

Imputation of missing medical data using GANs and graph representation learning

STUDENT: VEBJØRN HALVORSEN

SUPERVISORS: BENJAMIN RICAUD AND LUIS ALFREDO MOCTEZUMA

UiT - The Arctic University of Norway

November 22, 2022

Contents

1	Introduction	3
2	Missing data	3
2.1	Types of missing data	4
2.2	Missing completely at random (MCAR)	4
2.3	Missing at random (MAR)	4
2.4	Missing not at random (MNAR)	5
3	Simple imputation algorithms	5
3.1	Mean imputation	5
3.2	Imputation using most frequent	5
3.3	k-Nearest Neighbor Imputation	6
3.4	Multiple Imputation	7
3.5	Calculating error and score	8
4	Advanced imputation algorithms	8
4.1	Multivariate Imputation by Chained Equations	8
4.1.1	MICE steps	9
4.2	MissForest imputation	10
4.3	GAIN imputation	11

5	Generative Adversarial Networks (GAN)	11
5.1	Terminology	11
5.1.1	Generative	11
5.1.2	Adversarial	12
5.1.3	Networks	12
5.2	Generator	12
5.3	Discriminator	12
5.4	G vs D	13
5.5	Backpropagation	13
6	Generative Adversarial Imputation Network (GAIN)	15
6.1	Generator	16
6.2	Discriminator	17
6.3	Objective	17
7	The dataset	18
7.1	Pre-processing data	20
7.1.1	Standardization	20
7.1.2	One-Hot Encoding	20
8	Results from different papers	21
9	My results	21
9.1	Available imputation algorithms	21
9.2	Results	22

1 Introduction

A common problem when working with medical data is that some data can be missed because it was deemed unnecessary, unimportant, accidentally not recorded, because it is difficult or dangerous to collect or because it was never collected. Missing data are an important problem in machine learning as most of the algorithms expect a fixed size feature vector as input, filled with numerical values. A well-established solution is the use of data imputation algorithms, which sometimes can fill the datasets with wrong information. This may produce biased or not well-fitted machine learning models.

This project aims to test current state-of-the-art methods and propose a new architecture that can outperform current algorithms, completing the missing values into own and third-party datasets in most accurate way. The first part of the project will test the best possible methods and optimize the parameters to increase the performance. Based on this, the second part (masters semester) will propose a new ML/DL architecture. This new architecture should have 2 goals: 1) improve the quality of the imputation 2) be more explainable and give insight on which features are important for guessing the missing values. The second goal is particular useful for understanding relationship between features and revealing hidden correlations between them. Combining actual methods with a graph approach is envisioned.

2 Missing data

In several fields of data science and machine learning, missing data is a typical occurrence. Unfortunately, it may be difficult to deal with successfully, and there is rarely no one-size-fits-all answer. Especially in machine learning, depending on how you manage missing values, they can have significant impact on your models and the problem you're trying to solve [10].

Missing data often occurs when performing some sort of data collection. Whether it comes from health care, psychology studies, or some other area of data harvesting. With today's modern technology, we have the opportunity to use algorithms in combination with coding, to process the data and make predictions with it. The importance of data imputation might not be intuitive at first, but in health care for instance, the

predictions from the machine learning algorithms can be the difference between life and death. Since the algorithm used, close to no exception need complete datasets, missing data can be problematic. When we are introduced to a dataset with missing values, data imputation proposes a solution to make the dataset complete. Imputation simply means replacing the missing values with some guessed or estimated ones and there are many different ways of doing so.

2.1 Types of missing data

There exist three different types of missing data that is generally categorized. Missing completely at random (MCAR), missing at random (MAR) and missing not at random (MNAR). Each form of missing data may appear in your data, or even a mix of several types of missing data.

2.2 Missing completely at random (MCAR)

This is missing data that follows no known pattern. The values chosen for this type imputation often comes down to a coin flip. MCAR values are often identified when the remaining non missing values in the dataset can not be used for predicting the missing values. They are missing completely at random.

This type of missing data is however not usually the case. One example of this is data generated randomly on purpose. In short, assuming the data are missing completely at random is often not a reasonable assumption unless the methods used while producing the data is very well known [10].

2.3 Missing at random (MAR)

For MAR data we are allowed to assume that the missing data can be imputed using the remaining data that are not missing. This implicates that there exist some correlation or dependencies between the data in the dataset. This does however not mean that it should be or is possible to predict the missing values with 100% accuracy. Rather it makes us able to require a probabilistic relationship which hopefully is far better than a coin flip [10].

2.4 Missing not at random (MNAR)

Sometimes the case is that there exist no way to confidently make a conclusion about what value to impute for the missing data. Lets say the data set with missing data came from examination about peoples different incomes. People with very high or very low income might refuse to answer for privacy reasons or simply some arbitrary reason we dont know about. When there exist a good reason for the missing data it is categorized as missing not at random. When we are dealing with this type of missing data we cannot use any of the standard imputations method to get accurate values [10].

3 Simple imputation algorithms

3.1 Mean imputation



	col1	col2	col3	col4	col5
0	2	5.0	3.0	6	NaN
1	9	NaN	9.0	0	7.0
2	19	17.0	NaN	9	NaN

	col1	col2	col3	col4	col5
0	2.0	5.0	3.0	6.0	7.0
1	9.0	11.0	9.0	0.0	7.0
2	19.0	17.0	6.0	9.0	7.0

Figure 1: Simple illustration of small dataset applied with mean imputation [10]

This imputation method uses column-wise mean for filling in missing values (see figure 1). This is a very fast and easy to implement method but its generally pretty terrible compared to other algorithms and methods. Column-wise mean imputation only works on columns and not rows, therefore missing the correlation that might be between features. This make this algorithm not very accurate.

3.2 Imputation using most frequent

This is another really simple imputation method. As the name suggests, the imputed values are just the most frequent value from each column.

3.3 k-Nearest Neighbor Imputation

This algorithm finds the K "nearest" (most similar) rows and average their values. KNN imputation assumes only numerical data and not categorical. One can imagine having some distance between the datapoints using the features, and the finding the k closest datapoints to the one with missing values and use there values in order to impute the missing values.

