

# Dataset4-Capital\_Punishment\_output\_4

October 7, 2021

```
[1]: # Parameters
std = 1
mean = 0
```

## 1 Dataset 4 - Capital Punishment

### 1.1 Parameters

```
[2]: #ABC_Generator
std = 1
mean = 1
prior = 0

#Discriminator
hidden_nodes = 7
```

### 1.2 Import Libraries and Dataset

```
[3]: import warnings
warnings.filterwarnings('ignore')
```

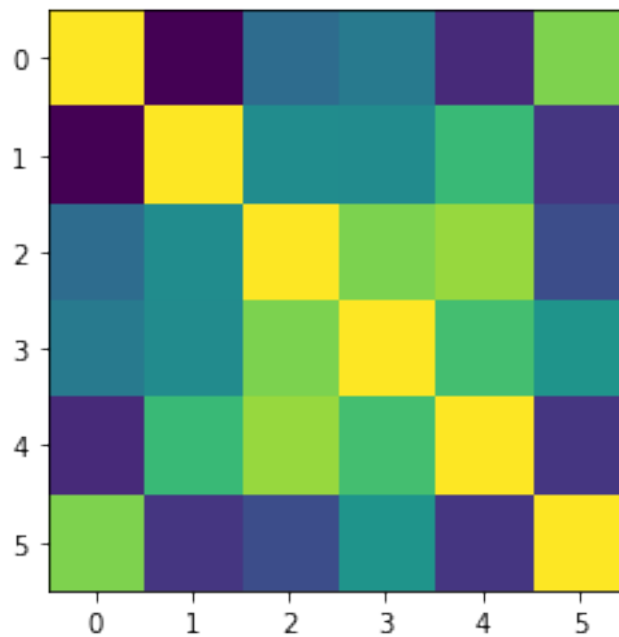
```
[4]: import cpunishDataset
import train_test
import ABC_train_test
import network
import statsModel
import performanceMetrics
import dataset
import sanityChecks

import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statistics import mean
```

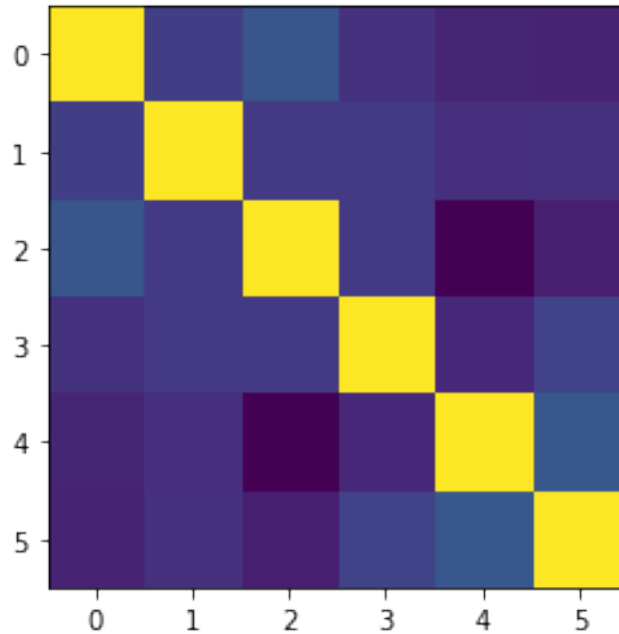
```
import pandas as pd
%matplotlib inline
```

```
[5]: #Load the dataset
X,Y = cpunishDataset.cpunish_data()
n_features = 6
```

```
[[ 1.          -0.78987947 -0.15604396 -0.05494505 -0.5731902  0.65634511]
 [-0.78987947  1.          0.08140819  0.07700775  0.42573825 -0.50722951]
 [-0.15604396  0.08140819  1.          0.64395604  0.72111026 -0.37030281]
 [-0.05494505  0.07700775  0.64395604  1.          0.46225016  0.13747769]
 [-0.5731902   0.42573825  0.72111026  0.46225016  1.          -0.50367775]
 [ 0.65634511 -0.50722951 -0.37030281  0.13747769 -0.50367775  1.          ]]
```



```
[[ 1.          0.06373626  0.16043956  0.01978022 -0.02857143 -0.03736264]
 [ 0.06373626  1.          0.04615385  0.04615385  0.00659341  0.01538462]
 [ 0.16043956  0.04615385  1.          0.04615385 -0.14725275 -0.05054945]
 [ 0.01978022  0.04615385  0.04615385  1.          -0.01538462  0.08131868]
 [-0.02857143  0.00659341 -0.14725275 -0.01538462  1.          0.16483516]
 [-0.03736264  0.01538462 -0.05054945  0.08131868  0.16483516  1.          ]]
```



## 2 Stats Model

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

No handles with labels found to put in legend.

