

Dataset1-Regression_output_11

October 7, 2021

1 Dataset 1 - Regression

1.1 Import Libraries

```
[1]: import train_test
import ABC_train_test
import regressionDataset
import network
import statsModel
import performanceMetrics
import dataset
import sanityChecks
import torch
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from torch.utils.data import Dataset, DataLoader
from torch import nn
import warnings
warnings.filterwarnings('ignore')
```

1.2 Parameters

General Parameters

1. Number of Samples

Discriminator Parameters

1. Size : number of hidden nodes

ABC-Generator parameters are as mentioned below: 1. mean : 1 ($\beta \sim N(\beta^*, \sigma)$ where β^* are coefficients of statistical model) or 1 ($\beta \sim N(0, \sigma)$) 2. std : $\sigma = 1, 0.1, 0.01$ (standard deviation)

```
[2]: n_features = 10
sample_size = 100
#Discriminator Parameters
hidden_nodes = 25
#ABC Generator Parameters
mean = 1
```

```
variance = 0.001
```

1.3 Dataset

Generate a random regression problem

$Y = 1 + \beta_1 x_1 + \beta_2 x_2 + \beta_2 x_3 + \dots + \beta_n x_n + N(0, \sigma)$ where $\sigma = 0.1$

```
[3]: X,Y = regressionDataset.regression_data(sample_size,n_features)
```

	X1	X2	X3	X4	X5	X6	X7 \
0	-0.611893	1.115631	1.031223	-0.028719	1.762731	0.599901	-0.468322
1	-0.820147	-2.044464	-0.276468	2.041207	0.327716	0.157858	0.982870
2	-0.323860	-0.146093	0.295811	0.598968	1.072365	-0.354907	0.022846
3	-1.512640	-0.138579	-0.760355	-2.203243	0.066267	1.017670	0.884575
4	1.098851	0.731226	-0.253979	-0.169836	-0.220666	-0.353923	1.655579

	X8	X9	X10	Y
0	0.566095	-0.449700	-0.713518	107.849998
1	0.998651	0.102835	-0.273093	226.394817
2	-0.055615	-0.079051	-0.683425	50.054821
3	-0.441100	-0.356051	0.051186	-296.560934
4	-0.670714	2.424403	-0.626481	83.607402

1.4 Stats Model

```
[4]: [coeff,y_pred] = statsModel.statsModel(X,Y)
```

No handles with labels found to put in legend.

```
=====
                        OLS Regression Results
=====
Dep. Variable:          Y      R-squared:                1.000
Model:                  OLS    Adj. R-squared:            1.000
Method:                 Least Squares    F-statistic:        2.491e+07
Date:                   Thu, 07 Oct 2021    Prob (F-statistic):    2.61e-282
Time:                   18:59:34    Log-Likelihood:        600.34
No. Observations:      100    AIC:                   -1179.
Df Residuals:          89    BIC:                   -1150.
Df Model:               10
Covariance Type:        nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
const	-3.816e-17	6.34e-05	-6.02e-13	1.000	-0.000	0.000
x1	0.2479	7.08e-05	3500.112	0.000	0.248	0.248
x2	0.1276	6.73e-05	1896.396	0.000	0.127	0.128
x3	0.5276	6.57e-05	8028.136	0.000	0.527	0.528
x4	0.6114	6.73e-05	9087.917	0.000	0.611	0.612
x5	0.0402	6.55e-05	614.477	0.000	0.040	0.040

x6	0.1687	6.62e-05	2549.528	0.000	0.169	0.169
x7	0.2851	6.74e-05	4231.809	0.000	0.285	0.285
x8	0.4573	7.03e-05	6507.055	0.000	0.457	0.457
x9	0.1293	7.09e-05	1824.513	0.000	0.129	0.129
x10	0.0065	6.67e-05	97.737	0.000	0.006	0.007

Omnibus:	1.452	Durbin-Watson:	1.736
Prob(Omnibus):	0.484	Jarque-Bera (JB):	1.053
Skew:	0.241	Prob(JB):	0.591
Kurtosis:	3.140	Cond. No.	1.78

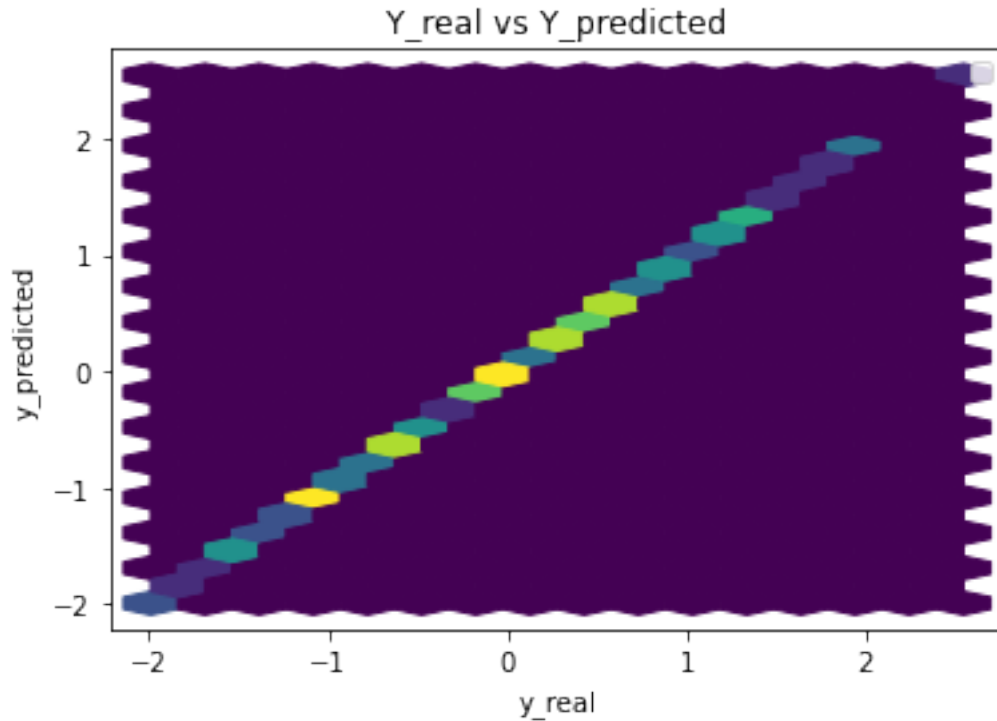
Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Parameters: const -3.816392e-17

x1	2.479428e-01
x2	1.276246e-01
x3	5.276010e-01
x4	6.113902e-01
x5	4.024706e-02
x6	1.687399e-01
x7	2.851491e-01
x8	4.572634e-01
x9	1.292844e-01
x10	6.522461e-03

dtype: float64



Performance Metrics

Mean Squared Error: 3.5728418730010814e-07

Mean Absolute Error: 0.0004842240492158185

Manhattan distance: 0.04842240492158184

Euclidean distance: 0.005977325382644885

2 Generator and Discriminator Networks

GAN Generator

