Deep Learning Assignment 4 - Report

Students

- **Student name** - Leor Ariel Rose, **ID** - 208373365
- **Student name** - Aviv Rovshitz, **ID** - 307974162

Analysis of the dataset

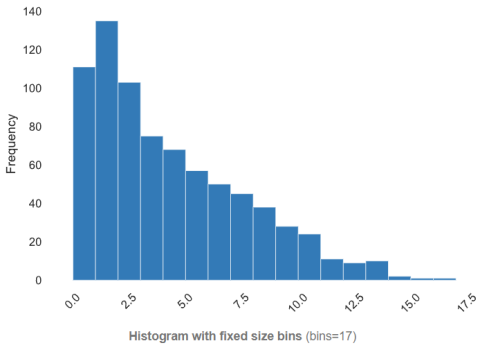
In this section we will present the analysis of the dataset by visual techniques (EDA) and by adding additional written information. All analysis of the dataset was done using pandas_profiling and additional analysis (we did not put everything in the report) can be found in “diabetes_eda.html” & “german_credit_eda.html” html files.

Diabetes dataset

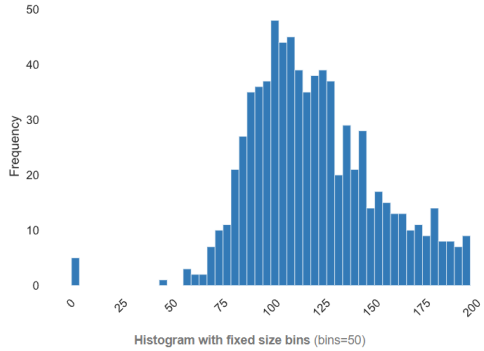
Some general information about the dataset:

Number of variables	Number of observations	Number of Missing cells	Number of Duplicate rows	Number of Numeric columns	Number of Categorical columns
9	768	9	0	8	1

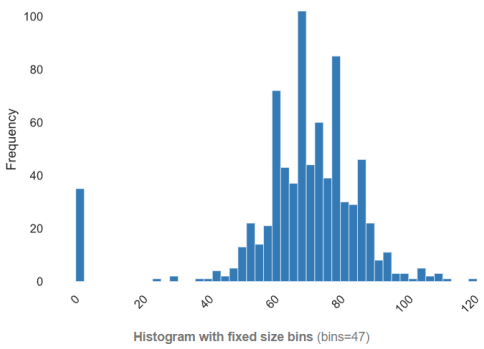
Distribution of each class and feature in the dataset:



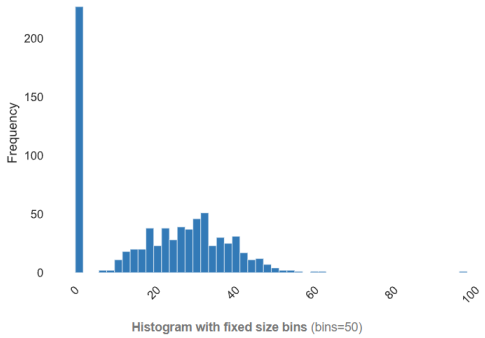
Preg



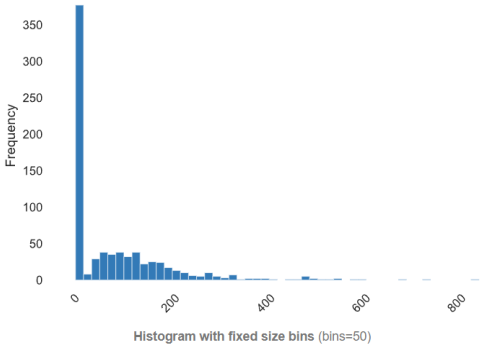
Plas



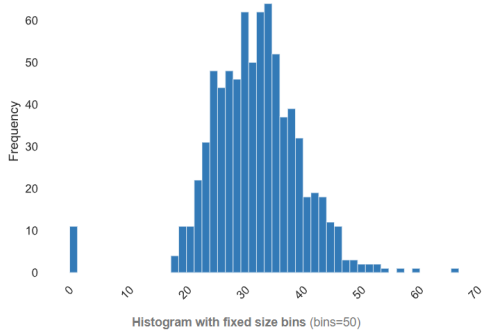
Pres



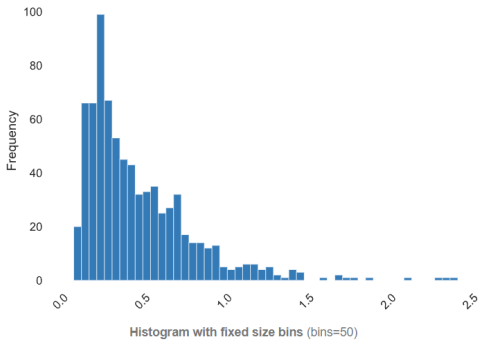
Skin



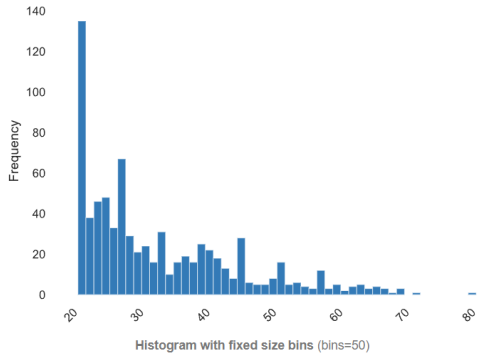
Insu



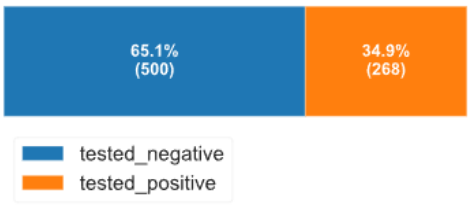
Mass



Pedi



Age



Class

Some records from the dataset:

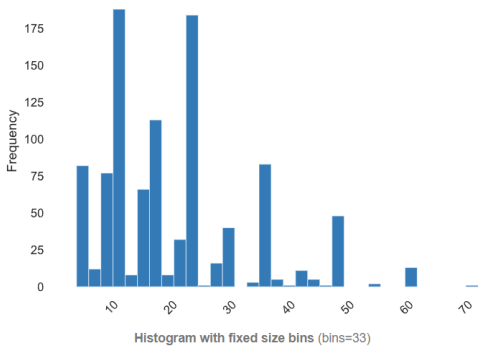
	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6.0	148.0	72.0	35.0	0.0	33.6	0.627	50.0	tested_positive
1	1.0	85.0	66.0	29.0	0.0	26.6	0.351	31.0	tested_negative
2	8.0	183.0	64.0	0.0	0.0	23.3	0.672	32.0	tested_positive
3	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0	tested_negative
4	0.0	137.0	40.0	35.0	168.0	43.1	2.288	33.0	tested_positive
5	5.0	116.0	74.0	0.0	0.0	25.6	0.201	30.0	tested_negative
6	3.0	78.0	50.0	32.0	88.0	31.0	0.248	26.0	tested_positive
7	10.0	115.0	0.0	0.0	0.0	35.3	0.134	29.0	tested_negative
8	2.0	197.0	70.0	45.0	543.0	30.5	0.158	53.0	tested_positive
9	8.0	125.0	96.0	0.0	0.0	0.0	0.232	54.0	tested_positive

German Credit dataset

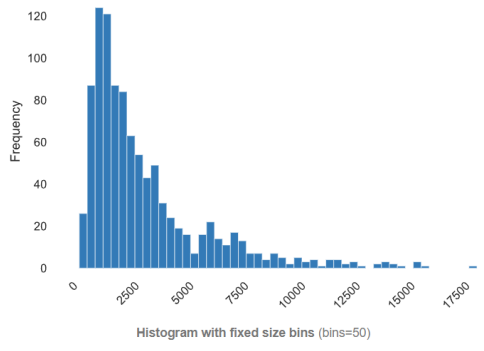
Some general information about the dataset:

Number of variables	Number of observations	Number of Missing cells	Number of Duplicate rows	Number of Numeric columns	Number of Categorical columns
21	1000	0	0	3	18

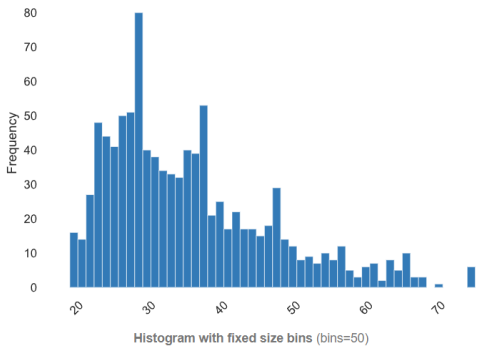
Distribution of each class and feature in the dataset:



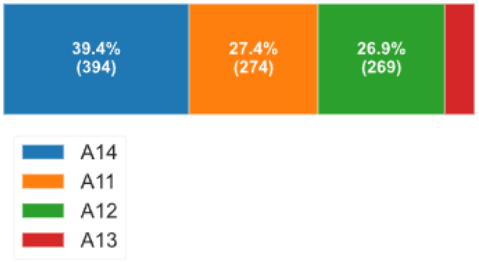
2



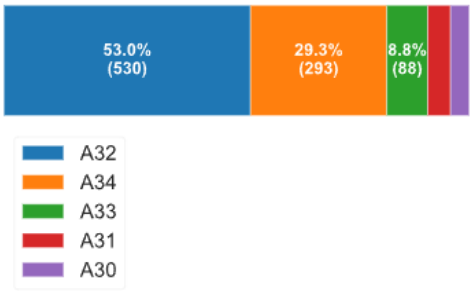
5


Frequency
Histogram with fixed size bins (bins=50)

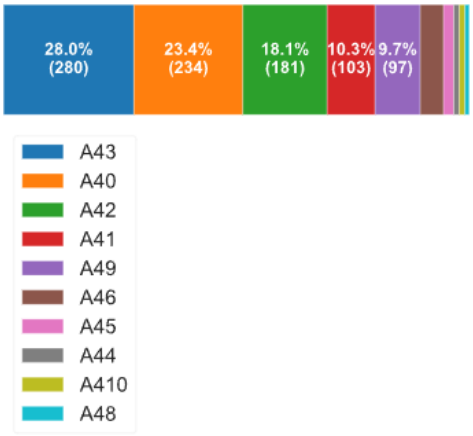
13


39.4%
(394)
27.4%
(274)
26.9%
(269)
A14
A11
A12
A13

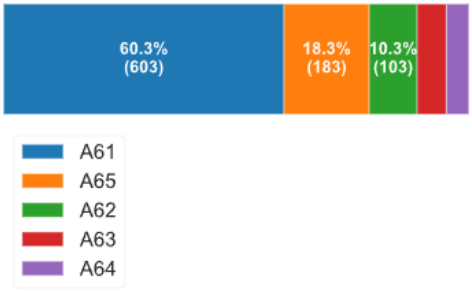
1


53.0%
(530)
29.3%
(293)
8.8%
(88)
A32
A34
A33
A31
A30

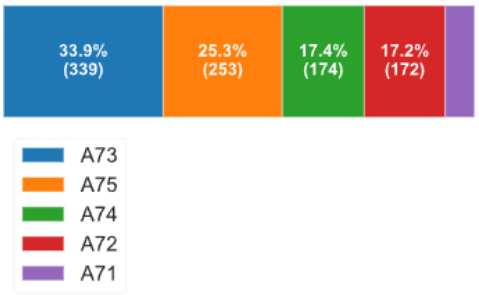
3


28.0%
(280)
23.4%
(234)
18.1%
(181)
10.3%
(103)
9.7%
(97)
A43
A40
A42
A41
A49
A46
A45
A44
A410
A48

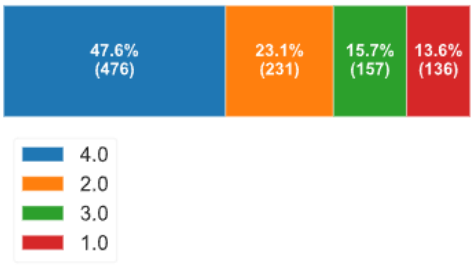
4


60.3%
(603)
18.3%
(183)
10.3%
(103)
A61
A65
A62
A63
A64

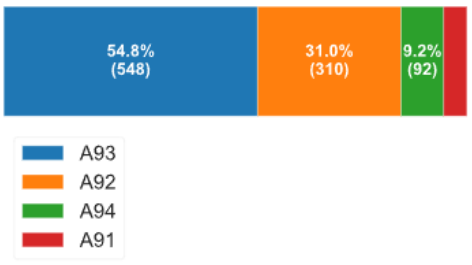
6


33.9%
(339)
25.3%
(253)
17.4%
(174)
17.2%
(172)
A73
A75
A74
A72
A71

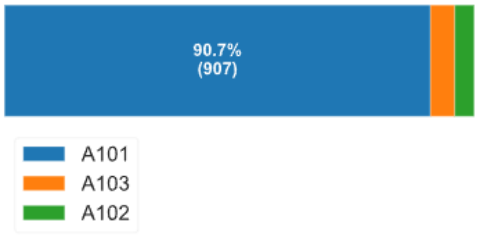
7


47.6%
(476)
23.1%
(231)
15.7%
(157)
13.6%
(136)
4.0
2.0
3.0
1.0

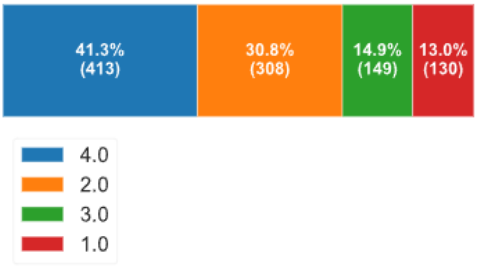
8


54.8%
(548)
31.0%
(310)
9.2%
(92)
A93
A92
A94
A91

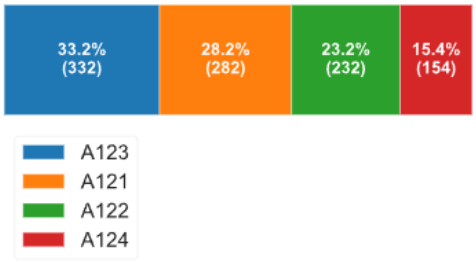
9



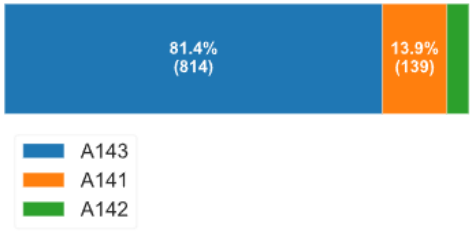
10



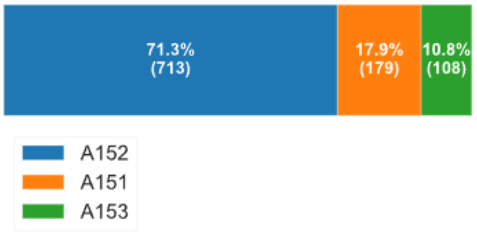
11



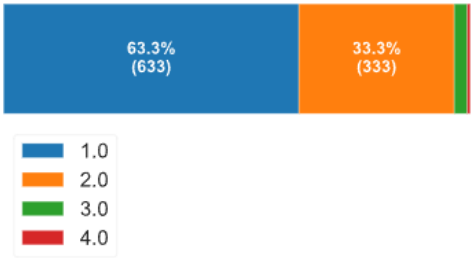
12



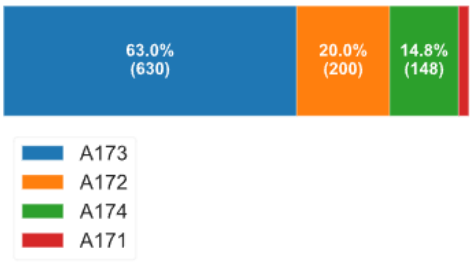
14



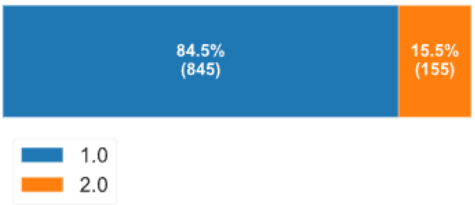
15



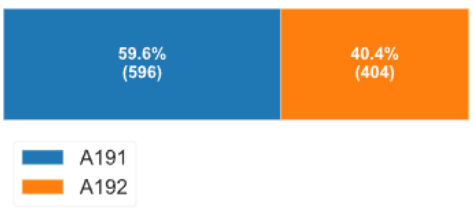
16



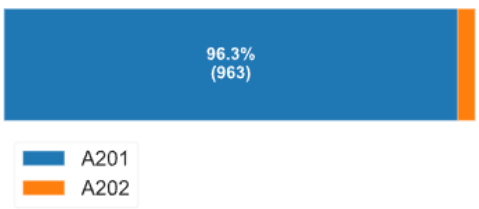
17



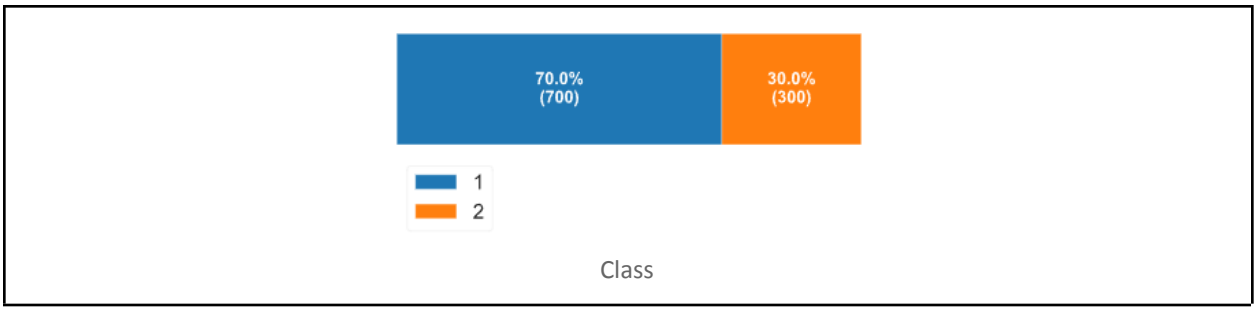
18



19



20



Some records from the dataset:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	class
0	A11	6.0	A34	A43	1169.0	A65	A75	4.0	A93	A101	4.0	A121	67.0	A143	A152	2.0	A173	1.0	A192	A201	1
1	A12	48.0	A32	A43	5951.0	A61	A73	2.0	A92	A101	2.0	A121	22.0	A143	A152	1.0	A173	1.0	A191	A201	2
2	A14	12.0	A34	A46	2096.0	A61	A74	2.0	A93	A101	3.0	A121	49.0	A143	A152	1.0	A172	2.0	A191	A201	1
3	A11	42.0	A32	A42	7882.0	A61	A74	2.0	A93	A103	4.0	A122	45.0	A143	A153	1.0	A173	2.0	A191	A201	1
4	A11	24.0	A33	A40	4870.0	A61	A73	3.0	A93	A101	4.0	A124	53.0	A143	A153	2.0	A173	2.0	A191	A201	2
5	A14	36.0	A32	A46	9055.0	A65	A73	2.0	A93	A101	4.0	A124	35.0	A143	A153	1.0	A172	2.0	A192	A201	1
6	A14	24.0	A32	A42	2835.0	A63	A75	3.0	A93	A101	4.0	A122	53.0	A143	A152	1.0	A173	1.0	A191	A201	1
7	A12	36.0	A32	A41	6948.0	A61	A73	2.0	A93	A101	2.0	A123	35.0	A143	A151	1.0	A174	1.0	A192	A201	1
8	A14	12.0	A32	A43	3059.0	A64	A74	2.0	A91	A101	4.0	A121	61.0	A143	A152	1.0	A172	1.0	A191	A201	1
9	A12	30.0	A34	A40	5234.0	A61	A71	4.0	A94	A101	2.0	A123	28.0	A143	A152	2.0	A174	1.0	A191	A201	2

Dataset Preprocessing

Preprocessing was performed on both datasets (Diabetes dataset, German Credit dataset) in the same manner.

