPClassifier_real	0.2350	0.2300	0.7021
RandomForestClassifier_fake	0.1800	0.2000	0.1628
RandomForestClassifier_real	0.5250	0.1850	0.1268

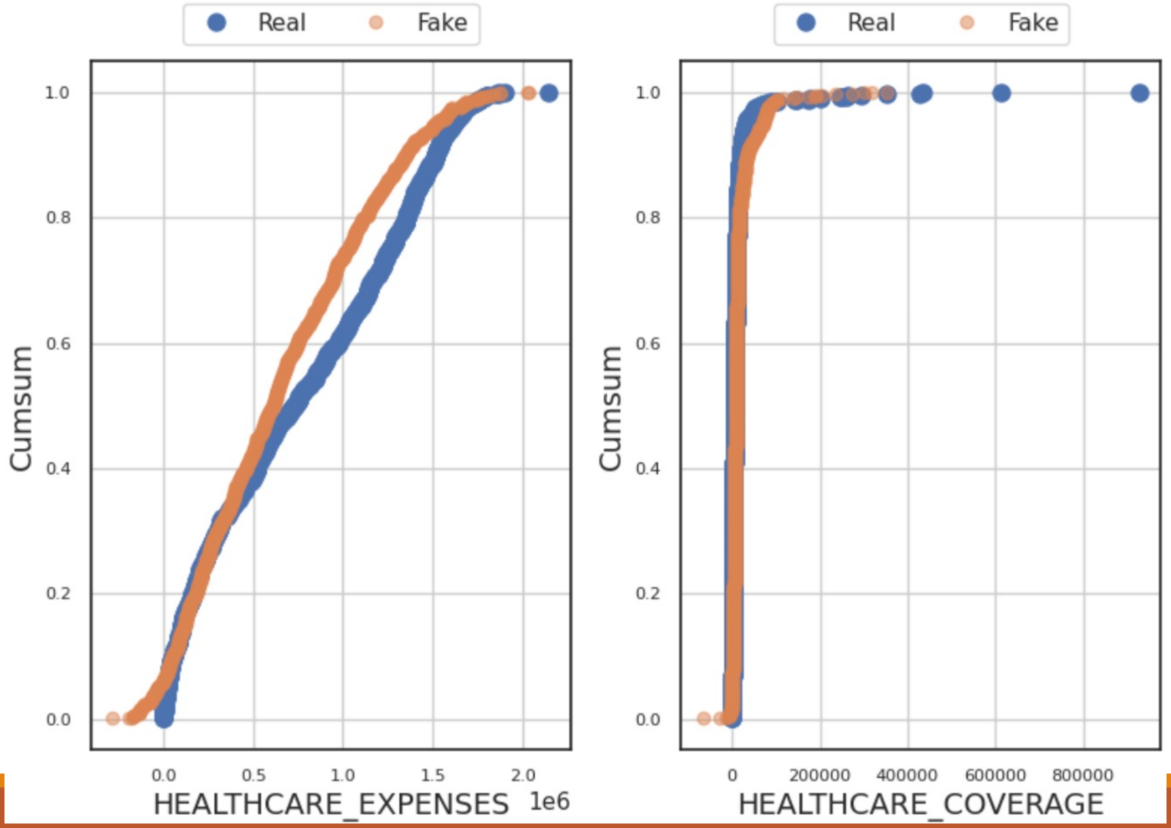
CT-GAN. Experiments with code

- Evaluating the produced simples using TableEvaluator Python library
  - https://baukebrenninkmeijer.github.io/table-evaluator/
- It provides a series of metrics and visualizations to compare the distribution of real data and synthetic data
  - Absolute log mean and standard deviation of numeric data