```
[5]: class Generator(nn.Module):

    def __init__(self, n_input):
        super().__init__()
        self.output = nn.Linear(n_input, 1)

    def forward(self, x):
        x = self.output(x)
        return x
```

GAN Discriminator

```
[6]: class Discriminator(nn.Module):
```

```

def __init__(self,n_input,n_hidden):

    super().__init__()
    self.hidden = nn.Linear(n_input,n_hidden)
    self.output = nn.Linear(n_hidden,1)
    self.relu = nn.ReLU()

def forward(self, x):
    x = self.hidden(x)
    x = self.relu(x)
    x = self.output(x)
    return x

```

ABC Generator

The ABC generator is defined as follows:

$Y = 1 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots + \beta_n x_n + N(0, \sigma)$ where $\sigma = 0.1$

$\beta_i \sim N(0, \sigma^*)$ when $\mu = 0$ else

$\beta_i \sim N(\beta_i^*, \sigma^*)$ where β_i^* s are coefficients obtained from stats model

Parameters : μ and σ^*

σ^* takes the values 0.01,0.1 and 1

```

[7]: def ABC_pre_generator(x_batch,coeff,variance,mean,device):

    coeff_len = len(coeff)

    if mean == 0:
        weights = np.random.normal(0,variance,size=(coeff_len,1))
        weights = torch.from_numpy(weights).reshape(coeff_len,1)
    else:
        weights = []
        for i in range(coeff_len):
            weights.append(np.random.normal(coeff[i],variance))
        weights = torch.tensor(weights).reshape(coeff_len,1)

    y_abc = torch.matmul(x_batch,weights.float())
    gen_input = torch.cat((x_batch,y_abc),dim = 1).to(device)
    return gen_input

```

3 GAN Model

```

[8]: real_dataset = dataset.CustomDataset(X,Y)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

```

```
[9]: generator = Generator(n_features+2)
discriminator = Discriminator(n_features+2,hidden_nodes)

criterion = torch.nn.BCEWithLogitsLoss()
gen_opt = torch.optim.Adam(generator.parameters(), lr=0.01, betas=(0.5, 0.999))
disc_opt = torch.optim.Adam(discriminator.parameters(), lr=0.01, betas=(0.5, 0.
↪999))
```

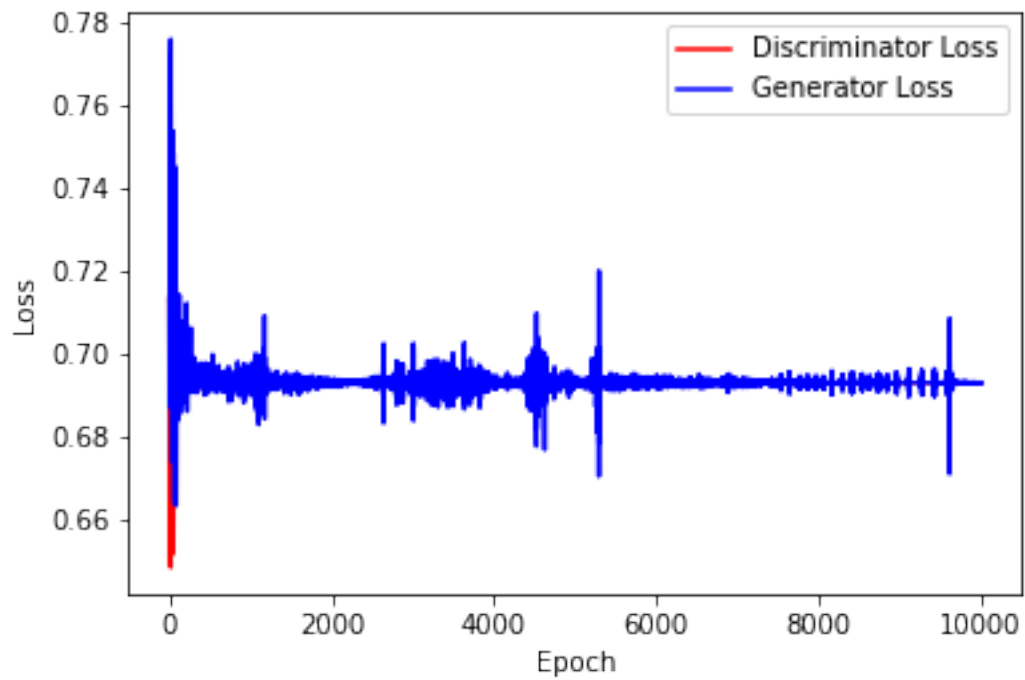
```
[10]: print(generator)
print(discriminator)
```