So, the idea of the kNN methods is simply to find the "k" samples in the dataset that are similar or close in space and use these "k" samples to impute the value of of the incomplete datapoints. First we find the closest datapoints by calculating the euclidean distance. Lets look at a simple example.

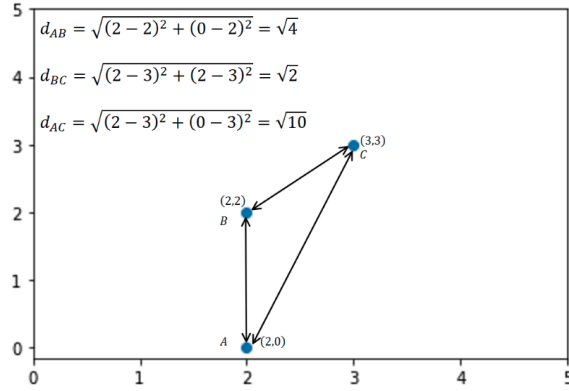


Figure 2: A simple example of 2D points and euclidean distance [6]

From figure 2 we can see that the nearest neighbour to point A is point B, and the nearest neighbour for point B is C. This example uses datapoints with complete data. In order to calculate the distance when there is missing data in a datapoint we use a modified version of euclidean distance formula called nan euclidean distance.

$$d_{xy} = \sqrt{\text{weight} \times \text{squared distance from present coordinates}} \quad (1)$$

where,

$$\text{weight} = \frac{\text{Total number of coordinates}}{\text{Number of present coordinates}} \quad (2)$$

So, the euclidean distance between (4, nan, 8) and (2, 3, 4) is:

$$\sqrt{\frac{3}{2}((4-2)^2 + (8-4)^2)} = 30$$

Lets say we want to use the $k = 2$ nearest neighbours of a datapoint with missing values to impute its value. Then we would take the mean of the the values in the the k nearest datapoints dimension which correspond to the missing values dimension. If the two closest datapoints to $(4, \text{nan}, 8)$ where $(2, 3, 4)$ and $(1, 1, 1)$ we could replace the nan value with $\frac{3+1}{2} = 2$ [6].

3.4 Multiple Imputation

The imputation methods mention until now does not reflect the uncertainty about the predicted values. Instead of imputing single values for the missing data, a multiple imputation method replaces each missing one with a set of probable ones that represent the uncertainty of the missing value to impute [2]. The missing values are imputed M times to produce M complete datasets using an appropriate model that incorporates random variation. Multiple imputation can be summarized in three distinct phases:

- The missing data is filled M times to generate M complete data sets.
- The M complete data sets are analyzed by using standard procedures.
- The results from the M complete data sets are combined for the inference.

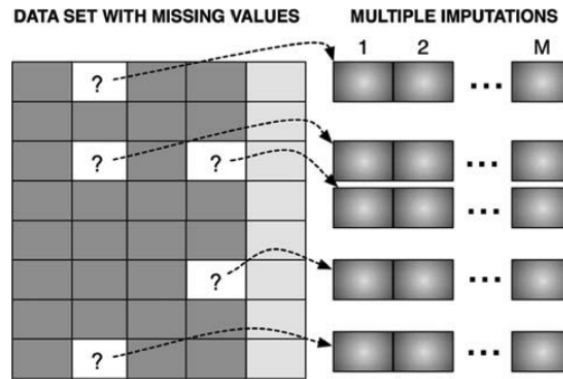


Figure 3: Schematic representation of multiple imputations. The missing values are imputed M times to produce M complete data sets using an appropriate model that incorporates random variation

3.5 Calculating error and score

In order to know how well the imputation algorithms perform we need some sort of loss or error function. In this article [9], it was used RMSE (Root mean squared error) 3 for the standard numeric dataset.

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{i=1}^N (\text{real}_i - \text{imputed}_i)^2} \quad (3)$$

where N represents the number of samples.

For another dataset consisting of images of numbers between 0 and 9, FID (see equation 4) (Fréchet inception distance) was used as error measurement. FID measures similarity between the generated images with missing values and the real ones.

$$\text{FID}(i, r) = \|\mu_i - \mu_r\|_2^2 + \text{Tr} \left(\sum_i + \sum_r - 2 \left(\sum_i \sum_r \right)^{\frac{1}{2}} \right) \quad (4)$$

where (i, r) represents the imputed image and the real image. μ_i and Σ_i are the mean-value and covariance matrix of the imputed images vectors, the same as μ_r and Σ_r . Tr denotes the trace of the matrix.

4 Advanced imputation algorithms

4.1 Multivariate Imputation by Chained Equations

Mice is a particular multiple imputation technique and operates under the assumption that the given variables used in the imputation procedure are of the MAR type (see section 2.3). Using MICE when the data is not MAR type could result in biased imputation estimates [1].

