Dataset1-Regression_output_0

October 19, 2021

1 Dataset 1 - Regression

1.1 Experiment Details

The aim of the experiment is to verify if the: 1. ABC_GAN model corrects model misspecification 2. ABC_GAN model performs better and converges faster than a simple C-GAN model

In the experiment we predict the distribution that represents the real data and simulate realistic fake data points using statistical mode, C-GAN and ABC-GAN model with 3 priors. We analyze and compare their performance using metrics like mean squared error, mean absolute error , manhattan distance and euclidean distance between y_{real} and y_{pred}

The models are as follows:

1. The statistical model assumes the distribution $Y = \beta X + \mu$ where $\mu \sim N(0, 1)$
2. The Conditional GAN consists of
    1. Generator with 2 hidden layers with 100 nodes each and ReLu activation.
    2. Discriminator with 2 hidden layers with 25 and 50 nodes and ReLu activation. We use Adam's optimser and BCE Logit Loss to train the model. The input to the Generator of the GAN is (x,e) where x are the features and $e \sim N(0, 1)$. The discriminator output is linear.
3. The ABC GAN Model consists of
    1. ABC generator is defined as follows:
        1. $Y = 1 + \beta_1 x_1 + \beta_2 x_2 + \beta_2 x_3 + ... + \beta_n x_n + N(0, \sigma)$ where $\sigma = 0.1$
        2. $\beta_i \sim N(0, \sigma^*)$ when $\mu = 0$ else $\beta_i \sim N(\beta_i^*, \sigma^*)$ where $\beta_i^* s$ are coefficients obtained from statistical model
        3. σ^* takes the values 0.01,0.1 and 1

    2. C-GAN network is as defined above. However the input to the Generator of the GAN is (x, y_{abc}) where y_{abc} is the output of the ABC Generator.

1.2 Import Libraries

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
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
```

1.3 Parameters

General Parameters

1. Number of Samples
2. Number of features

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)

[3]:
```
n_features = 10
n_samples= 100

#ABC Generator Parameters
mean = 1
variance = 0.001
```

[4]:
```
# Parameters
n_samples = 10
n_features = 10
mean = 1
variance = 1
```

1.4 Dataset

Generate a random regression problem

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

[5]:
```
X,Y = regressionDataset.regression_data(n_samples,n_features)
```

```
         X1        X2        X3        X4        X5        X6        X7  \
0  0.578186 -0.745305 -0.409734  0.844205 -0.002063  0.525307  0.432641
1  2.477230 -1.276228  0.106391 -0.223284  0.512581  0.260654 -1.105014
2  1.582062  0.773356  0.045144  1.339291  1.184795  0.417705  0.319639
3 -0.617066 -0.467011 -0.099947  1.241433 -0.825883 -0.455972  0.946513
4 -1.521098  1.060374  0.359814 -0.313856 -0.196631  0.698228 -0.610070

         X8        X9       X10           Y
```

```
0 -0.325435  0.424304  0.558874   18.729500
1 -1.026321  0.450231  0.834778  103.918801
2 -1.195508  0.389476  0.182137  265.752306
3  0.167525  0.322680 -0.395029  -77.533465
4 -1.315396 -1.982429  0.177500  -50.083371
```

1.5 Stats Model

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

```
No handles with labels found to put in legend.
```

OLS Regression Results

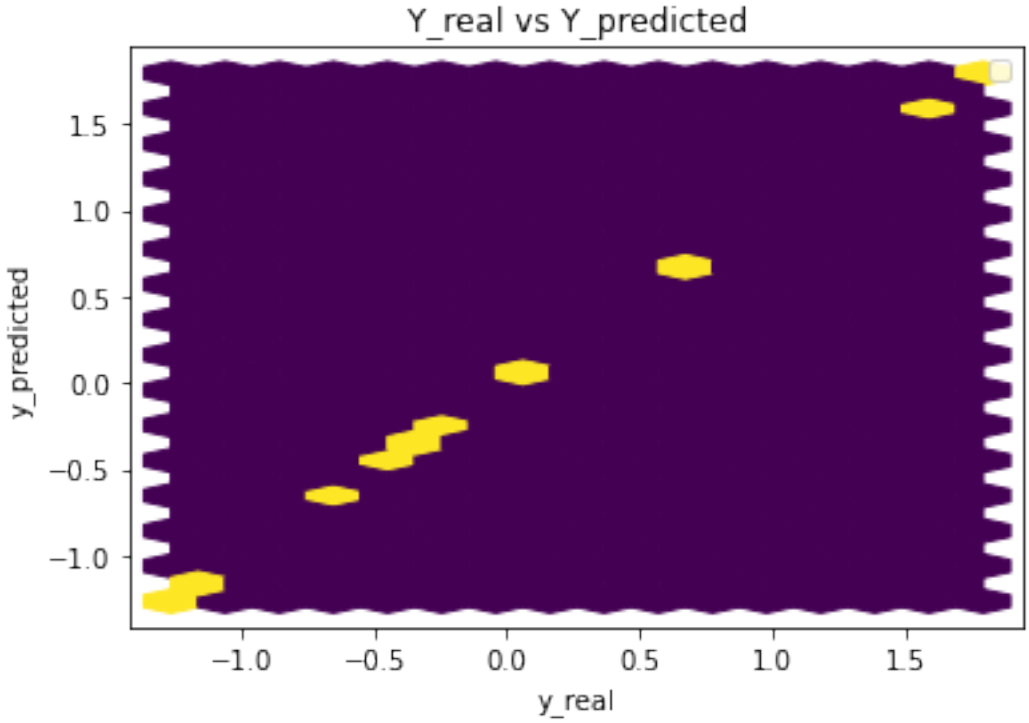
Dep. Variable:	Y	R-squared:	1.000
Model:	OLS	Adj. R-squared:	nan
Method:	Least Squares	F-statistic:	nan
Date:	Tue, 19 Oct 2021	Prob (F-statistic):	nan
Time:	22:38:09	Log-Likelihood:	332.37
No. Observations:	10	AIC:	-644.7
Df Residuals:	0	BIC:	-641.7
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.441e-16	inf	0	nan	nan	nan
x1	0.5318	inf	0	nan	nan	nan
x2	0.4302	inf	0	nan	nan	nan
x3	0.2424	inf	0	nan	nan	nan
x4	0.1620	inf	0	nan	nan	nan
x5	0.1306	inf	0	nan	nan	nan
x6	-0.0298	inf	-0	nan	nan	nan
x7	0.1533	inf	0	nan	nan	nan
x8	-0.1565	inf	-0	nan	nan	nan
x9	0.1270	inf	0	nan	nan	nan
x10	0.3218	inf	0	nan	nan	nan

Omnibus:	2.217	Durbin-Watson:	1.893
Prob(Omnibus):	0.330	Jarque-Bera (JB):	0.962
Skew:	0.755	Prob(JB):	0.618
Kurtosis:	2.832	Cond. No.	32.5

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The input rank is higher than the number of observations.
Parameters:  const    4.440892e-16
```

```
x1       5.318420e-01
x2       4.302306e-01
x3       2.423668e-01
x4       1.620454e-01
x5       1.305914e-01
x6      -2.975714e-02
x7       1.533023e-01
x8      -1.565390e-01
x9       1.270455e-01
x10      3.218287e-01
dtype: float64
```




```
Performance Metrics
Mean Squared Error: 7.918075780359243e-31
Mean Absolute Error: 7.896461262646426e-16
Manhattan distance: 7.896461262646426e-15
Euclidean distance: 2.8139075642883584e-15
```