### OLS Regression Results

```
=====
Dep. Variable:          EXECUTIONS    R-squared:                0.409
Model:                  OLS           Adj. R-squared:          -0.098
Method:                 Least Squares F-statistic:              0.8073
Date:                   Thu, 07 Oct 2021 Prob (F-statistic):       0.595
Time:                   15:03:06      Log-Likelihood:          -16.184
No. Observations:       14           AIC:                      46.37
Df Residuals:           7           BIC:                      50.84
Df Model:                6
Covariance Type:        nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
const	0	0.291	0	1.000	-0.687	0.687
x1	-0.3071	0.167	-1.835	0.109	-0.703	0.089
x2	-0.0727	0.210	-0.347	0.739	-0.569	0.423
x3	-0.0780	0.406	-0.192	0.853	-1.039	0.883
x4	0.2846	0.539	0.528	0.614	-0.989	1.558
x5	0.6329	0.622	1.017	0.343	-0.839	2.104

x6	-0.1115	1.356	-0.082	0.937	-3.318	3.095
=====						
Omnibus:	0.509	Durbin-Watson:	0.960			
Prob(Omnibus):	0.775	Jarque-Bera (JB):	0.285			
Skew:	0.314	Prob(JB):	0.867			
Kurtosis:	2.696	Cond. No.	8.10			
=====						

Notes:

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

Parameters: const 0.000000

x1 -0.307149

x2 -0.072734

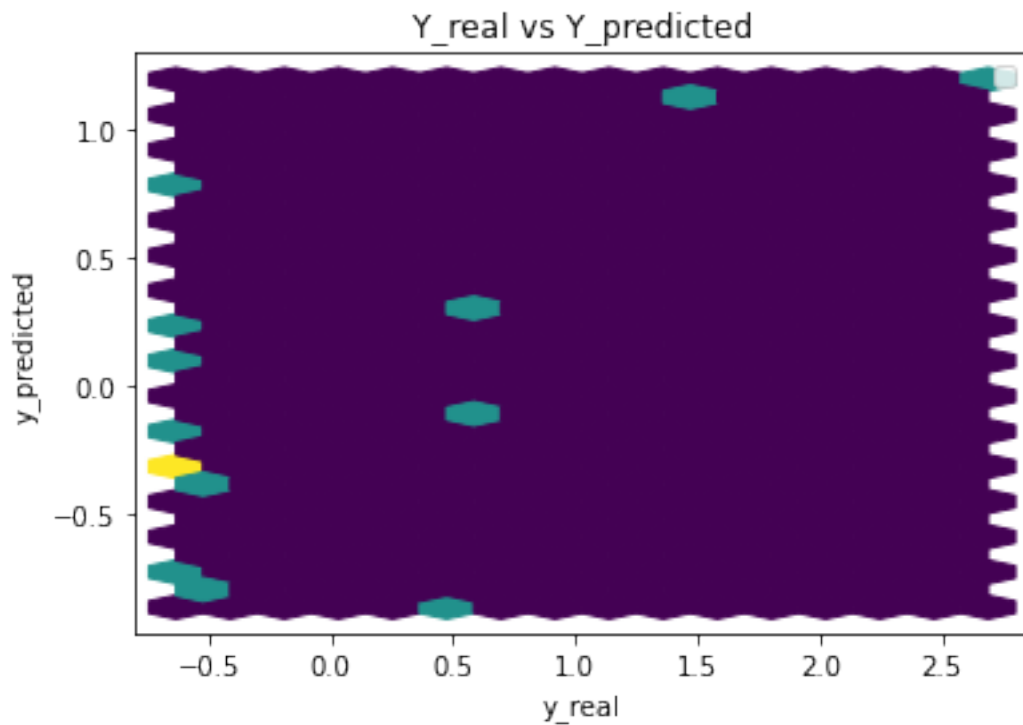
x3 -0.077971

x4 0.284622

x5 0.632853

x6 -0.111469

dtype: float64



Performance Metrics

Mean Squared Error: 0.5910184401346179

Mean Absolute Error: 0.6107560111864074

Manhattan distance: 8.550584156609704

Euclidean distance: 2.876501027617521

### 3 Generator and Discriminator Networks

#### GAN Generator

```
[7]: class Generator(nn.Module):
    def __init__(self, n_input):
        super().__init__()
        #Input to Output Layer Linear Transformation
        self.output = nn.Linear(n_input, 1)

    def forward(self, x):
        #Pass the input tensor through the operations
        x = self.output(x)
        return x
```

#### GAN Discriminator

```
[8]: class Discriminator(nn.Module):
    def __init__(self, n_input, hiddenNodes):
        super().__init__()
        self.hidden = nn.Linear(n_input, hiddenNodes)
        self.output = nn.Linear(hiddenNodes, 1)
        #Define LeakyRelu Activation and sigmoid output
        self.sigmoid = nn.Sigmoid()
        self.leakyRelu = nn.LeakyReLU()

    def forward(self, x):
        #Pass the input tensor through the operations
        x = self.hidden(x)
        x = self.leakyRelu(x)
        x = self.output(x)
        x = self.sigmoid(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  $\beta_i^*$ s are coefficients obtained from stats model

Parameters :  $\mu$  and  $\sigma^*$

```
[9]: 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

```

## 4 GAN Model

```

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

```

```

[11]: 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))

```

```

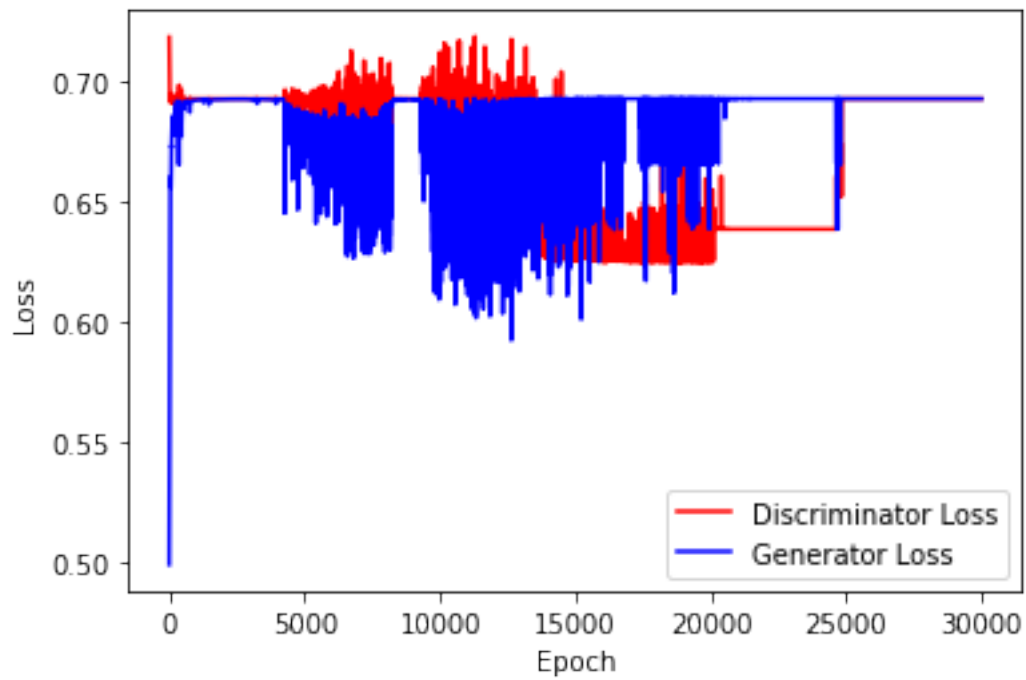
[12]: sample_size = len(real_dataset)
n_epochs = 30000
batch_size = sample_size