4.1.1 MICE steps

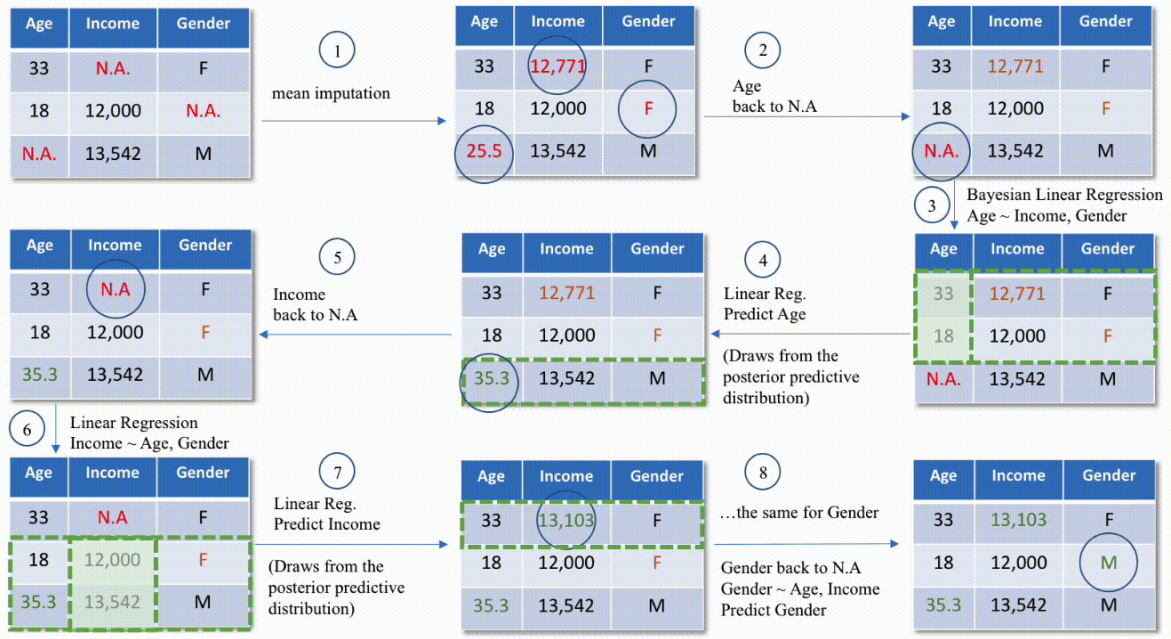


Figure 4: Illustration of a single MICE algorithm iteration

By following the illustration in figure 4 one can observe how the MICE iteration goes to action.

- Step 1: Imputing the missing values in the dataset by using very simple imputation like mean imputation. These imputed values are only temporary.
- Step 2: Set back one of the mean imputed values, but only one. Now we only have one missing value in the entire dataset. From the illustration in figure 4 this is the age variable.
- Step 3: Regress age on income and gender by the use of linear regression on the remaining dataset. We are dropping all the rows where age is missing when fitting the regression. In the illustration 4 until now, age is the dependent variable and income and gender are the independent variables.
- Step 4: Now we use the fitted regression model and impute a value for the relevant missing value, in our case the age value. Now we have imputed one value and we are ready to set back another of the original missing values.

- Step 5: Now we repeat steps 2-4 for all the variables that was originally missing in the dataset, until all the missing data has been imputed by the use of linear regression [1].

When all missing variables have been imputed the way illustrated in figure 4 and explained in the steps above, one iteration of the MICE algorithm is complete. When using this algorithm one can choose how many iterations/cycles they want. The main idea of doing many iterations of complete imputations is to avoid dependency on the order of which the values are imputed. For example in illustration 4 age is imputed first, and then the regressed imputed value for age is then later used to impute the other missing values. If the order of missing values imputed was change to for instance income variable before age, the imputed values might have changed. The illustrated iteration also contains a very small dataset. The impact from change of order would be much greater as the datasets become bigger. So, at the end of multiple cycles the goal is to have the parameters governing the imputation, for example the coefficients in the regression model should in some sense have converged and become stable.

4.2 MissForest imputation

This advanced imputation method introduces Random forest to the mix. The missForest algorithm can be summarized as follows [4]:

1. Initialization in order to utilize Random Forest algorithm. All the missing variables will be temporary imputed using mean imputation (most frequent class for categorical data).
2. The imputation process starts and is done sequentially for the variables in the data in either ascending or descending order of variable missingness. The data is split into two parts: The first for non missing variables and the second for missing variables. The non missing portion is used as training set, and the missing portion as test set. The missing values in the missing variables are replaced by predictions from the random forest models.
3. When all the variables with missing data is imputed one iteration of the missForest algorithm is complete. The iterations continuous as long as the relative sum of squared difference between the current and previous imputation result

decreases (result gets better), or to some self defined iteration limit. When the sum of squared differences start to increase the missForest algorithm no longer improves and will output the previous imputation as the final result.

KANSKJE IKKE HELT FERDIG.....

4.3 GAIN imputation

This is a highly advanced and accurate imputation technique which involves the use of Generative Adversarial Network (see section 5). The generator (G) observes certain elements of a real data vector, imputes missing components based on what is observed, and returns a full vector. Next the discriminator (D) takes the imputed vector and tries to figure out which of the components were real and which were imputed by the generator. The discriminator is trained to minimize classification loss and the generator is trained to maximize the discriminators misclassification rate. Because of their contradictory goals these two networks are trained in an adversarial manner. A much more in depth explanation can be seen in section 5 and 6.

5 Generative Adversarial Networks (GAN)

GANs land in the category of machine learning frameworks and was originally designed by Ian Goodfellow and his colleagues in June of 2014. In short, it works by two neural networks which contest with each other in a zero sum game. One algorithm's gain is the other's loss. When given a training set of data, GANs learn to generate new data with the same statistical characteristics as the training set. One example of this might be images as training set, and the GANs generating new images that look at least superficially authentic to a human observer. GANs were originally proposed as a form of generative model for unsupervised learning. [3]

5.1 Terminology

5.1.1 Generative

As the name suggests this part of the algorithm generates data. The data that is generated is so called "fake" data which is not included in the ground truth.

5.1.2 Adversarial

This is the part where the "Generator" and "Discriminator" compete with each other to win. The generator trying to fake the data and the discriminator trying to not get "fooled".

5.1.3 Networks

This part stand for the networks being used by the generator and discriminator for training. Can be a deep convolutional network, can be a fully connected Neural network and so forth.

5.2 Generator

The generator has one job and that is to create fake data so close to the real data from the discriminators training set that the discriminator can not distinguish the generated data from the real one. In order for the generator to do so it has to be trained, often for many epochs. For the sake of explanation, we assume we want to create some fake images, but images are only one area of data where GANs are usefull. We start by sampling some noise z using a normal or uniform distribution. With z as input we use the generators neural network (often CNN since we are dealing with images) to generate a new image $x = G(z)$. In conceptual terms z represents the latent features of the generated images, for instance color and the shape and so forth. We don't control what features the model is learning, this is left to the automatic training process to decide. The generator alone cant do anything other than create new noise and this is where the discriminator comes into play. [5].