- Byte values were decoded to strings with the built-in Pandas str.decode("utf-8") method.
- Numeric values were transformed to the range of [-1, 1] using scikit-learn MinMaxScaler.
- Categorical columns not including the class column were transformed using the scikit-learn OneHotEncoder.
- Class column was transformed using the scikit-learn LabelEncoder.

All transformers were saved so later on we will be able to inverse the generated samples and compare them against the original dataset.

Model Architecture

In both parts (regular GAN & GAN with a twist) we implemented a conditional GAN in order to control the type of generated sample and because usually they are more stable and achieve better results.
Using some recommended tips (Tips for Training Stable Generative Adversarial Networks, How to Train a GAN? Tips and tricks to make GANs work) we applied the following in our architecture in this section to improve the CGAN:

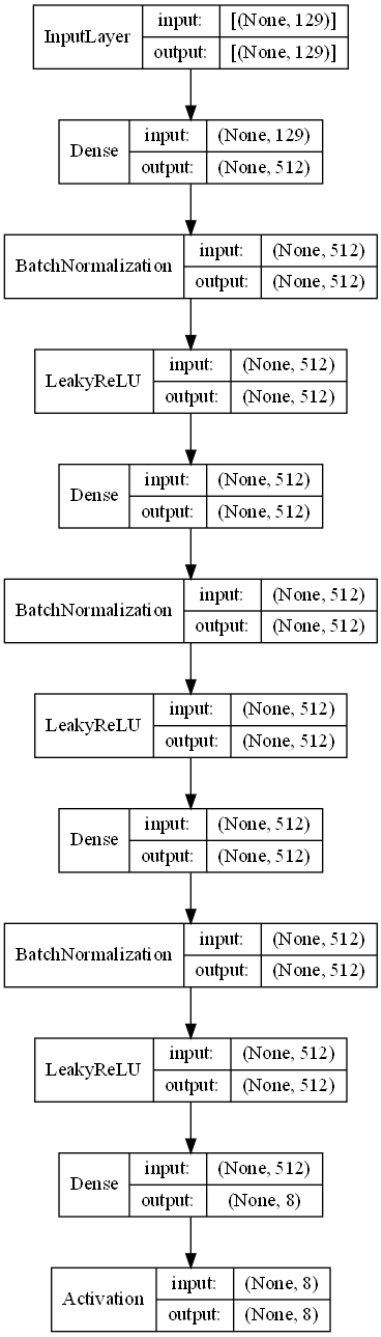
- **Optimizer** - we used Adam optimization with a learning rate of 0.002 and momentum of 0.5.
- **Weight initialization** - we initialized all weights using a zero-centered gaussian distribution with a standard deviation of 0.02.
- **Activation functions** - we used Leaky ReLU in the generator except in the output layer where we used tanh and Leaky ReLU in the discriminator except in the output layer where we used sigmoid.
- **Batch normalization** - we used batch normalization in both the discriminator and generator models, except the output of the generator and input to the discriminator.
- **Dropout** - we used a dropout of 50 percent for discriminator.
- **Latent space sampling** - we sampled noise from a gaussian distribution as input to the generator.
- **BatchNorm** - we used mini batches of all real or all fake.
- **Label smoothing** - we applied label smoothing in the discriminator, with small random noise.
- **Noise** - we added gaussian noise to the labels and inputs in the discriminator.

The only change between both parts is what we add to the generator and discriminator input to make a conditional GAN (In regular GAN we add the class label and in GAN with a twist we add the real/fake confidence).

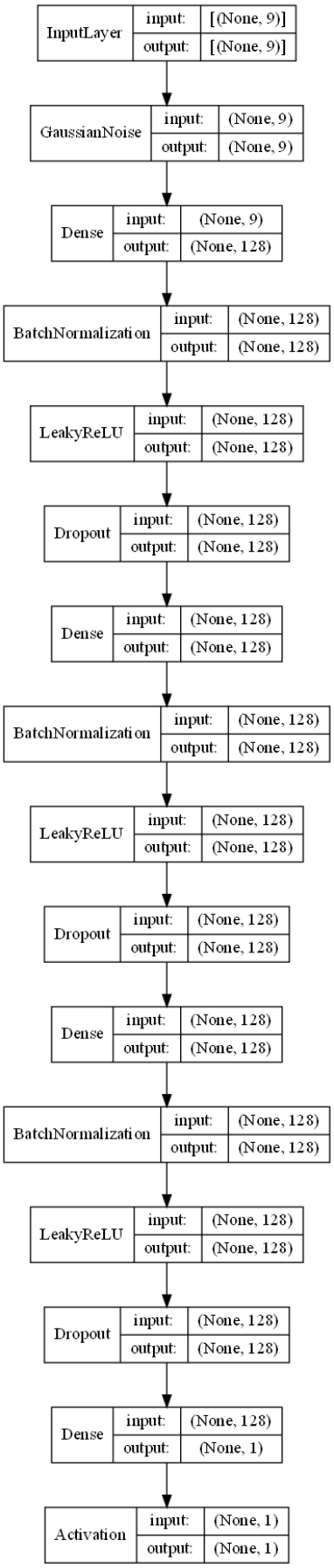
Part 1 – Regular GAN

Diabetes dataset

The architecture of the generator is the following:

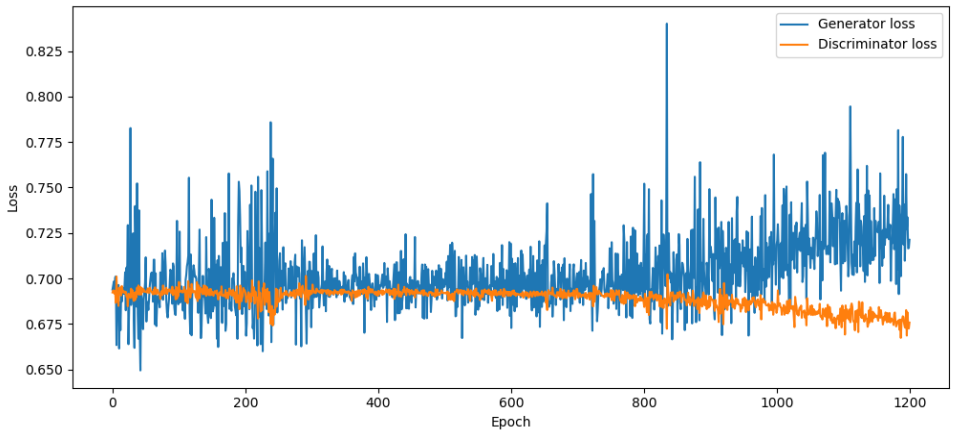


The architecture of the discriminator is the following:

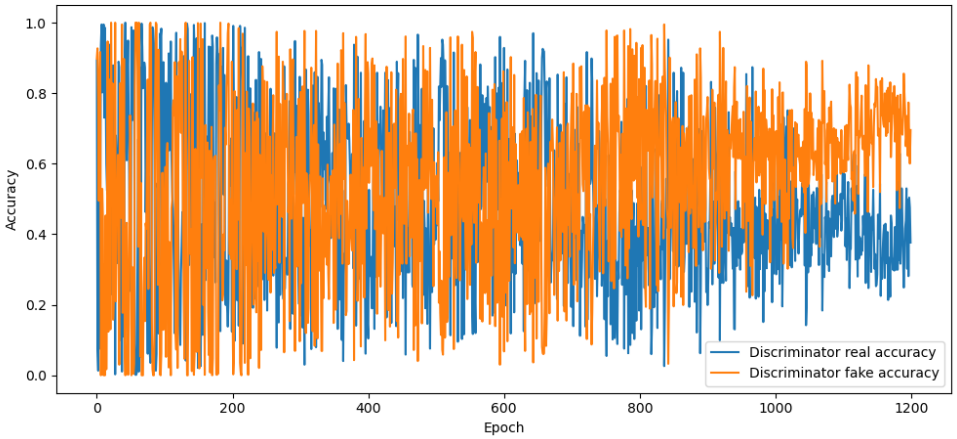

InputLayer
input: [(None, 9)]
output: [(None, 9)]
GaussianNoise
input: (None, 9)
output: (None, 9)
Dense
input: (None, 9)
output: (None, 128)
BatchNormalization
input: (None, 128)
output: (None, 128)
LeakyReLU
input: (None, 128)
output: (None, 128)
Dropout
input: (None, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 128)
BatchNormalization
input: (None, 128)
output: (None, 128)
LeakyReLU
input: (None, 128)
output: (None, 128)
Dropout
input: (None, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 128)
BatchNormalization
input: (None, 128)
output: (None, 128)
LeakyReLU
input: (None, 128)
output: (None, 128)
Dropout
input: (None, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 1)
Activation
input: (None, 1)
output: (None, 1)

Generator and the discriminator training metrics:

The loss and accuracy of the generator and the discriminator can tell us a lot about the training process. First let's look at the loss:



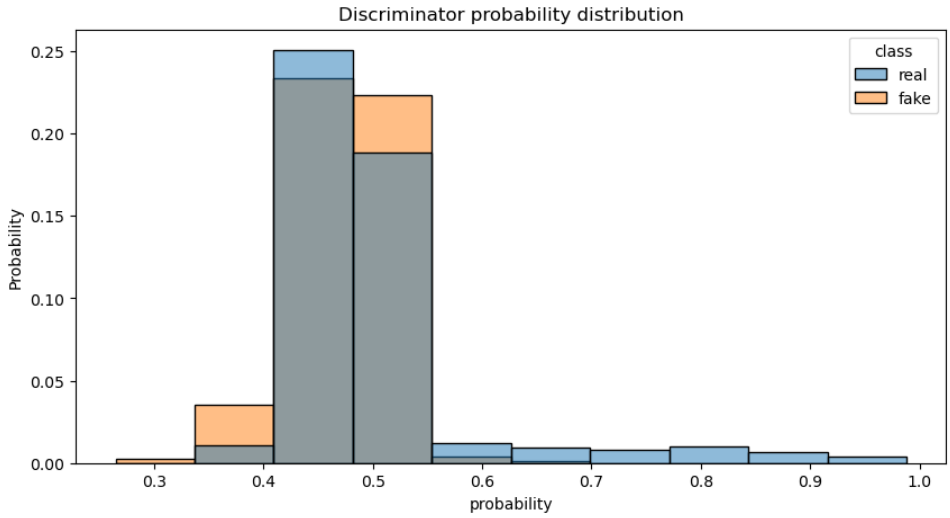
As we can see the generator has a lot of variance (going back and forth) compared to the discriminator loss. This makes us believe that the discriminator is the consistent leader. We can also see that at around epoch 1000 the discriminator starts to get really good causing an increase in the generator loss. So data that does manage to fool the discriminator should be high quality. Secondly, let's take a look at the accuracy:



As we can see the real and fake accuracy have a lot of variance (going back and forth). The discriminator is a lot more sure about the fake data then the real data but still isn't 100 percent sure what is fake. In our opinion this means the generator was able to confuse the discriminator.

Discriminator probability distribution:

Another way to evaluate the GAN performance is by looking at the probabilities the discriminator gives to fake generated samples. We generated 768 samples (the same number of the original dataset samples) and this is the probability distribution:



As we can see, most of the samples have a probability to be real in between [0.4, 0.5]. In the higher probabilities [0.6, 1] we have more real samples and In the lower probabilities [0, 0.3] we have more fake samples. Overall, these probabilities to be real have a good distribution for real and fake samples.

Samples that did fool the discriminator:

Our GAN managed to create 42 samples out of 768 that we requested that did fool the discriminator. Let's take a look at a few of these samples:

	preg	plas	pres	skin	insu	mass	pedi	age	class
28	0.467070	62.516998	69.791580	25.083942	3.188053	32.834141	0.863031	27.304157	tested_negative
31	0.289234	80.954025	80.417511	24.245792	11.989375	35.539864	0.992346	30.573620	tested_positive
34	0.840393	67.750168	58.226730	18.566761	8.766100	26.092045	0.584342	24.355286	tested_negative
69	2.108303	74.819183	0.974388	0.044224	0.013766	16.719324	0.119461	21.154465	tested_negative
80	5.418090	103.439896	78.136711	0.000000	0.000000	22.659266	0.514277	69.465408	tested_negative
99	3.694934	152.995834	0.000167	0.000531	0.000000	30.123156	0.235413	26.932753	tested_positive
111	2.055166	130.138260	85.832443	0.000000	0.000000	25.763359	0.581437	58.209488	tested_positive
119	0.056038	53.324032	62.070263	21.344994	25.809223	31.995808	0.840214	22.393963	tested_negative
126	0.202205	58.161591	68.528221	27.098307	19.958298	36.878944	0.638671	23.556580	tested_negative
134	0.316175	60.767830	73.049904	31.156050	14.658954	39.951027	0.695744	24.776836	tested_negative

Samples that didn't fool the discriminator:

Our GAN managed to create 726 samples out of 768 that we requested that didn't fool the discriminator. Let’s take a look at a few of these samples:

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	2.749295	106.208733	62.475708	17.651957	39.325359	29.934866	0.218589	23.806908	tested_positive
1	2.306576	102.701515	64.339111	26.941715	120.877106	35.170876	0.433919	26.697441	tested_negative
2	1.039154	100.964317	64.320000	20.222012	95.323662	30.784491	0.535398	28.938505	tested_positive
3	1.459134	74.451736	55.623055	28.026606	40.246330	28.879293	0.239118	23.732988	tested_negative
4	0.926391	79.767273	76.826950	22.369547	16.828386	33.214603	0.895167	33.560009	tested_positive
5	0.460393	65.328377	65.436584	28.949244	17.803438	33.504997	0.528179	22.447119	tested_negative
6	13.311133	117.544350	83.728958	23.357840	27.761549	29.419146	0.583043	54.204292	tested_positive
7	10.246860	79.144524	73.981918	31.314011	7.587933	35.933495	0.381676	38.248909	tested_negative
8	2.557977	178.888962	68.249870	21.933054	157.740692	33.126682	0.578837	31.933979	tested_positive
9	3.205697	153.133118	58.505005	27.169399	111.620110	29.939083	0.350592	23.764996	tested_positive

Fake and real data similarity evaluation using table_evaluter:

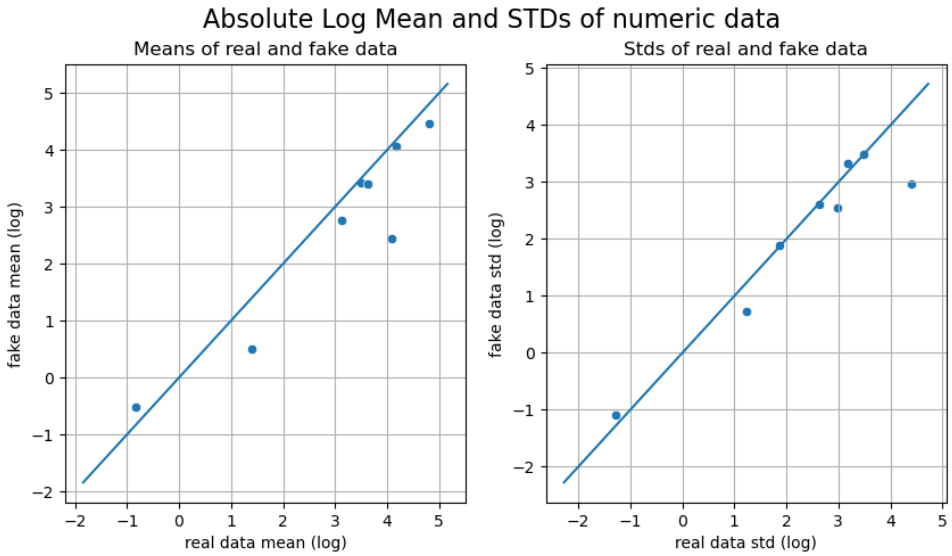
For the samples that did fool the discriminator, we evaluated the similarity to the real data using TableEvaluator - a library to evaluate how similar a synthesized dataset is to real data. In other words, it tries to give an indication into how real our fake data is.