```
Generator(
  (output): Linear(in_features=12, out_features=1, bias=True)
)
Discriminator(
  (hidden): Linear(in_features=12, out_features=25, bias=True)
  (output): Linear(in_features=25, out_features=1, bias=True)
  (relu): ReLU()
)
```

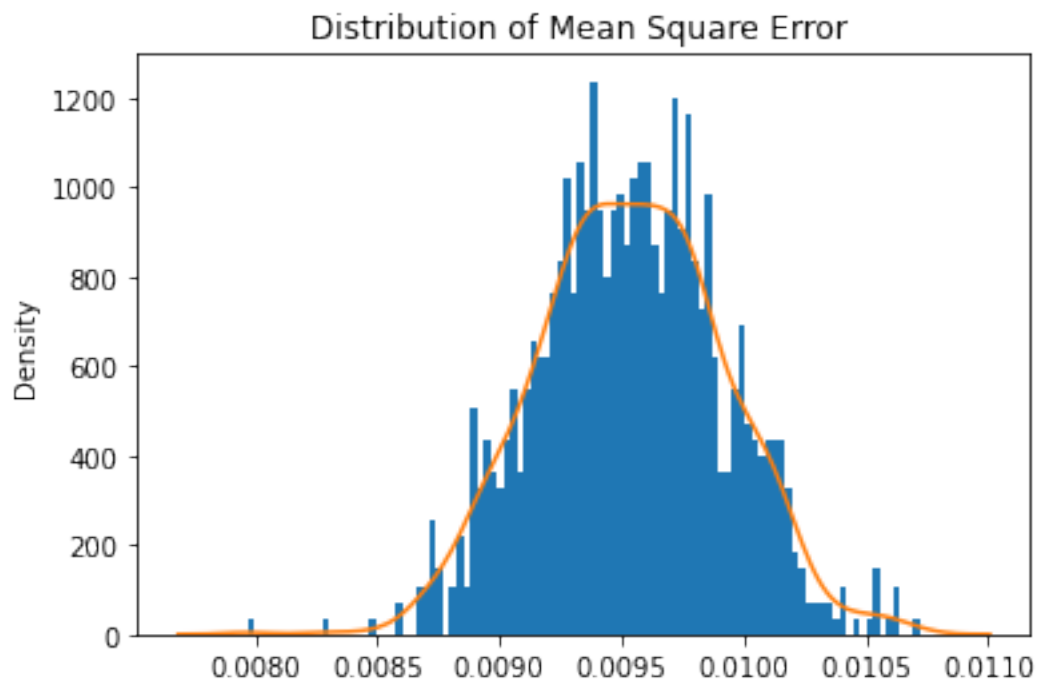
```
[11]: n_epochs = 5000
batch_size = sample_size//2
```