5.3 Discriminator

The discriminators biggest desire is to be able to separate the real from the fake images. with differences no human would be able to spot with the naked eye. In order to train the discriminator for this purpose, the discriminator is trained by looking at real images and generated images separately. See figure 5. The output given from the discriminator comes in the form of probability that the input image comes from the real images and not the generated ones. If the input is real we want the output to be 1. If the input is generated we want the output to be 0. In order to train the discriminator we use

the same approach as for a deep neural network classifier. By repeating this process over some given number of epochs, the discriminator is able to provide us with the specific features of the images that contribute to real images. On the other hand we want the generator to generate images with output equal to 1, in other words fool the discriminator by creating very good fakes. We can train the discriminator by backpropogating the output error all the way back to the generator, so the generator is able to create images that are closer to fool the discriminator [5].

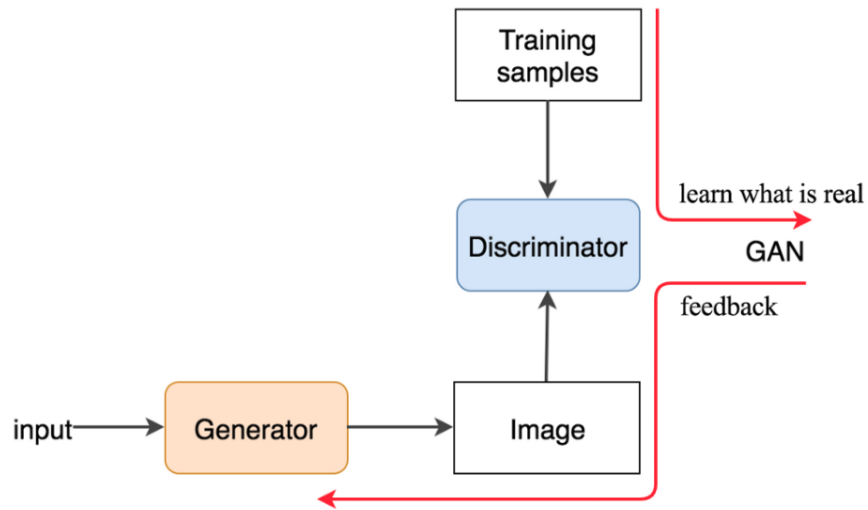


Figure 5: Illustration of the architectural characteristics of a GAN.

5.4 G vs D

Now we let the discriminator and the generator fight through the epochs. Eventually, the discriminator is able to identify very tiny differences between real and generated images, and the generator creates images that the discriminator is unable to distinguish from the real ones. Eventually, GAN will converge and produce images that look as natural as the real images.

5.5 Backpropagation

As mentioned earlier, the discriminator outputs some probability that tells us if the image is more likely real or more likely fake. Lets call this value $D(x)$. Our objective

is to maximise the chance to recognize real images as real and generated images as fakes. In order to measure the loss of the output we use *cross-entropy* [5]. So for the discriminator the objective becomes 5:

$$\max_D V(D) = \underbrace{E_{x \sim p_{data}(x)}[\log D(x)]}_{\text{recognize real images better}} + \underbrace{E_{z \sim p_z(z)}[\log(1 - D(G(z)))]}_{\text{recognize generated images better}} \quad (5)$$

From the generators point of view, the objective becomes to generate highest possible value of $D(x)$ in order to fool the discriminator 6.

$$\min_G V(G) = \underbrace{E_{z \sim p_z(z)}[\log(1 - D(G(z)))]}_{\text{Optimize G that can fool the discriminator the most}} \quad (6)$$

GAN is often defined as a **minimax** game which G wants to minimize V while D wants to maximize it (see equation 7) [5].

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (7)$$

Once both objectives are defined, they are learned jointly by the alternating gradient descent. The generators parameters are recalculated first, then a single iteration of gradient descent is performed on the discriminator using the real and the generated images. Then the sides are flipped. Now the discriminators parameters are recalculated and the generator is trained for another iteration. Both the networks are trained alternatively until the generator produces images of high enough quality. See figure 6 for illustration and 7.

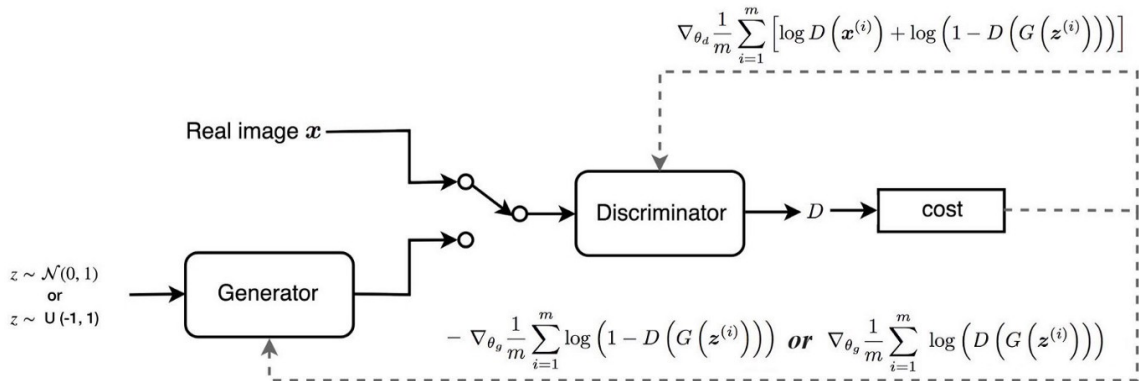


Figure 6: Illustration of the GANs training cycle [5].