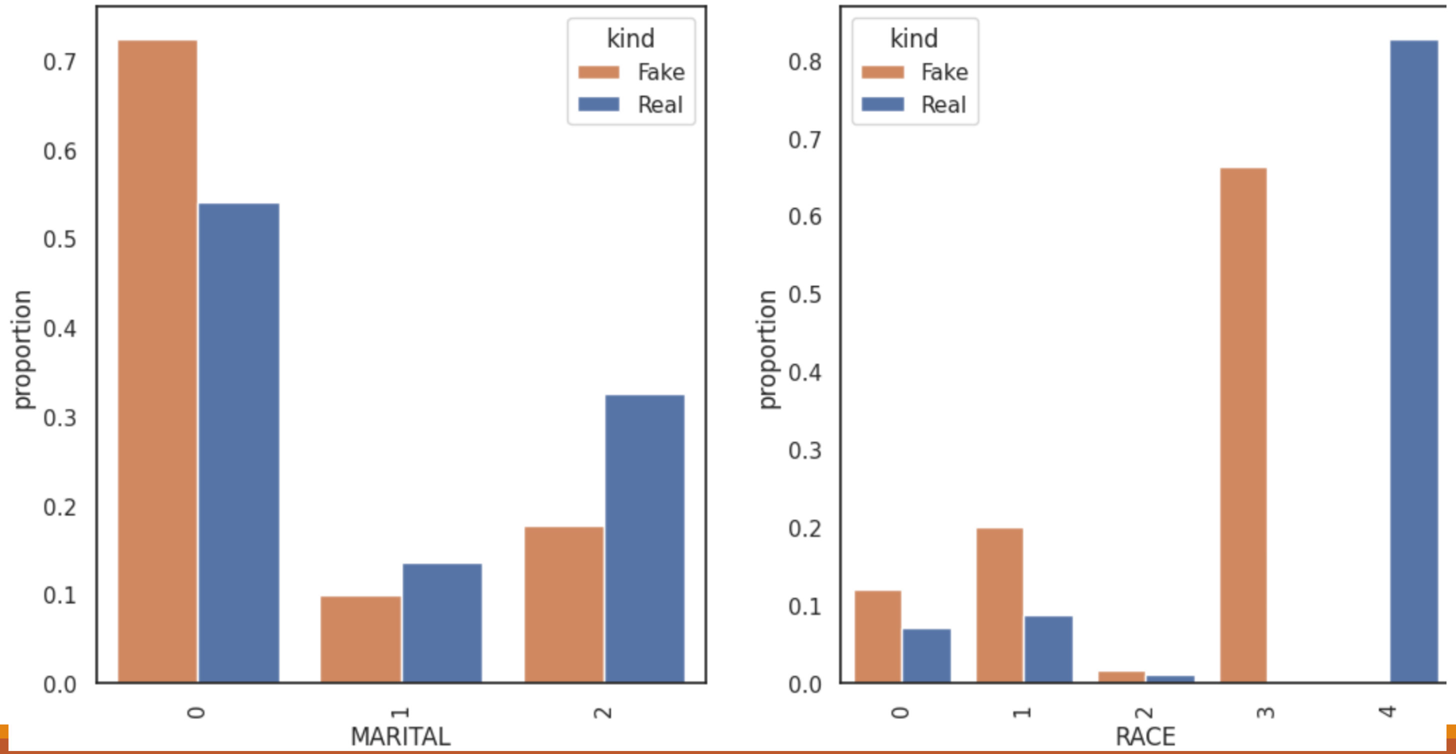
CT-GAN. Experiments with code

- Evaluating the produced simples using TableEvaluator Python library
  - https://baukebrenninkmeijer.github.io/table-evaluator/
- It provides a series of metrics and visualizations to compare the distribution of real data and synthetic data
  - Cummulative sums