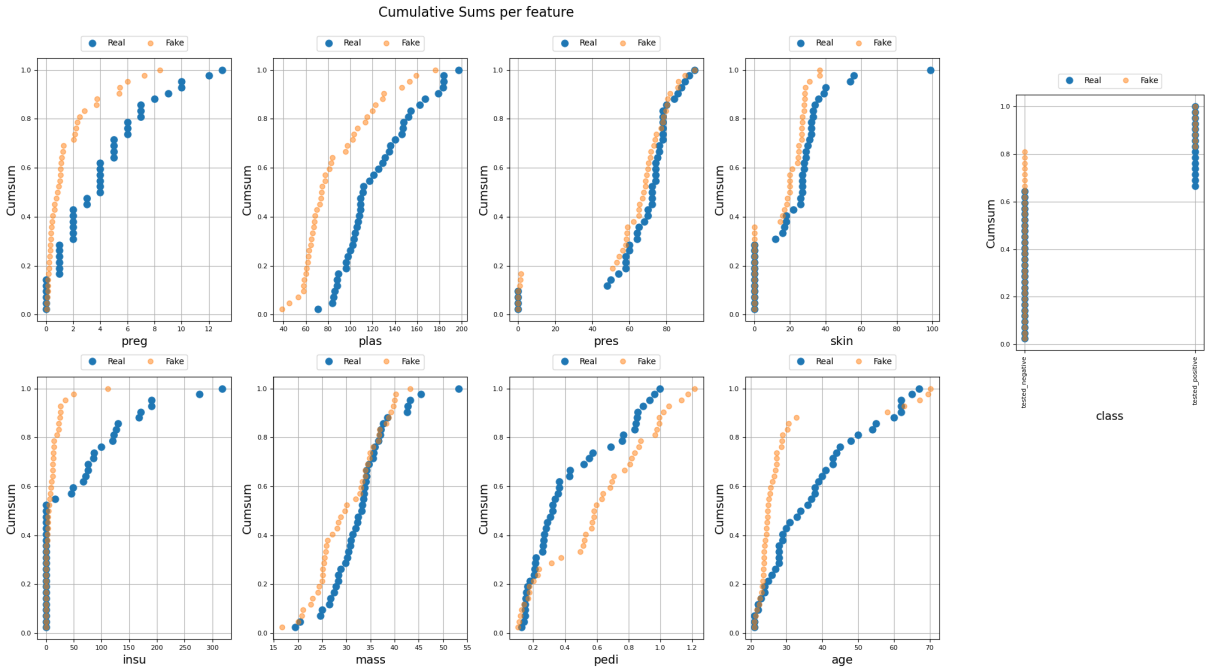
First evaluation is a visualized statistics in order to see if the GAN is able to capture properties. we take a look at:

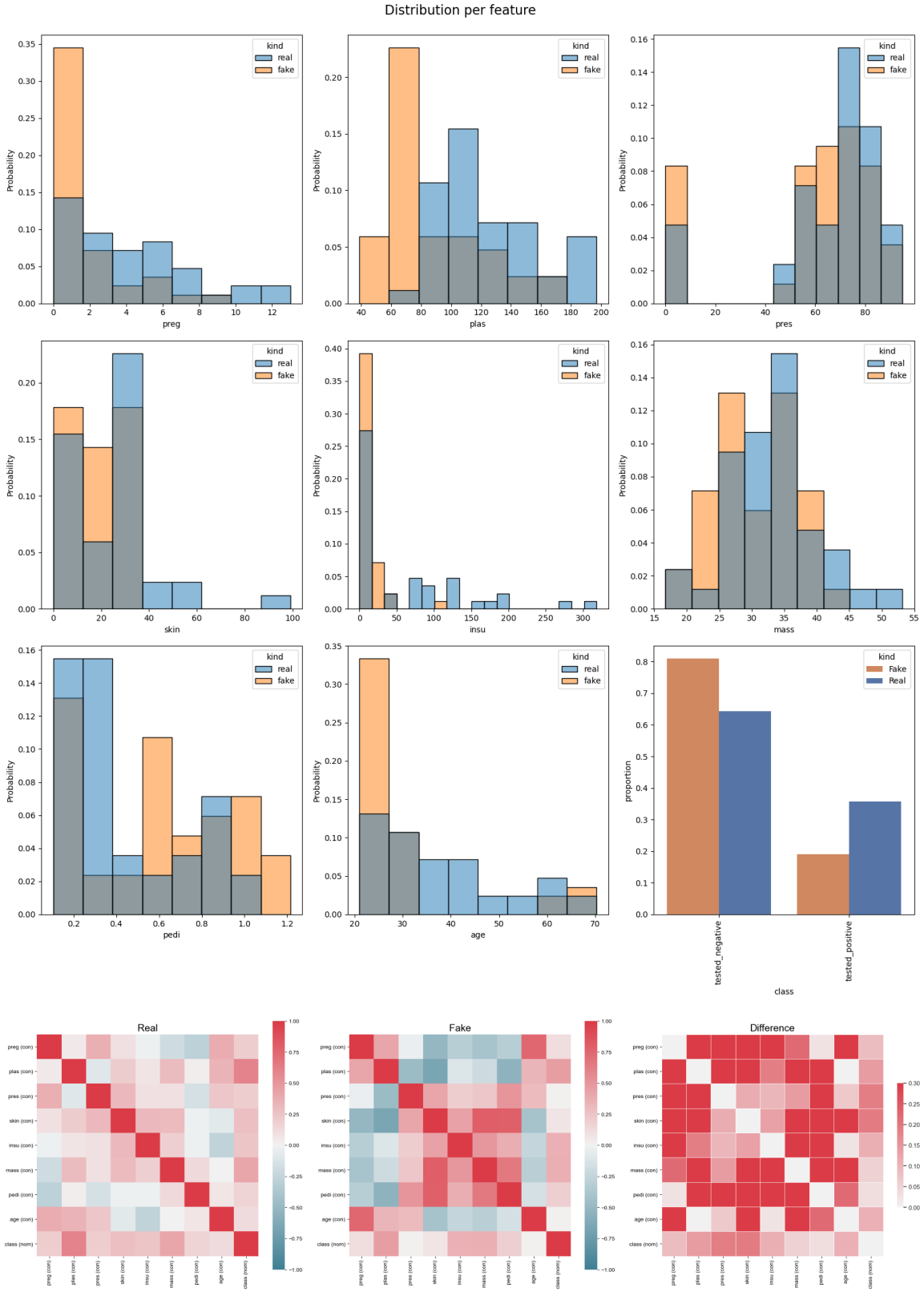
1. Columnwise mean and standard deviation
2. Cumulative Sums
3. Column Correlations

Second evaluation of the performance using TableEvaluator is machine learning efficiency. This evaluates the performance of using synthetic data as training data for machine learning. In other words, we train the model on real\fake data and test it on fake\real data respectively to see if the results are similar.

Visualized statistics:






Distribution per feature
kind
real
fake
Probability
preg
plas
pres
skin
insu
mass
pedi
age
Fake
Real
proportion
tested_negative
tested_positive
class
Real
Fake
Difference

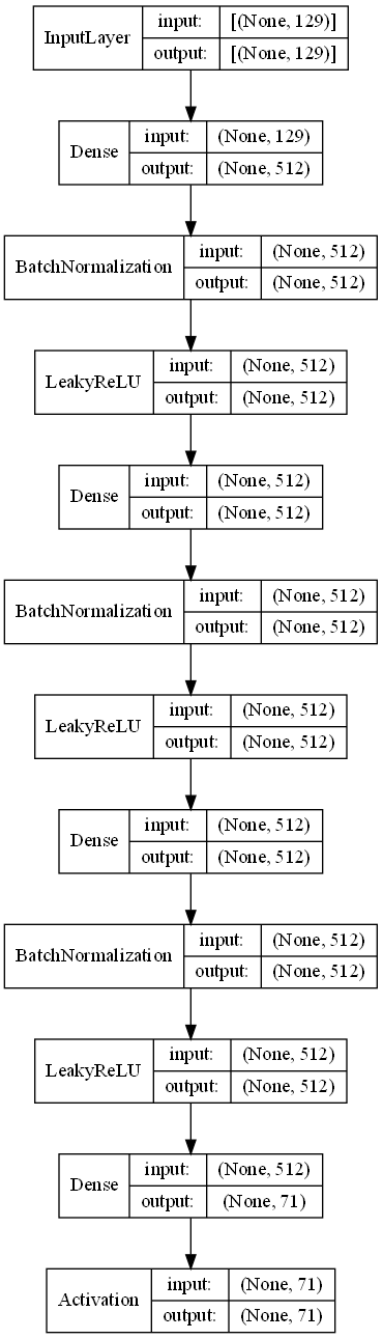
Machine learning efficiency:

```
Classifier F1-scores and their Jaccard similarities::
                           f1_real  f1_fake  jaccard_similarity
index
DecisionTreeClassifier_fake   0.6667   0.6667              0.6364
DecisionTreeClassifier_real   0.7778   0.5556              0.3846
LogisticRegression_fake       0.6667   0.8889              0.6364
LogisticRegression_real       0.6667   0.5556              0.5000
MLPClassifier_fake            0.5556   0.3333              0.6364
MLPClassifier_real            0.5556   0.5556              0.3846
RandomForestClassifier_fake   0.4444   0.4444              0.3846
RandomForestClassifier_real   0.5556   1.0000              0.3846

Privacy results:
                                        result
Duplicate rows between sets (real/fake)  (0, 0)
nearest neighbor mean                    2.0970
nearest neighbor std                     0.6203

Miscellaneous results:
                                Result
Column Correlation Distance RMSE  0.3418
Column Correlation distance MAE   0.2547

Results:
                                                result
Basic statistics                                0.9274
Correlation column correlations                 0.4516
Mean Correlation between fake and real columns  0.8553
1 - MAPE Estimator results                      0.7518
Similarity Score                                0.7465
```

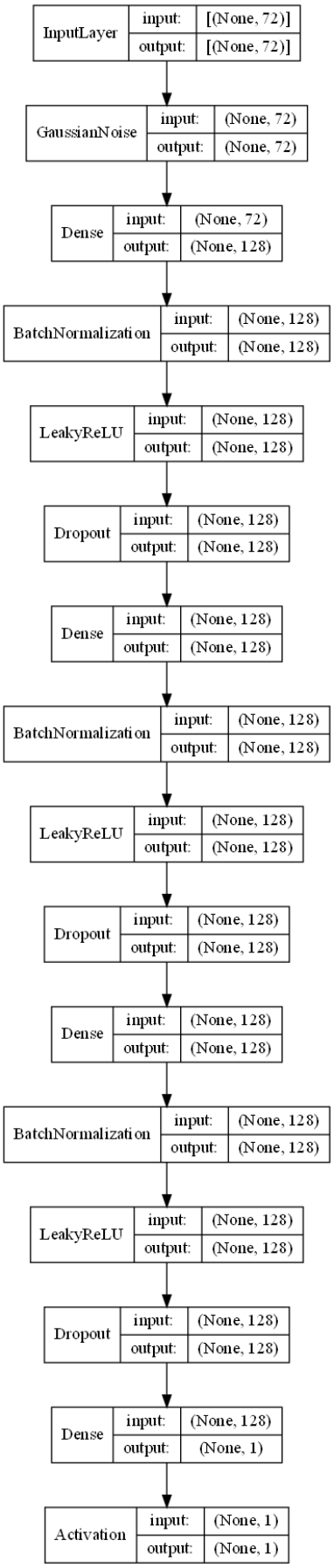
We can see that our data isn't perfect but it was able to learn and be similar by statistical measures. And by machine learning efficiency we see that training our model using fake data gives more or less the same results as training with real data.

German Credit dataset

The architecture of the generator is the following:

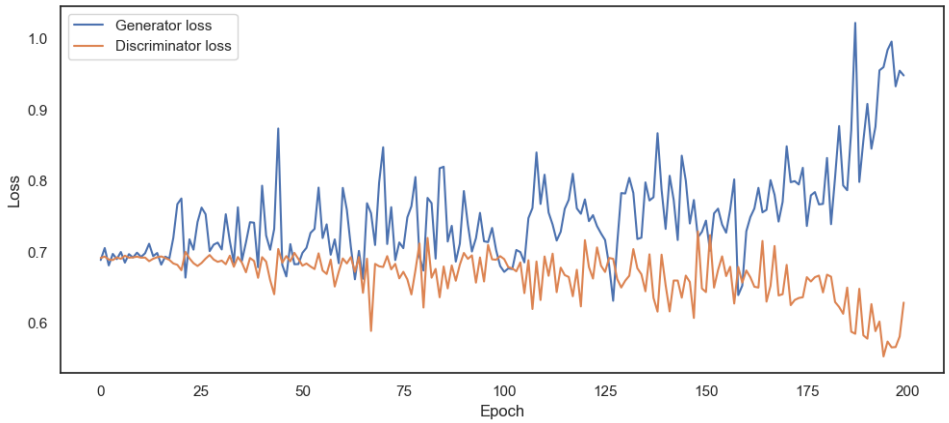


The architecture of the discriminator is the following:

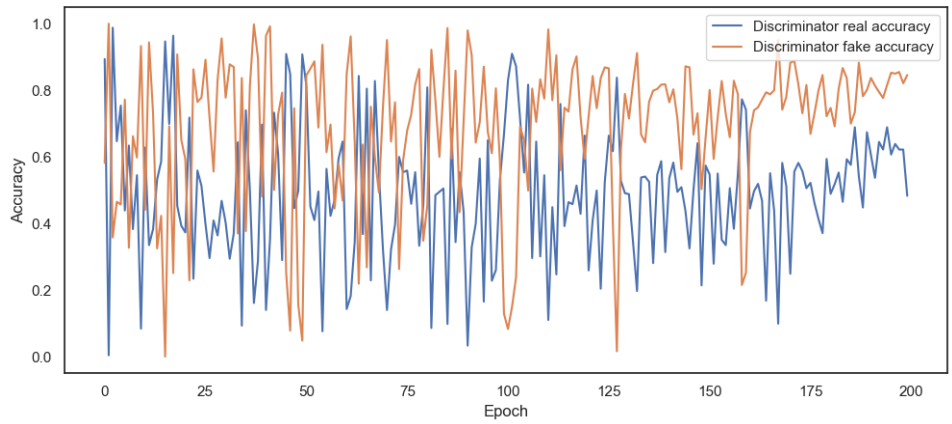

InputLayer
input: [(None, 72)]
output: [(None, 72)]
GaussianNoise
input: (None, 72)
output: (None, 72)
Dense
input: (None, 72)
output: (None, 128)
BatchNormalization
input: (None, 128)
output: (None, 128)
LeakyReLU
input: (None, 128)
output: (None, 128)
Dropout
input: (None, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 128)
BatchNormalization
input: (None, 128)
output: (None, 128)
LeakyReLU
input: (None, 128)
output: (None, 128)
Dropout
input: (None, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 128)
BatchNormalization
input: (None, 128)
output: (None, 128)
LeakyReLU
input: (None, 128)
output: (None, 128)
Dropout
input: (None, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 1)
Activation
input: (None, 1)
output: (None, 1)

Generator and the discriminator training metrics:

The loss and accuracy of the generator and the discriminator can tell us a lot about the training process. First let's look at the loss:



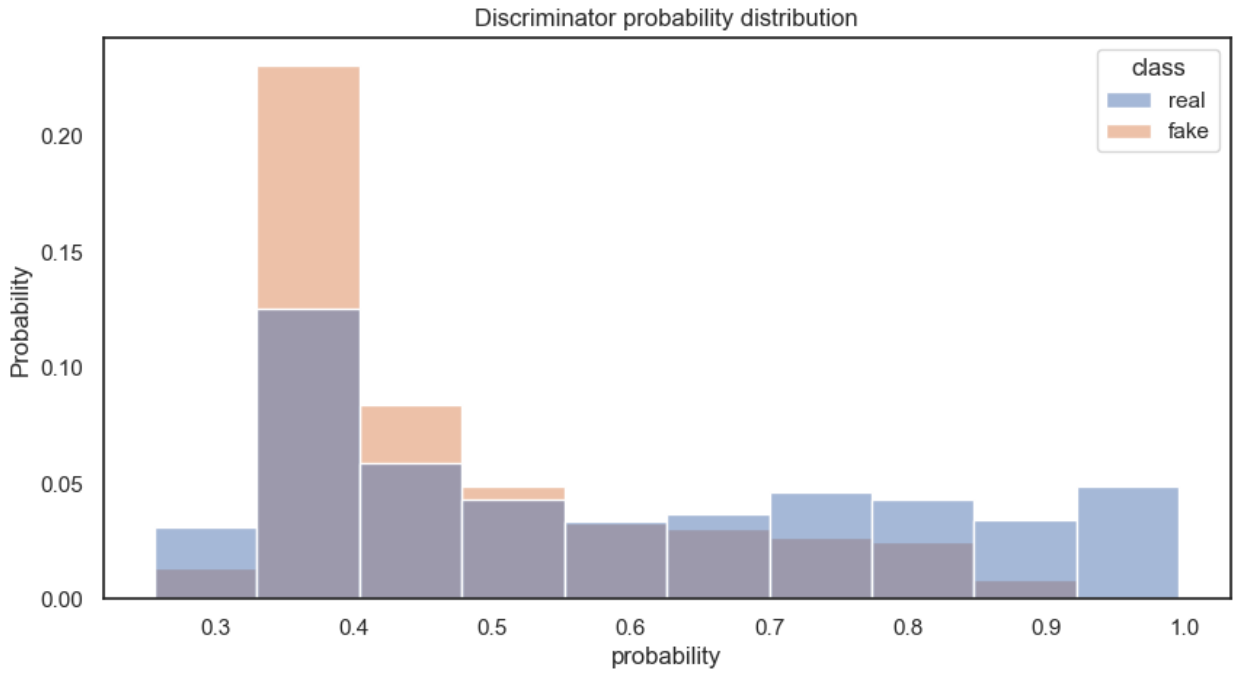
As we can see the generator has a lot of variance (going back and forth) compared to the discriminator loss. This makes us believe that the discriminator is the consistent leader. We can also see that at around epoch 175 the discriminator starts to get really good, causing an increase in generator loss. So data that does manage to fool the discriminator should be high quality. Secondly, let's take a look at the accuracy:



As we can see the real and fake accuracy have a lot of variance (going back and forth). The discriminator is a lot more sure about the fake data then the real data but still isn't 100 percent sure what is fake. In our opinion this means the generator was able to confuse the discriminator.