1.6 Common Training Parameters (GAN & ABC_GAN)

[7]:
```
n_epochs = 5000
error = 0.001
batch_size = n_samples//2
```

1.7 GAN Model

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

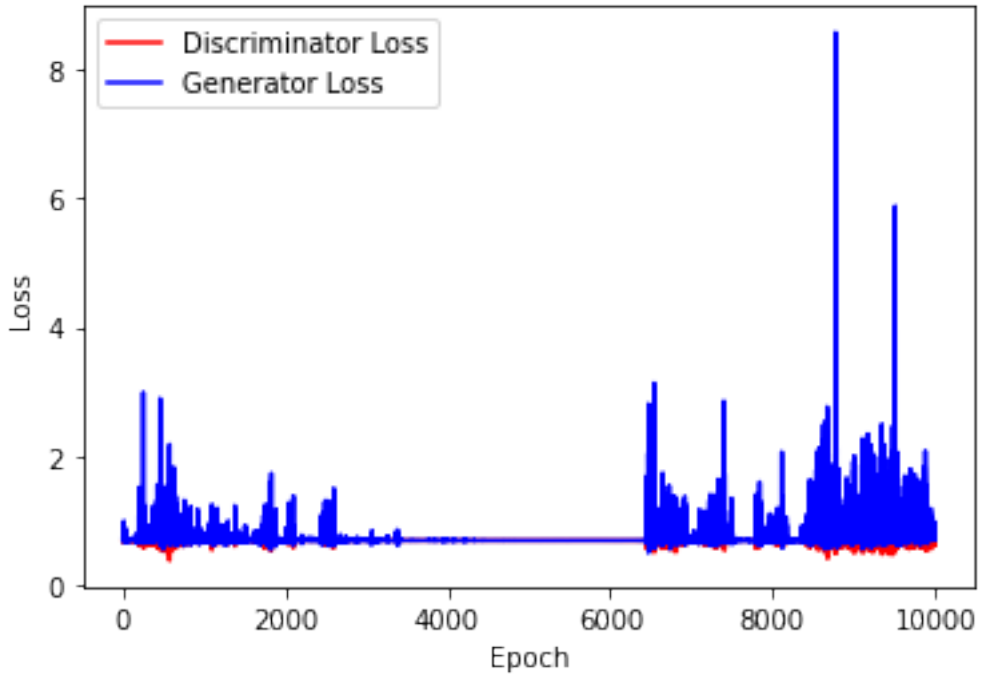
Training GAN for n_epochs number of epochs

[9]:
```
generator = network.Generator(n_features+2)
discriminator = network.Discriminator(n_features+2)

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))
```

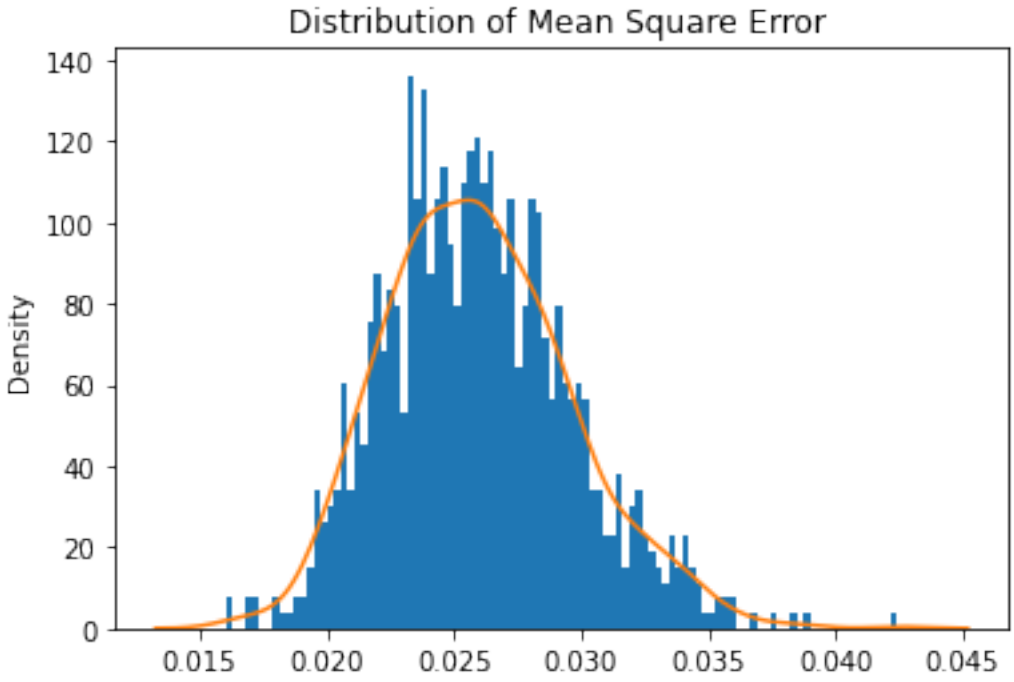
[10]:
```
print(generator)
print(discriminator)
```

```
Generator(
  (hidden1): Linear(in_features=12, out_features=100, bias=True)
  (hidden2): Linear(in_features=100, out_features=100, bias=True)
  (output): Linear(in_features=100, out_features=1, bias=True)
  (relu): ReLU()
)
Discriminator(
  (hidden1): Linear(in_features=12, out_features=25, bias=True)
  (hidden2): Linear(in_features=25, out_features=50, bias=True)
  (output): Linear(in_features=50, out_features=1, bias=True)
  (relu): ReLU()
)
```