```
[12]: # Parameters
sample_size = 10000
mean = 0
std = 1
```

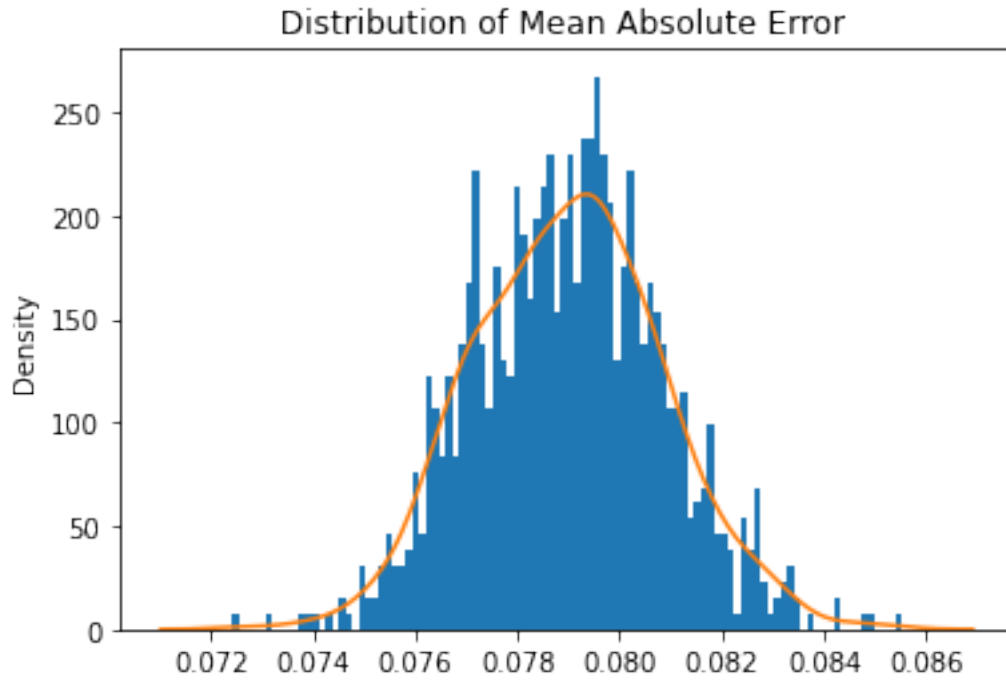
```
[13]: train_test.
↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,
↪n_epochs,criterion,device)
```



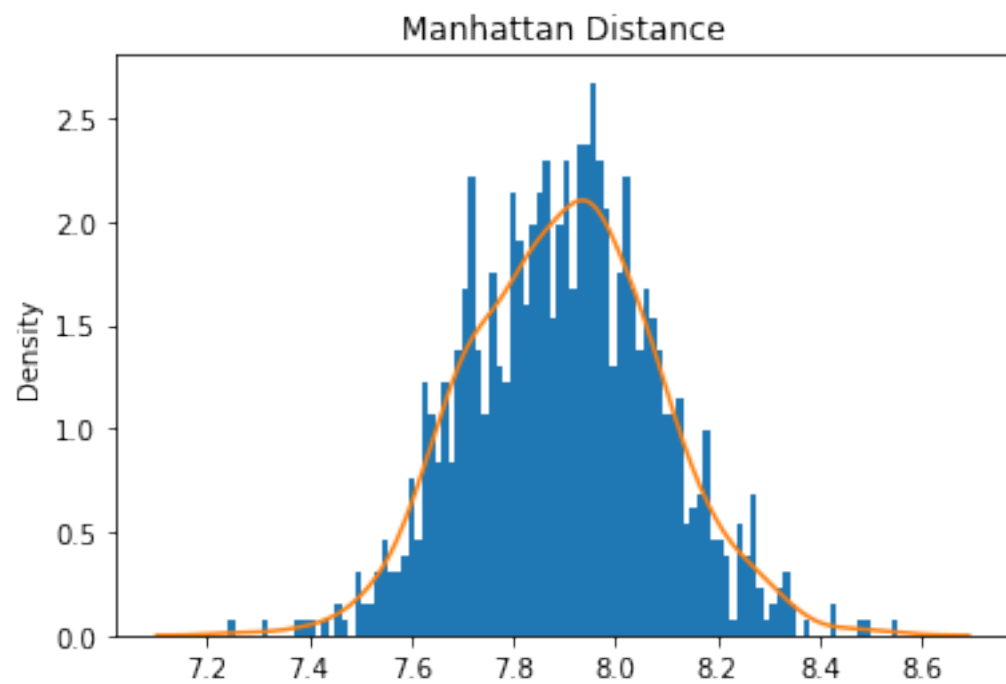
```
[14]: train_test.test_generator(generator,real_dataset,device)
```



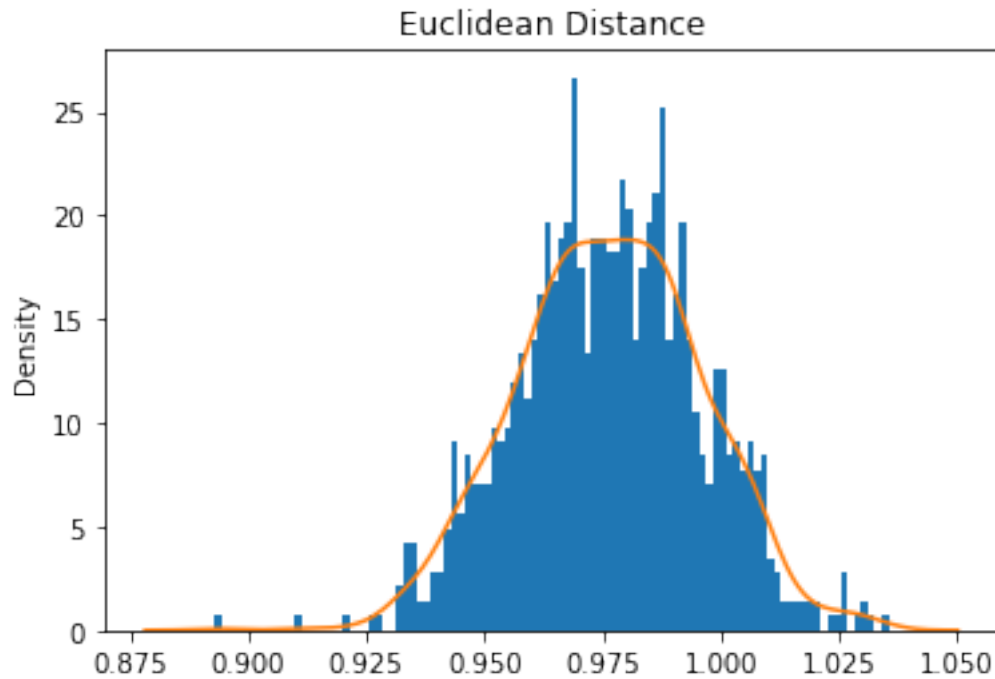
Mean Square Error: 0.009537023074622517



Mean Absolute Error: 0.07899878918020986



Mean Manhattan Distance: 7.899878918020986



Mean Euclidean Distance: 7.899878918020986

4 ABC GAN Model

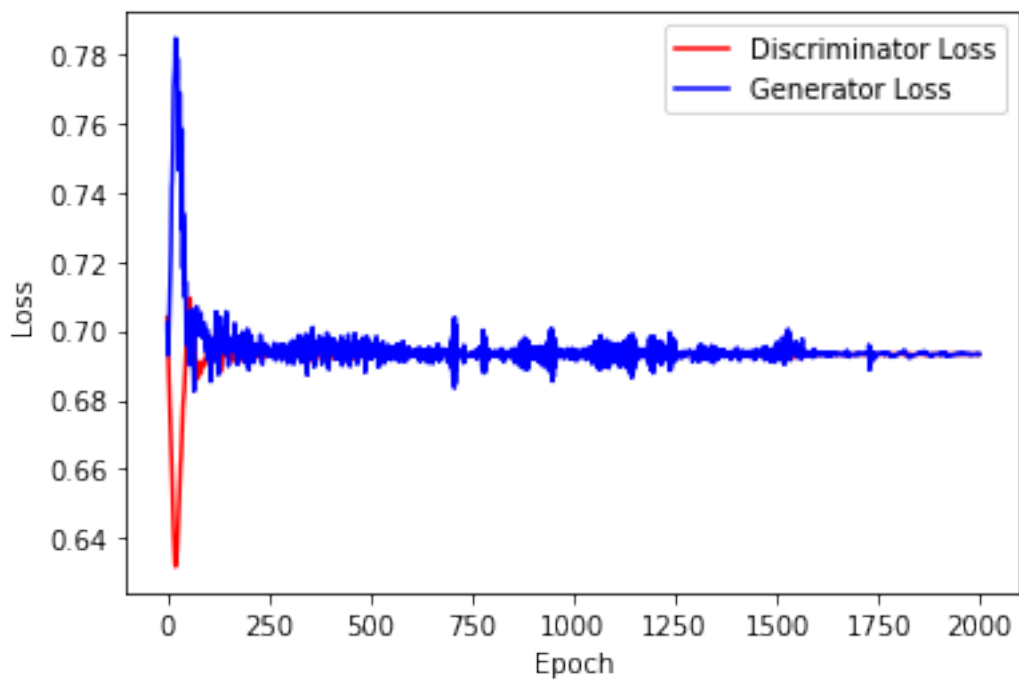
Training the network

```
[15]: gen = Generator(n_features+2)
      disc = Discriminator(n_features+2,hidden_nodes)

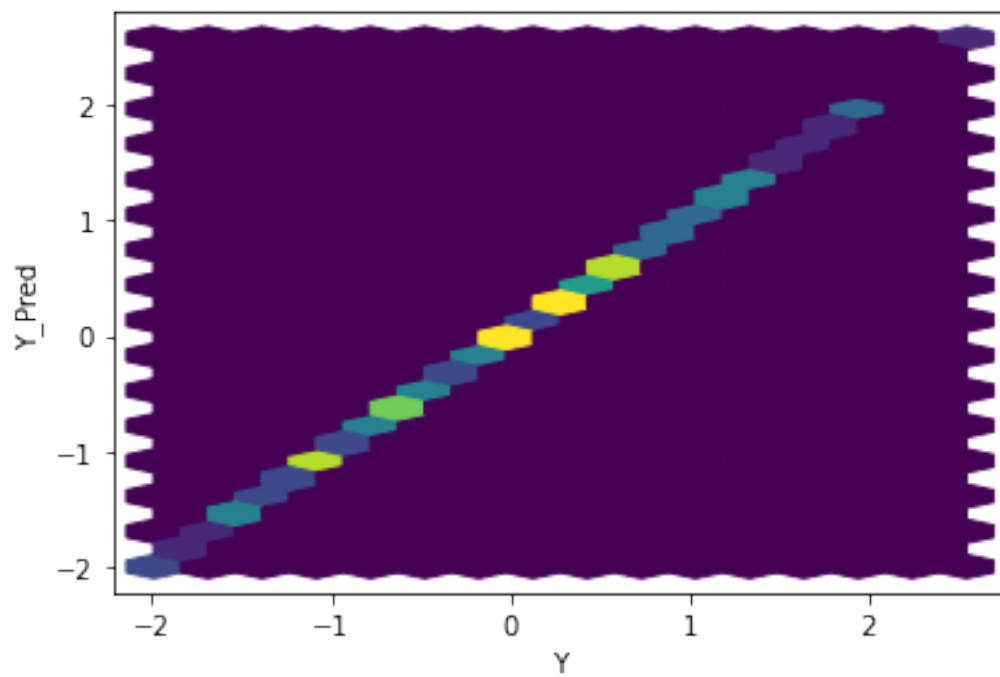
      criterion = torch.nn.BCEWithLogitsLoss()
      gen_opt = torch.optim.Adam(gen.parameters(), lr=0.01, betas=(0.5, 0.999))
      disc_opt = torch.optim.Adam(disc.parameters(), lr=0.01, betas=(0.5, 0.999))