```

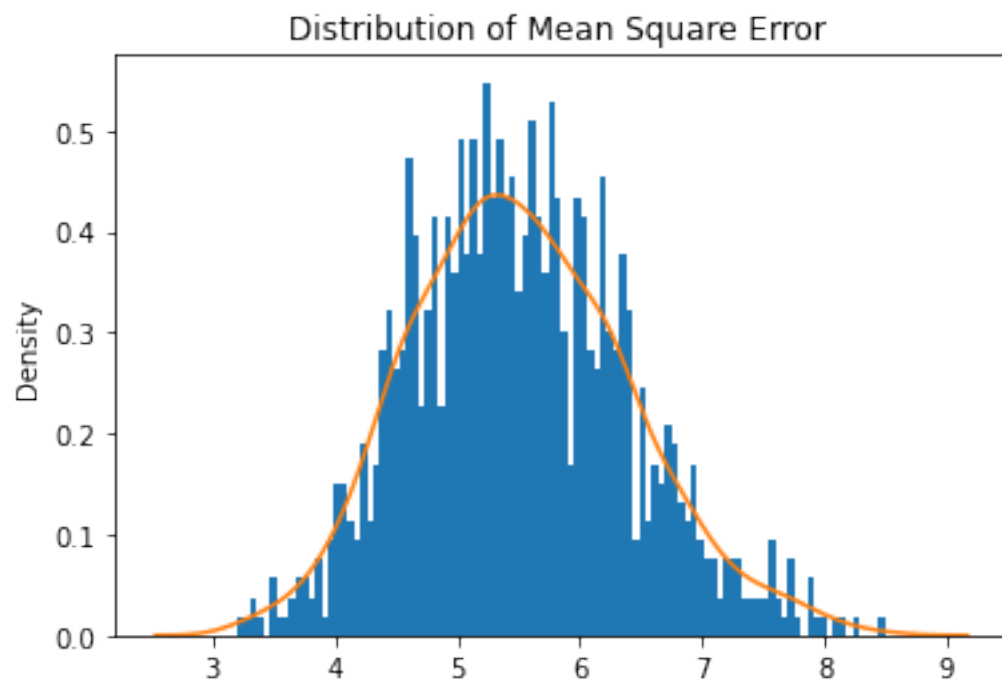
```

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

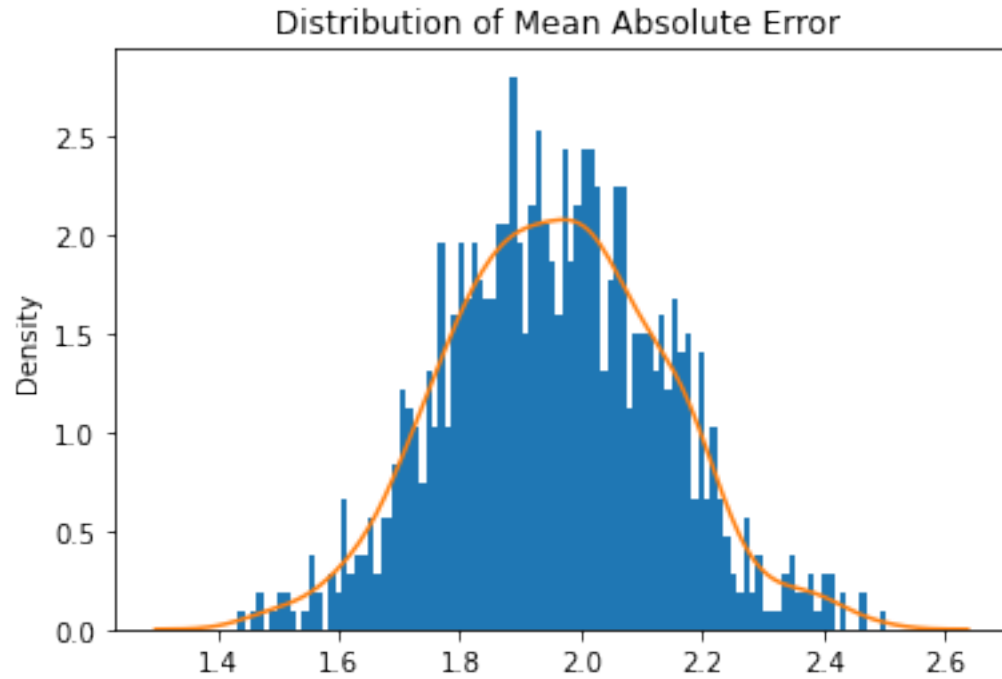
```



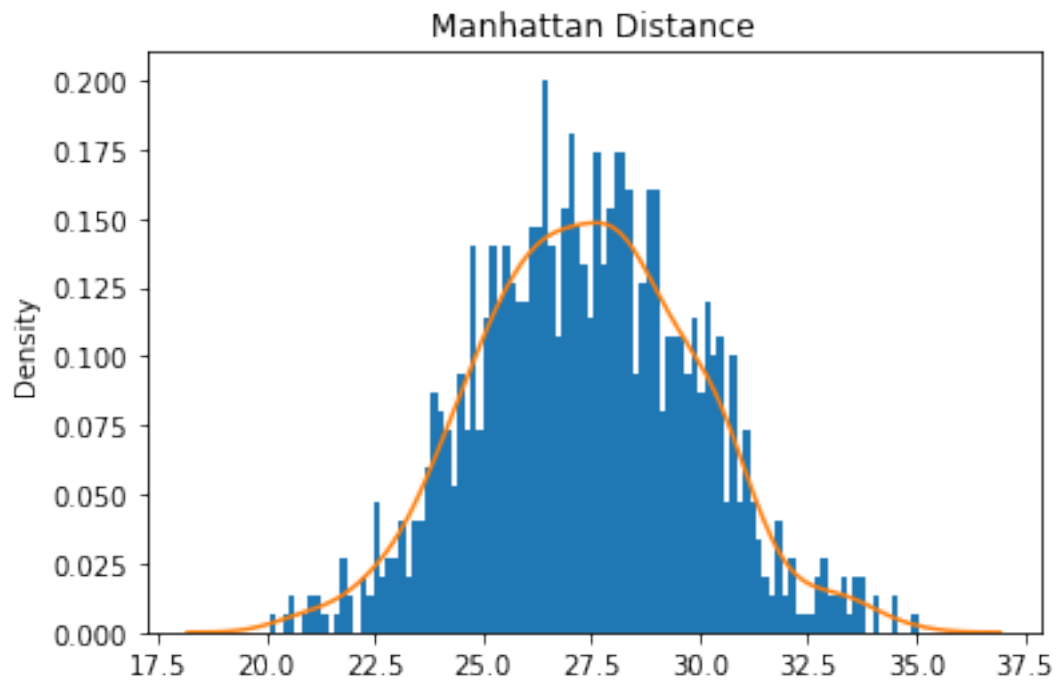
```
[14]: train_test.test_generator(generator,real_dataset,device)
```