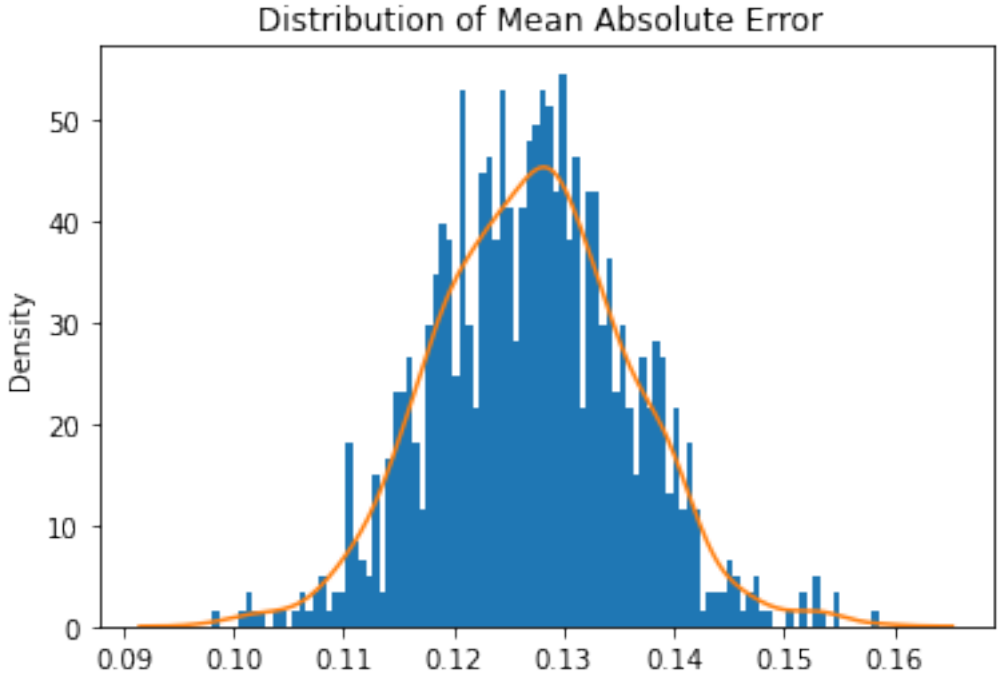
[11]:
```
train_test.
↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
↪n_epochs,criterion,device)
```



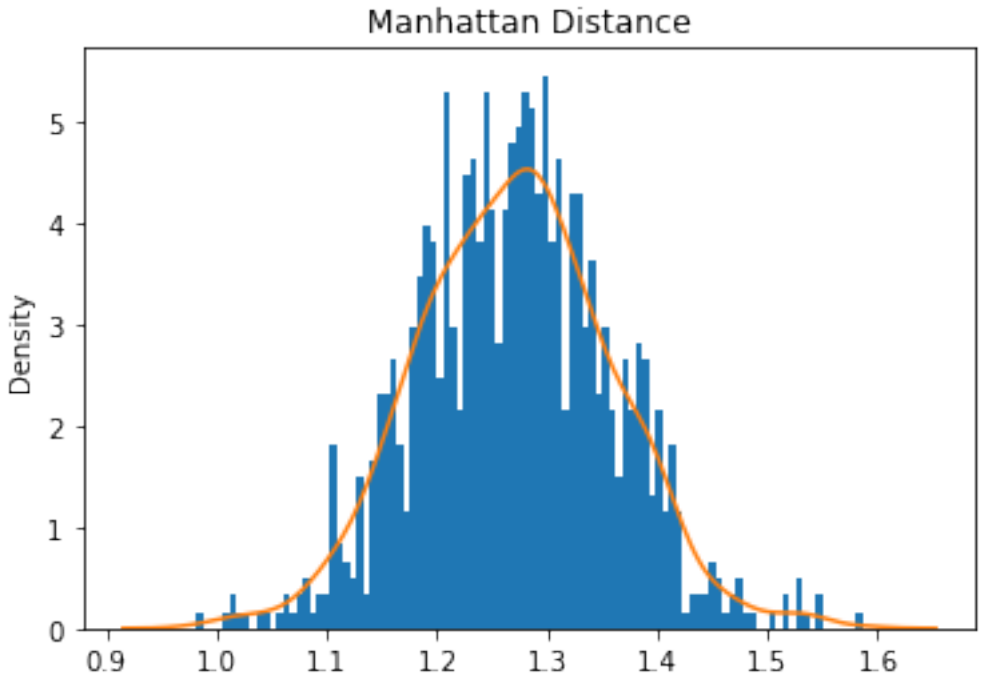
[12]:
```
train_test.test_generator(generator,real_dataset,device)
```