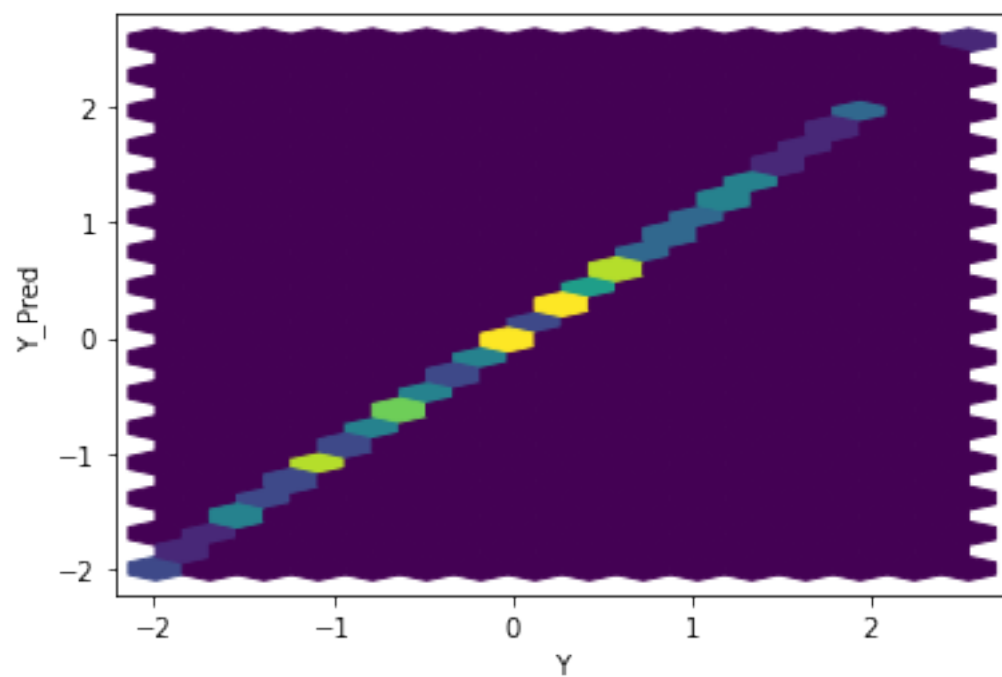
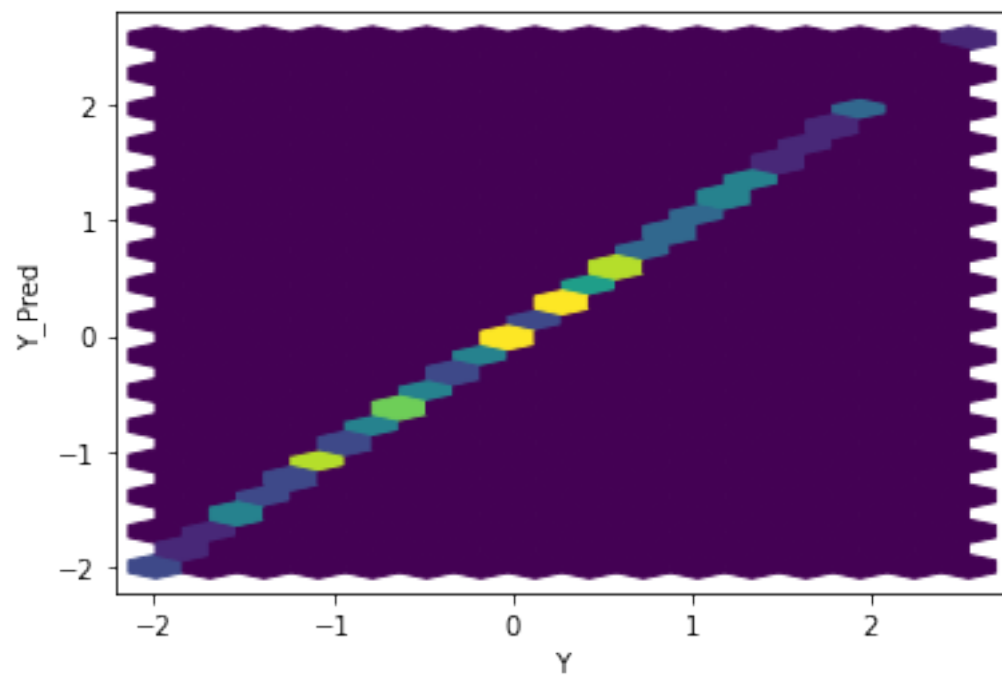
[16]: n_epoch_abc = 2000
      batch_size = sample_size//2