Mean Square Error: 5.5130342450474705

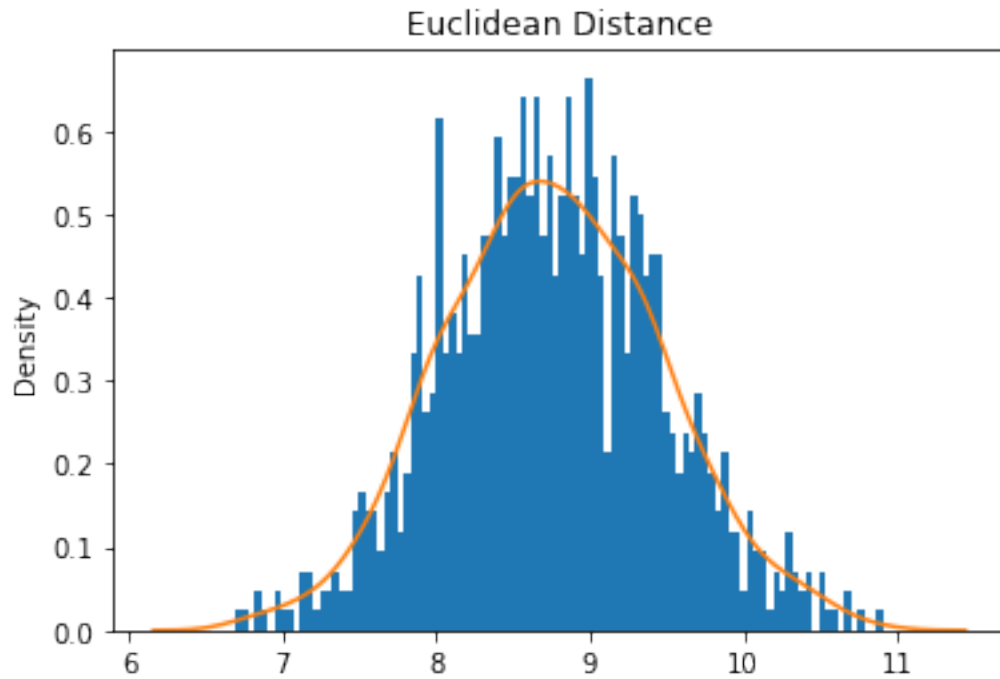


Mean Absolute Error: 1.9563203257481967





Mean Manhattan Distance: 27.388484560474755



Mean Euclidean Distance: 27.388484560474755

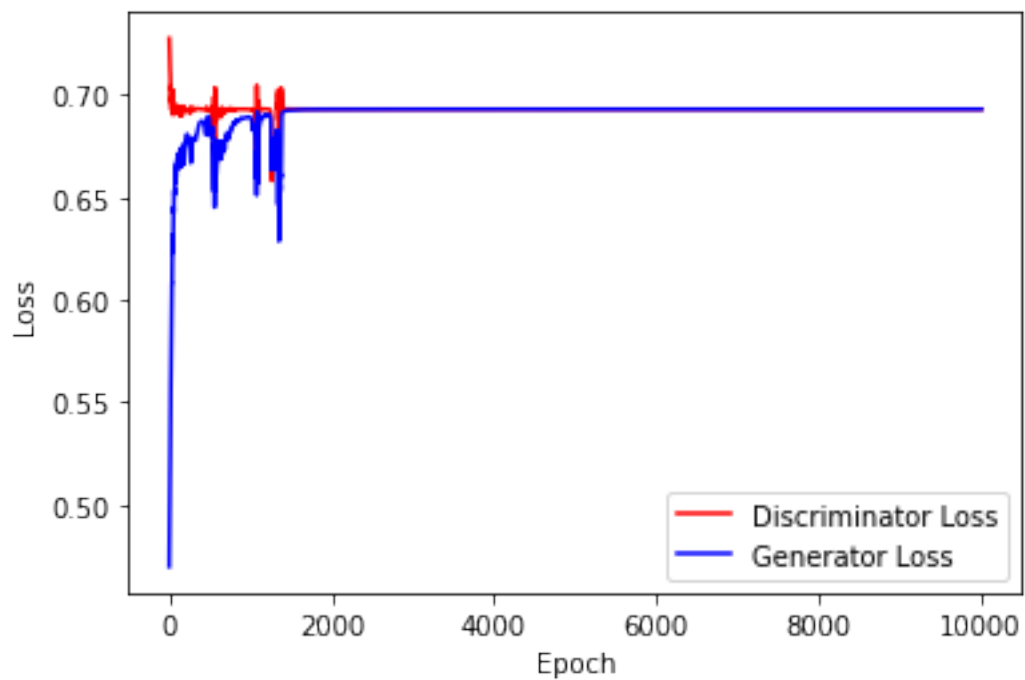
## 5 ABC GAN Model

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