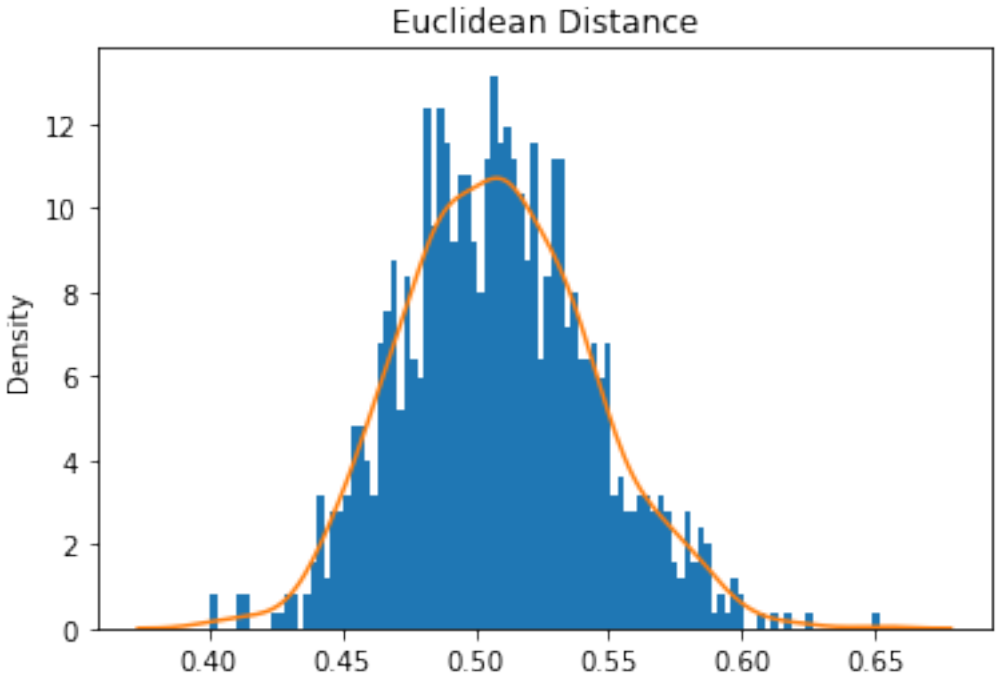
Mean Square Error: 0.025961736511581065



Mean Absolute Error: 0.127064150133729

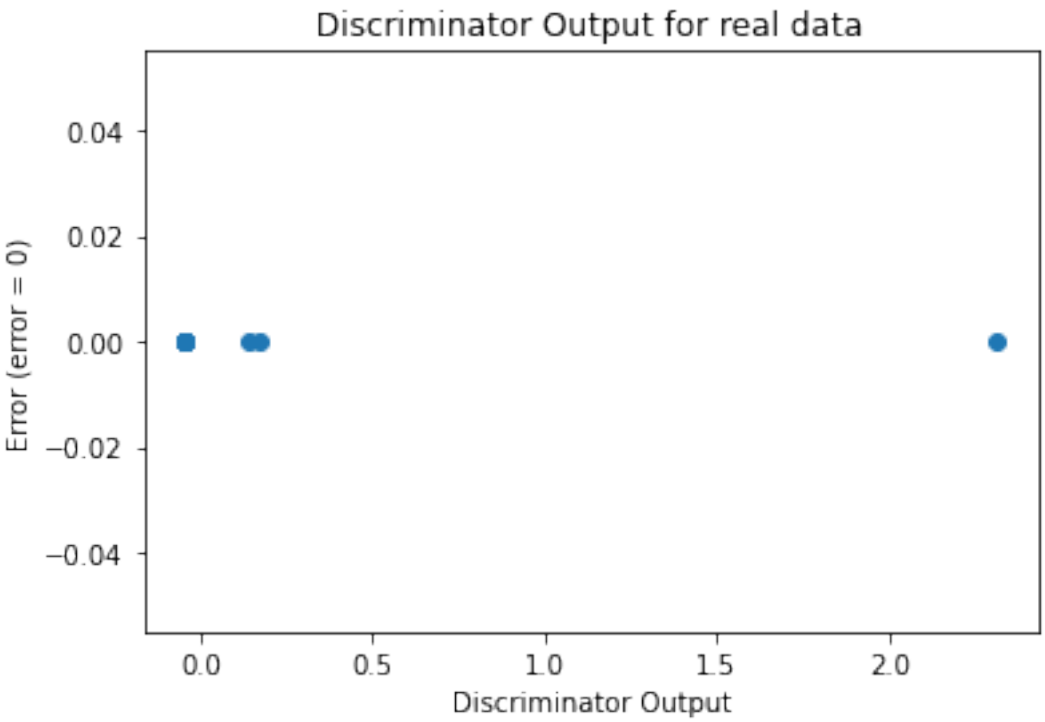


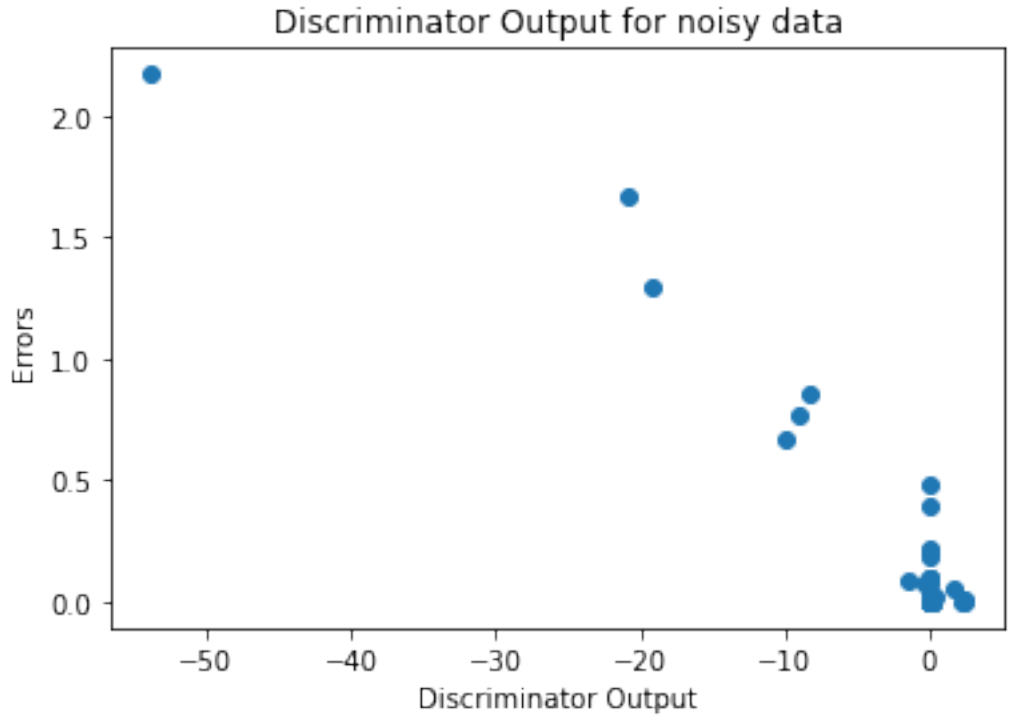
```
Mean Manhattan Distance: 1.2706415013372898
```




```
Mean Euclidean Distance: 0.508261118364835
```

[13]:
```
sanityChecks.discProbVsError(real_dataset,discriminator,device)
```


Discriminator Output for real data
Error (error = 0)
0.04
0.02
0.00
−0.02
−0.04
0.0
0.5
1.0
1.5
2.0
Discriminator Output

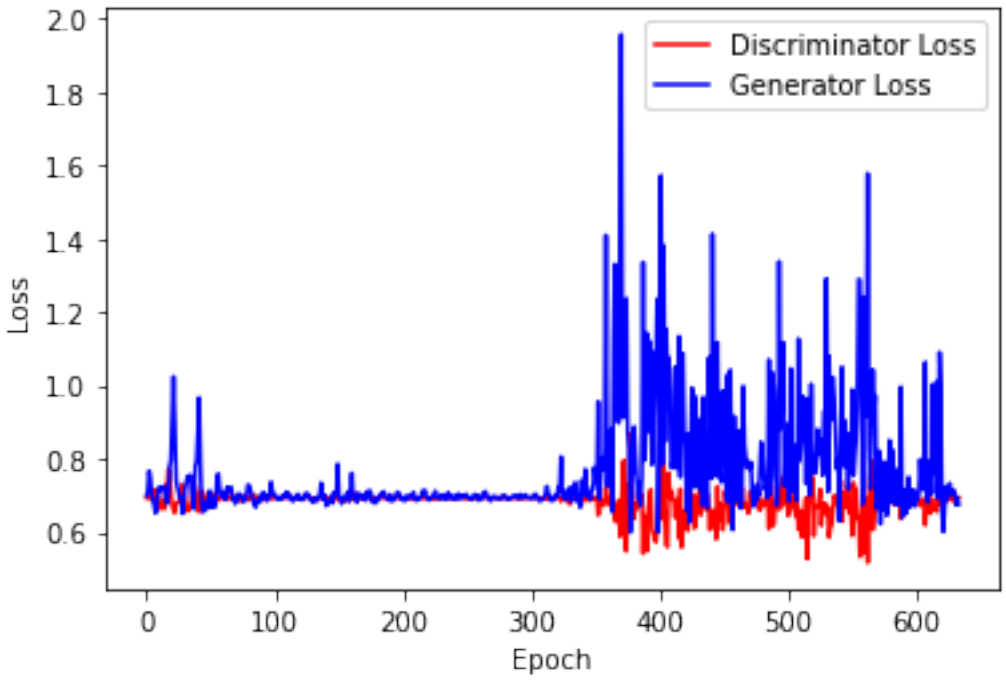

Discriminator Output for noisy data
Errors
2.0
1.5
1.0
0.5
0.0
−50
−40
−30
−20
−10
0
Discriminator Output