Discriminator probability distribution:

Another way to evaluate the GAN performance is by looking at the probabilities the discriminator gives to fake generated samples. We generated 1000 samples (the same number of the original dataset samples) and this is the probability distribution:



As we can see, in the higher probabilities [0.9, 1] we have more real samples and In the lower probabilities [0, 0.5] we have more fake samples. Overall, these probabilities to be real have a good distribution for real and fake samples.

<u>Samples that did fool the discriminator:</u>

Our GAN managed to create 317 samples out of 1000 that we requested that did fool the discriminator. Let's take a look at a few of these samples:

	1	2	3	4	5	6	7	8	9	10	...	12	13	14	15	16	17	18	19	20	class
0	A12	12.3181	A32	A49	2,138.7583	A61	A72	2.0000	A93	A101	...	A121	32.7501	A143	A152	1.0000	A173	1.0000	A192	A201	1
1	A12	23.9420	A32	A49	2,974.2639	A61	A72	3.0000	A94	A101	...	A124	36.1297	A143	A152	1.0000	A173	1.0000	A192	A201	2
3	A14	15.6978	A34	A42	1,860.0465	A61	A75	4.0000	A93	A101	...	A123	30.7811	A143	A152	2.0000	A173	2.0000	A191	A201	1
12	A14	23.2432	A32	A49	3,756.1301	A62	A73	4.0000	A93	A101	...	A121	33.1127	A143	A152	1.0000	A173	1.0000	A192	A201	1
13	A12	20.4912	A32	A49	1,436.8519	A61	A72	2.0000	A93	A101	...	A124	27.8001	A143	A152	1.0000	A173	1.0000	A192	A201	2
20	A14	11.3845	A34	A42	791.4850	A61	A73	4.0000	A93	A101	...	A123	24.3747	A143	A152	2.0000	A173	1.0000	A191	A201	1
23	A14	11.5547	A34	A42	742.6724	A61	A75	4.0000	A93	A101	...	A121	30.9703	A143	A152	2.0000	A173	1.0000	A191	A201	1
25	A12	11.4434	A32	A43	4,454.6025	A65	A75	2.0000	A91	A101	...	A124	49.8635	A143	A153	1.0000	A174	1.0000	A192	A201	1
26	A12	18.9103	A32	A41	2,808.5757	A65	A75	4.0000	A93	A101	...	A124	36.5761	A143	A151	1.0000	A173	1.0000	A192	A201	1
36	A14	19.2543	A32	A43	3,000.6758	A65	A75	4.0000	A93	A101	...	A124	32.6560	A143	A152	1.0000	A173	1.0000	A192	A201	1

Samples that didn't fool the discriminator:

Our GAN managed to create 683 samples out of 1000 that we requested that didn't fool the discriminator. Let's take a look at a few of these samples:

	1	2	3	4	5	6	7	8	9	10	...	12	13	14	15	16	17	18	19	20	class
2	A12	17.8272	A34	A43	2,957.6274	A61	A75	2.0000	A93	A101	...	A121	41.3774	A143	A152	1.0000	A173	1.0000	A191	A201	1
4	A14	7.4975	A34	A42	1,014.8570	A61	A71	4.0000	A93	A101	...	A123	26.2912	A143	A152	2.0000	A173	1.0000	A191	A201	2
5	A14	22.2235	A32	A49	2,019.1740	A61	A73	4.0000	A93	A101	...	A123	26.8397	A143	A152	1.0000	A173	1.0000	A191	A201	1
6	A12	20.7837	A32	A49	1,520.2126	A61	A73	3.0000	A94	A101	...	A124	29.6804	A143	A152	1.0000	A173	1.0000	A192	A201	1
7	A14	11.0457	A34	A40	3,062.4209	A61	A75	2.0000	A93	A101	...	A121	45.0632	A143	A151	1.0000	A173	1.0000	A191	A201	1
8	A14	23.9226	A32	A43	4,085.5029	A65	A75	4.0000	A93	A101	...	A123	34.5515	A143	A152	2.0000	A173	1.0000	A192	A201	1
9	A12	31.0905	A32	A49	2,060.2715	A61	A73	4.0000	A93	A101	...	A123	25.3709	A143	A152	1.0000	A173	1.0000	A191	A201	2
10	A11	22.0236	A32	A42	4,502.1270	A61	A72	4.0000	A93	A101	...	A124	35.6831	A142	A153	1.0000	A173	2.0000	A192	A201	2
11	A11	24.4131	A32	A43	4,990.0542	A61	A72	4.0000	A93	A101	...	A124	39.0658	A143	A153	1.0000	A173	2.0000	A191	A201	2
14	A12	10.4194	A32	A40	1,479.2245	A61	A72	2.0000	A92	A101	...	A121	34.4801	A143	A151	1.0000	A173	1.0000	A191	A201	1

Fake and real data similarity evaluation using table_evaluter:

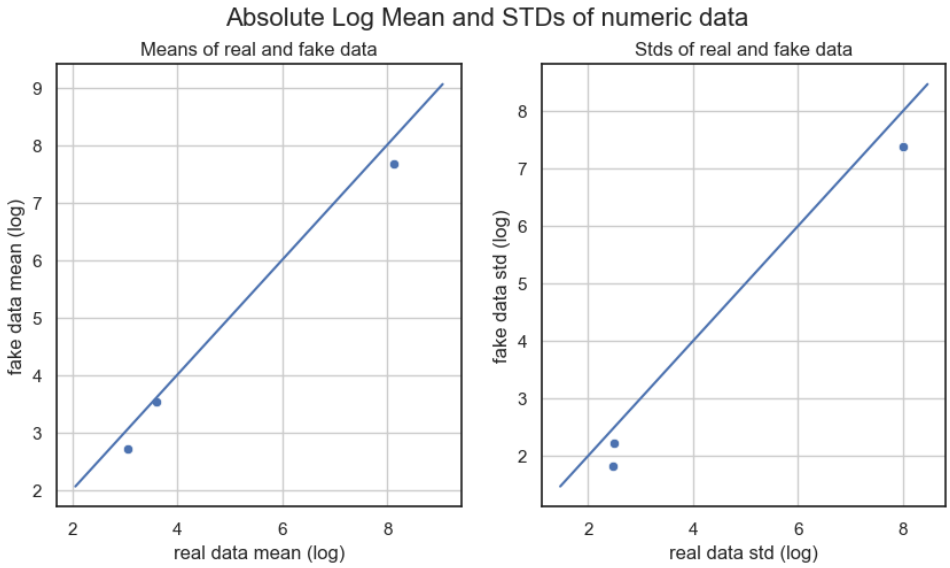
For the samples that did fool the discriminator, we evaluated the similarity to the real data using TableEvaluator - a library to evaluate how similar a synthesized dataset is to real data. In other words, it tries to give an indication into how real our fake data is.

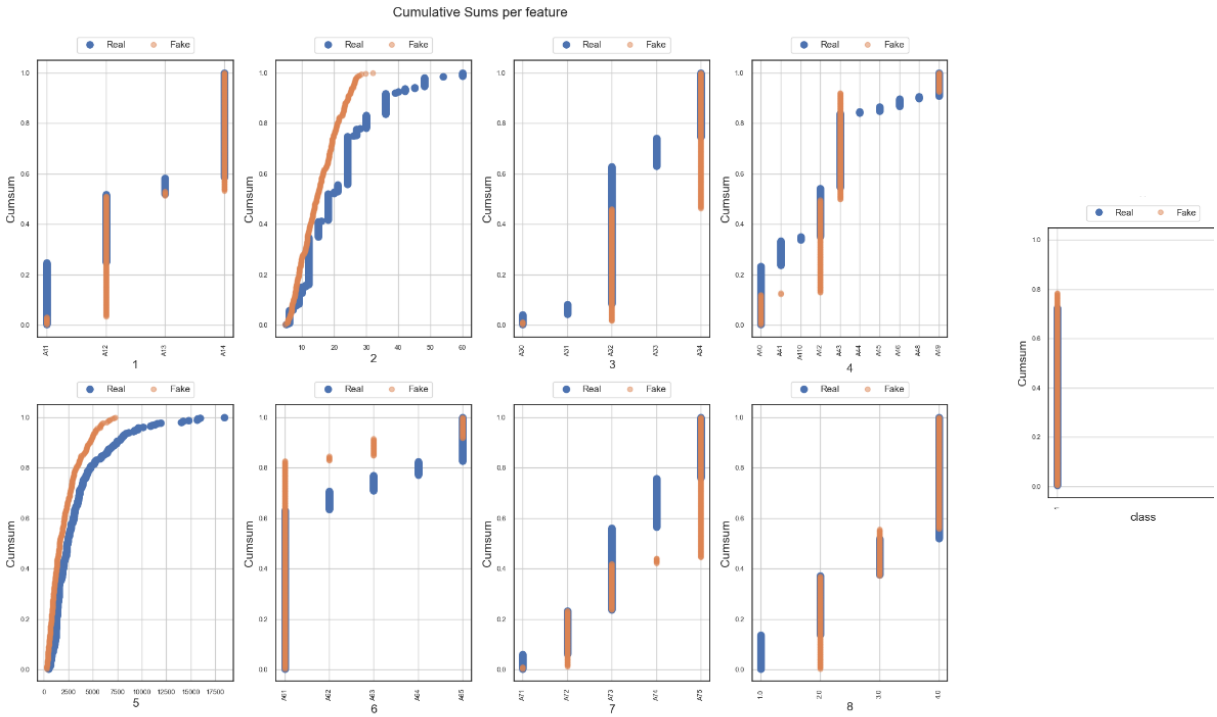
First evaluation is a visualized statistics in order to see if the GAN is able to capture properties. we take a look at:

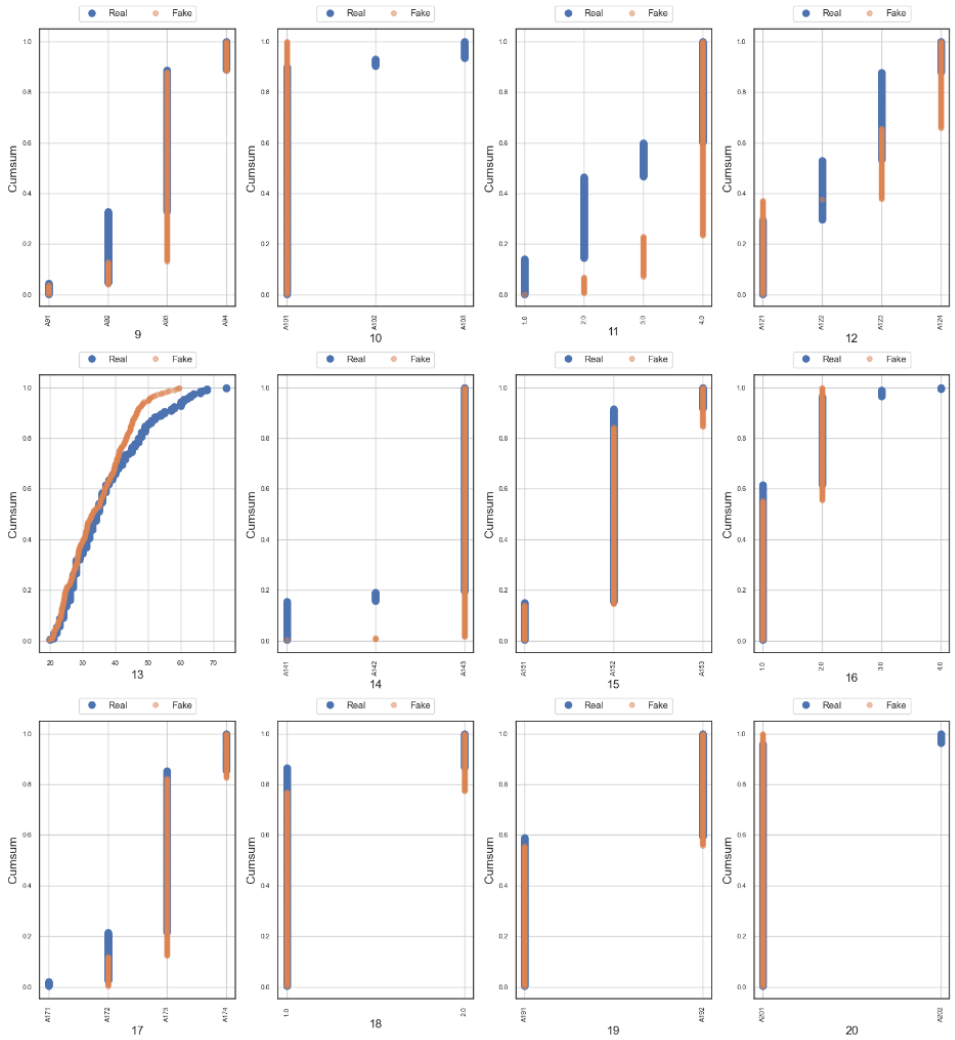
4. Columnwise mean and standard deviation
5. Cumulative Sums
6. Column Correlations

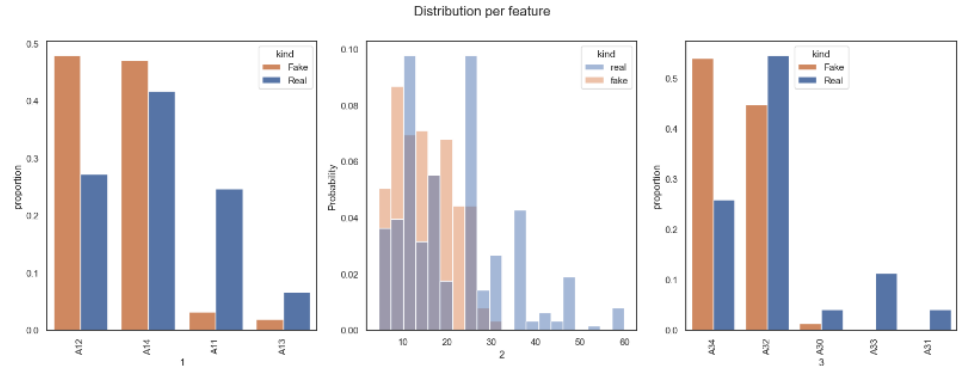
Second evaluation of the performance using TableEvaluator is machine learning efficiency. This evaluates the performance of using synthetic data as training data for machine learning. In other words, we train the model on real\fake data and test it on fake\real data respectively to see if the results are similar.

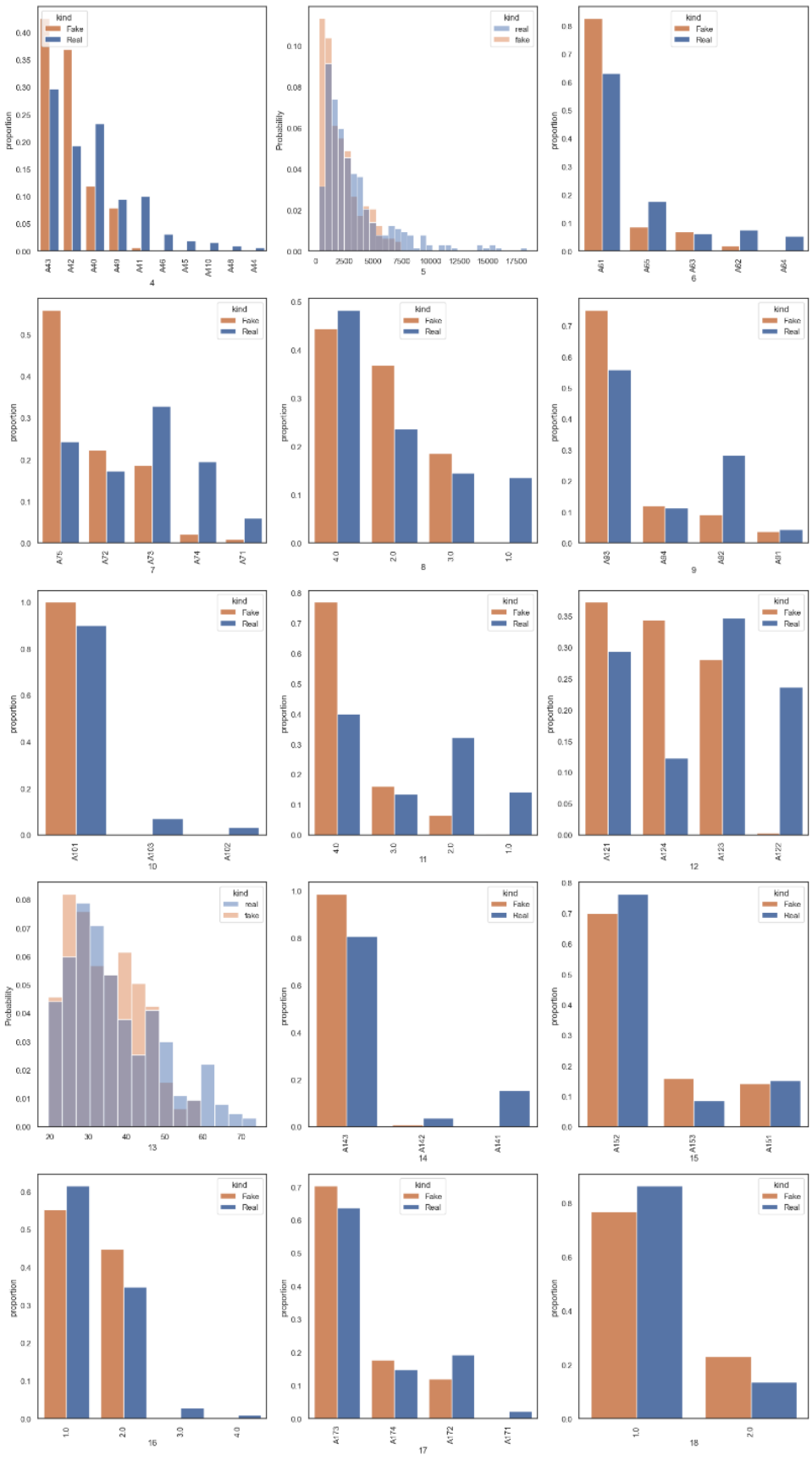
Visualized statistics:

