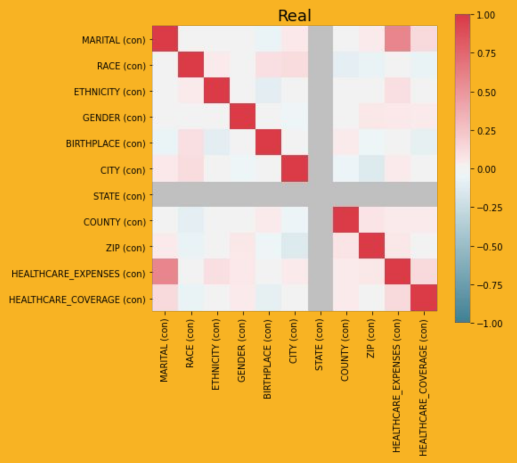
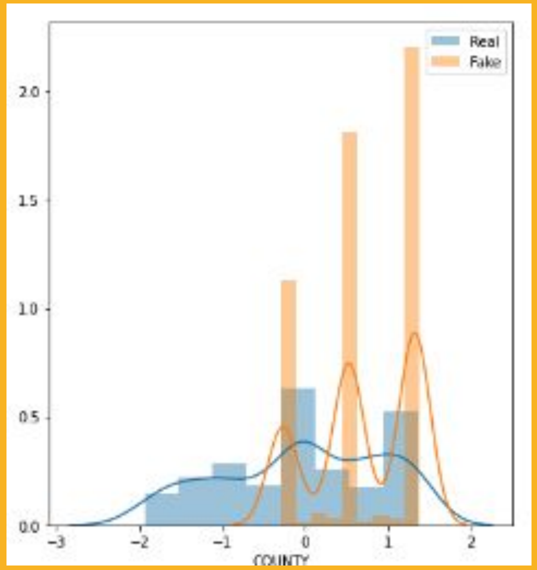


DPFree DATA GENERATION



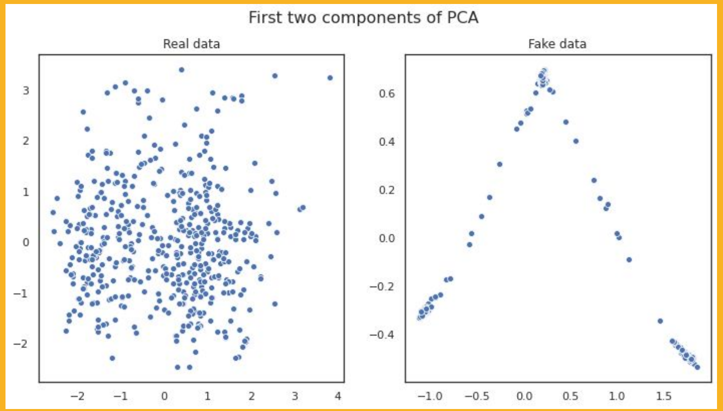
The future of Data Science

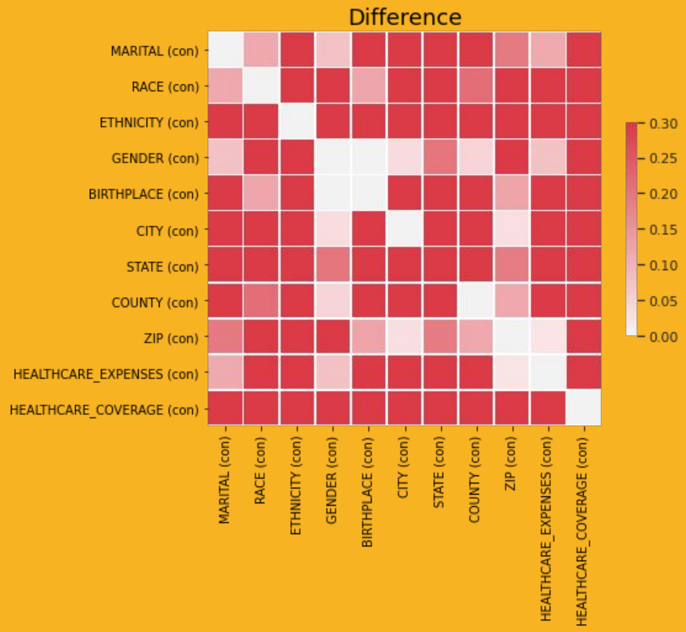
The future of humanity

Sayash Raaj,

IIT Madras

Silver Medalist





Credentials-

IIT Madras

Harvard HPAIR Core Tech Associate

Amazon USA Intl' App Contest Winne

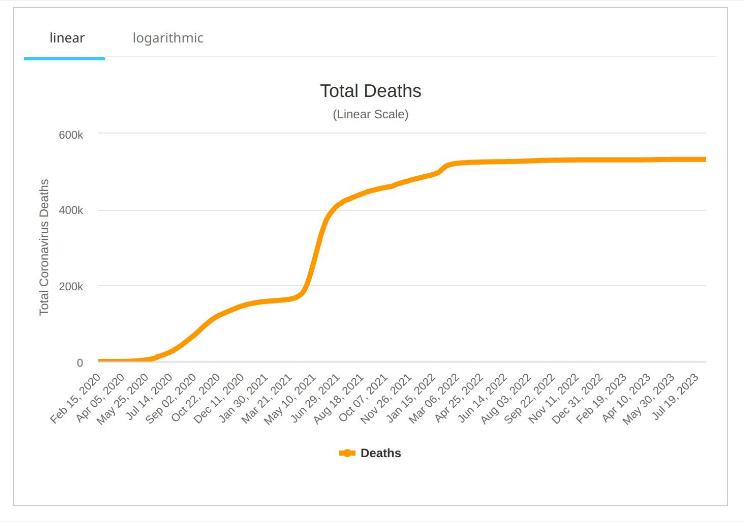
Intel AI Global Impact Festival Winnerr

SMEC Indo-Australian **Scholarship'22**

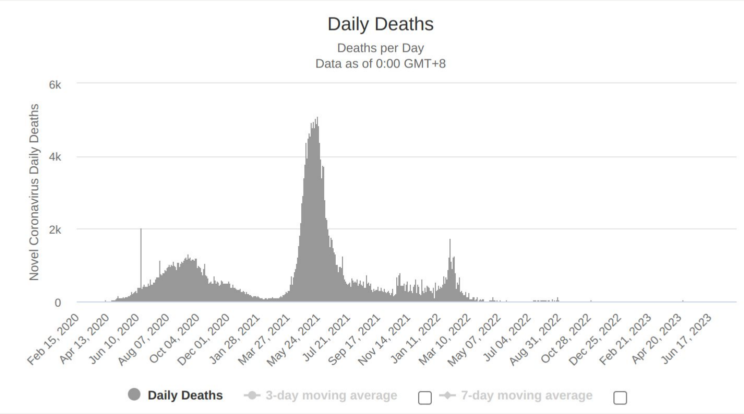
Kalidas Madhavpeddi IITM **Scholarship'22**

Real-Life Use Case: India

- Covid-19 pandemic disturbed India.
- 531,000 deaths.



- Real-time covid data was inaccessible. Reasons:
  - Privacy of medical data
  - Inefficient logistics (late delivery of data)
  - Lack of manual workforce

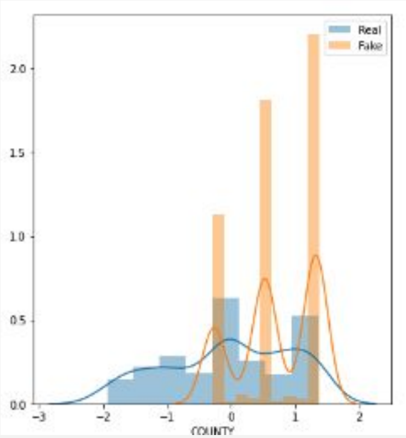


- Data-driven solutions could have saved lives
- Data was unavailable, and delayed. It cost real, human lives.