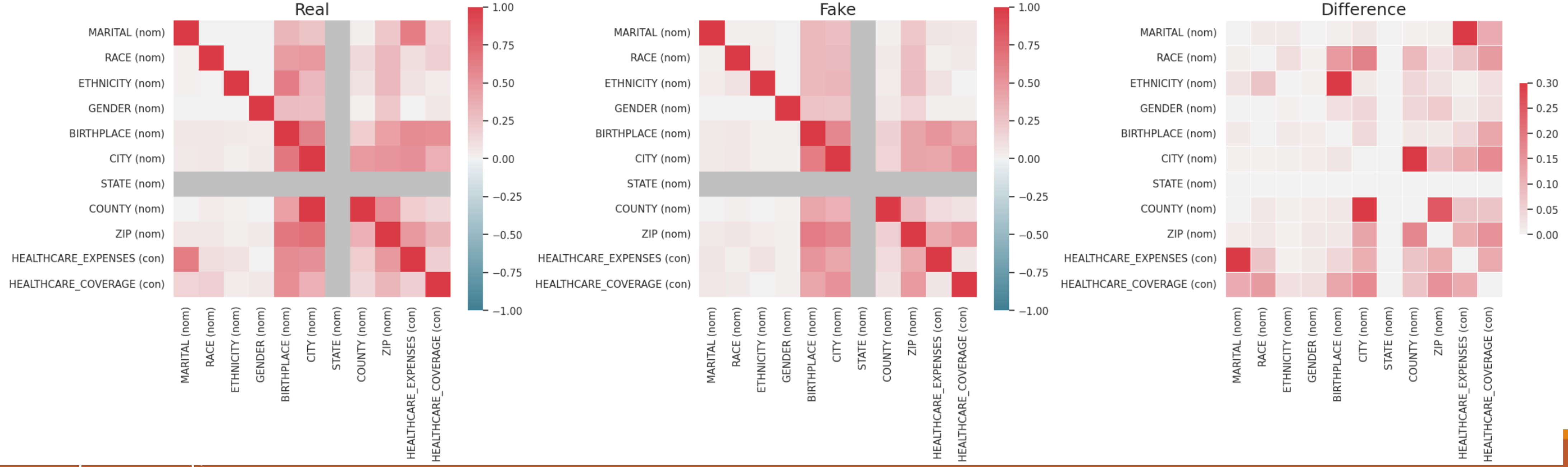
CT-GAN. Experiments with code

- Evaluating the produced simples using TableEvaluator Python library
  - https://baukebrenninkmeijer.github.io/table-evaluator/
- It provides a series of metrics and visualizations to compare the distribution of real data and synthetic data
  - Distributions per feture