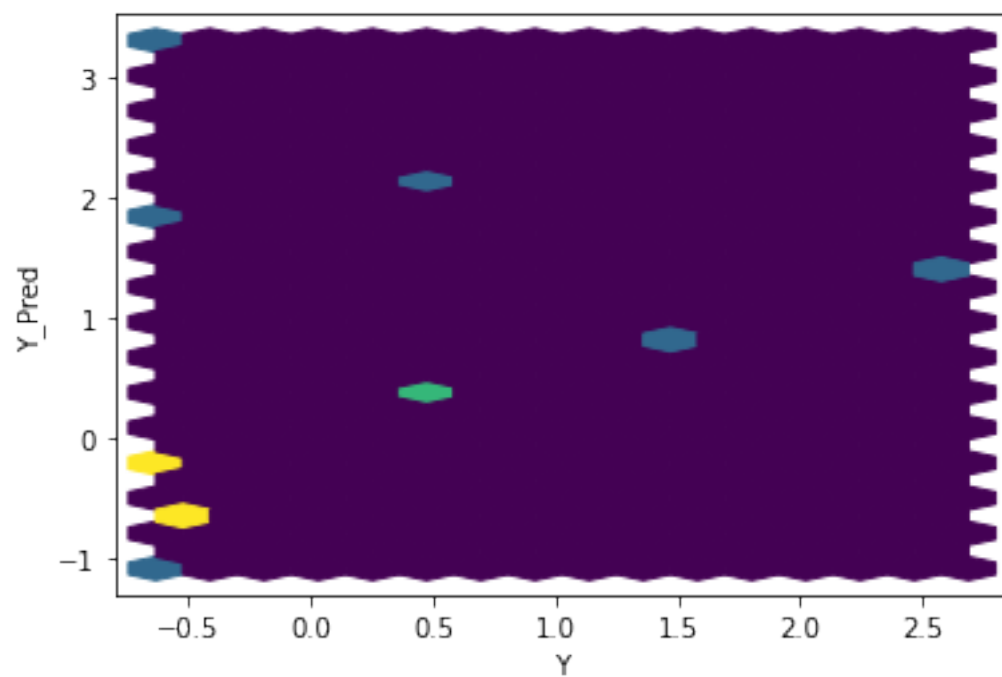
      criterion = torch.nn.BCEWithLogitsLoss()
      gen_opt = torch.optim.Adam(gen.parameters(), lr=0.01, betas=(0.5, 0.999))
      disc_opt = torch.optim.Adam(disc.parameters(), lr=0.01, betas=(0.5, 0.999))
```

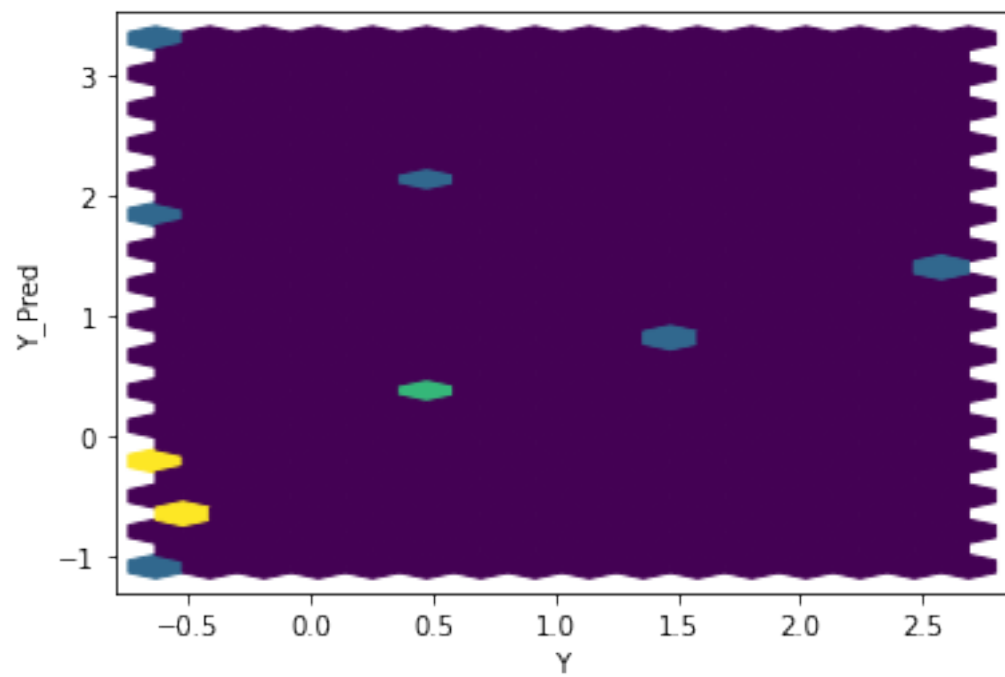
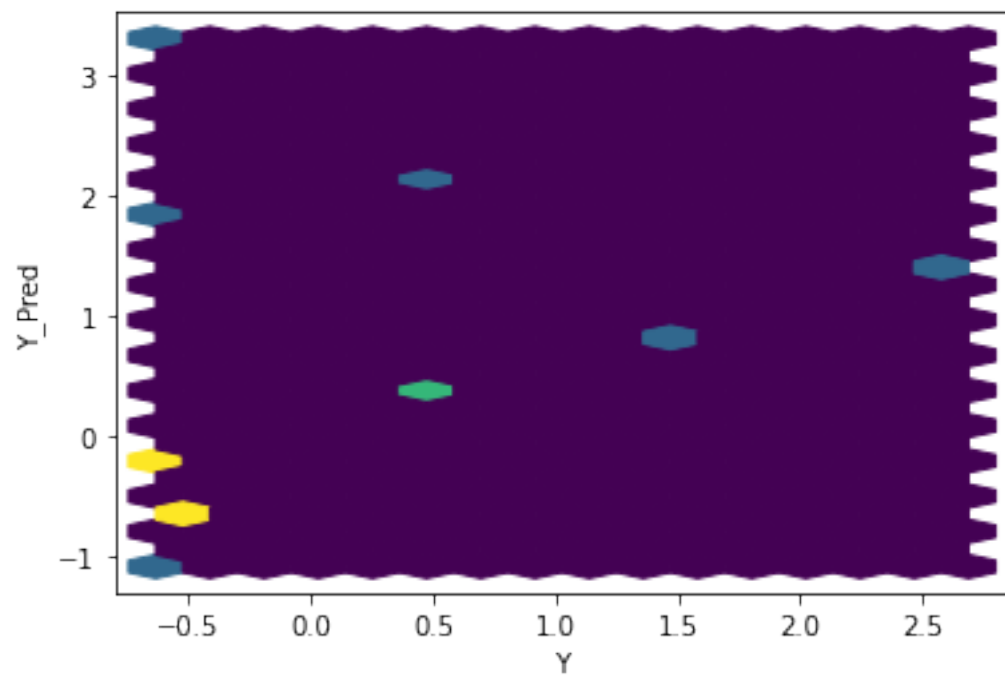