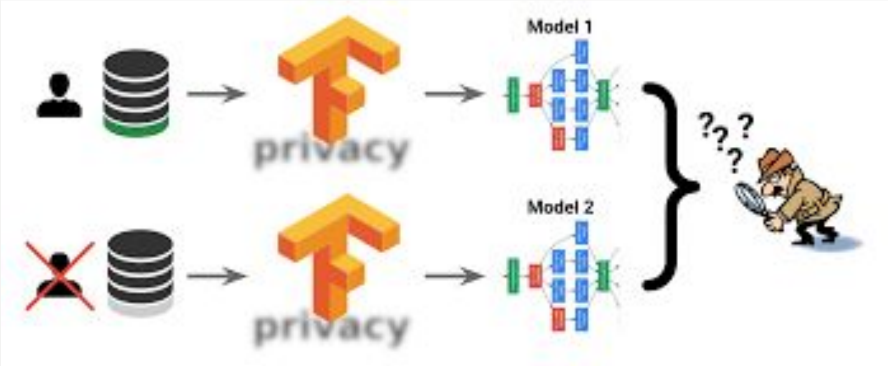
Problem Solved

- ~~Privacy of medical data~~
- ~~Inefficient logistics (late delivery of data)~~
- ~~Lack of manual workforce~~

- PII-free, Secure, **GDPR-compliant** shareable data
- **95.83%** Efficient, Faster Data generation
- **100%** Automation achieved

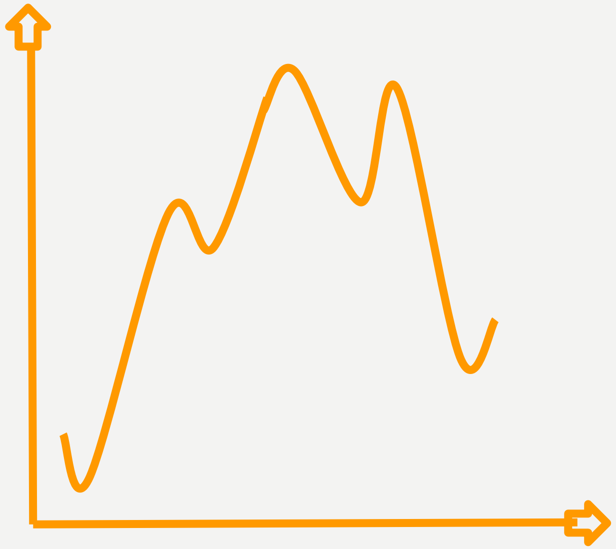


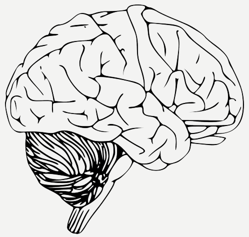
DELIVERABLES



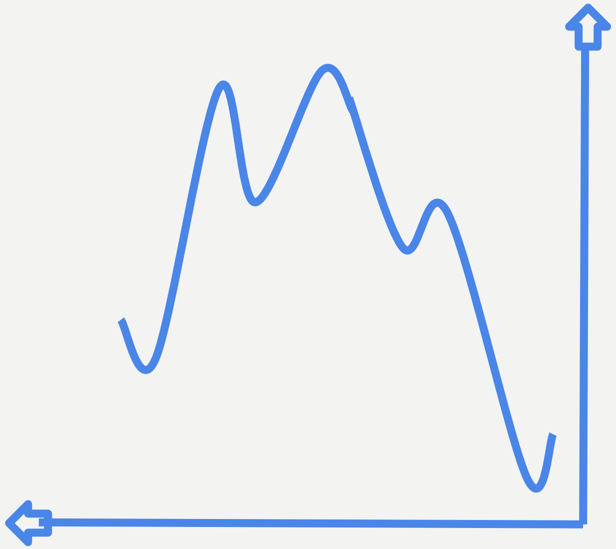
- DPFree Data Generation
- GAN based approach: Modular
- Customisable for each dataset: Domain-agnostic
- Privacy guaranteed
- Complies with all data privacy regulations including GDPR compliance: biggest mark of trust
- Differential Privacy