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k , is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Sample minibatch of m examples $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\mathbf{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D(\mathbf{x}^{(i)}) + \log \left(1 - D(G(\mathbf{z}^{(i)})) \right) \right].$$

end for

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log \left(1 - D(G(\mathbf{z}^{(i)})) \right).$$

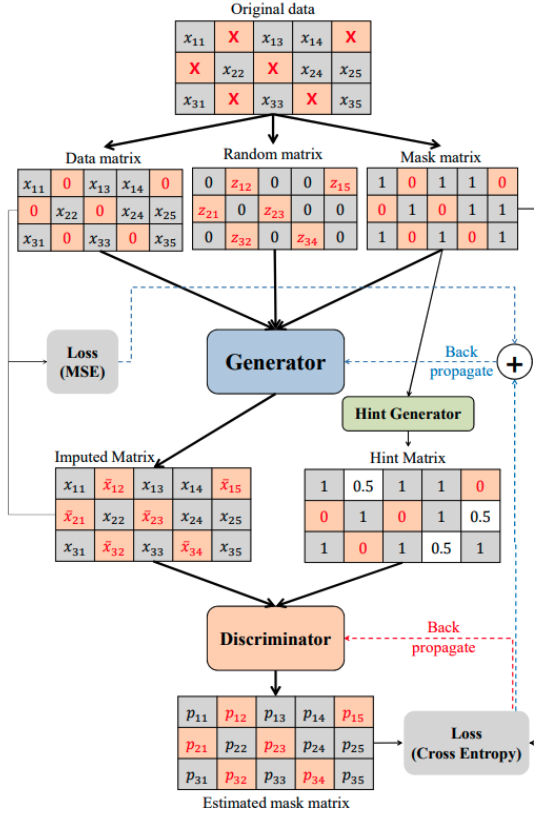
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Figure 7: Pseudo code for how the GAN is trained [5].

6 Generative Adversarial Imputation Network (GAIN)

GAIN is a modification of a normal GAN, where the I simply stand for Imputation. In short, the idea behind this algorithm is as follows. The Generator (described above) takes the vector of real data which has some missing values and impute them accordingly. The newly imputed data is then fed back to the Discriminator which then tries to figure out which data was originally missing. See figure 8a.



(a) Illustration of the architecture which makes the GAIN algorithm work [12]

Algorithm 1 Pseudo-code of GAIN

while training loss has not converged **do**

(1) Discriminator optimization

 Draw k_D samples from the dataset $\{(\tilde{\mathbf{x}}(j), \mathbf{m}(j))\}_{j=1}^{k_D}$

 Draw k_D i.i.d. samples, $\{\mathbf{z}(j)\}_{j=1}^{k_D}$, of $\mathbf{Z}$

 Draw k_D i.i.d. samples, $\{\mathbf{b}(j)\}_{j=1}^{k_D}$, of $\mathbf{B}$

for $j = 1, \dots, k_D$ **do**

$\tilde{\mathbf{x}}(j) \leftarrow G(\tilde{\mathbf{x}}(j), \mathbf{m}(j), \mathbf{z}(j))$

$\hat{\mathbf{x}}(j) \leftarrow \mathbf{m}(j) \odot \tilde{\mathbf{x}}(j) + (\mathbf{1} - \mathbf{m}(j)) \odot \mathbf{x}(j)$

$\mathbf{h}(j) = \mathbf{b}(j) \odot \mathbf{m}(j) + 0.5(\mathbf{1} - \mathbf{b}(j))$

end for

 Update D using stochastic gradient descent (SGD)

$$\nabla_D - \sum_{j=1}^{k_D} \mathcal{L}_D(\mathbf{m}(j), D(\hat{\mathbf{x}}(j), \mathbf{h}(j)), \mathbf{b}(j))$$

(2) Generator optimization

 Draw k_G samples from the dataset $\{(\tilde{\mathbf{x}}(j), \mathbf{m}(j))\}_{j=1}^{k_G}$

 Draw k_G i.i.d. samples, $\{\mathbf{z}(j)\}_{j=1}^{k_G}$ of $\mathbf{Z}$

 Draw k_G i.i.d. samples, $\{\mathbf{b}(j)\}_{j=1}^{k_G}$ of $\mathbf{B}$

for $j = 1, \dots, k_G$ **do**

$\mathbf{h}(j) = \mathbf{b}(j) \odot \mathbf{m}(j) + 0.5(\mathbf{1} - \mathbf{b}(j))$

end for

 Update G using SGD (for fixed D)

$$\nabla_G \sum_{j=1}^{k_G} \mathcal{L}_G(\mathbf{m}(j), \hat{\mathbf{m}}(j), \mathbf{b}(j)) + \alpha \mathcal{L}_M(\mathbf{x}(j), \tilde{\mathbf{x}}(j))$$

end while

(b) Pseudo code of the GAIN algorithm

Figure 8

From figure 8a we can see some different matrices. We have the original matrix with missing values which becomes a Data matrix with zeros for missing values, a Random matrix and a Mask matrix. The mask matrix tells the generator which values are missing and which ones that are present. Ones for present and Zeros for missing. The random matrix applies randomness to the imputed values, so they are not the same every time. We also have something called a Hint matrix which is an addition to the Discriminator and makes sure that the Discriminator *forces* the Generator to learn.

6.1 Generator

The generator G takes the original ($\tilde{X}$), mask (M) and random matrix (Z) as input (see figure 8a). Let G be a function and Z be d -dimensional noise which is independent of all other variables. Then we define the random variables $\tilde{X}$, $\hat{X} \in x$ by

$$\bar{X} = G(\tilde{X}, M, (1 - M) \odot Z) \quad (8)$$

$$\hat{X} = M \odot \tilde{X} + (1 - M) \odot \bar{X} \quad (9)$$

where $\odot$ symbolises element-wise multiplication. $\bar{X}$ corresponds to the vector of imputed values. Something to keep in mind is that G outputs a value for every component, even if its value was observed. $\hat{X}$ is the complete imputed data vector which has the non-missing values from $\tilde{X}$ and each missing value replaced with the use of the matrix $\bar{X}$. This is similar to a normal GAN, but in order to make the element-wise multiplication work as intended the noise passed to the generator is of the form $(1 - M) \odot Z$.

6.2 Discriminator

As in a standard GAN algorithm, also here the discriminator is introduced, but with some key differences. Unlike the standard GAN where portions of the data is complete fake or completely real, the output dealt with from GAIN consist of some components that are real and some that are fake. Instead of identifying if the entire vector is real or fake, this version of the discriminator tries to separate the real components (not missing) from the fake (imputed). For this reason the Mask matrix come in handy. By defining the $\mathbf{H}$ -matrix in different ways, it is possible to control the amount of information contained in $\mathbf{H}$ about the $\mathbf{M}$ -matrix (Mask) in particular. It is shown that if not enough information about the mask matrix is provided to the Discriminator, for example leaving out the hinting mechanism entirely, we are left with several distributions that the Generator could produce which the Discriminator would be *happy* with, which is not desirable [12].