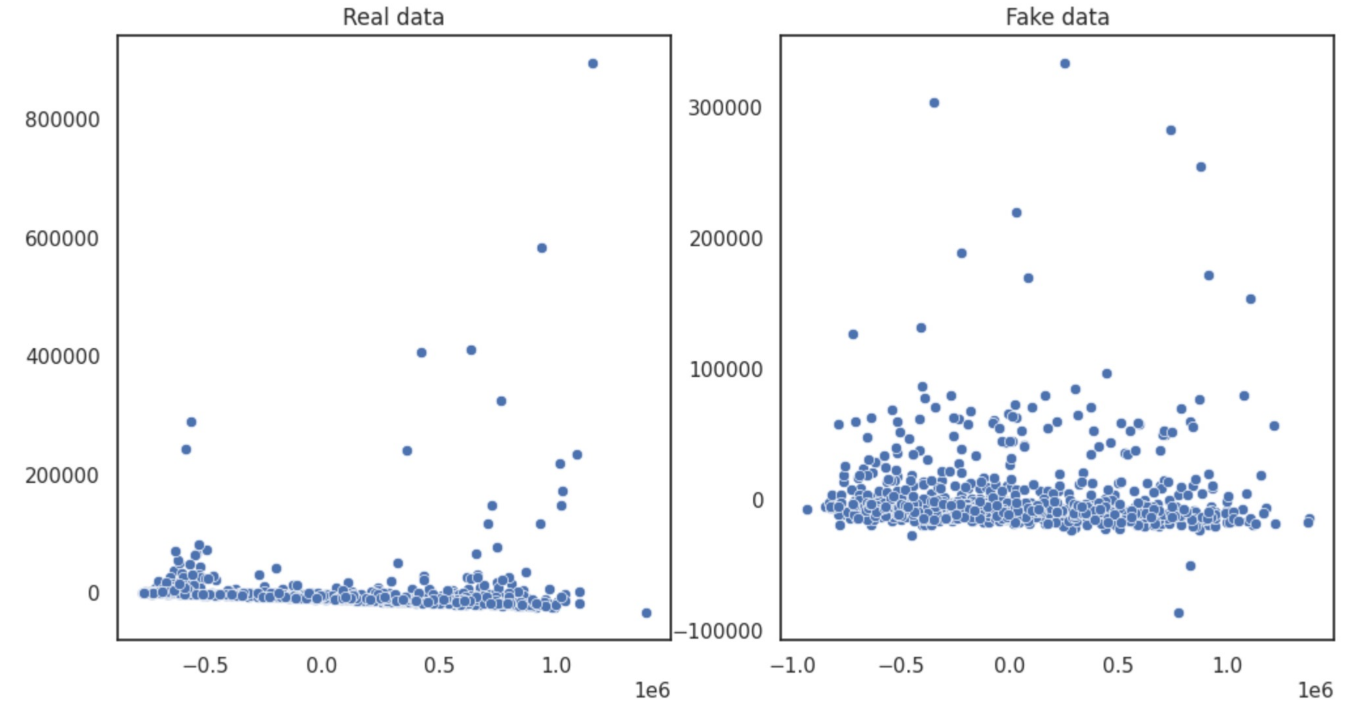
CT-GAN. Experiments with code

- Evaluating the produced simples using TableEvaluator Python library
  - https://baukebrenninkmeijer.github.io/table-evaluator/
- It provides a series of metrics and visualizations to compare the distribution of real data and synthetic data
  - Difference between real and fave correlations among features



CT-GAN. Experiments with code

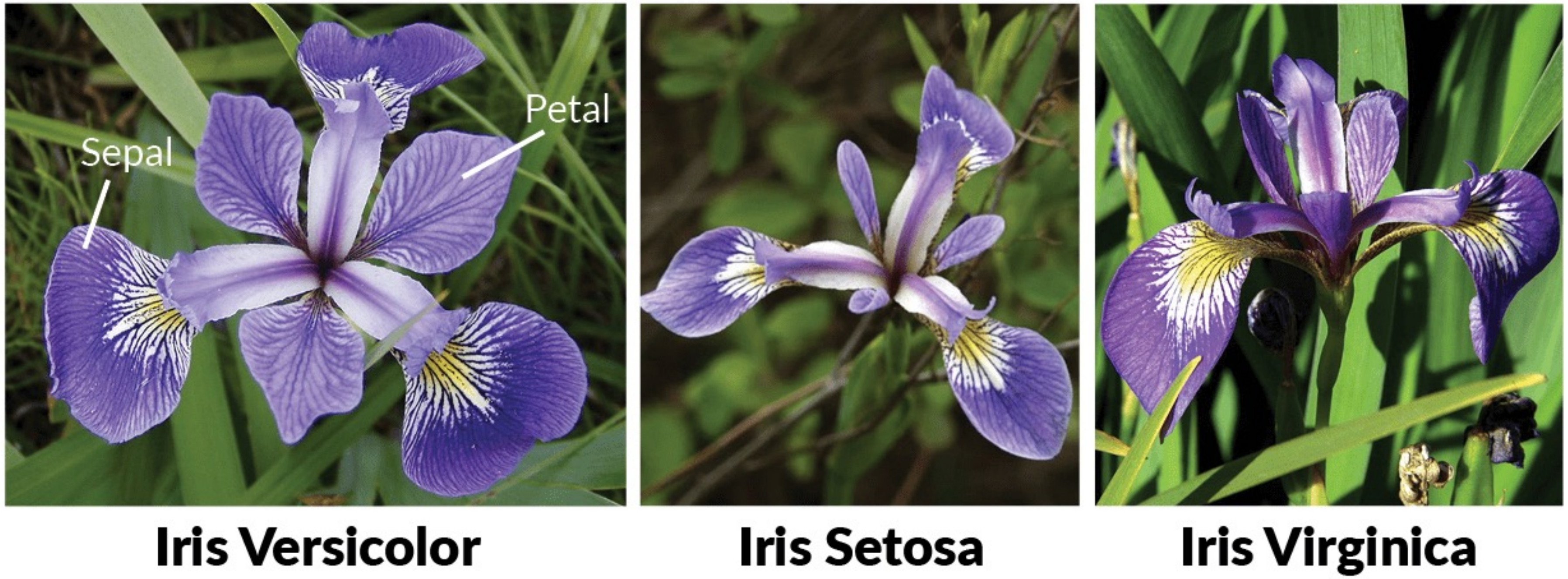
- Evaluating the produced simples using TableEvaluator Python library
  - https://baukebrenninkmeijer.github.io/table-evaluator/
- It provides a series of metrics and visualizations to compare the distribution of real data and synthetic data
  - First two PCA components