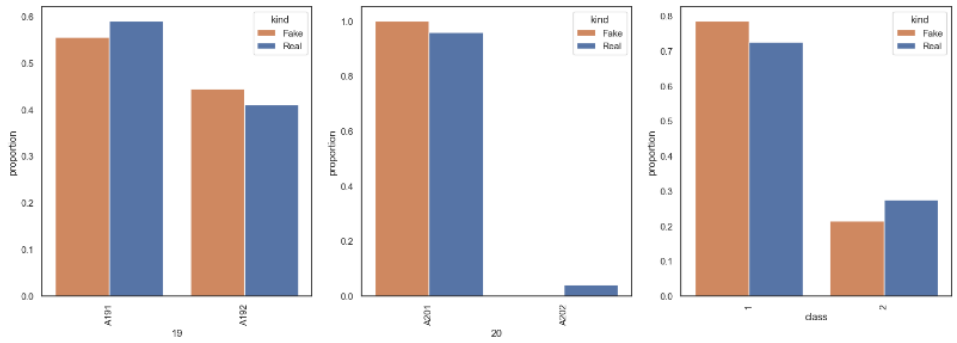


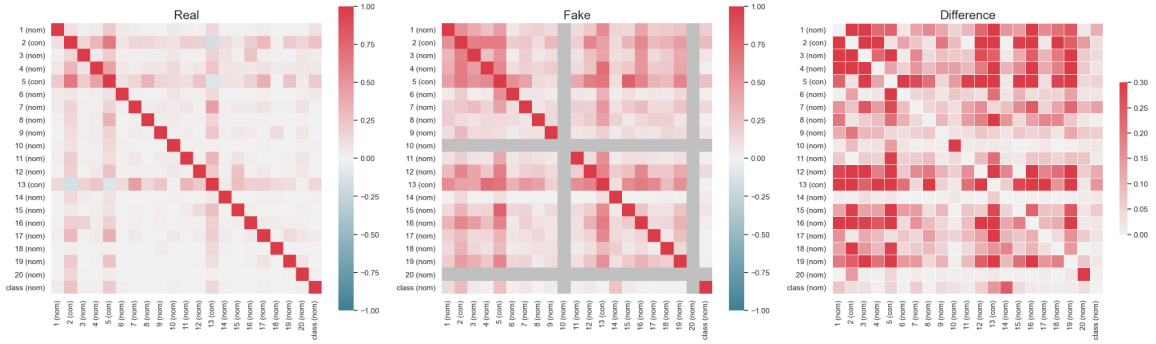


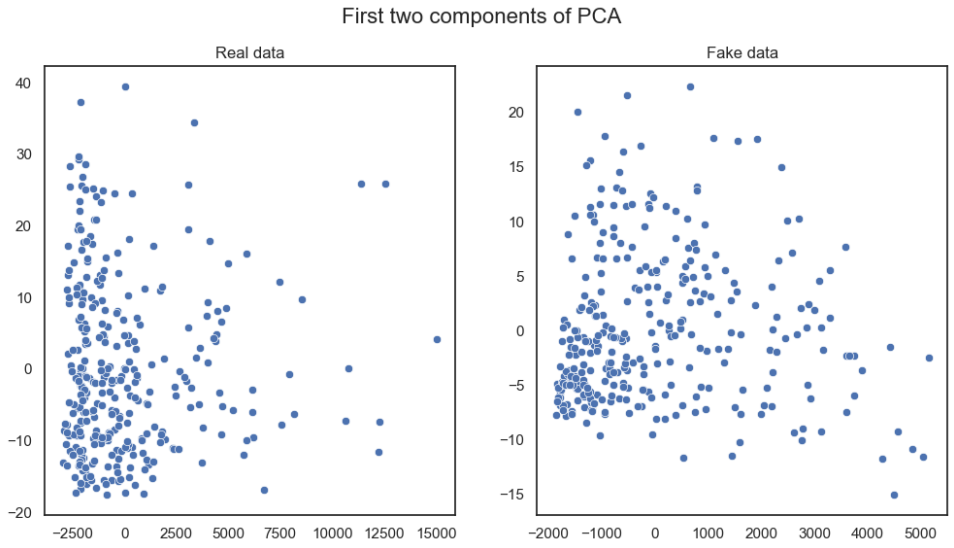

Real
Fake
Cumsum
9
10
11
12
13
14
15
16
17
18
19
20


Distribution per feature
kind
Fake
Real
proportion
A12
A14
A11
A13
1
real
fake
Probability
2
A34
A32
A30
A33
A31
3


kind
Fake
Real
real
fake
proportion
Probability
A43
A42
A40
A49
A41
A46
A45
A410
A48
A44
4
0
2500
5000
7500
10000
12500
15000
17500
5
A61
A65
A63
A62
A64
6
A75
A72
A73
A74
A71
7
4.0
2.0
3.0
1.0
8
A93
A94
A92
A91
9
A101
A103
A102
10
11
A121
A124
A123
A122
12
13
A143
A142
A141
14
A152
A153
A151
15
16
A173
A174
A172
A171
17
18







Machine learning efficiency:

```
Classifier F1-scores and their Jaccard similarities::
                            f1_real  f1_fake  jaccard_similarity
index
DecisionTreeClassifier_fake   0.7500   0.8438              0.5238
DecisionTreeClassifier_real   0.6562   0.4062              0.3333
LogisticRegression_fake       0.7500   0.8594              0.7067
LogisticRegression_real       0.7656   0.5938              0.4713
MLPClassifier_fake            0.7500   0.8125              0.6842
MLPClassifier_real            0.7656   0.5000              0.3474
RandomForestClassifier_fake   0.7969   0.8438              0.7534
RandomForestClassifier_real   0.7031   0.7188              0.6623

Privacy results:
                                        result
Duplicate rows between sets (real/fake)  (0, 0)
nearest neighbor mean                    3.7493
nearest neighbor std                     0.3973

Miscellaneous results:
                                  Result
Column Correlation Distance RMSE  0.2010
Column Correlation distance MAE   0.1319

Results:
                                                result
Basic statistics                                0.9930
Correlation column correlations                 0.5714
Mean Correlation between fake and real columns  0.5436
1 - MAPE Estimator results                      0.8266
Similarity Score                                0.7336
```

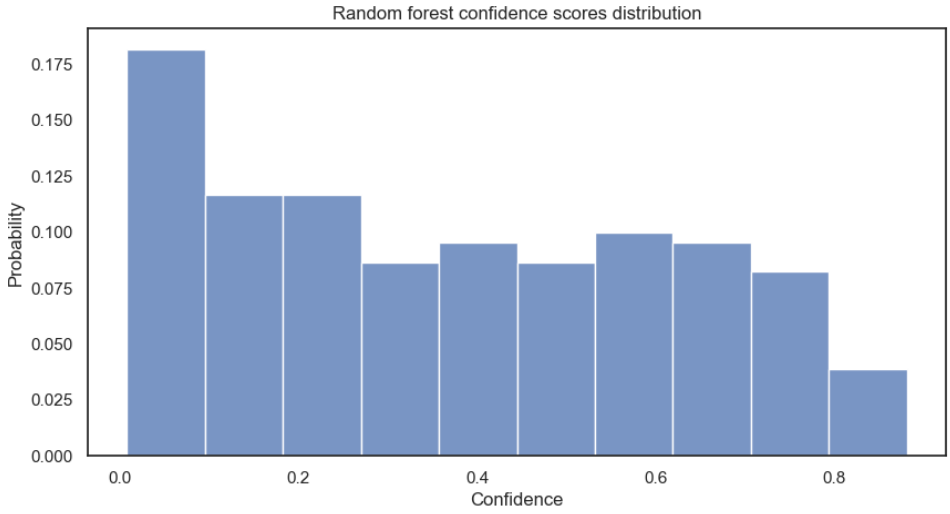
We can see that our data isn't perfect but it was able to learn and be similar by statistical measures. And by machine learning efficiency we see that training our model using fake data gives more or less the same results as training with real data.

Part 2 – GAN with a twist

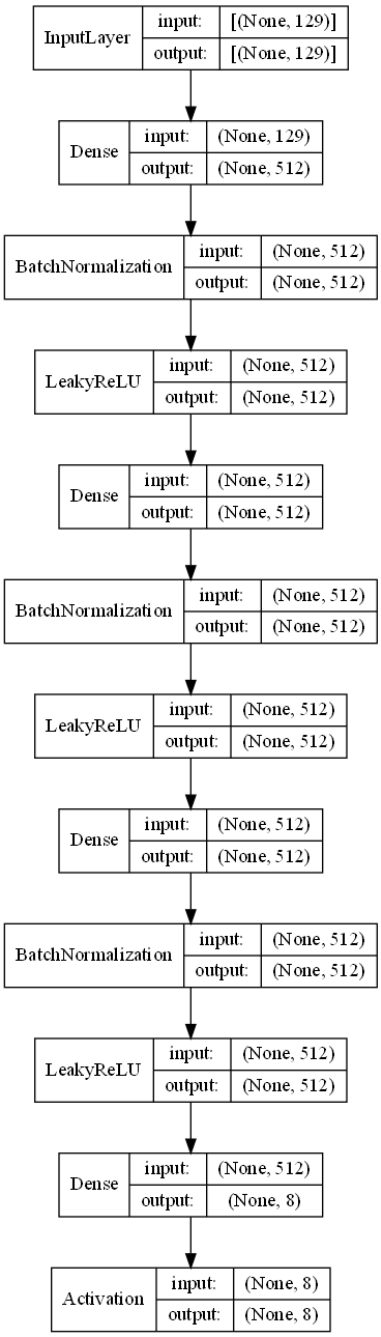
Diabetes dataset

RandomForest classifier's performance:

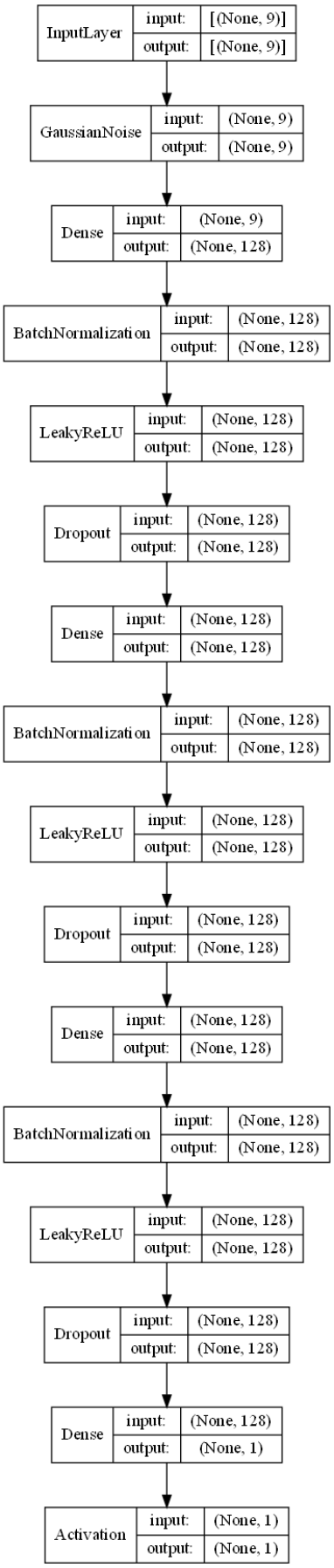
Train score	Test score	Confidence score			
		Min	Max	Median	Mean
0.773	0.736	0.007	0.882	0.353	0.373



The architecture of the generator is the following:

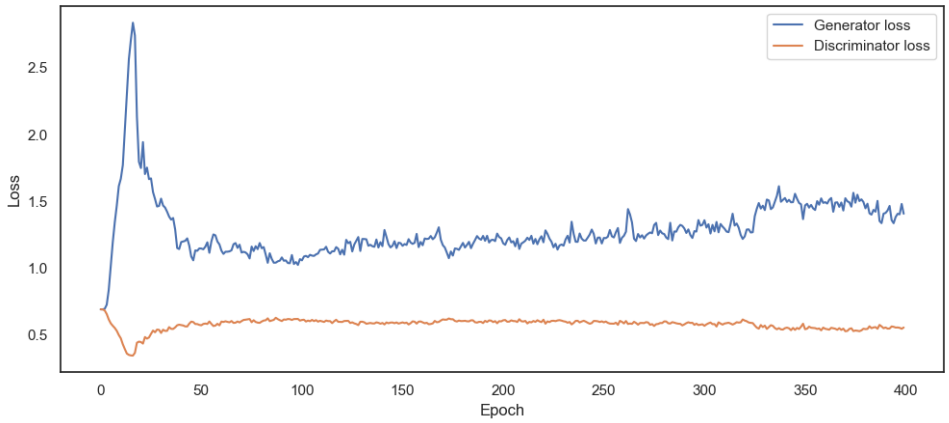


The architecture of the discriminator is the following:

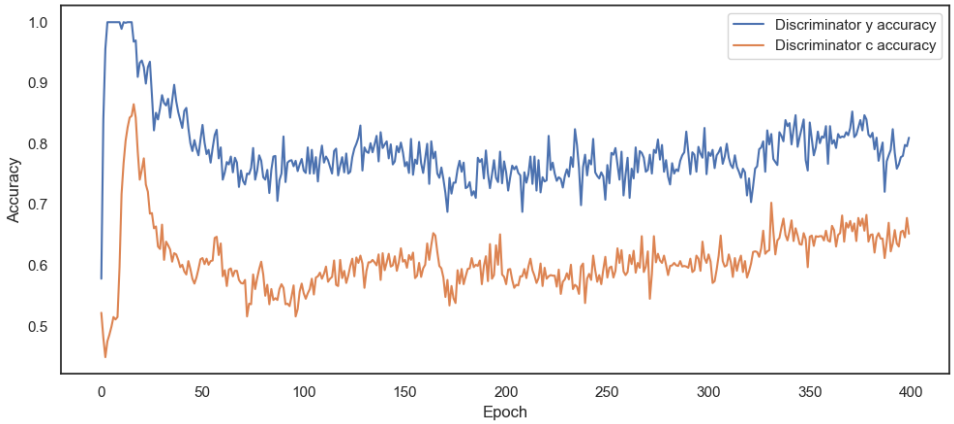

InputLayer
input: [(None, 9)]
output: [(None, 9)]
GaussianNoise
input: (None, 9)
output: (None, 9)
Dense
input: (None, 9)
output: (None, 128)
BatchNormalization
input: (None, 128)
output: (None, 128)
LeakyReLU
input: (None, 128)
output: (None, 128)
Dropout
input: (None, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 128)
BatchNormalization
input: (None, 128)
output: (None, 128)
LeakyReLU
input: (None, 128)
output: (None, 128)
Dropout
input: (None, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 128)
BatchNormalization
input: (None, 128)
output: (None, 128)
LeakyReLU
input: (None, 128)
output: (None, 128)
Dropout
input: (None, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 1)
Activation
input: (None, 1)
output: (None, 1)

Generator and the discriminator training metrics:

The loss and accuracy of the generator and the discriminator can tell us a lot about the training process. First let's look at the loss:



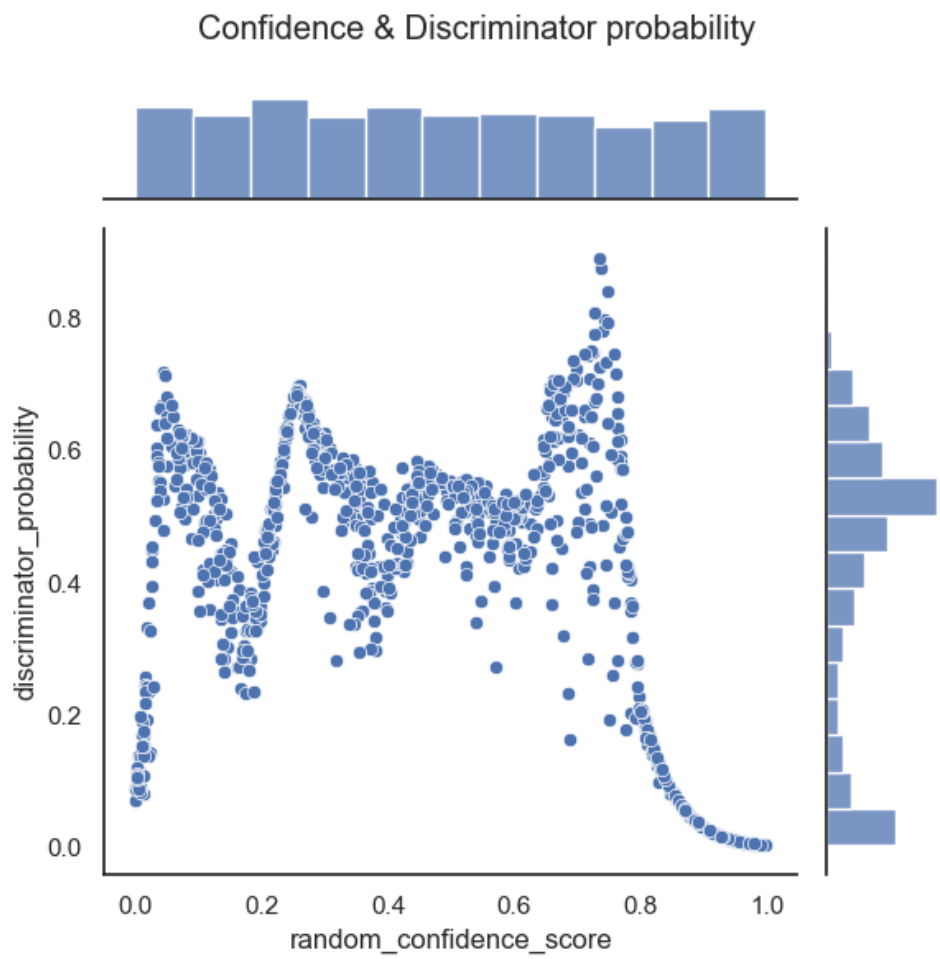
As we can see the generator has a lot of variance (going back and forth) compared to the discriminator loss. This makes us believe that the discriminator is the consistent leader. We can also see that the discriminator is pretty constant and does not change a lot. These metrics do not tell us much about the data generation and are a little weird indicating a problem in the data generation. Secondly, let's take a look at the accuracy:



As we can see the c and y accuracy have a lot of variance (going back and forth). The discriminator is a lot more sure about the y data then the real data but still isn't 100 percent sure what is y. In our opinion this generator was not very successful in data generation and the discriminator was not sure about c and y because of the similarity.