Working Visuals





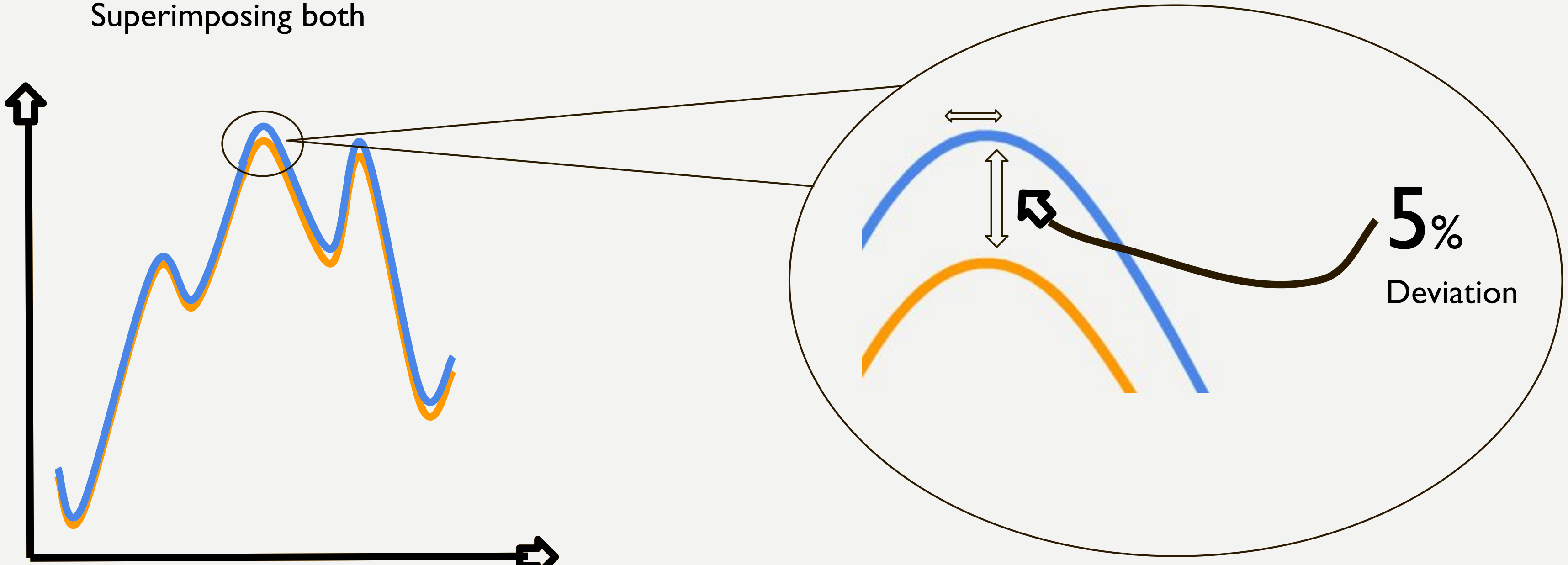
Original Data



My solution: Masked PII, Encrypted data

95% Accuracy

Superimposing both



My solution replicates the original data with 95% Accuracy

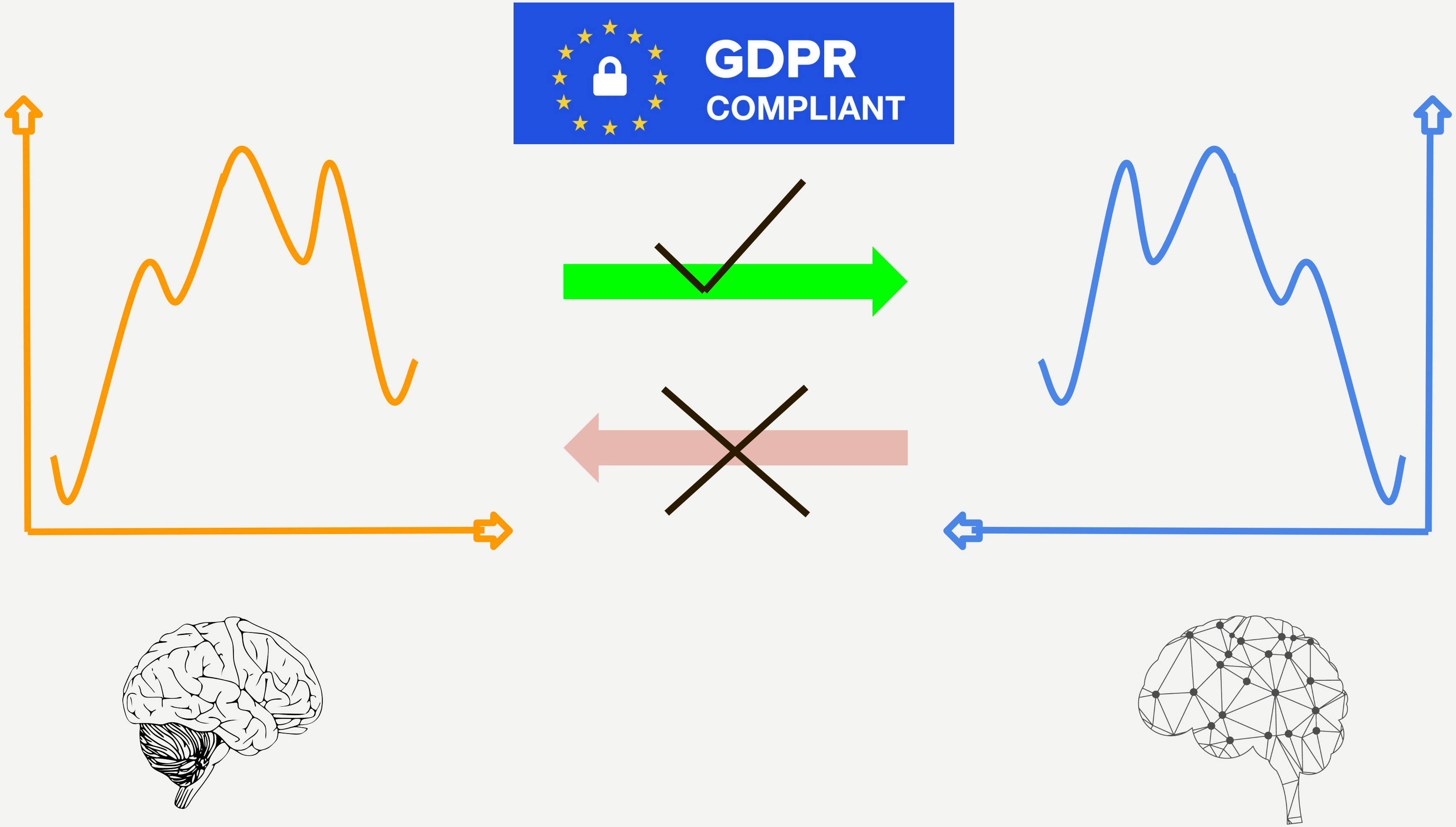
5% Deviation gives birth to **Hashing: Irreversible Encryption of Data**

Intrinsically, Masking **PII**, improving **security** with unbreakable encryption

95% Accuracy 5% Deviation: **Hashing: Irreversible Encryption of Data**

Ensures Privacy

Observations are untraceable, untracked, backtracking-resistant



Original Data

My solution: Masked PII, Encrypted data

1. EXTRACT COLUMNS TO SIMULATE

Remove unnecessary columns and encode all data

```
[ ] import pandas as pd

    df = pd.read_csv('csv/patients.csv')
    df.drop(['Id', 'BIRTHDATE', 'DEATHDATE', 'SSN', 'DRIVERS', 'PASSPORT', 'PREFIX',
            'FIRST', 'ADDRESS', 'LAST', 'SUFFIX', 'MAIDEN','LAT', 'LON',], axis=1, inplace=True)
    print(df.columns)
```

```
Index(['MARITAL', 'RACE', 'ETHNICITY', 'GENDER', 'BIRTHPLACE', 'CITY', 'STATE',
       'COUNTY', 'ZIP', 'HEALTHCARE_EXPENSES', 'HEALTHCARE_COVERAGE'],
      dtype='object')
```

Next, read patients data and remove fields such as id, date, SSN, name etc. Note, that we are trying to generate synthetic data which can be used to train our deep learning models for some other tasks. For such a model, we don't require fields like id, date, SSN etc.

```
# data configuration


file_name = "csv/patients.csv"
columns_to_drop = ['Id', 'BIRTHDATE', 'DEATHDATE', 'SSN', 'DRIVERS', 'PASSPORT', 'PREFIX', 'FIRST', 'ADDRESS', 'LAST', 'SUFFIX', 'MAIDEN','LAT', 'LON']
categorical_features = ['MARITAL', 'RACE', 'ETHNICITY', 'GENDER', 'BIRTHPLACE', 'CITY', 'STATE', 'COUNTY', 'ZIP']
continuous_features = ['HEALTHCARE_EXPENSES', 'HEALTHCARE_COVERAGE']
col1, col2 = 'num_of_doors', 'price'
col_group_by = 'body_style'
```

- User decides the columns to contain in the simulated dataset
- Customizable as per user needs

- Pre-processing continuous variables- binning

Next, we will encode all continious features to equally sized bins. First, lets find the minimum and maximum values for `HEALTHCARE_EXPENSES` and `HEALTHCARE_COVERAGE` and then create bins based on these values.

```
import numpy as np

for column in continuous_features:
  min = df[column].min()
  max = df[column].max()
  feature_bins = pd.cut(df[column], bins=np.linspace(min, max, 21), labels=False)
  df.drop([column], axis=1, inplace=True)
  df = pd.concat([df, feature_bins], axis=1)
  print(df)
```

```
      MARITAL  RACE  ETHNICITY  ...  ZIP  HEALTHCARE_COVERAGE  HEALTHCARE_EXPENSES
0           0     4          0  ...    2              1334.88                  2.0
1           0     4          1  ...  132              3204.49                  7.0
2           0     4          1  ...    3              2606.40                  5.0
3           0     4          1  ...   68              8756.19                  8.0
4          -1     4          1  ...  125              3772.20                  5.0
...       ...   ...        ...  ...  ...                  ...                  ...
1166        0     0          0  ...  130             32086.31                 15.0
1167        1     4          1  ...   80              3130.52                  9.0
1168        1     4          1  ...   -1             52391.24                 14.0
1169        0     4          1  ...   98             13157.00                 12.0
1170        0     4          1  ...  102             26565.65                 14.0

[1171 rows x 11 columns]
      MARITAL  RACE  ETHNICITY  ...  ZIP  HEALTHCARE_EXPENSES  HEALTHCARE_COVERAGE
0           0     4          0  ...    2                  2.0                  0.0
1           0     4          1  ...  132                  7.0                  0.0
2           0     4          1  ...    3                  5.0                  0.0
3           0     4          1  ...   68                  8.0                  0.0
4          -1     4          1  ...  125                  5.0                  0.0
...       ...   ...        ...  ...  ...                  ...                  ...
1166        0     0          0  ...  130                 15.0                  0.0
1167        1     4          1  ...   80                  9.0                  0.0
1168        1     4          1  ...   -1                 14.0                  1.0
1169        0     4          1  ...   98                 12.0                  0.0
1170        0     4          1  ...  102                 14.0                  0.0

[1171 rows x 11 columns]
```