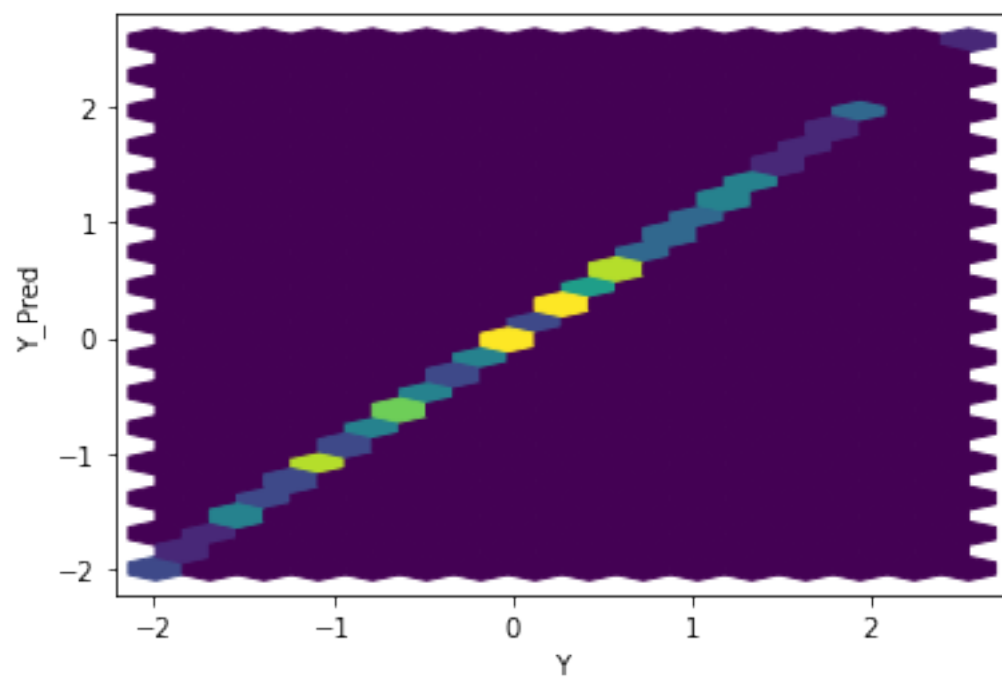
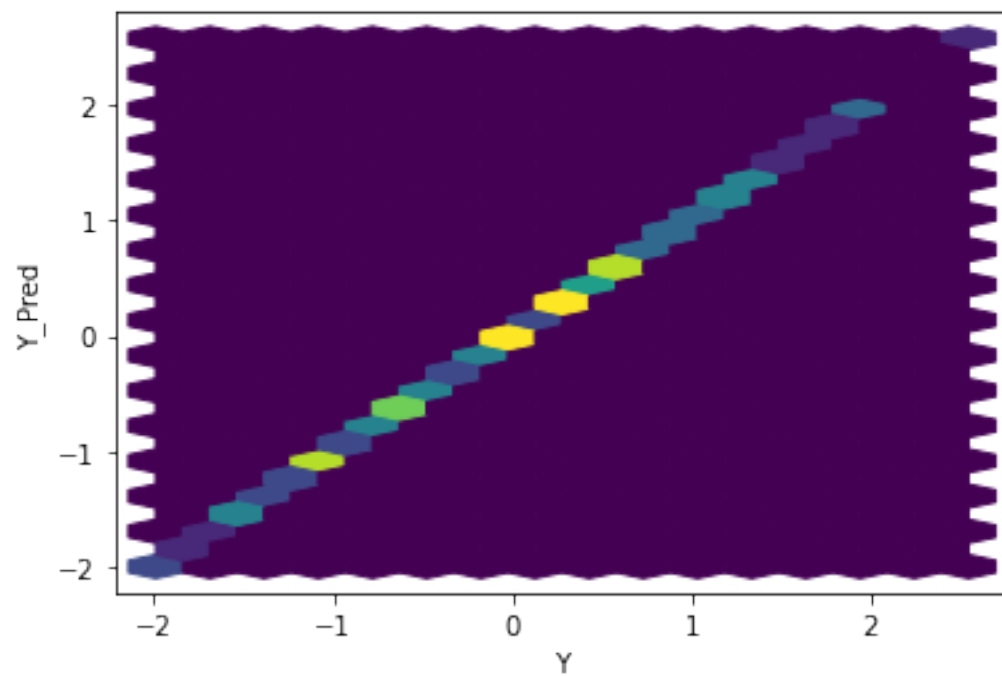
[17]: ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,
      ↪batch_size, n_epoch_abc,criterion,coeff,mean,variance,device)
```

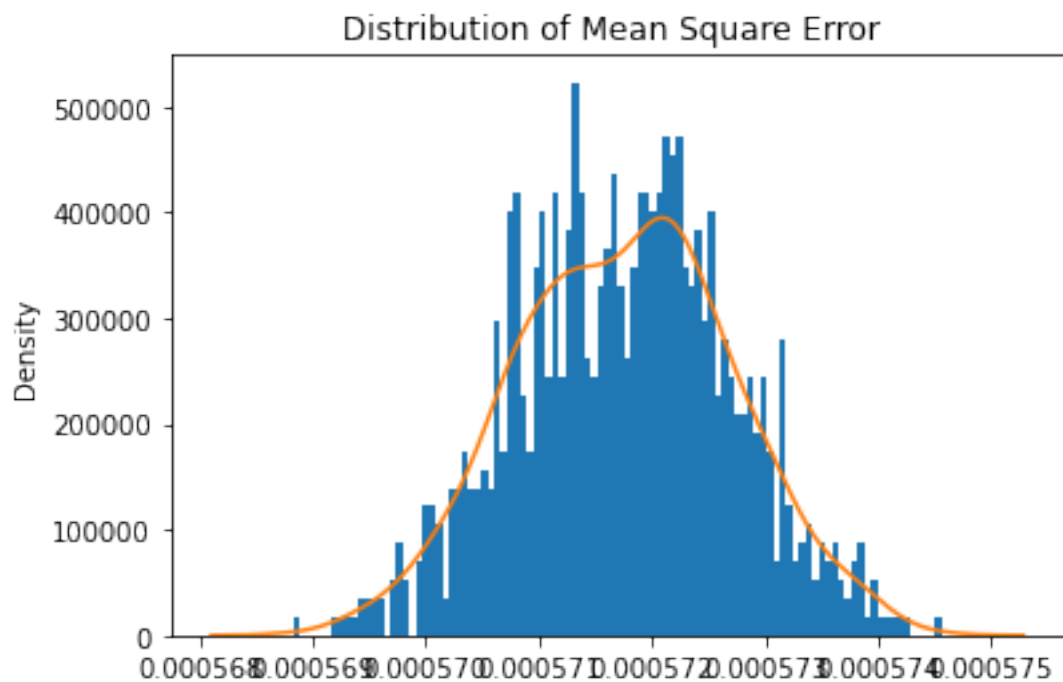