```
[16]: n_epoch_abc = 10000
      batch_size = sample_size
```

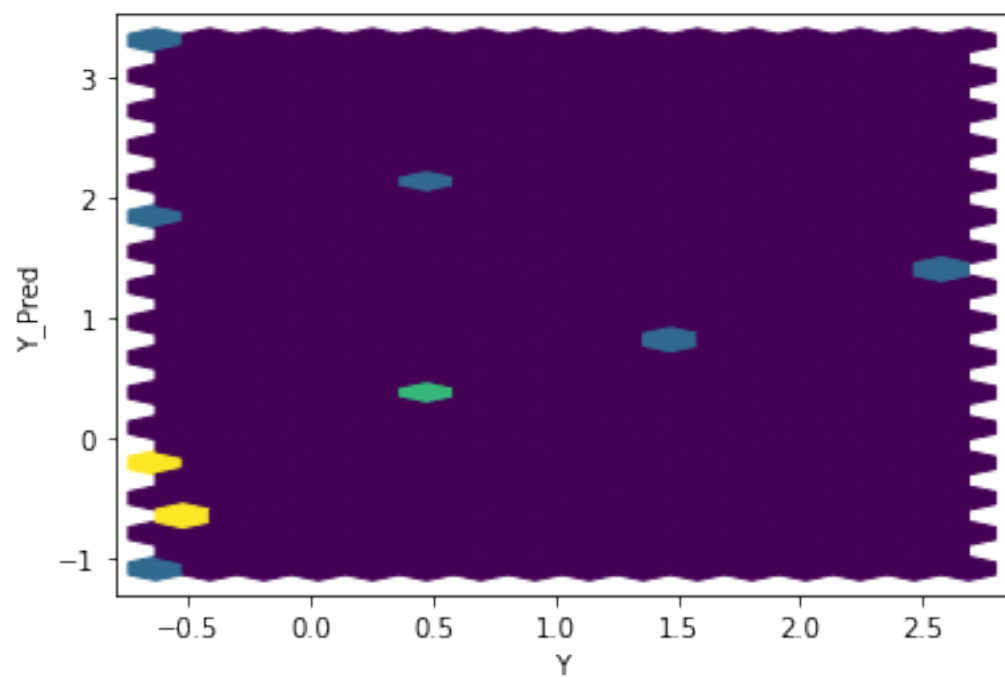
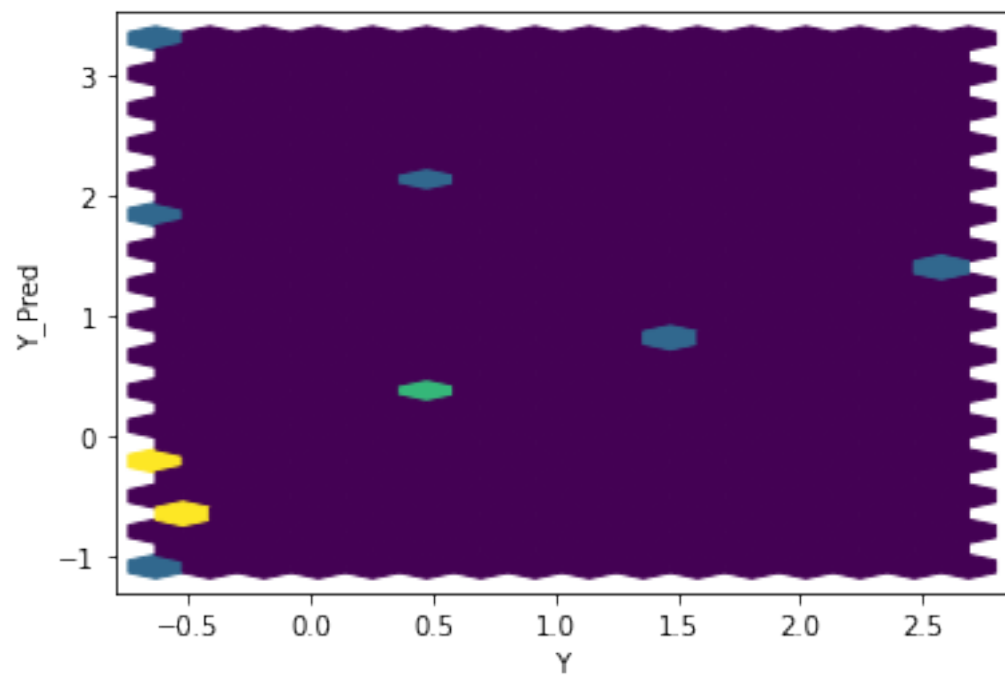
```
[17]: ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,
      ↪ batch_size, n_epoch_abc,criterion,coeff,mean,std,device)
```

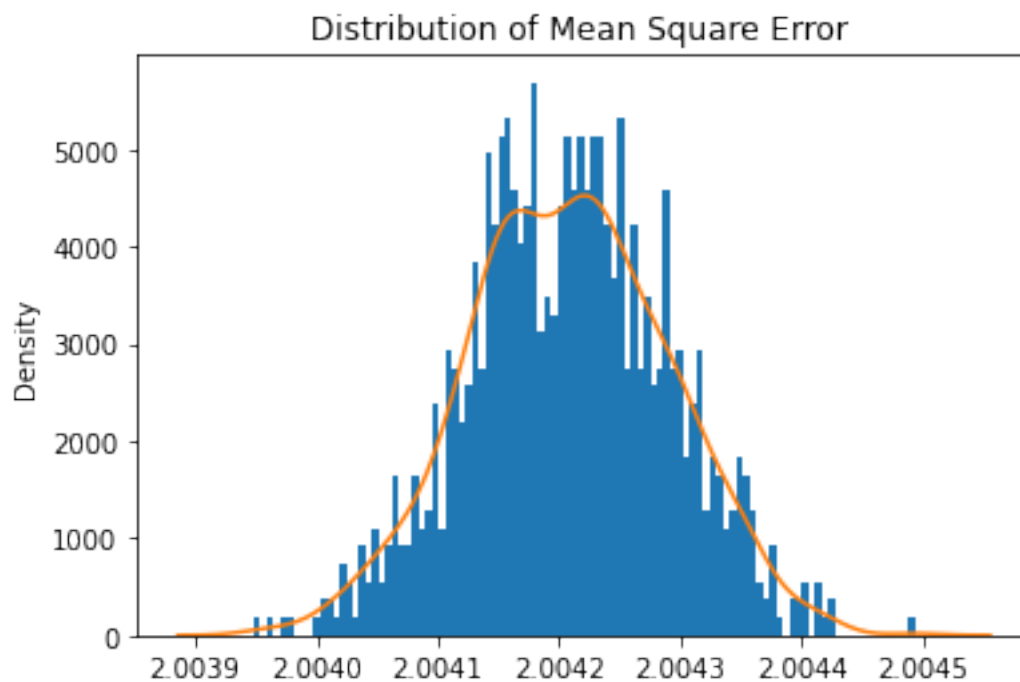