Next, we will encode all categorical features to integer values. We are simply encoding the features to numerica hot encoding as its not required for GANs.

```
[ ] for column in categorical_features:
      df[column] = df[column].astype('category').cat.codes

    df.head()
```

	MARITAL	RACE	ETHNICITY	GENDER	BIRTHPLACE	CITY	STATE	COUNTY	ZIP	HEALTHCARE_EXPEN	
0	0	4	0	1	136	42	0	6	2	271227.08	1334.88
1	0	4	1	1	61	186	0	8	132	793946.01	3204.49
2	0	4	1	1	236	42	0	6	3	574111.90	2606.40
3	0	4	1	0	291	110	0	8	68	935630.30	8756.19
4	-1	4	1	1	189	24	0	12	125	598763.07	3772.20

- Pre-processing categorical variables- encoding

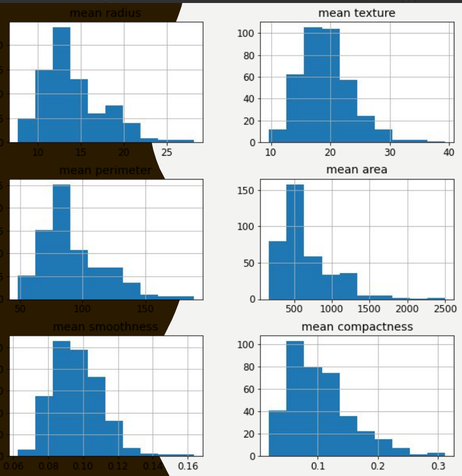
Transform the data

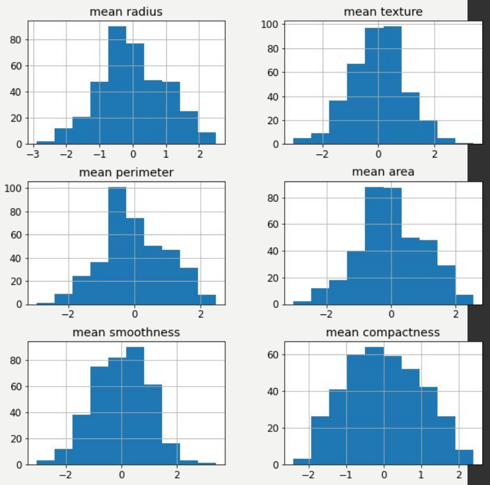
Next, we apply `PowerTransformer` on all the fields to get a Gaussian distribution for the data.

```
from sklearn.preprocessing import PowerTransformer


df[df.columns] = PowerTransformer(method='yeo-johnson', standardize=True, copy=True).fit_transform(df[df.columns])

print(df)
```

```
        MARITAL      RACE  ...  HEALTHCARE_EXPENSES  HEALTHCARE_COVERAGE
0      0.334507  0.461541  ...            -0.819522            -0.187952
1      0.334507  0.461541  ...             0.259373            -0.187952
2      0.334507  0.461541  ...            -0.111865            -0.187952
3      0.334507  0.461541  ...             0.426979            -0.187952
4     -1.275676  0.461541  ...            -0.111865            -0.187952
...         ...       ...  ...                  ...                  ...
1166   0.334507 -2.207146  ...             1.398831            -0.187952
1167   1.773476  0.461541  ...             0.585251            -0.187952
1168   1.773476  0.461541  ...             1.275817             5.320497
1169   0.334507  0.461541  ...             1.016430            -0.187952
1170   0.334507  0.461541  ...             1.275817            -0.187952

[1171 rows x 11 columns]
```

Normalisation of variables for better sampling from distribution





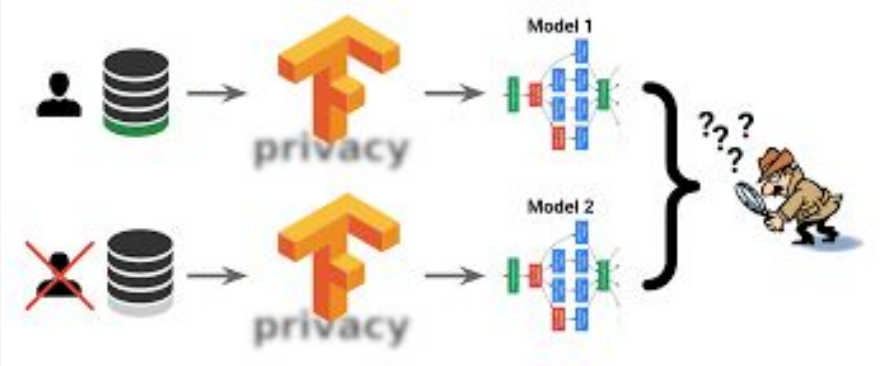
Final input dataframe output

```
df
```

	MARITAL	RACE	ETHNICITY	GENDER	BIRTHPLACE	CITY	STATE	COUNTY	ZIP	HEALTHCARE_EXPENSES	HEALTHCARE_COVERAGE
0	0.323410	0.462029	-3.059874	1.040975	0.080041	-0.953186	0.0	-0.603714	-0.329374	-0.852459	-0.18795
1	0.323410	0.462029	0.326811	1.040975	-0.781806	1.009515	0.0	-0.108844	1.084032	0.178401	-0.18795
2	0.323410	0.462029	0.326811	1.040975	1.086562	-0.953186	0.0	-0.603714	-0.240309	-0.201706	-0.18795
3	0.323410	0.462029	0.326811	-0.960637	1.596780	0.025263	0.0	-0.108844	0.798438	0.359778	-0.18795
4	-1.266799	0.462029	0.326811	1.040975	0.627873	-1.276494	0.0	1.132069	1.059763	-0.201706	-0.18795
...	...	...	...	...	...	...	...	...	...	...	...
1166	0.323410	-2.165654	-3.059874	-0.960637	-0.180194	-1.107466	0.0	-0.108844	1.077217	1.504460	-0.18795
1167	1.797699	0.462029	0.326811	1.040975	1.265789	-1.337494	0.0	-1.035152	0.866443	0.535933	-0.18795
1168	1.797699	0.462029	0.326811	-0.960637	-0.593921	0.604114	0.0	0.472068	-1.037056	1.351976	5.32049
1169	0.323410	0.462029	0.326811	-0.960637	1.086562	0.604114	0.0	0.472068	0.953153	1.037127	-0.18795
1170	0.323410	0.462029	0.326811	-0.960637	1.578669	0.604114	0.0	0.472068	0.970481	1.351976	-0.18795