6.3 Objective

The objective is very similar to a normal GAN. We want to train the discriminator to maximize the probability of correctly predicting M . The generator we train to minimize probability of the discriminator predicting M . In order to do so we need to define the quantity $V(D, G)$ to be:

$$V(D, G) = \mathbb{E}_{\hat{\mathbf{X}}, \mathbf{M}, \mathbf{H}} \left[\mathbf{M}^T \log D(\hat{\mathbf{X}}, \mathbf{H}) + (\mathbf{1} - \mathbf{M})^T \log(1 - D(\hat{\mathbf{X}}, \mathbf{H})) \right] \quad (10)$$

As for the normal GAN we now define a minimax problem:

$$\min_G \max_D V(D, G) \quad (11)$$

We also define a loss function $\mathcal{L}$:

$$\mathcal{L}(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^d [a_i \log(b_i) + (1 - a_i) \log(1 - b_i)] \quad (12)$$

And then, by writing $\hat{\mathbf{M}} = D(\hat{\mathbf{X}}, \mathbf{H})$ [12], we can rewrite equation 11 as:

$$\min_G \max_D \mathbb{E}[\mathcal{L}(\mathbf{M}, \hat{\mathbf{M}})] \quad (13)$$

In figure 8b we can see a simple pseudo code for how to GAIN algorithm works code wise.

7 The dataset

The datasets given from the supervisors is a large medical datasets. One consisting of complete data with shape (13100, 89) and the other with missing data and shape (1385, 89). Each row represents one patient or subject, and the columns consist of different medical features such as height, weigh, different medical states both physical and mental and also different socioeconomic features such as disability benefits. Exactly what the features represent are not that important, but the correlation between them is. Many of the algorithms tested on the dataset use correlation between features in order to impute missing values accurately (MAR type). Below are a couple of tables representing start, middle and end of the datasets columns with the first seven rows.

Alder	Kjonn	Roker	Utd	Vekt	Hoyde	BMI
65.0	2.0	0.0	5.0	73.0	170.0	25.26
44.0	1.0	0.0	1.0	102.0	190.0	28.25
51.0	2.0	0.0	1.0	89.0	168.0	31.53
37.0	2.0	1.0	2.0	79.0	168.0	27.99
81.0	1.0	0.0	4.0	74.0	185.0	21.62
47.0	2.0	0.0	2.0	75.0	170.0	25.95
76.0	2.0	0.0	4.0	74.0	173.0	24.73

Table 1: First seven rows and first seven columns from the dataset without missing values

RfSynovpre	RfSpondtypeIs misk	RfSpondtypeDegen	RfDegskol	OpIndSme	OpIndSmeType	OpIndParese
0.0	0.0	0.0	0.0	1.0	3.0	0.0
0.0	0.0	0.0	0.0	1.0	3.0	0.0
0.0	0.0	0.0	0.0	1.0	2.0	0.0
0.0	0.0	0.0	0.0	1.0	3.0	0.0
0.0	0.0	0.0	0.0	1.0	2.0	0.0
0.0	0.0	0.0	0.0	1.0	3.0	0.0
0.0	0.0	0.0	0.0	1.0	2.0	0.0

Table 2: First seven rows and 7 columns from the middle from the dataset without missing values

SmRyPre	SmBePre	OswTotPre	EQ5DPre	Failure_GPE	Worse_GPE	success_GPE
9.0	9.0	36.0	0.159	0.0	0.0	0.0
5.0	5.0	50.0	0.516	0.0	0.0	0.0
8.0	9.0	58.0	-0.113	0.0	0.0	0.0
5.0	6.0	26.0	0.689	0.0	0.0	0.0
8.0	7.0	56.0	0.516	0.0	1.0	0.0
5.0	6.0	22.0	0.760	0.0	0.0	0.0
9.0	9.0	46.0	0.587	0.0	0.0	0.0

Table 3: First seven rows and 7 last columns from the dataset without missing values

From table 1, 2 and 3 we can see a little sample of some of the 89 different features present in the dataset. Something that is noticeable is the different data units. Some contain numbers in the range (0, 1), others are binary and some are categorical. Also

one important factor is that the last three columns acts as ground truth for a potential machine learning algorithm.

7.1 Pre-processing data

In order for machine learning and imputation algorithms to have the highest accuracy possible some data pre-processing is often needed. This might involve standardization, handling categorical data, One-Hot Encoding and so forth [7].

7.1.1 Standardization

Standardization is needed in order to keep each feature, at least at the beginning, equally important. If we for instance look at the weight and age columns in table 1, we can see from the present datapoints and imagine, that weight will be higher than age in several cases and this will often implicate that algorithms will treat weight more importantly (weighted) than age. The numbers are simply not on the same scale but the algorithms don't have that information to take into account. In simple terms standardization calculates the mean and standard deviation column wise and subtract the mean and divide by the standard deviation for each value in the column. See equation 14.

$$z = \frac{x_i - \mu}{\sigma} \quad (14)$$

7.1.2 One-Hot Encoding

Often when dealing with datasets, and especially health sector datasets, the features have values based on the different categories/classes. For instance the *Kjonn* and *UTD* features seen in table 1. In *Kjonn* feature which in English translates to gender, we can see some variables having 1 and some 2 as values. The whole column consist of these two values. One other way of representing this binary column is to have two columns, both containing only 1s and 0s as values. They might have been named *Kjonn1* and *Kjonn2* or something similar. The first column would correspond to the values from the original column being equal to 1, and the second column to the values being equal to 2. This essentially is called one-hot encoding. See the simple example of how one-hot encoding works in equation 15 below.

$$\begin{array}{ccc}
K_{jonn} & & K_{jonn1} \quad K_{jonn2} \\
\left[\begin{array}{c} 2 \\ 1 \\ 2 \\ 2 \\ 1 \\ 2 \\ 2 \end{array} \right] & = & \left[\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{array} \right] \left[\begin{array}{c} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{array} \right]
\end{array} \tag{15}$$

So, if a column contains n classes we would get n columns after one hot encoding. By changing columns containing many classes into many columns with one class, we make the life of imputation algorithms a lot easier.