```
[18]: ABC_train_test.test_generator(gen,real_dataset,coeff,mean,std,device)
```

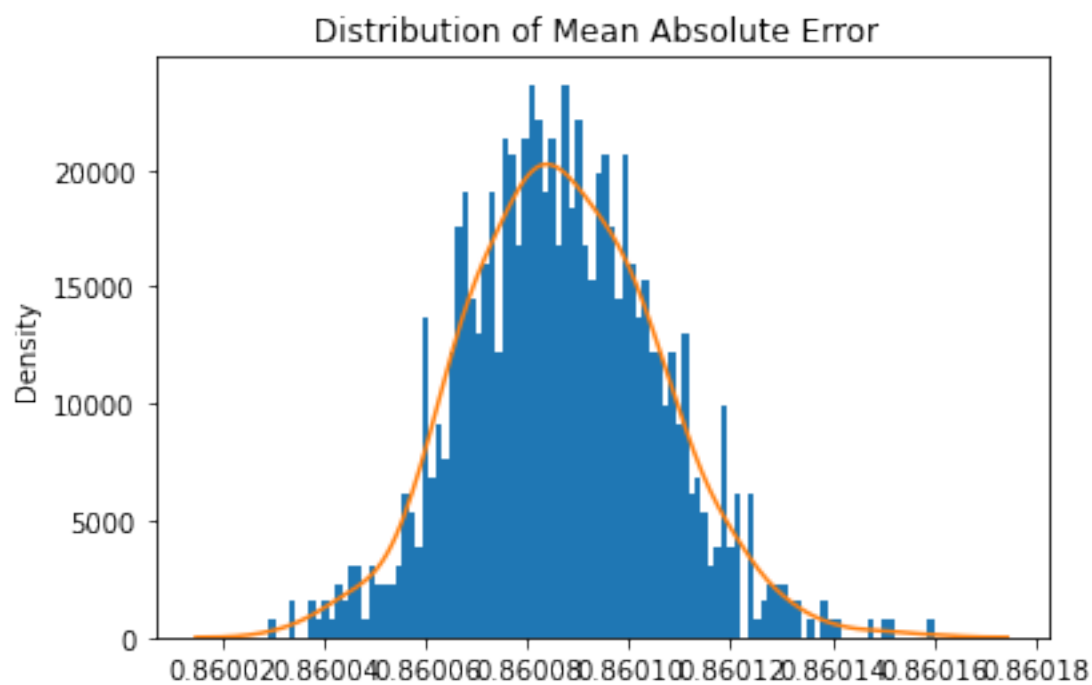




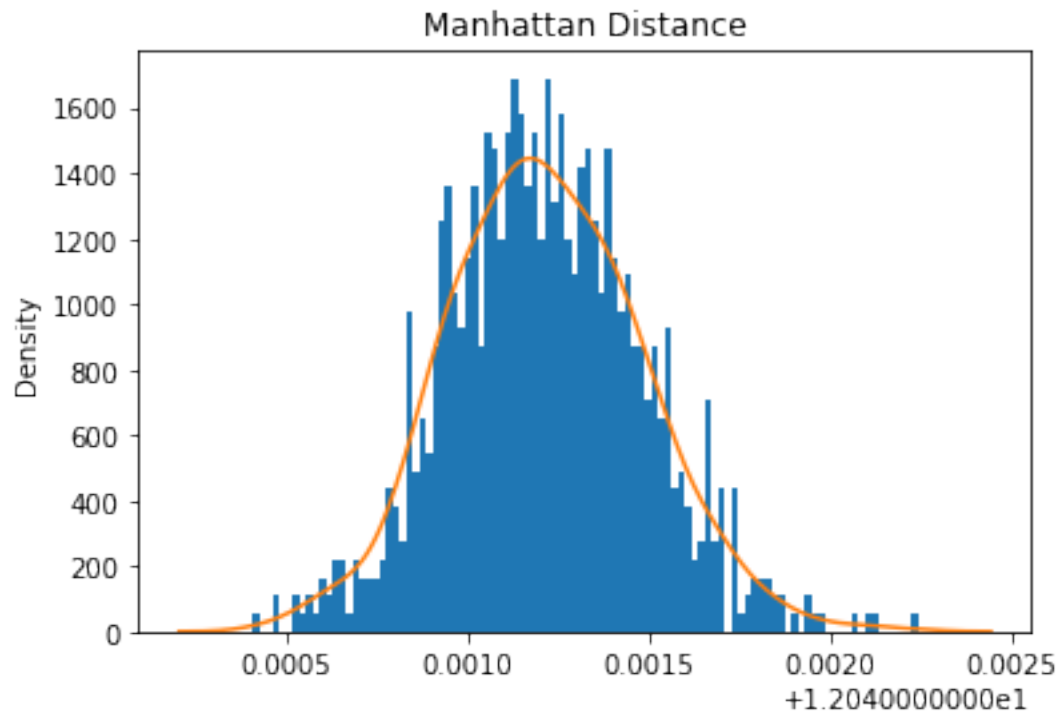




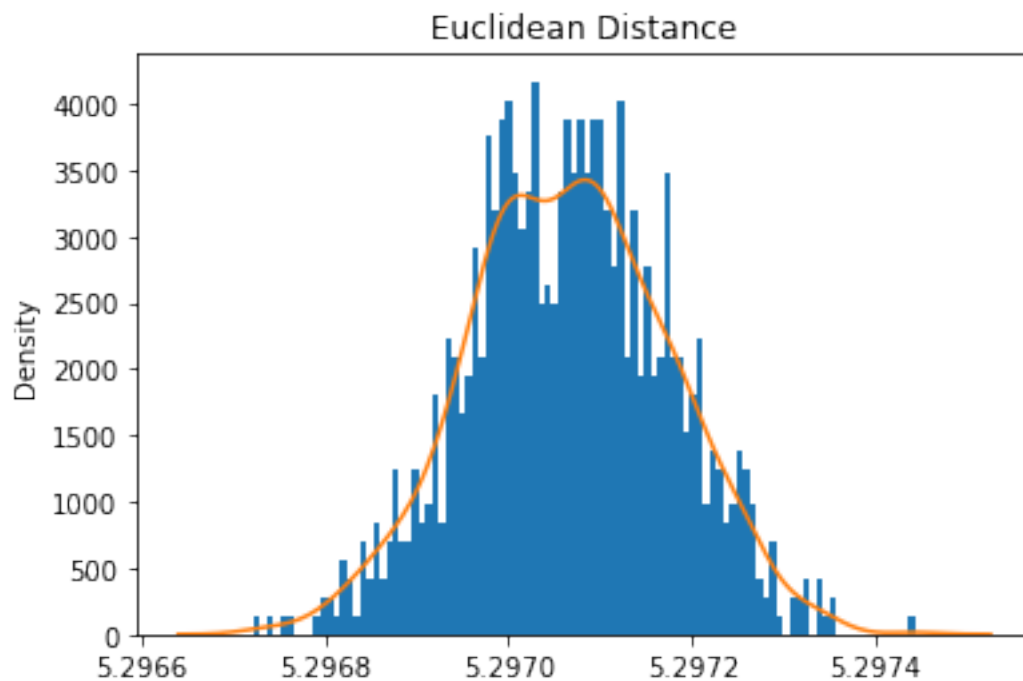
Mean Square Error: 2.004207386437904



Mean Absolute Error: 0.8600866886994668  
Mean Manhattan Distance: 12.041213641792536

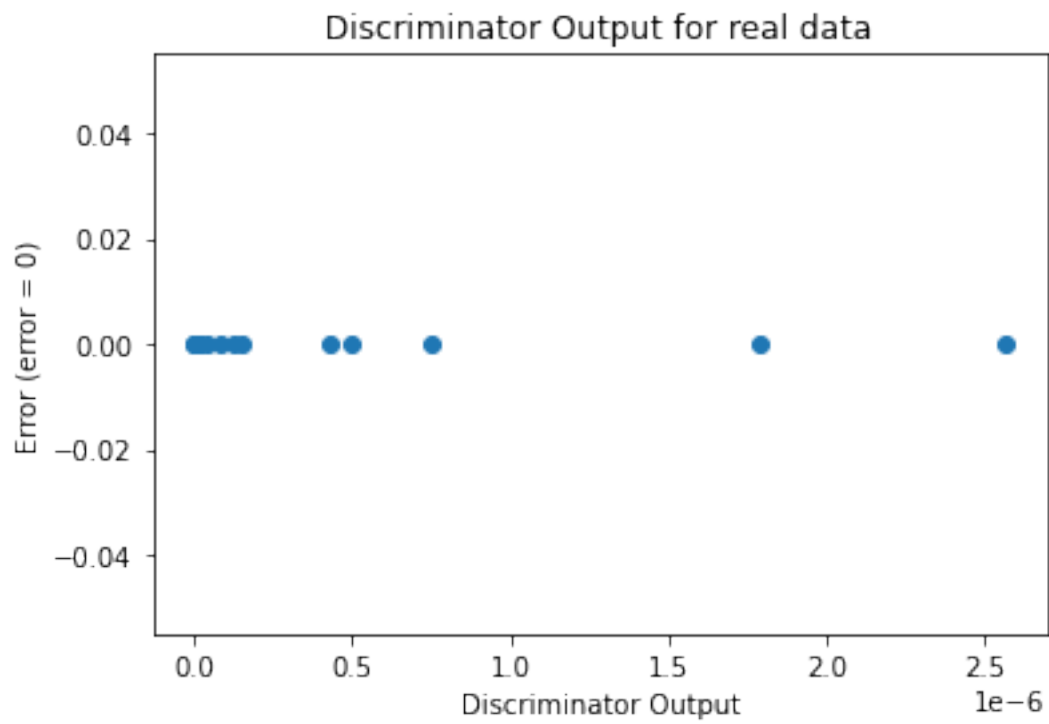


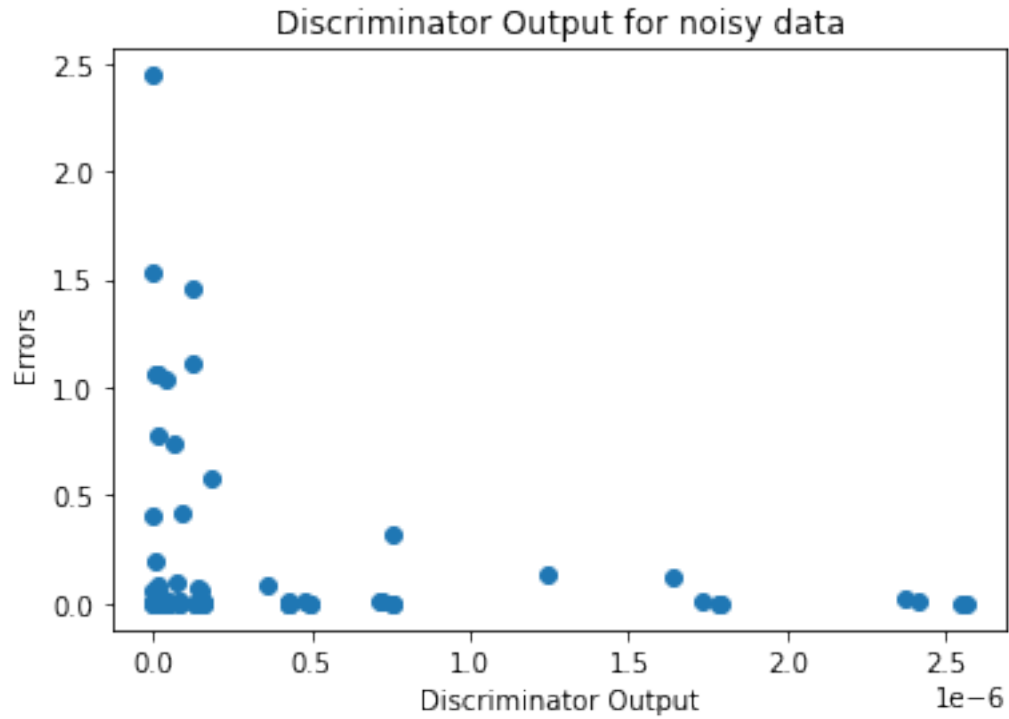
Mean Euclidean Distance: 5.297065545963925



## Sanity Check

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





### Visualisation of Trained GAN Generator

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

output.weight Parameter containing:

tensor([[ 1.8453e-02, -5.0662e-01, 4.4187e-01, 5.1976e-01, 7.4512e-01,  
 -4.5813e-01, -9.2004e-01, -3.1576e-05]], requires\_grad=True)

output.bias Parameter containing:

tensor([0.4823], requires\_grad=True)