```
[18]: ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

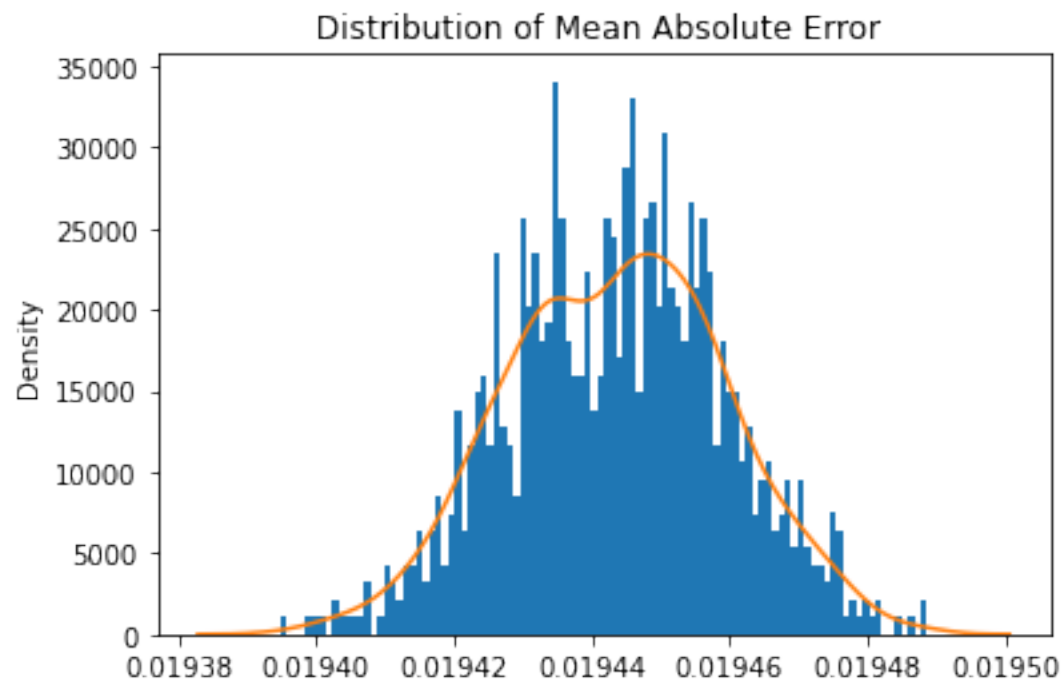




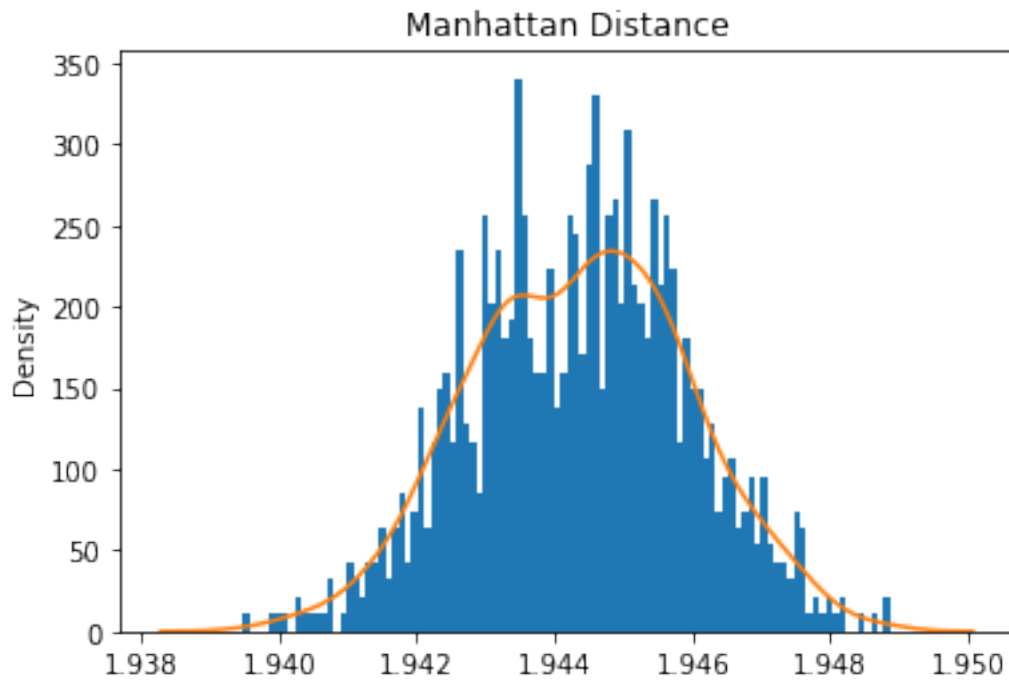




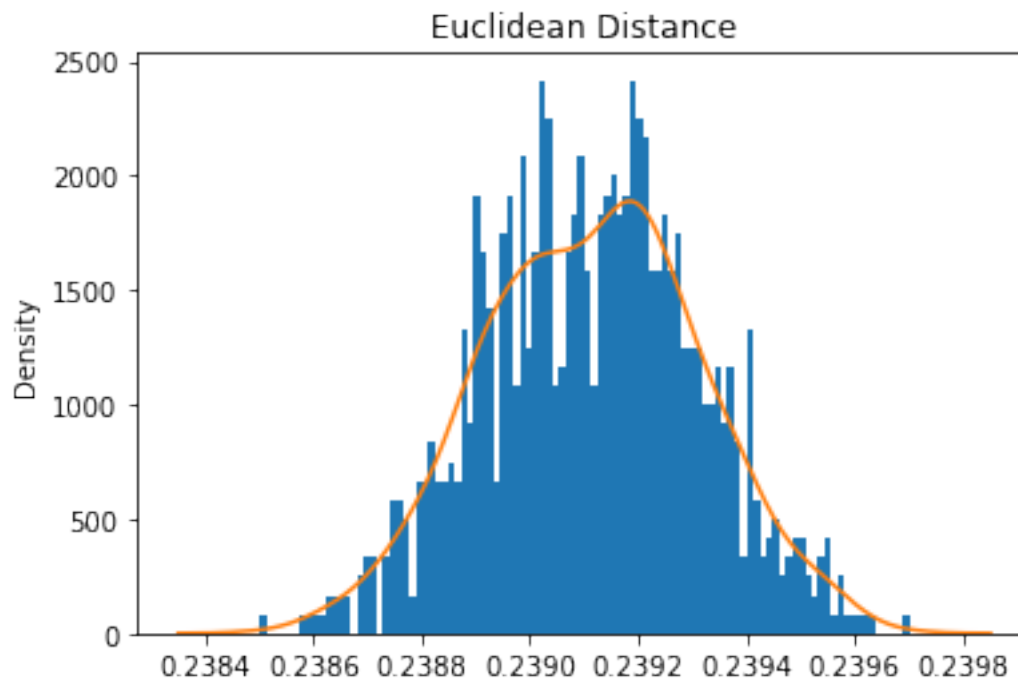
Mean Square Error: 0.0005717580698745176