Training GAN until mse of y_pred is > 0.1 or n_epochs < 30000

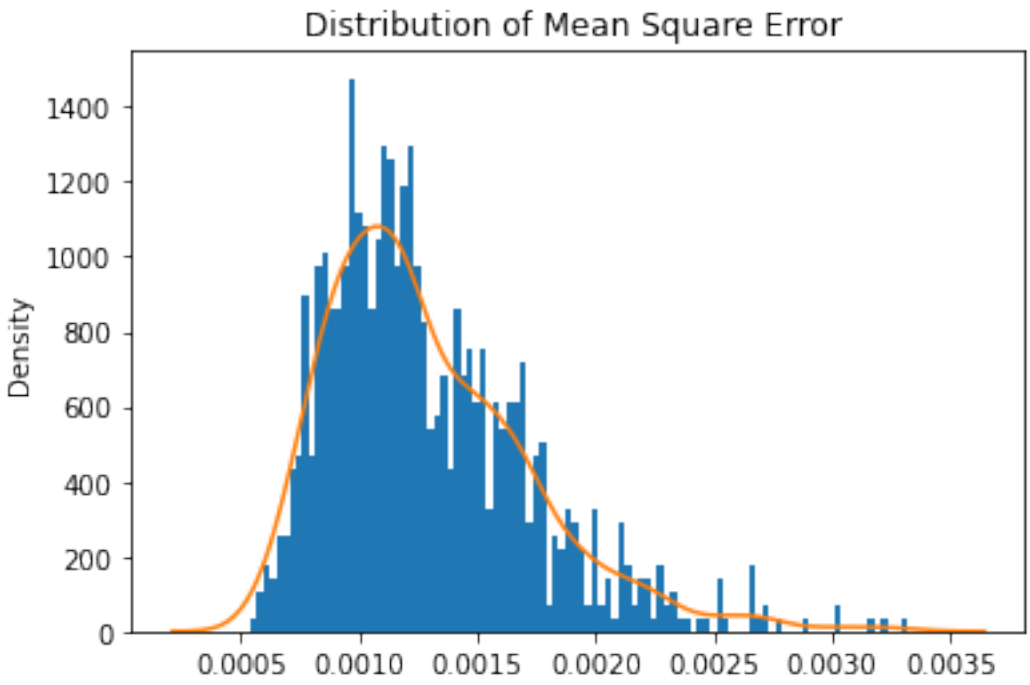
[14]:
```
generator = network.Generator(n_features+2)
discriminator = network.Discriminator(n_features+2)
criterion = torch.nn.BCEWithLogitsLoss()
gen_opt = torch.optim.Adam(generator.parameters(), lr=0.01, betas=(0.5, 0.999))
disc_opt = torch.optim.Adam(discriminator.parameters(), lr=0.01, betas=(0.5, 0.
↪999))
```

[15]:
```
train_test.
↪training_GAN_2(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,error,crite
```