1171 rows × 11 columns

DIFFERENTIAL PRIVACY

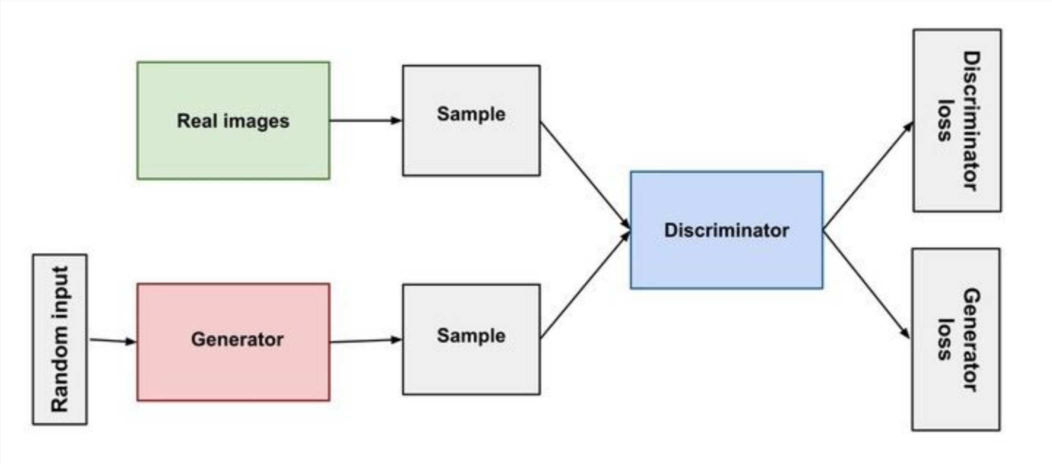


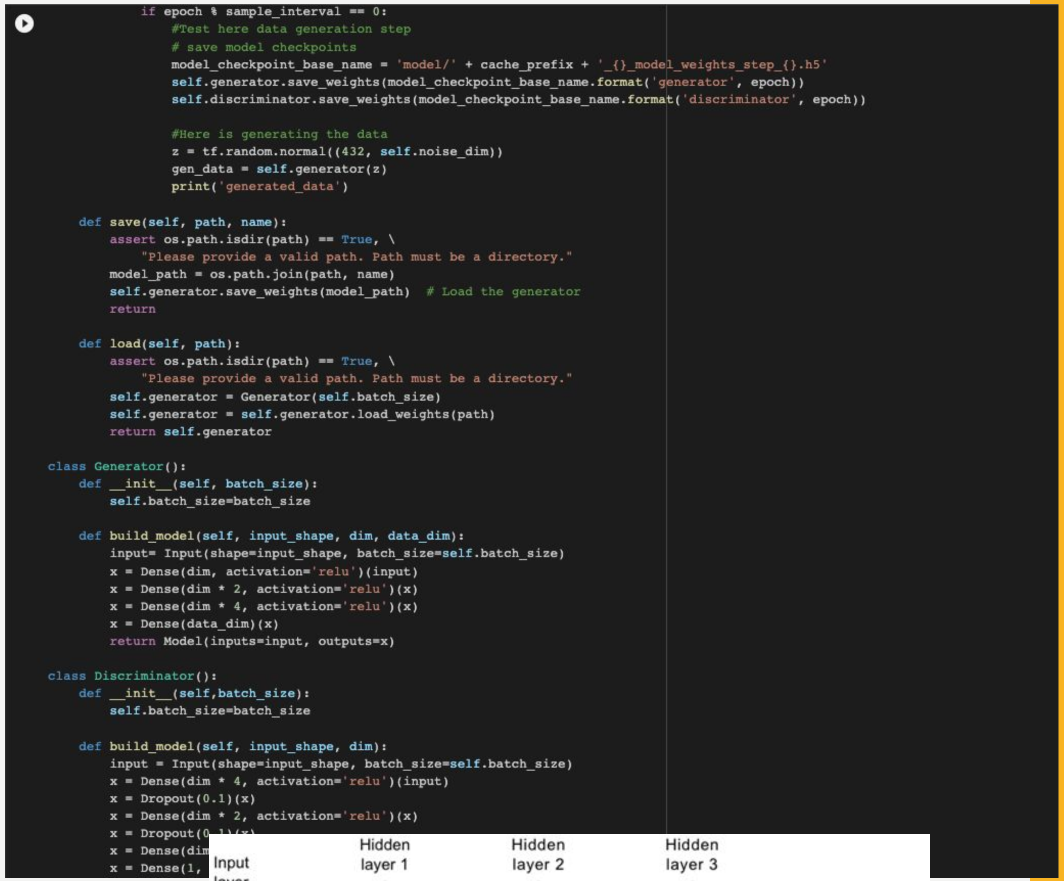
- Differential privacy (DP) is a system for publicly sharing information about a dataset by describing the patterns of groups within the dataset while withholding information about individuals in the dataset
- Describe the statistical properties of data, but abstract individual observations
- Beneficial for sharing private data on-the-go with null restrictions in health domain and other public protected datasets
- Complies with all data privacy regulations including GDPR compliance

GAN- TRAINING THE MODEL

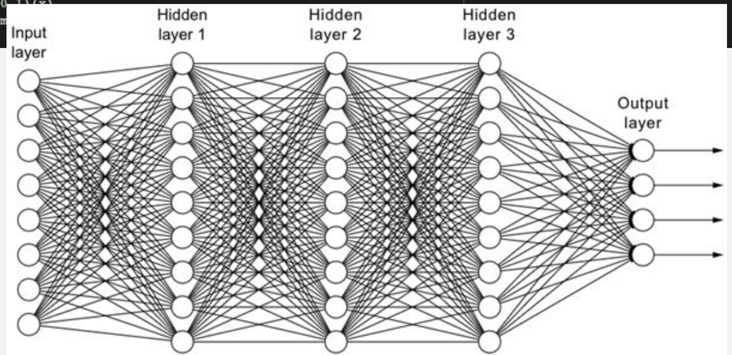
For the first iteration, we have used a classic GAN solution to iterate upon after inspecting and obtaining further results

Other method in progress is using Bayesian Networks





```
            if epoch % sample_interval == 0:
                #Test here data generation step
                # save model checkpoints
                model_checkpoint_base_name = 'model/' + cache_prefix + '_{}_model_weights_step_{}.h5'
                self.generator.save_weights(model_checkpoint_base_name.format('generator', epoch))
                self.discriminator.save_weights(model_checkpoint_base_name.format('discriminator', epoch))

                #Here is generating the data
                z = tf.random.normal((432, self.noise_dim))
                gen_data = self.generator(z)
                print('generated_data')

    def save(self, path, name):
        assert os.path.isdir(path) == True, \
            "Please provide a valid path. Path must be a directory."
        model_path = os.path.join(path, name)
        self.generator.save_weights(model_path)  # Load the generator
        return

    def load(self, path):
        assert os.path.isdir(path) == True, \
            "Please provide a valid path. Path must be a directory."
        self.generator = Generator(self.batch_size)
        self.generator = self.generator.load_weights(path)
        return self.generator

class Generator():
    def __init__(self, batch_size):
        self.batch_size=batch_size

    def build_model(self, input_shape, dim, data_dim):
        input= Input(shape=input_shape, batch_size=self.batch_size)
        x = Dense(dim, activation='relu')(input)
        x = Dense(dim * 2, activation='relu')(x)
        x = Dense(dim * 4, activation='relu')(x)
        x = Dense(data_dim)(x)
        return Model(inputs=input, outputs=x)

class Discriminator():
    def __init__(self,batch_size):
        self.batch_size=batch_size

    def build_model(self, input_shape, dim):
        input = Input(shape=input_shape, batch_size=self.batch_size)
        x = Dense(dim * 4, activation='relu')(input)
        x = Dropout(0.1)(x)
        x = Dense(dim * 2, activation='relu')(x)
        x = Dropout(0
        x = Dense(dim
        x = Dense(1,
```