8 Results from different papers

9 My results

By using the given dataset, python, some different python libraries and some code from github I was able to test some of these imputation algorithms myself. In order to test and produce some score, the ground truth of the imputation had to be known (see section 3.5). Therefore the testing was done on the complete portion of the medical dataset. This implicates that the missingness of the data has to be created artificially. The creating of missing data is done randomly, making the missing type MCAR (see section 2.2), but the assumption of the missingness is still going to be of type MAR (see section 2.3). One preprocessing technique that was used on the dataset was to use one-hot encoding on categorical columns. See section 7.1.2.

9.1 Available imputation algorithms

Python has become a very popular coding language, topping the worldwide PYPL index [8]. This means that there exist a waste number of libraries one can utilize if needed. This made it possible to implement and test different algorithms without having to write all the code myself. Also for the GAIN algorithm (see section 6) there existed a github page with available code [11]. In order to see performance for the

different algorithms missing rate in the range $\text{missing_rates} \in [0, 0.50]$ with stepsize of 0.10. Different number of samples was also used. Some of the algorithm was really slow, so testing with smaller number of samples was a nice way to speed up the process. Below is table of different libraries and algorithms used:

Algorithms	scikit-learn	impyute	fancyimpute	missingpy	github
Mean	*				
kNN	*		*		
MICE	*		*		
missForest				*	
EM		*			
GAIN					*