The confidence score effect on discriminator probability to be y:

Let's see if the confidence score (c) changes makes the discriminator believe it is y more or less.



As we can see in the range confidence scores [0, 0.1] and [0.8, 1] our GAN gives a smaller probability to be y but in range [0.1, 0.8] the probabilities are higher.

Samples that did fool the discriminator:

Our GAN managed to create 484 samples out of 1000 that we requested that did fool the discriminator. Let's take a look at a few of these samples:

	preg	plas	pres	skin	insu	mass	pedi	age	class
2	0.0036	83.8412	0.0003	0.0045	0.0000	33.6748	0.0899	21.0000	tested_negative
6	0.0132	97.5203	0.0005	0.0058	0.0001	37.8512	0.1160	21.0000	tested_negative
7	8.6169	96.8087	0.0124	4.4868	821.2357	66.6213	2.4017	26.6578	tested_positive
9	0.0001	74.3051	0.0000	0.0017	0.0000	28.3501	0.0787	21.0000	tested_negative
11	7.1129	125.5543	0.0121	10.8573	796.3124	66.7351	2.4132	27.1626	tested_positive
13	7.8048	97.1313	0.0076	0.2409	753.7131	65.7377	2.3946	22.7101	tested_positive
16	6.0867	146.0002	0.0036	53.7727	815.5380	67.0044	2.4159	23.3536	tested_positive
17	0.0120	91.3802	0.0004	0.0046	0.0002	35.6372	0.1042	21.0000	tested_negative
18	5.1368	109.3468	0.0003	0.0040	179.9114	63.8891	2.3417	21.0235	tested_positive
23	0.0010	78.3077	0.0001	0.0031	0.0000	32.4203	0.0826	21.0000	tested_negative

Samples that didn't fool the discriminator:

Our GAN managed to create 516 samples out of 1000 that we requested that didn't fool the discriminator. Let’s take a look at a few of these samples:

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	0.0026	198.9902	0.0000	99.0000	1.3907	67.1000	2.4200	21.0001	tested_positive
1	0.3636	197.3423	0.0002	98.9166	196.7814	67.0948	2.4199	21.0952	tested_positive
3	0.7627	119.4860	0.0014	0.0164	0.4732	51.9765	1.3079	21.0010	tested_negative
4	0.0020	80.0594	0.0004	0.0085	0.0000	32.6375	0.0867	21.0000	tested_negative
5	0.0289	198.8855	0.0000	98.9988	18.8434	67.0996	2.4200	21.0032	tested_positive
8	1.5920	119.1786	0.0008	0.0132	3.3880	57.1847	2.0076	21.0025	tested_negative
10	0.0012	79.2941	0.0002	0.0040	0.0000	32.7849	0.0831	21.0000	tested_negative
12	0.0001	198.9996	0.0000	99.0000	0.0131	67.1000	2.4200	21.0000	tested_positive
14	0.0003	198.9984	0.0000	99.0000	0.1147	67.1000	2.4200	21.0000	tested_positive
15	1.0608	193.4161	0.0008	97.6344	541.7171	67.0848	2.4195	21.3353	tested_positive

Fake and real data similarity evaluation using table_evaluter:

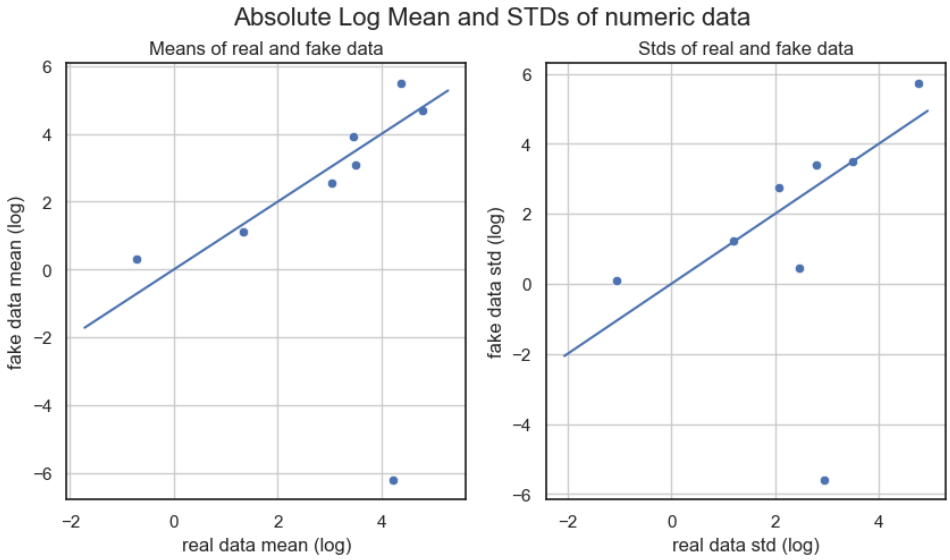
For the samples that did fool the discriminator, we evaluated the similarity to the real data using TableEvaluator - a library to evaluate how similar a synthesized dataset is to real data. In other words, it tries to give an indication into how real our fake data is.

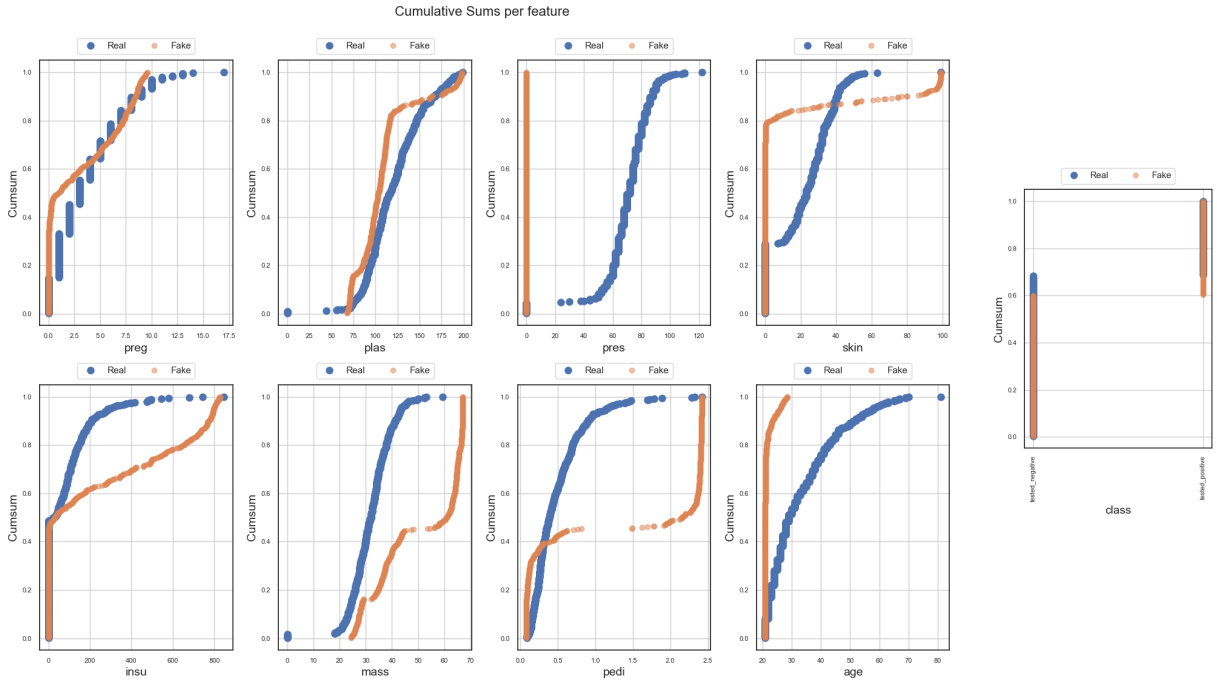
First evaluation is a visualized statistics in order to see if the GAN is able to capture properties. we take a look at:

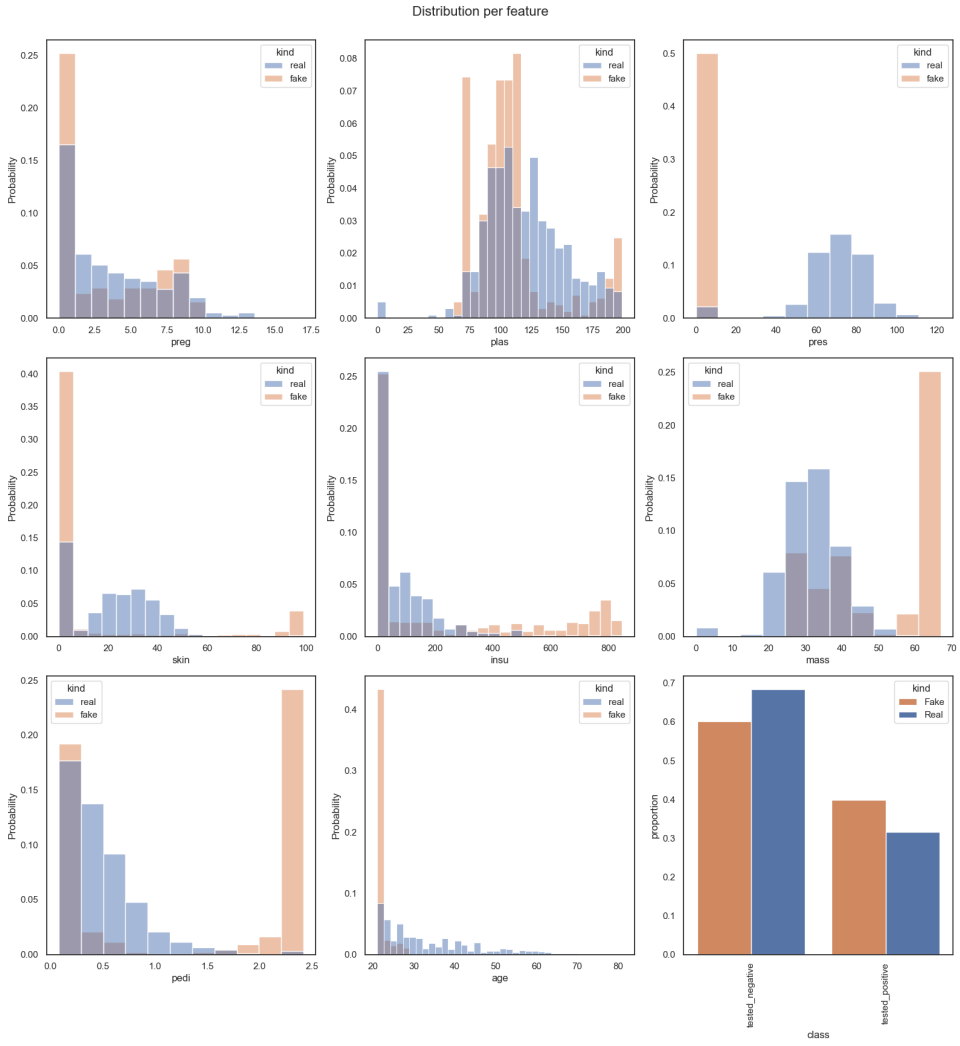
7. Columnwise mean and standard deviation
8. Cumulative Sums
9. Column Correlations

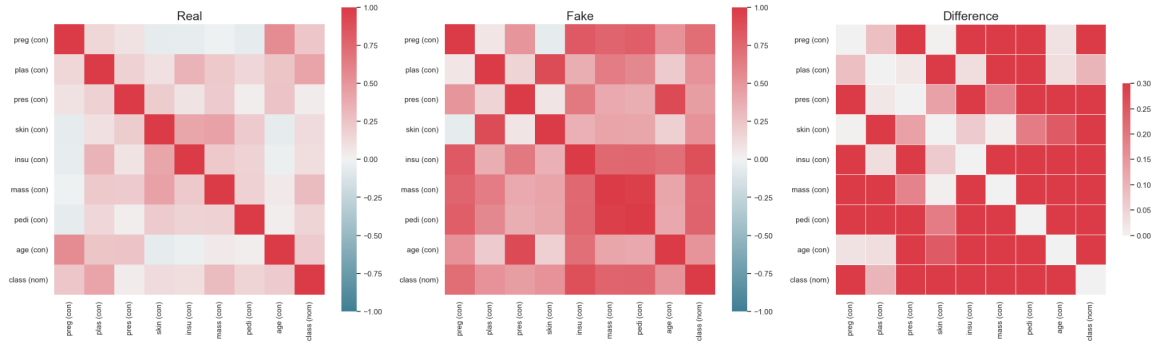
Second evaluation of the performance using TableEvaluator is machine learning efficiency. This evaluates the performance of using synthetic data as training data for machine learning. In other words, we train the model on real\fake data and test it on fake\real data respectively to see if the results are similar.

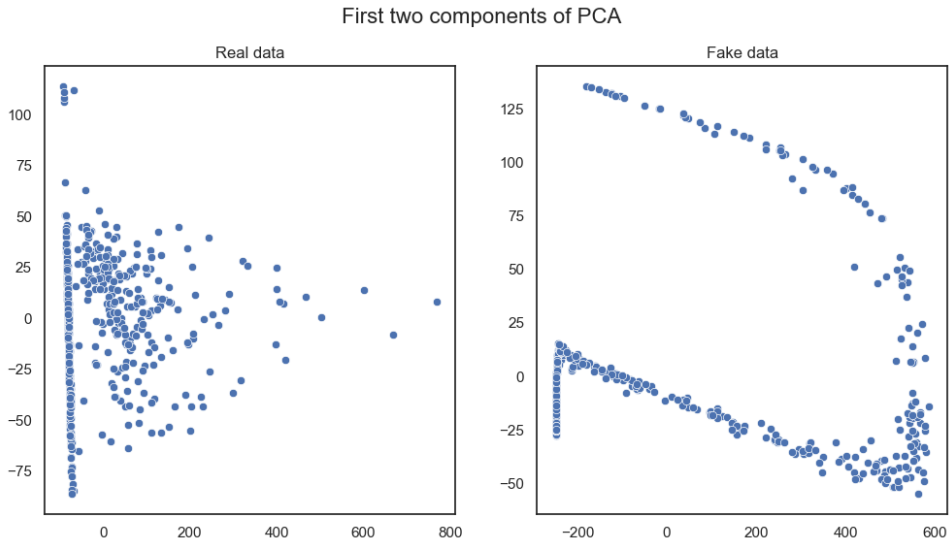
Visualized statistics:






Distribution per feature
kind
real
fake
Probability
preg
plas
pres
skin
insu
mass
pedi
age
class
Fake
Real
proportion
tested_negative
tested_positive


Real
Fake
Difference
preg (con)
plas (con)
pres (con)
skin (con)
insu (con)
mass (con)
pedi (con)
age (con)
class (nom)



Machine learning efficiency:

Classifier F1-scores and their Jaccard similarities::

index	f1_real	f1_fake	jaccard_similarity
DecisionTreeClassifier_fake	0.6392	0.9897	0.4809
DecisionTreeClassifier_real	0.7216	0.7526	0.5156
LogisticRegression_fake	0.7423	0.9897	0.5772
LogisticRegression_real	0.8041	0.7526	0.7321
MLPClassifier_fake	0.2371	0.9897	0.1279
MLPClassifier_real	0.7113	0.5052	0.3566
RandomForestClassifier_fake	0.8660	0.9897	0.7798
RandomForestClassifier_real	0.7732	0.7629	0.6441

Privacy results:

	result
Duplicate rows between sets (real/fake)	(0, 0)
nearest neighbor mean	2.3013
nearest neighbor std	0.8552

Miscellaneous results:

	Result
Column Correlation Distance RMSE	0.4519
Column Correlation distance MAE	0.3486

Results:

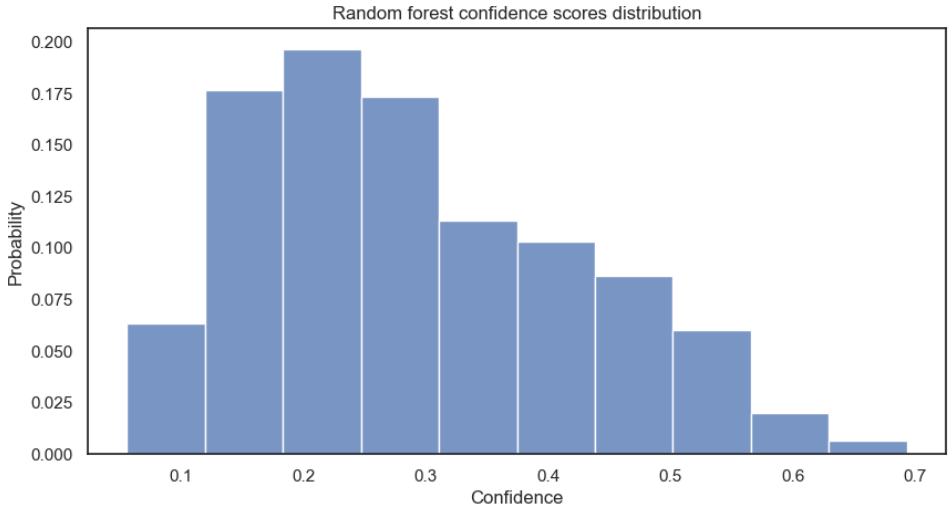
	result
Basic statistics	0.5040
Correlation column correlations	-0.0303
Mean Correlation between fake and real columns	0.7796
1 - MAPE Estimator results	0.4239
Similarity Score	0.4193

We can see that our data is very not similar to real data by statistical measures. And by machine learning efficiency we see that training our model using fake data gives worse results as training with real data.