Let's take a look at the Generator and Discriminator models.

```
[ ] synthesizer.generator.summary()
```

```
Model: "functional_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(32, 32)]                0
_________________________________________________________________
dense_16 (Dense)             (32, 128)                 4224
_________________________________________________________________
dense_17 (Dense)             (32, 256)                 33024
_________________________________________________________________
dense_18 (Dense)             (32, 512)                 131584
_________________________________________________________________
dense_19 (Dense)             (32, 11)                  5643
=================================================================
Total params: 174,475
Trainable params: 174,475
Non-trainable params: 0
_________________________________________________________________
```

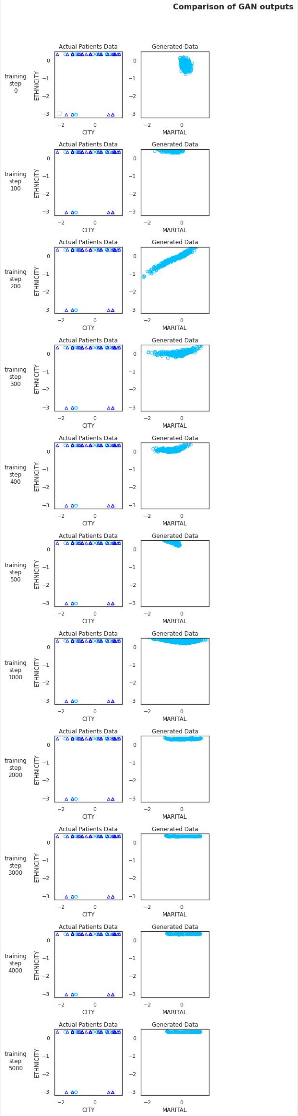
```
[ ] synthesizer.discriminator.summary()
```

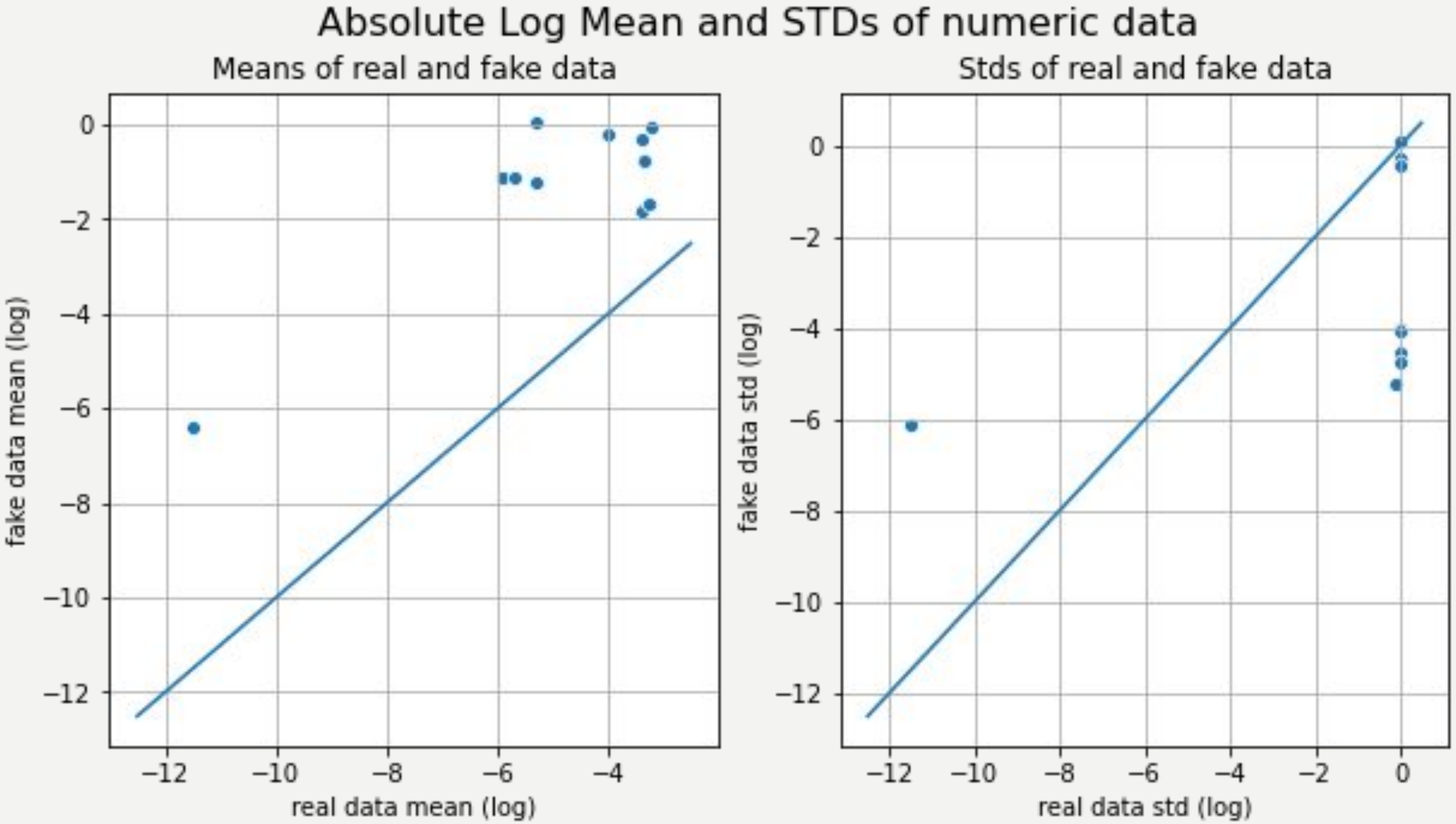
```
Model: "functional_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_8 (InputLayer)         [(32, 11)]                0
_________________________________________________________________
dense_20 (Dense)             (32, 512)                 6144
_________________________________________________________________
dropout_4 (Dropout)          (32, 512)                 0
_________________________________________________________________
dense_21 (Dense)             (32, 256)                 131328
_________________________________________________________________
dropout_5 (Dropout)          (32, 256)                 0
_________________________________________________________________
dense_22 (Dense)             (32, 128)                 32896
_________________________________________________________________
dense_23 (Dense)             (32, 1)                   129
=================================================================
Total params: 170,497
Trainable params: 0
Non-trainable params: 170,497
_________________________________________________________________
```

The Generator and Discriminator are neural networks of 3 hidden layers

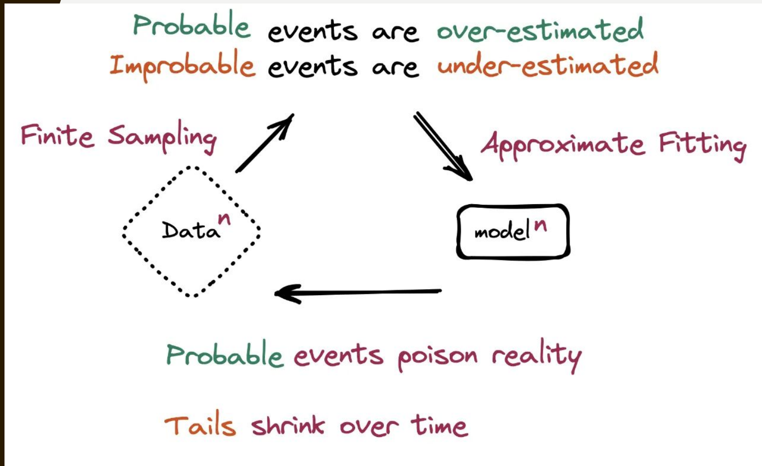
The neural networks are defined in the code and the snippet provided