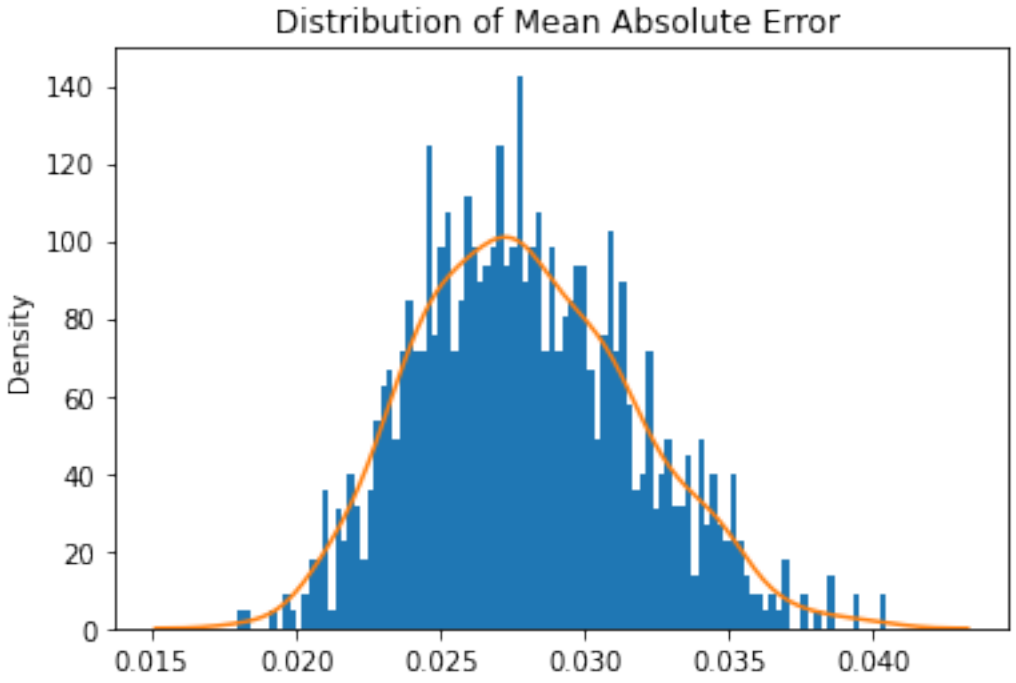
```
Number of epochs needed 317
```



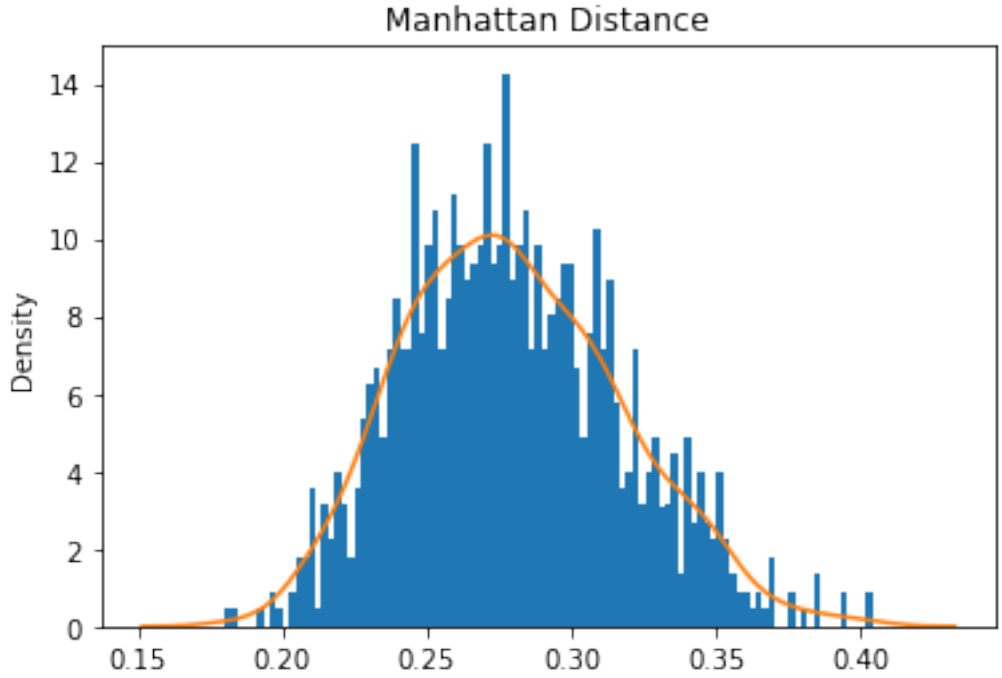
[16]:
```
train_test.test_generator(generator,real_dataset,device)
```



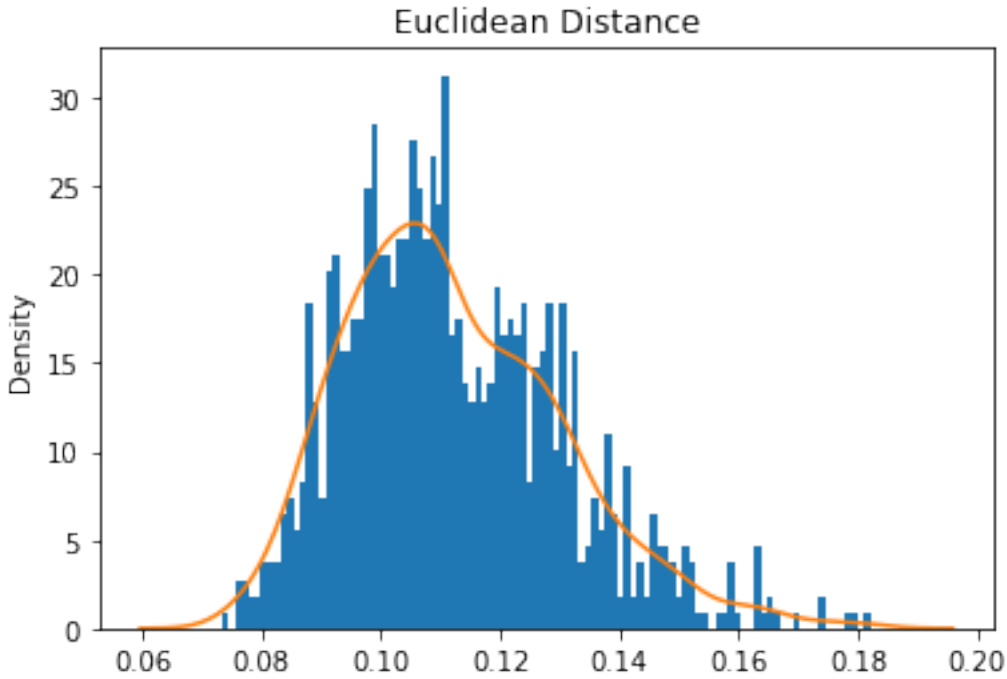

```
Mean Square Error: 0.0012923286336170088
```



```
Mean Absolute Error: 0.02795371982306242
```




```
Mean Manhattan Distance: 0.2795371982306242
```



```
Mean Euclidean Distance: 0.11220624392436895
```