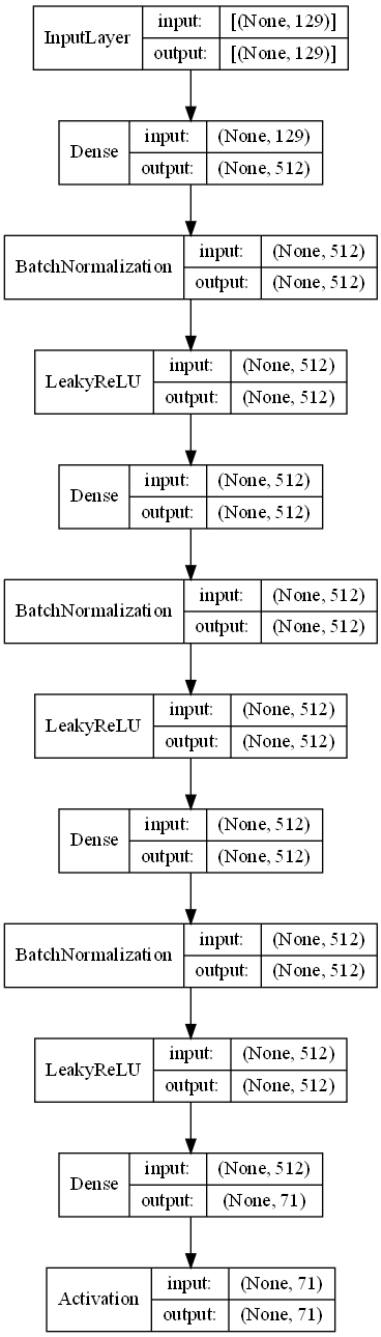
German Credit dataset

RandomForest classifier's performance:

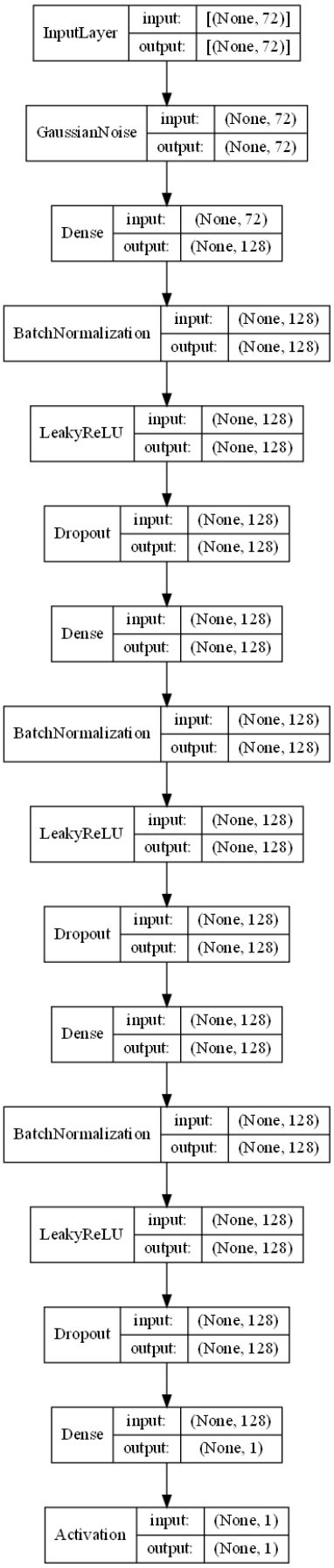
Train score	Test score	Confidence score			
		Min	Max	Median	Mean
0.741	0.737	0.056	0.693	0.279	0.294



The architecture of the generator is the following:

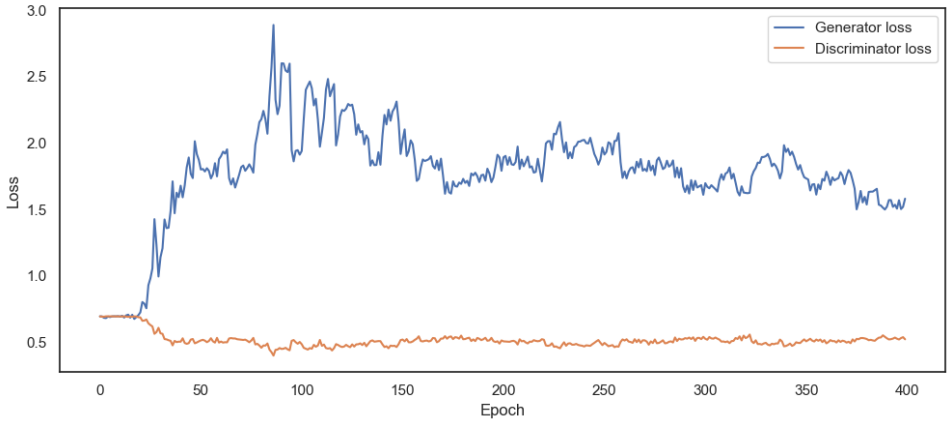


The architecture of the discriminator is the following:

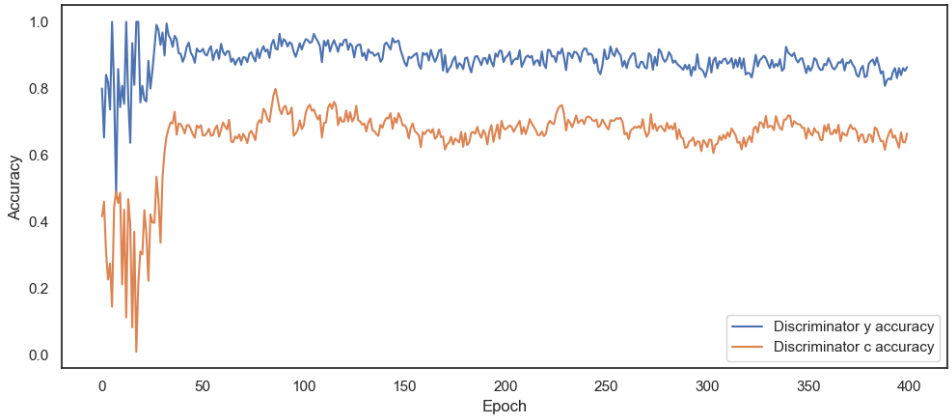

InputLayer
input: [(None, 72)]
output: [(None, 72)]
GaussianNoise
input: (None, 72)
output: (None, 72)
Dense
input: (None, 72)
output: (None, 128)
BatchNormalization
input: (None, 128)
output: (None, 128)
LeakyReLU
input: (None, 128)
output: (None, 128)
Dropout
input: (None, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 128)
BatchNormalization
input: (None, 128)
output: (None, 128)
LeakyReLU
input: (None, 128)
output: (None, 128)
Dropout
input: (None, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 128)
BatchNormalization
input: (None, 128)
output: (None, 128)
LeakyReLU
input: (None, 128)
output: (None, 128)
Dropout
input: (None, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 1)
Activation
input: (None, 1)
output: (None, 1)

Generator and the discriminator training metrics:

The loss and accuracy of the generator and the discriminator can tell us a lot about the training process. First let's look at the loss:



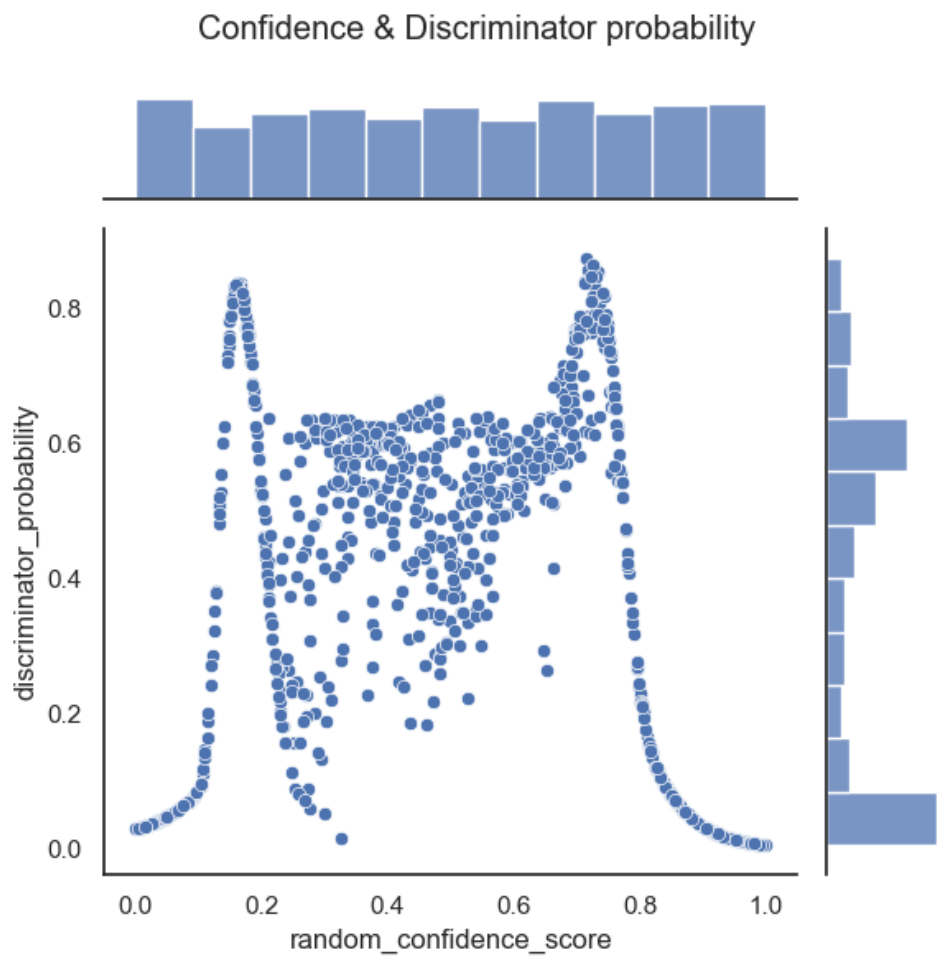
As we can see the generator has a lot of variance (going back and forth) compared to the discriminator loss. This makes us believe that the discriminator is the consistent leader. We can also see that the discriminator is pretty constant and does not change a lot. These metrics do not tell us much about the data generation and are a little weird indicating a problem in the data generation. Secondly, let's take a look at the accuracy:



As we can see the c and y accuracy have a lot of variance (going back and forth). The discriminator is a lot more sure about the y data then the real data but still isn't 100 percent sure what is y. In our opinion this generator was not very successful in data generation and the discriminator was not sure about c and y because of the similarity.

The confidence score effect on discriminator probability to be y:

Let's see if the confidence score (c) changes makes the discriminator believe it is y more or less.



As we can see in the range confidence scores [0, 0.1] and [0.8, 1] our GAN gives a smaller probability to be y but in range [0.1, 0.8] the probabilities are higher.

Samples that did fool the discriminator:

Our GAN managed to create 454 samples out of 1000 that we requested that did fool the discriminator. Let's take a look at a few of these samples:

	1	2	3	4	5	6	7	8	9	10	...	12	13	14	15	16	17	18	19	20	class
4	A11	71.8687	A32	A45	18,420.8984	A63	A72	1.0000	A93	A103	...	A123	19.0000	A142	A153	2.0000	A172	1.0000	A192	A201	2
6	A11	6.5634	A34	A45	14,295.8076	A65	A74	2.0000	A94	A103	...	A123	20.0901	A142	A152	3.0000	A173	2.0000	A192	A201	1
8	A11	18.4878	A34	A45	16,595.5645	A63	A71	2.0000	A94	A103	...	A123	19.0585	A142	A151	3.0000	A172	2.0000	A192	A201	2
11	A11	4.9940	A34	A45	12,749.1338	A65	A74	2.0000	A94	A103	...	A123	28.3713	A142	A152	3.0000	A173	2.0000	A192	A201	1
12	A11	4.3758	A34	A45	11,271.3945	A65	A74	2.0000	A94	A103	...	A123	48.4034	A142	A152	3.0000	A173	2.0000	A192	A201	1
20	A11	4.0062	A34	A40	4,979.4521	A65	A74	3.0000	A94	A103	...	A123	74.4663	A143	A151	3.0000	A173	2.0000	A191	A201	1
22	A11	4.0040	A34	A40	4,244.2627	A65	A74	3.0000	A94	A103	...	A123	74.6387	A143	A151	3.0000	A173	2.0000	A191	A202	1
24	A11	71.9755	A30	A41	18,423.9453	A63	A72	1.0000	A93	A103	...	A123	19.0000	A142	A153	2.0000	A173	1.0000	A192	A201	2
25	A11	4.0481	A34	A45	6,888.2568	A65	A74	2.0000	A94	A103	...	A123	72.1009	A143	A152	3.0000	A173	2.0000	A191	A201	1
26	A11	4.2079	A34	A45	10,371.5811	A65	A74	2.0000	A94	A103	...	A123	58.9223	A143	A152	3.0000	A173	2.0000	A191	A201	1

Samples that didn't fool the discriminator:

Our GAN managed to create 516 samples out of 1000 that we requested that didn't fool the discriminator. Let’s take a look at a few of these samples:

	1	2	3	4	5	6	7	8	9	10	...	12	13	14	15	16	17	18	19	20	class
0	A11	71.9934	A30	A41	18,423.9980	A63	A72	1.0000	A93	A103	...	A123	19.0000	A142	A153	2.0000	A173	1.0000	A192	A201	2
1	A14	4.0000	A34	A40	537.3464	A65	A74	3.0000	A94	A101	...	A123	74.9972	A143	A151	2.0000	A173	2.0000	A191	A202	1
2	A11	4.1783	A34	A45	10,032.6973	A65	A74	2.0000	A94	A103	...	A123	63.8912	A143	A152	3.0000	A173	2.0000	A191	A201	1
3	A11	23.9207	A34	A45	16,640.6719	A63	A71	2.0000	A94	A103	...	A123	19.0344	A142	A151	3.0000	A172	2.0000	A192	A201	1
5	A11	4.5831	A34	A45	12,071.4697	A65	A74	2.0000	A94	A103	...	A123	35.4762	A142	A152	3.0000	A173	2.0000	A192	A201	1
7	A11	71.9993	A30	A41	18,424.0000	A61	A72	1.0000	A92	A103	...	A122	19.0000	A142	A153	2.0000	A173	1.0000	A192	A201	2
9	A11	71.9977	A30	A41	18,424.0000	A61	A72	1.0000	A93	A103	...	A123	19.0000	A142	A153	2.0000	A173	1.0000	A192	A201	2
10	A11	71.9951	A30	A41	18,424.0000	A61	A72	1.0000	A93	A103	...	A123	19.0000	A142	A153	2.0000	A173	1.0000	A192	A201	2
13	A11	71.9976	A30	A41	18,424.0000	A61	A72	1.0000	A93	A103	...	A123	19.0000	A142	A153	2.0000	A173	1.0000	A192	A201	2
14	A11	68.2401	A32	A45	18,235.2598	A63	A72	2.0000	A93	A103	...	A123	19.0001	A142	A151	2.0000	A172	1.0000	A192	A201	2

Fake and real data similarity evaluation using table_evaluter:

For the samples that did fool the discriminator, we evaluated the similarity to the real data using TableEvaluator - a library to evaluate how similar a synthesized dataset is to real data. In other words, it tries to give an indication into how real our fake data is.