Example

- IRIS → **CTGAN**

  https://drive.google.com/file/d/12541AomVyxpSowrLbCGvvpGKDt9-LLyQ/view?usp=sharing



Iris Versicolor

Iris Setosa

Iris Virginica

Example

- Medical data → **CTGAN**

  https://drive.google.com/file/d/1RDencmohwZiB-7d5JlHNVhCLd8dNMWXw/view?usp=sharing

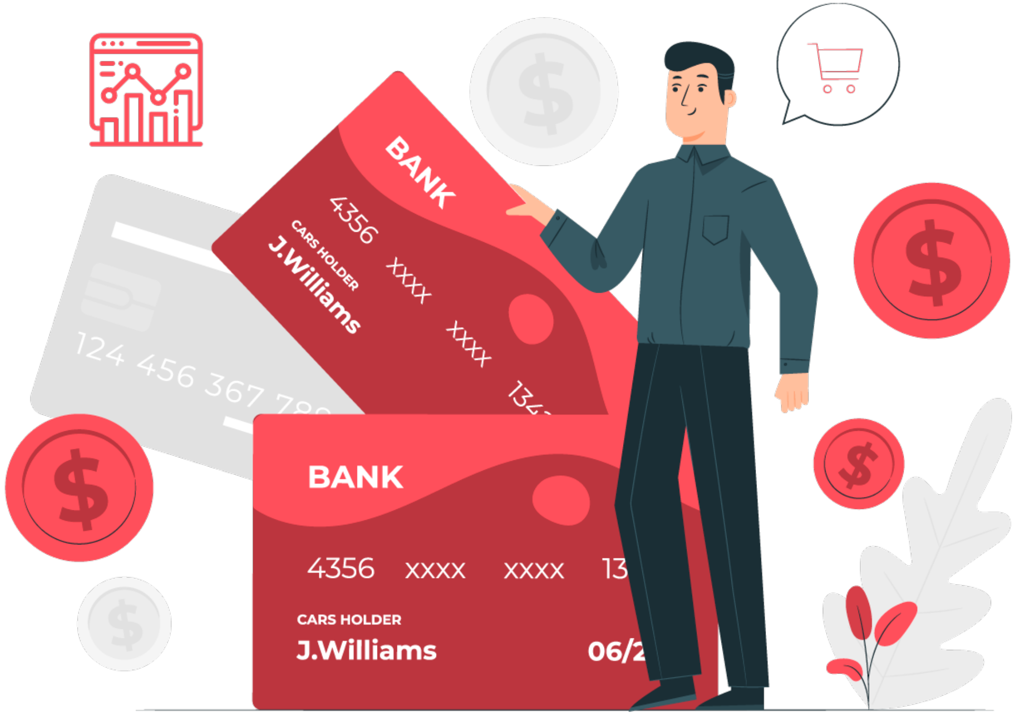
Exercise

- Credit Card Analysis Data → **CTGAN**

https://github.com/jamaltoutouh/curso-ciencia-de-datos-python/blob/main/correct_synthetic_credit_analysis.csv

Download file from:

https://github.com/jamaltoutouh/curso-ciencia-de-datos-python/blob/main/correct_synthetic_credit_analysis.csv



Massachusetts
Institute of
Technology

ANYSCALE LEARNING FOR ALL

Thanks!

Comments?

JAMAL TOUTOUH
toutouh@mit.edu
jamal.es
necol.net
@jamtou