EVALUATION METRICS

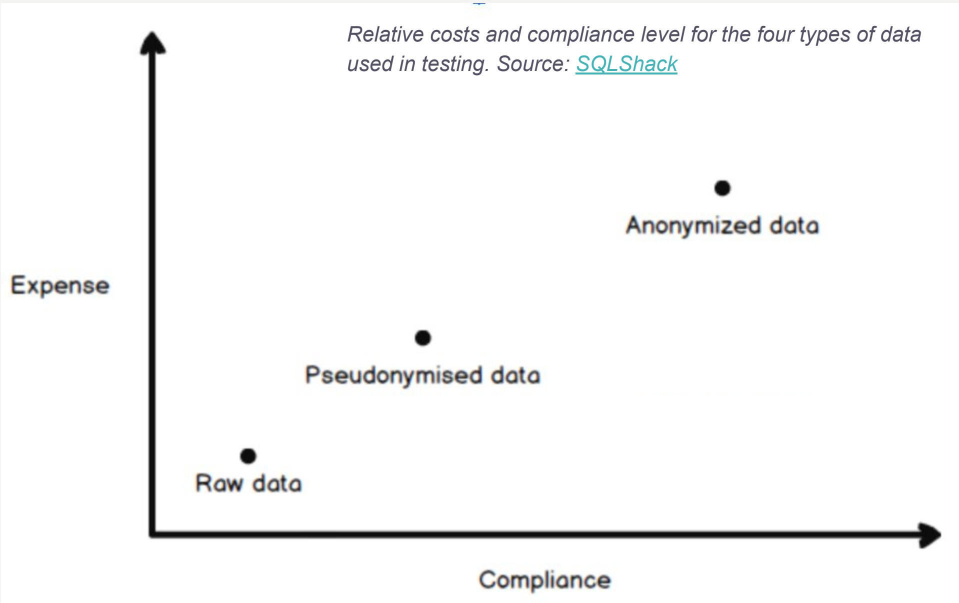




Why DPFree Data is better?



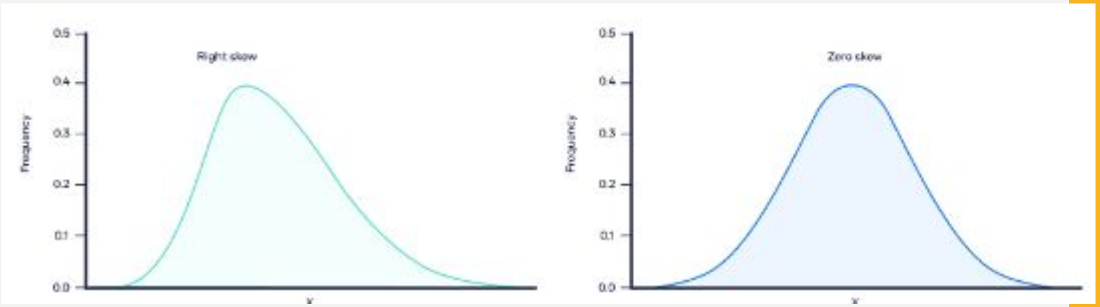
Simulated average accuracy: ***90.16%****, original: 85.79%.*
Average improvement: 4.37%



most other solutions mask PII or directly encrypt it, keeping the rest of the data vulnerable. This can be easily reverse engineered causing a data leak

https://www.kaggle.com/ronitf/heart-disease-uci

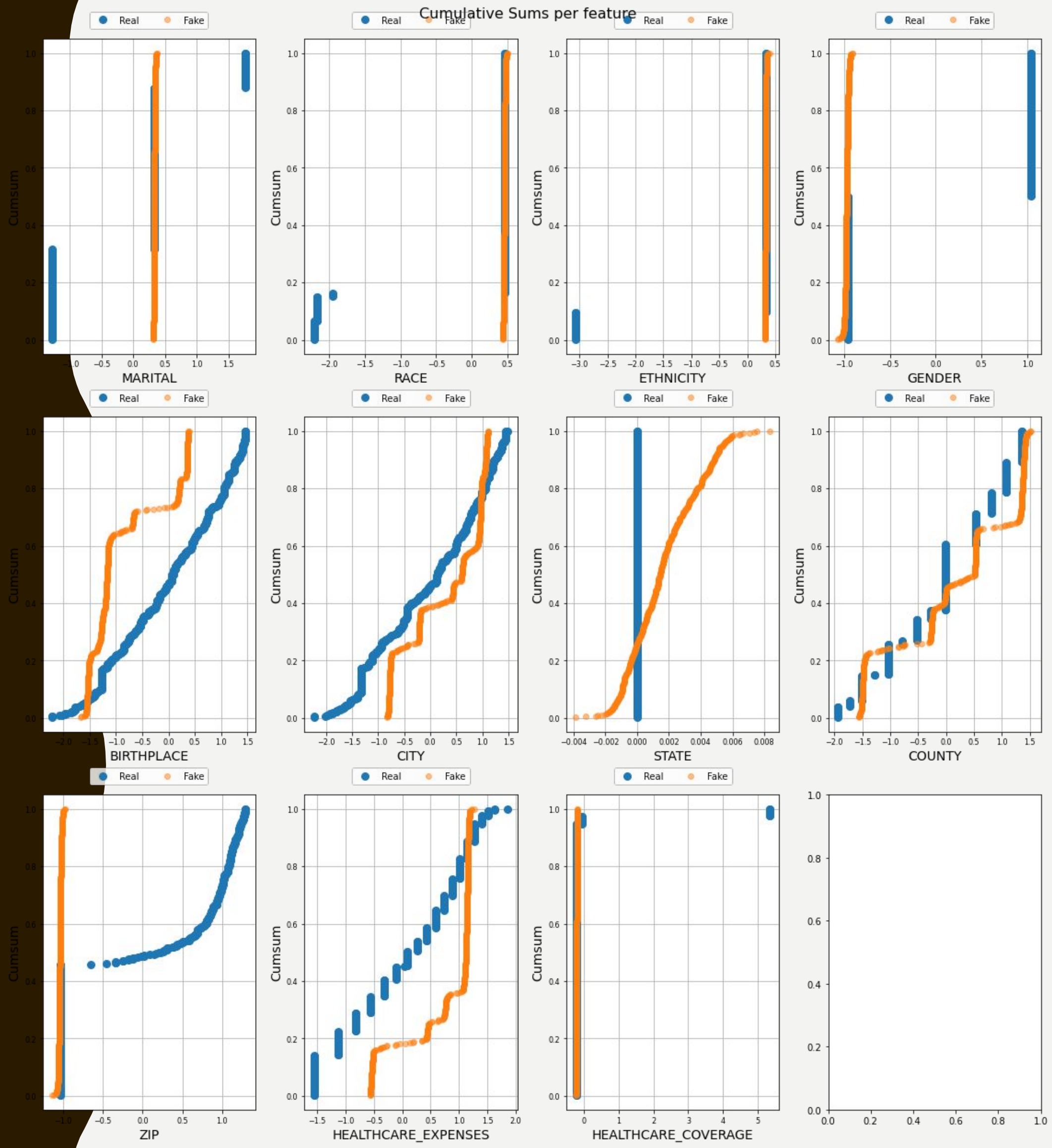
Using a popular Heart Disease dataset that was skewed almost 2-to-1 towards male patients, we added synthetically generated patient records to a training set to boost female representation in the dataset.



observations from the underlying statistical distribution and correlations.