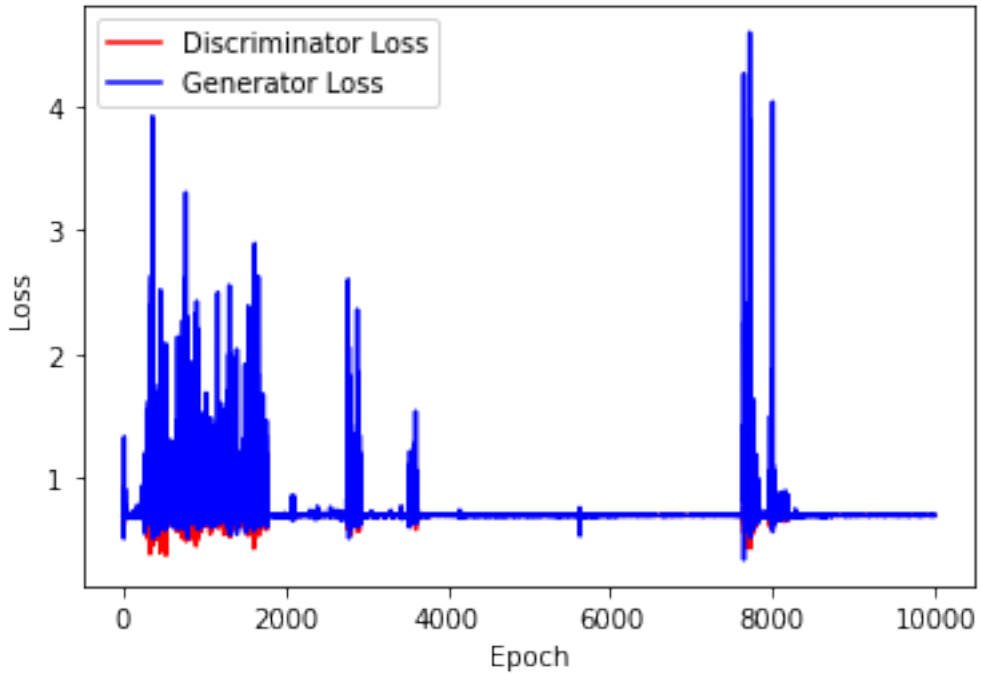
2 ABC GAN Model

2.0.1 Training the network

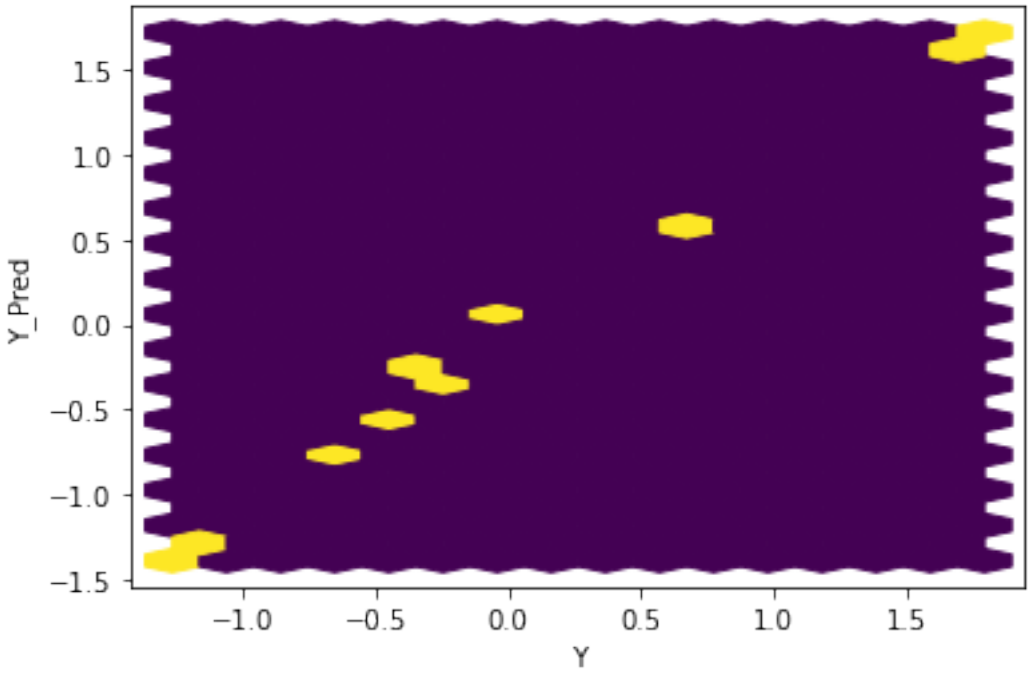
Training ABC-GAN for n_epochs number of epochs

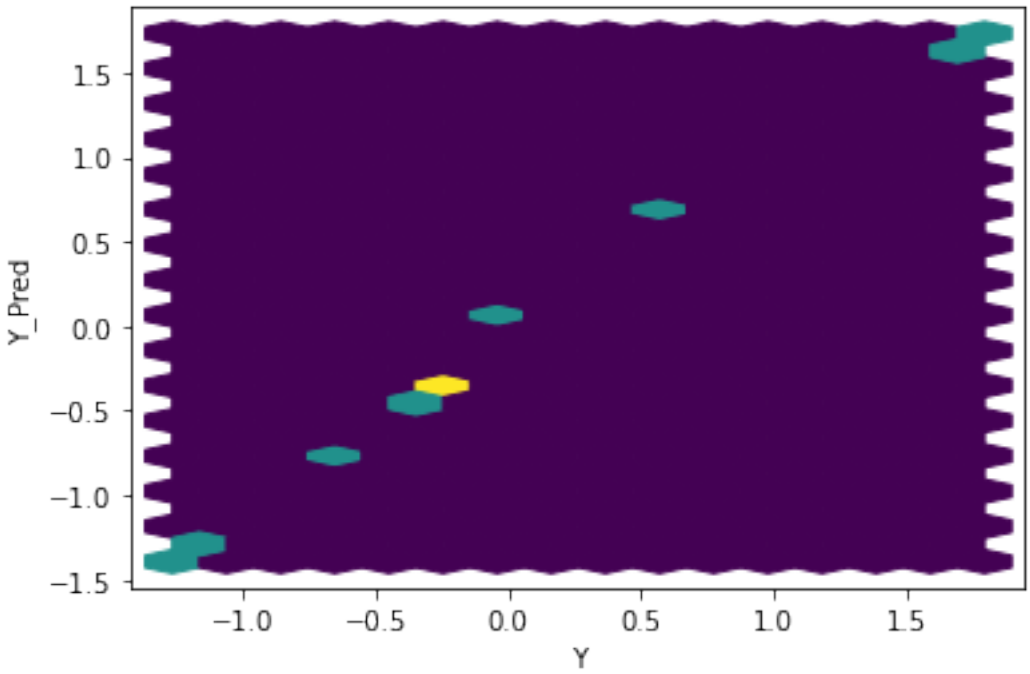
[17]:
```
gen = network.Generator(n_features+2)
disc = network.Discriminator(n_features+2)

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))
```

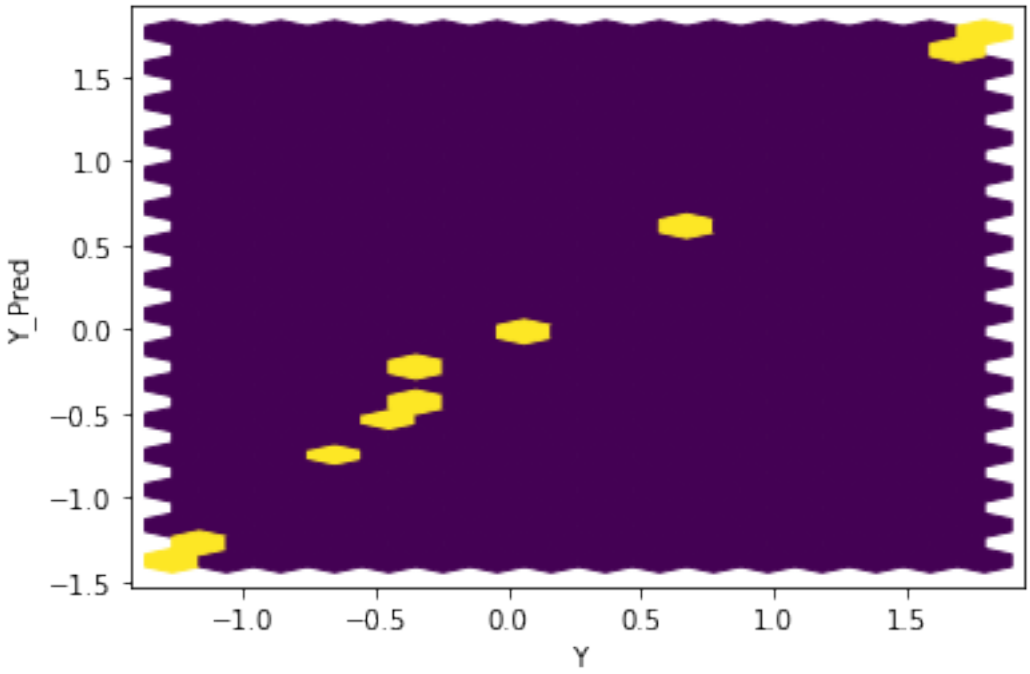
[18]:
```
ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,␣
↪batch_size, n_epochs,criterion,coeff,mean,variance,device)
```