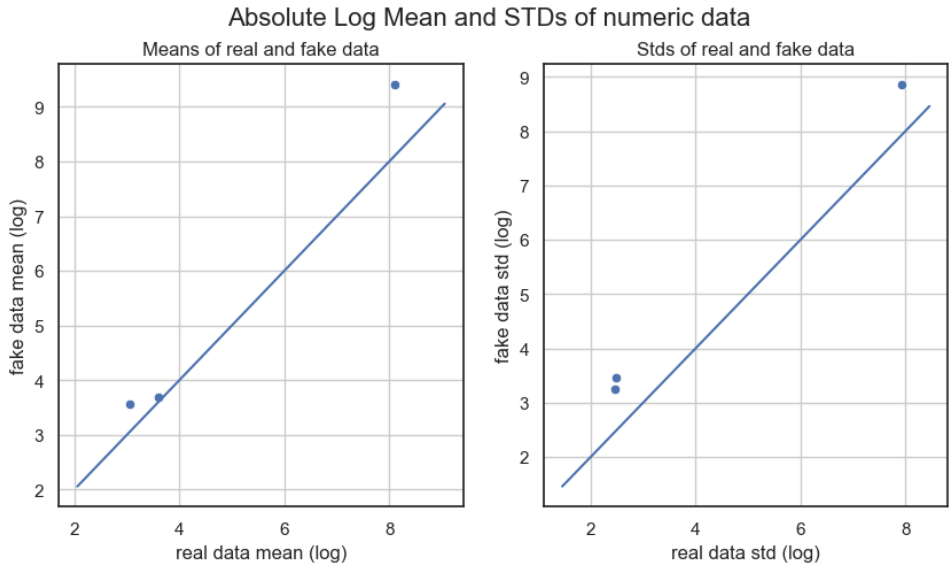
First evaluation is a visualized statistics in order to see if the GAN is able to capture properties. we take a look at:

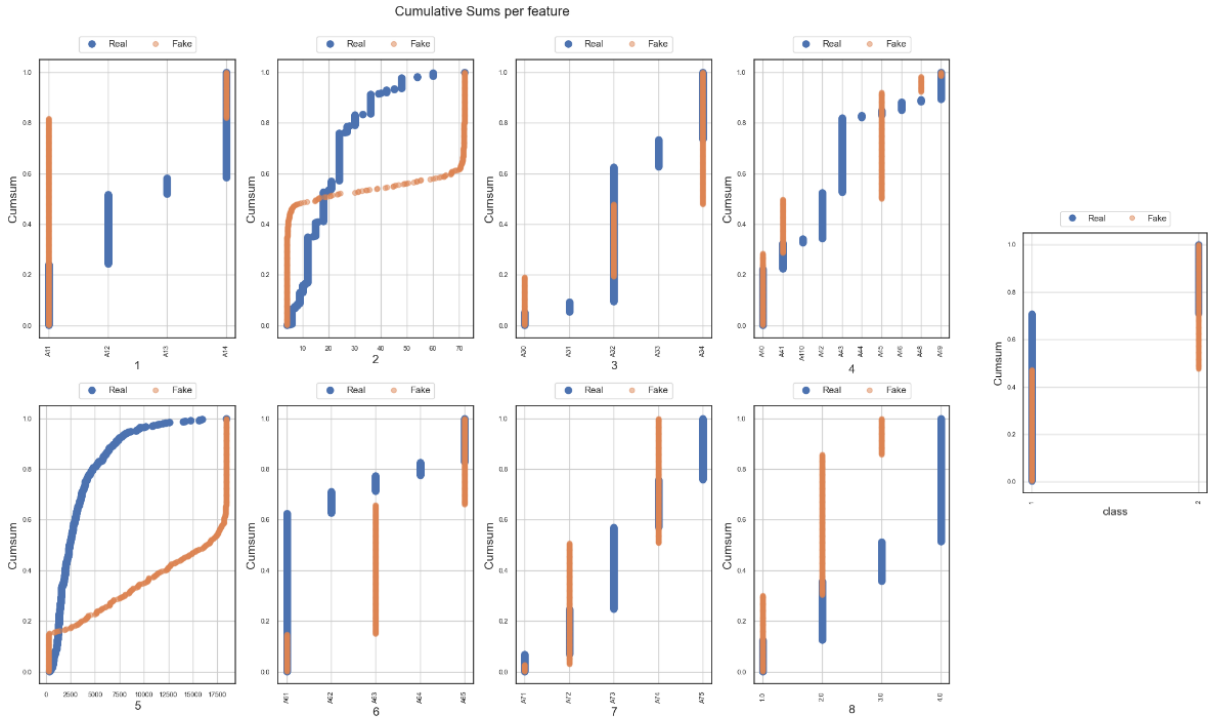
10. Columnwise mean and standard deviation
11. Cumulative Sums
12. Column Correlations

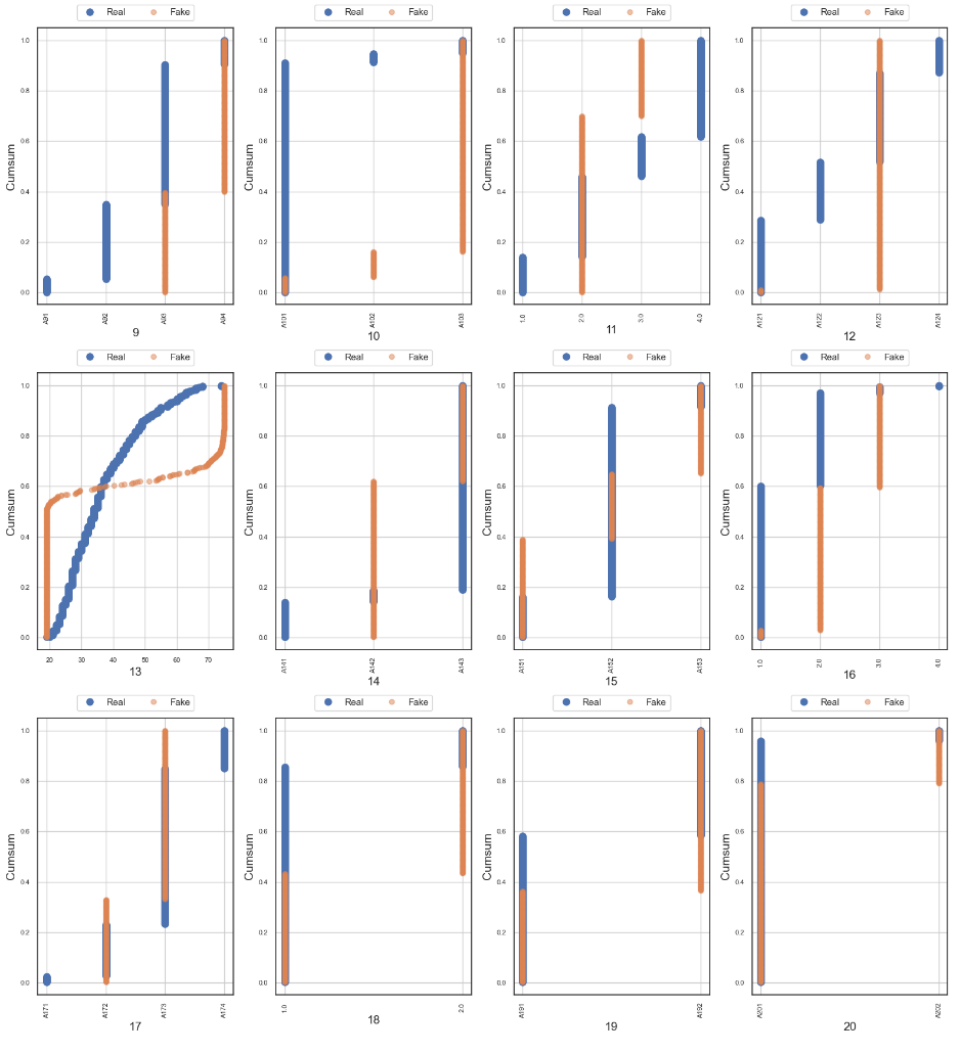
Second evaluation of the performance using TableEvaluator is machine learning efficiency. This evaluates the performance of using synthetic data as training data for machine learning. In

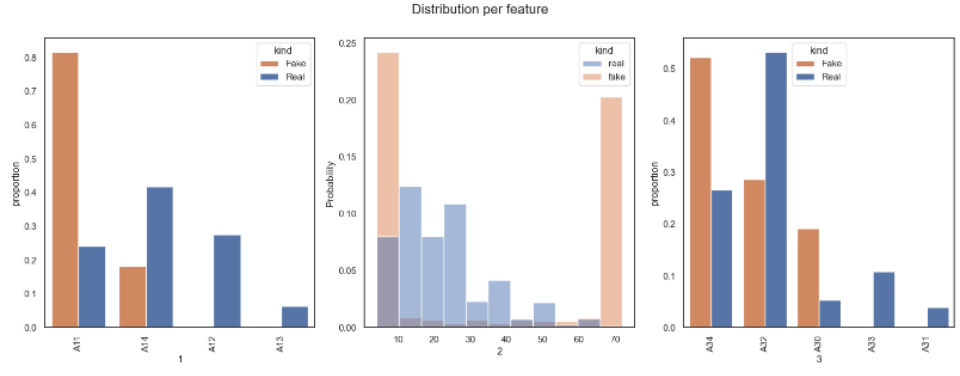
other words, we train the model on real\fake data and test it on fake\real data respectively to see if the results are similar.

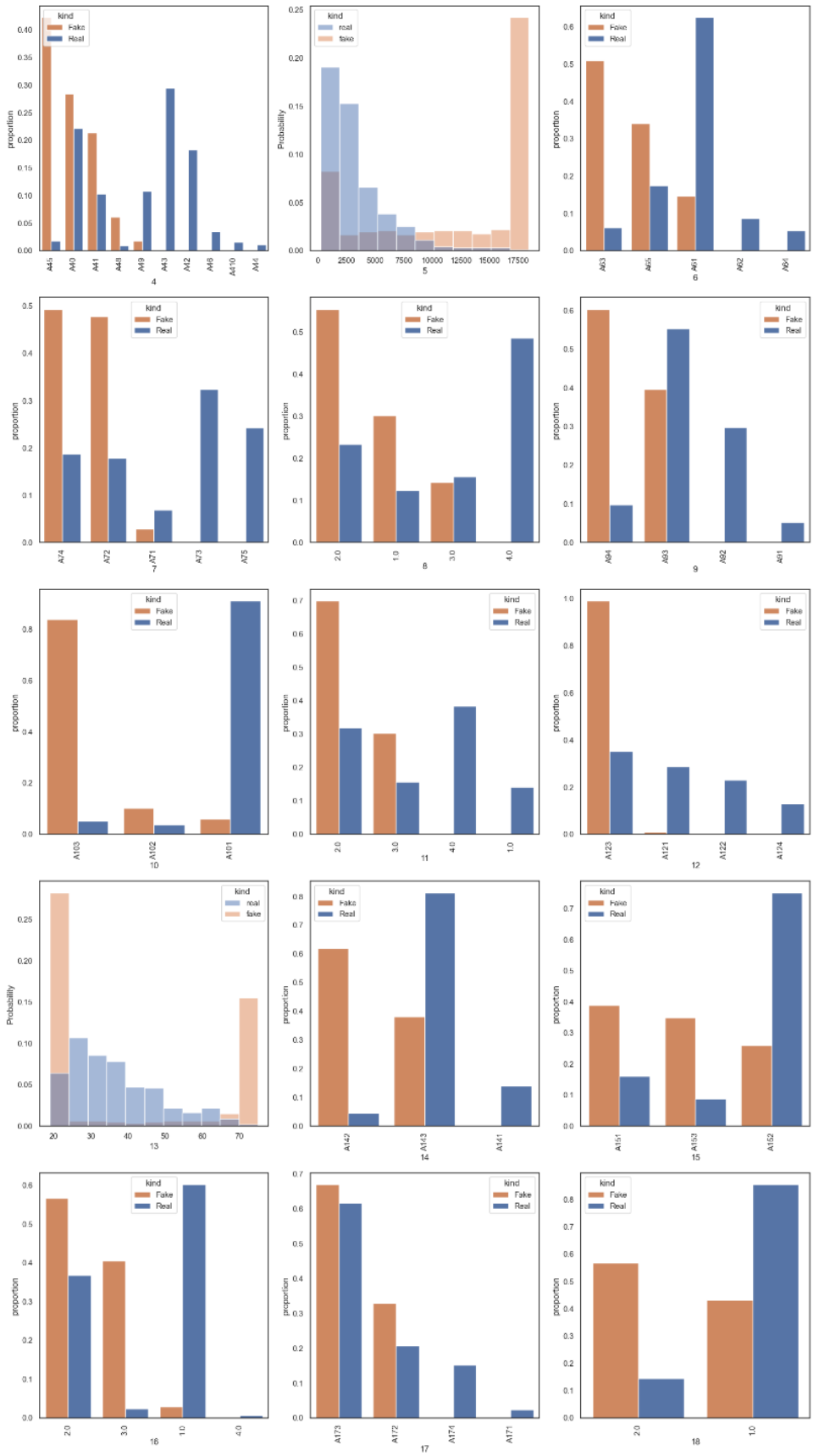
Visualized statistics:

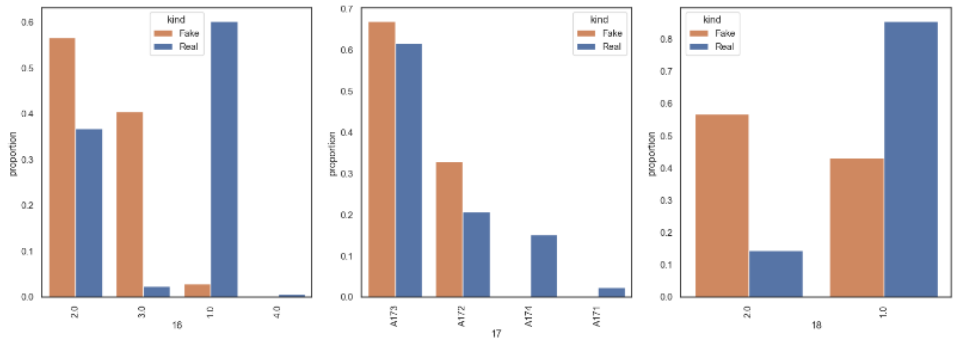


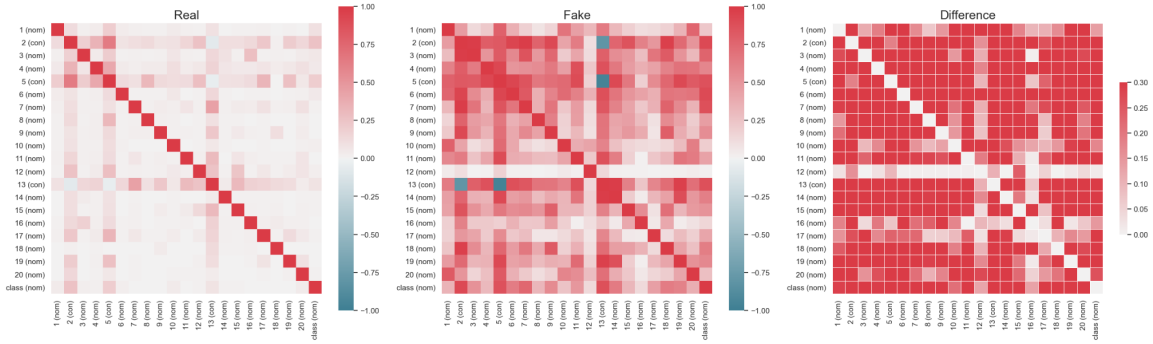


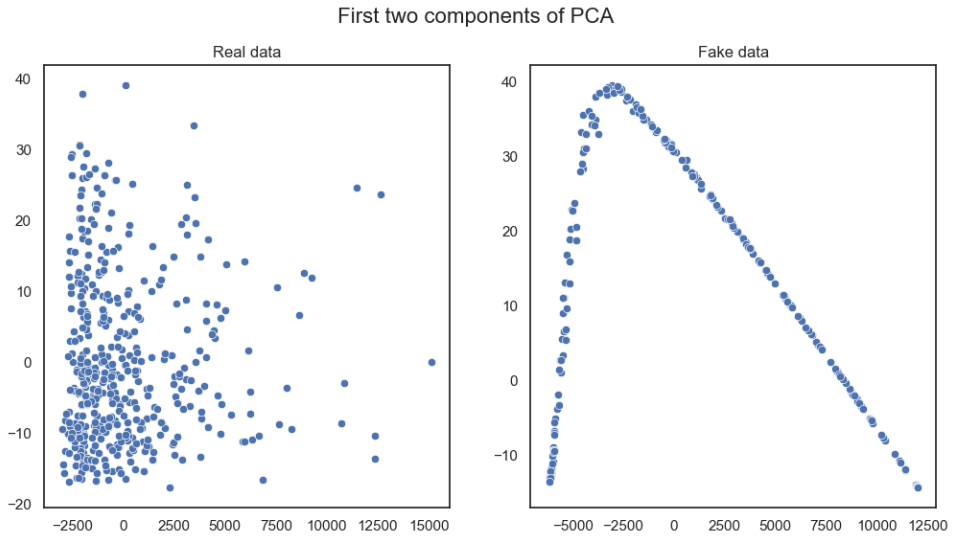

Real
Fake
Cumsum
9
10
11
12
13
14
15
16
17
18
19
20


Distribution per feature
kind
Fake
Real
proportion
1
Probability
real
fake
2
3


kind
Fake
Real
real
fake
proportion
Probability
A45
A40
A41
A48
A49
A43
A42
A46
A410
A44
0
2500
5000
7500
10000
12500
15000
17500
4
5
A63
A65
A61
A62
A64
6
A74
A72
A71
A73
A75
7
2.0
1.0
3.0
4.0
8
A94
A93
A92
A91
9
A103
A102
A101
10
11
A123
A121
A122
A124
12
13
A142
A143
A141
14
A151
A153
A152
15
16
A173
A172
A174
A171
17
18







Machine learning efficiency:

```
Classifier F1-scores and their Jaccard similarities::
                            f1_real  f1_fake  jaccard_similarity
index
DecisionTreeClassifier_fake   0.8022   0.9890              0.6852
DecisionTreeClassifier_real   0.6484   0.3407              0.1895
LogisticRegression_fake       0.5165   1.0000              0.3481
LogisticRegression_real       0.7582   0.7143              0.6396
MLPClassifier_fake            0.8352   0.9121              0.5965
MLPClassifier_real            0.5495   0.7363              0.3788
RandomForestClassifier_fake   0.5165   0.9890              0.3582
RandomForestClassifier_real   0.6923   0.7582              0.6697

Privacy results:
                                       result
Duplicate rows between sets (real/fake)  (0, 2)
nearest neighbor mean                    4.3418
nearest neighbor std                     0.3557

Miscellaneous results:
                                Result
Column Correlation Distance RMSE  0.4634
Column Correlation distance MAE   0.3887

Results:
                                                result
Basic statistics                                0.9021
Correlation column correlations                 0.3963
Mean Correlation between fake and real columns  0.4173
1 - MAPE Estimator results                      0.6070
Similarity Score                                0.5807
```

We can see that our data is very not similar to real data by statistical measures. And by machine learning efficiency we see that training our model using fake data gives worse results as training with real data.

Conclusion

We were able to create a conditional GAN and evaluate it using different approaches. Obviously we achieved better results when training the GAN using real data and not using a black box model. We still got some interesting results using the balack box approach.

Another additional steps that can be done in the future are:

- Change the GAN - different GAN types, different losses etc.
- Change preprocessing - there are different methods for preprocessing tabular data for GAN (CTGAN for example).