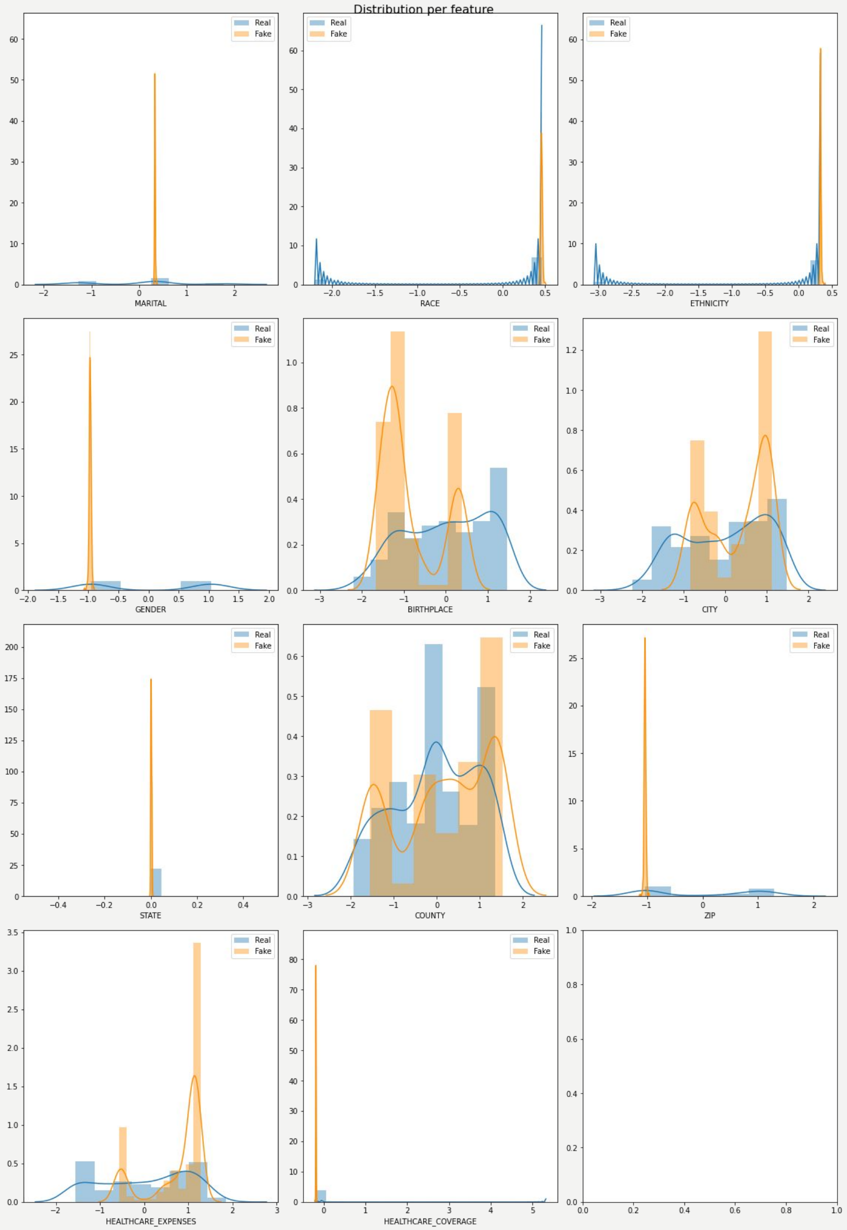
Rare cancers, Covid19, Financial data

EVALUATION METRICS



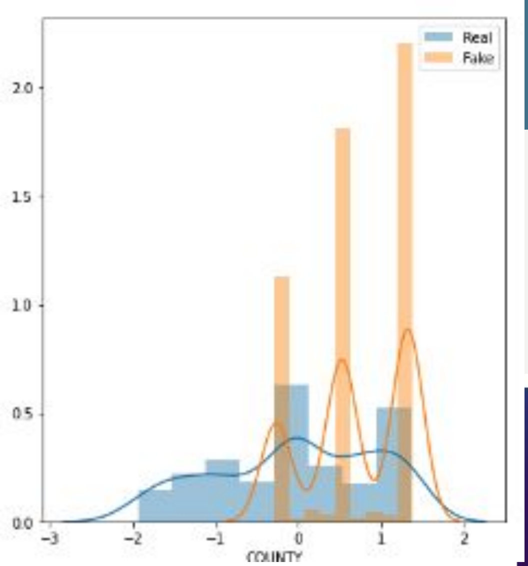
Cumulative sum

Distributions visualised



Why My Project **Wins**

Intel CEO Pat Gelsinger Congratulates Global Winners from Intel® AI Global Impact Festival 2022



[https://www.**bloomberg**.com/press-releases/2022-09-21/intel-announces-ai-global-impact-festival-grand-prize-winners](https://www.bloomberg.com/press-releases/2022-09-21/intel-announces-ai-global-impact-festival-grand-prize-winners)

San Jose, California, USA

Sayash Raaj – India

This solution generates artificially manufactured data using Intel® technologies that include Intel® Core™ i5, Intel® Optimization for TensorFlow, PyTorch, and Intel® Extension for Scikit-learn. This solution helps in mitigating the unavailability of data due to inefficient logistics, and sensitivity of data.

Global Industrial Testimonials

My claims backed with proofs from global corporations and experts

Intel USA

Global Winner 2022

Represented India

Broadridge Finance US

Global Winner 2023

Integrated a mathematical approach

Shell AI

Global Winner 2022

Transportational data

OpenText API

```python
# Function to authenticate and get a session token
def get_opentext_session_token(api_url, username, password):
    url = f"{api_url}/api/v1/auth"
    payload = {"username": username, "password": password}
    response = requests.post(url, data=json.dumps(payload), headers={"Content-Type": "application/json"})
    if response.status_code == 200:
        return response.json().get('ticket')
    else:
        raise Exception("Authentication failed")

# Function to upload a CSV file to OpenText Content Server
def upload_csv_to_opentext(api_url, session_token, csv_file_path, folder_id):
    url = f"{api_url}/api/v2/nodes"
    headers = {
        "OTCSTicket": session_token,
        "Content-Type": "multipart/form-data"
    }
    files = {
        'type': (None, '144'),  # 144 is the type for Document
        'parent_id': (None, folder_id),
        'file': open(csv_file_path, 'rb')
    }
    response = requests.post(url, headers=headers, files=files)
    if response.status_code == 200:
        return response.json()
    else:
        raise Exception("File upload failed")

# Function to download a CSV file from OpenText Content Server
def download_csv_from_opentext(api_url, session_token, file_id, download_path):
    url = f"{api_url}/api/v2/nodes/{file_id}/content"
    headers = {
        "OTCSTicket": session_token
    }
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        with open(download_path, 'wb') as file:
            file.write(response.content)
        return f"File downloaded successfully to {download_path}"
    else:
        raise Exception("File download failed")
```

opentext™

Using **Content Server REST API** for uploading and downloading tabular, relational csv files.

THANK YOU

DPFree DATA GENERATION

The future of Data Science

The future of humanity

Sayash Raaj,

IIT Madras

Special Thanks for conducting this event

The hard work during late nights while balancing academics at IIT Madras has been fueled by motivation adequately provided

If it saves a life, I will consider my work successful.

https://www.linkedin.com/in/sayashraaj/