Mean Absolute Error: 0.019443606442958117
Mean Manhattan Distance: 1.9443606442958117

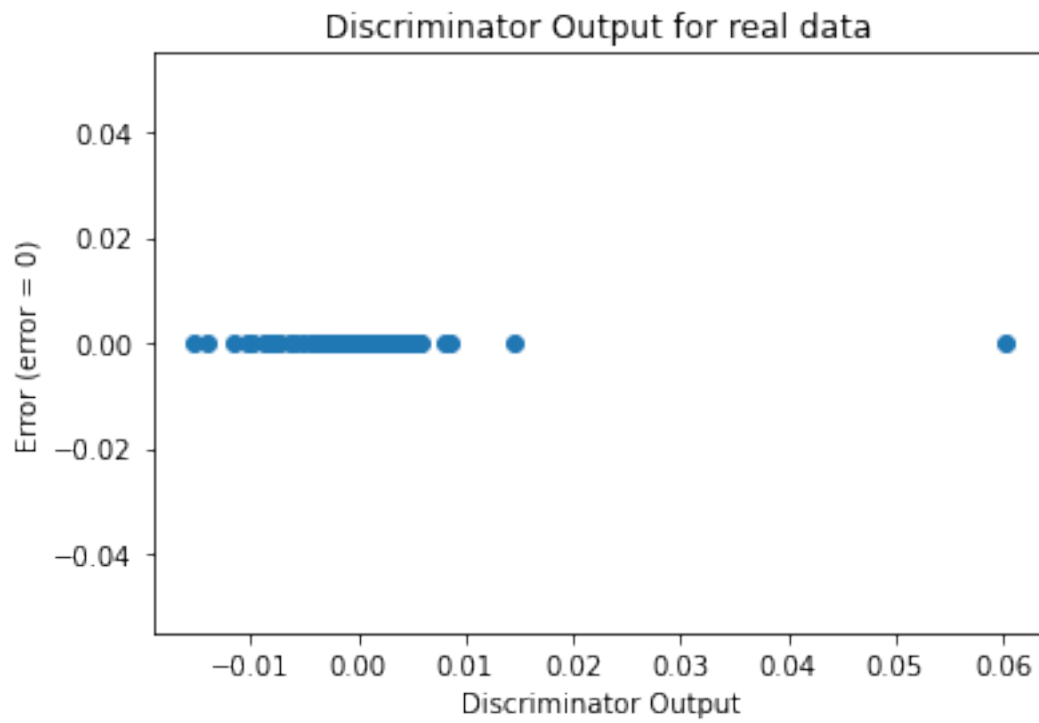


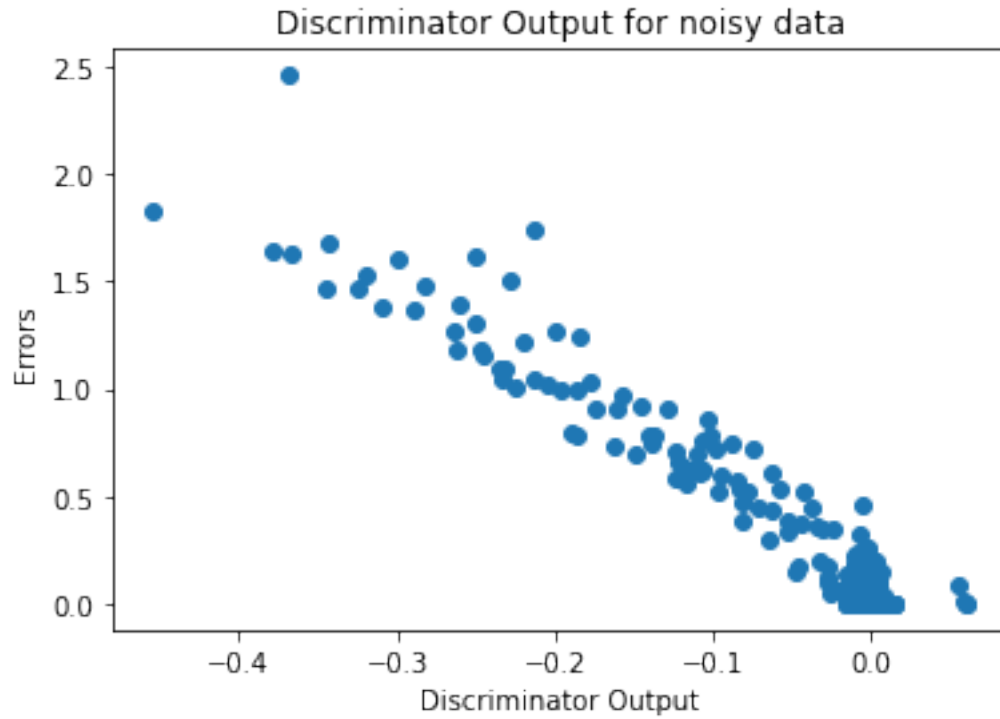
Mean Euclidean Distance: 0.2391145481478602



Sanity Checks

```
[19]: sanityChecks.discProbVsError(real_dataset,disc,device)
```





4.1 Visualization of trained GAN generator

```
[20]: for name, param in gen.named_parameters():  
      print(name,param)
```

output.weight Parameter containing:

tensor([[0.1433, 0.2494, 0.1212, 0.5385, 0.6093, 0.0281, 0.1526, 0.2844, 0.4626,
 0.1356, 0.0038, 0.0210]], requires_grad=True)

output.bias Parameter containing:

tensor([-0.1401], requires_grad=True)