9.2 Results

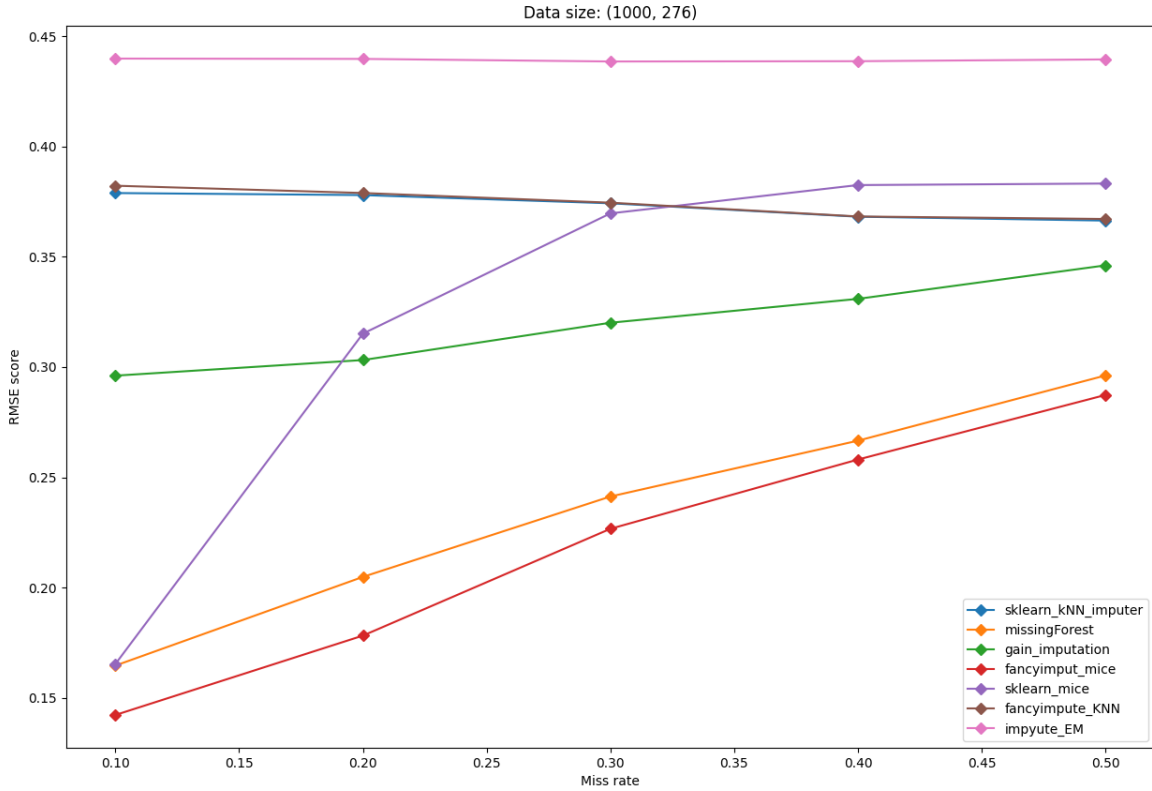


Figure 9: Results from the different algorithms with a sample size of 1000

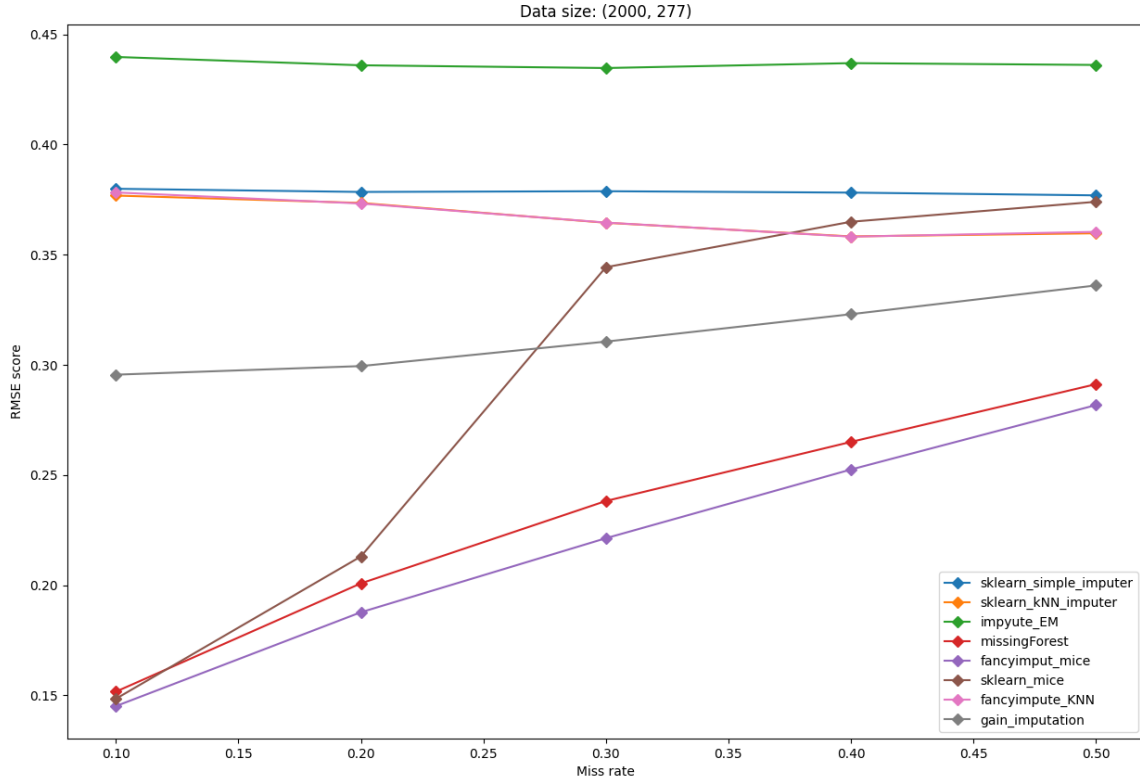


Figure 10: Results from the different algorithms with a sample size of 2000

From the results seen in figures 9 and 10 one can easily see that the best algorithms are the mice algorithm from the fancyimpute library and the missForest algorithm from the missingpy library. The estimation maximization from impute does the absolute worse, and I decided after testing on samples of size 1000 and 2000 that I was not going to spend more time and process power on that specific algorithm. Despite the fact that missForest algorithm seem to do a nice job it takes a lot of time finishing and a lot of computing power. Also, the gain algorithm does not bring the expected result relative to some of the other algorithms, but for now its at least better than guessing, which is most of what the mean algorithm from scikit learn is doing. Since the mean algorithm is so lightweight and quick, I decided to use it as benchmark when sometimes testing algorithms alone. One key factor is that some of the algorithms have parameters that can be tuned in order to hopefully improve results. The GAIN algorithm and the scikit learn MICE algorithm are tuneable. Something also interesting is the big difference between fancyimpute's and scikit learn's MICE algorithm results.

By a little investigation I found out that the only difference between them was a code parameter when instanciating the imputation object. Fancyimputes IterativeImputer was actually moved over to the scikit learn library, and now only acts as a pointer to scikit learns IterativeImputer. This makes keeping both fancyimputer's and

References

- [1] Melissa J. Azur et al. “Multiple imputation by chained equations: what is it and how does it work?” In: *International Journal of Methods in Psychiatric Research* 20.1 (Feb. 2011), pp. 40–49. ISSN: 1049-8931. DOI: 10.1002/mpr.329. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3074241/> (visited on 05/04/2022).
- [2] Pedro J. García-Laencina, José-Luis Sancho-Gómez, and Aníbal R. Figueiras-Vidal. “Pattern classification with missing data: a review”. en. In: *Neural Computing and Applications* 19.2 (). ISSN: 1433-3058. DOI: 10.1007/s00521-009-0295-6. URL: <https://doi.org/10.1007/s00521-009-0295-6> (visited on 02/17/2022).
- [3] *Generative adversarial network*. en. Page Version ID: 1077276508. Mar. 2022. URL: https://en.wikipedia.org/w/index.php?title=Generative_adversarial_network&oldid=1077276508 (visited on 03/16/2022).
- [4] Shangzhi Hong and Henry S. Lynn. “Accuracy of random-forest-based imputation of missing data in the presence of non-normality, non-linearity, and interaction”. In: *BMC Medical Research Methodology* 20.1 (July 2020), p. 199. ISSN: 1471-2288. DOI: 10.1186/s12874-020-01080-1. URL: <https://doi.org/10.1186/s12874-020-01080-1> (visited on 05/04/2022).
- [5] Jonathan Hui. *GAN — What is Generative Adversarial Networks GAN?* en. URL: <https://jonathan-hui.medium.com/gan-whats-generative-adversarial-networks-and-its-application-f39ed278ef09> (visited on 03/17/2022).
- [6] *KNNImputer — Way To Impute Missing Values*. en. July 2020. URL: <https://www.analyticsvidhya.com/blog/2020/07/knnimputer-a-robust-way-to-impute-missing-values-using-scikit-learn/> (visited on 02/16/2022).
- [7] Dhairya Kumar. *Introduction to Data Preprocessing in Machine Learning*. en. June 2021. URL: <https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d> (visited on 05/05/2022).
- [8] *Top Computer Languages 2021 - StatisticsTimes.com*. URL: <https://statisticstimes.com/tech/top-computer-languages.php> (visited on 05/09/2022).
- [9] Wei Wang, Yimeng Chai, and Yue Li. “GAGIN: generative adversarial guider imputation network for missing data”. en. In: *Neural Computing and Applications* (Jan. 2022). ISSN: 1433-3058. DOI: 10.1007/s00521-021-06862-2. URL: <https://doi.org/10.1007/s00521-021-06862-2> (visited on 02/16/2022).

- [10] Zachary Warnes. *Missing Value Handling — Missing Data Types*. en. Dec. 2021. URL: <https://towardsdatascience.com/missing-value-handling-missing-data-types-a89c0d81a5bb> (visited on 05/02/2022).
- [11] Jinsung Yoon. *Codebase for "Generative Adversarial Imputation Networks (GAIN)"*. original-date: 2018-07-09T13:42:44Z. May 2022. URL: <https://github.com/jsyoon0823/GAIN> (visited on 05/09/2022).
- [12] Jinsung Yoon, James Jordon, and Mihaela Schaar. "GAIN: Missing Data Imputation using Generative Adversarial Nets". en. In: *Proceedings of the 35th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2018, pp. 5689–5698. URL: <https://proceedings.mlr.press/v80/yoon18a.html> (visited on 05/04/2022).