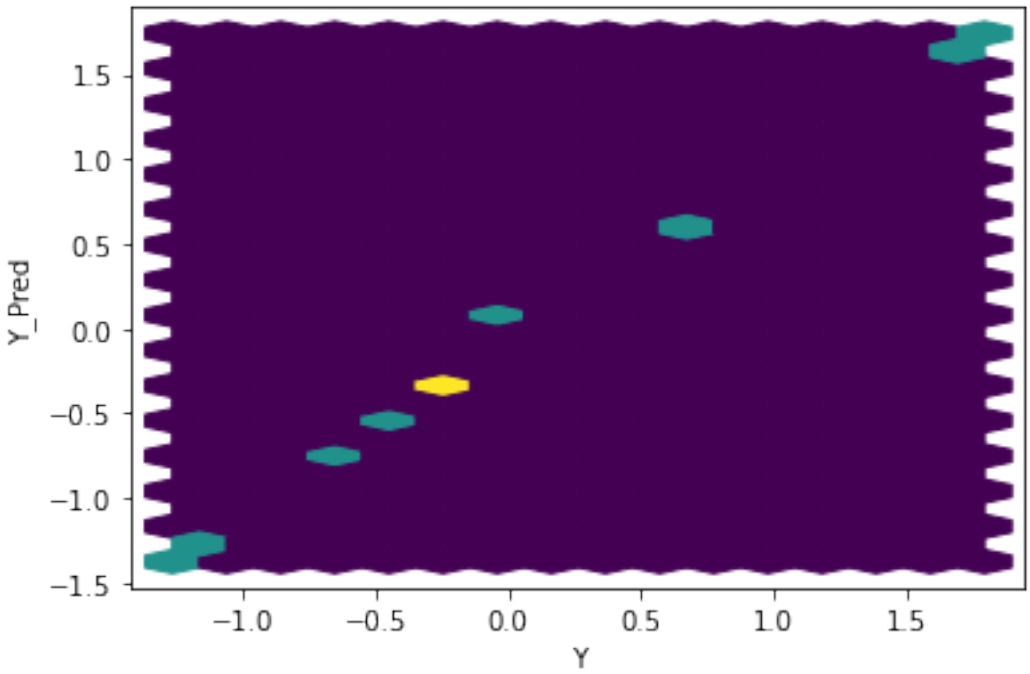


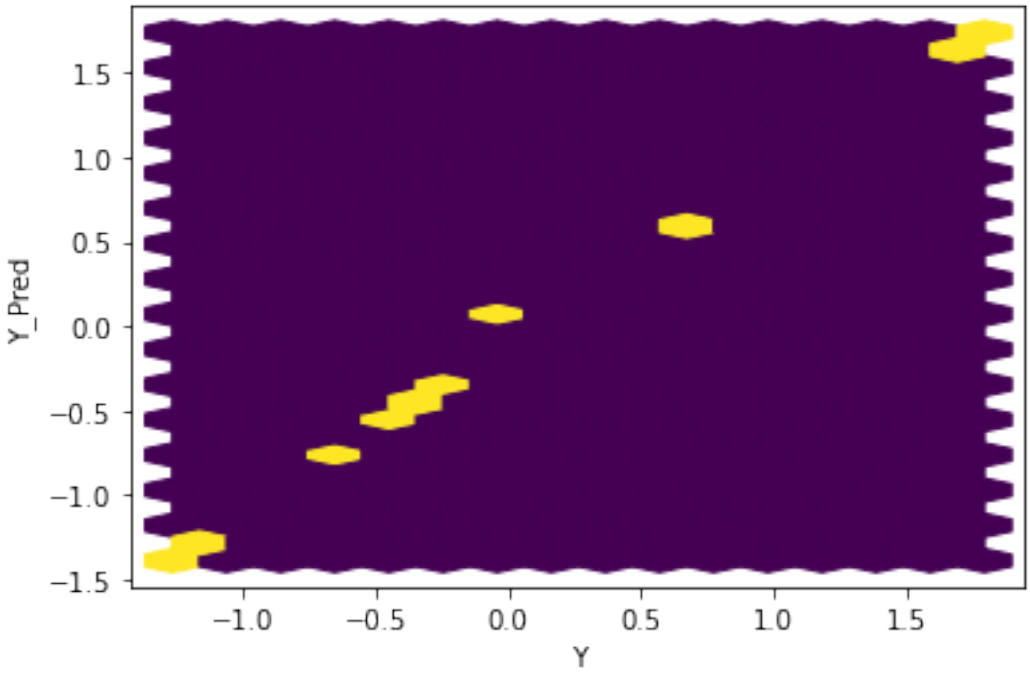
[19]:
```
ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

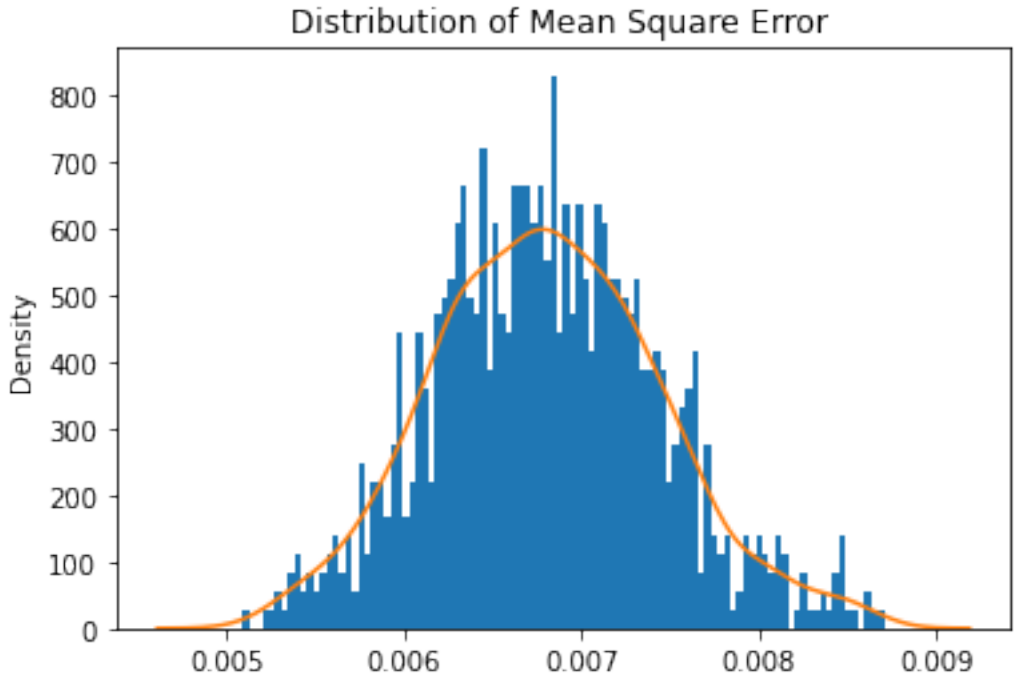

Y_Pred
Y
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Y_Pred
Y
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

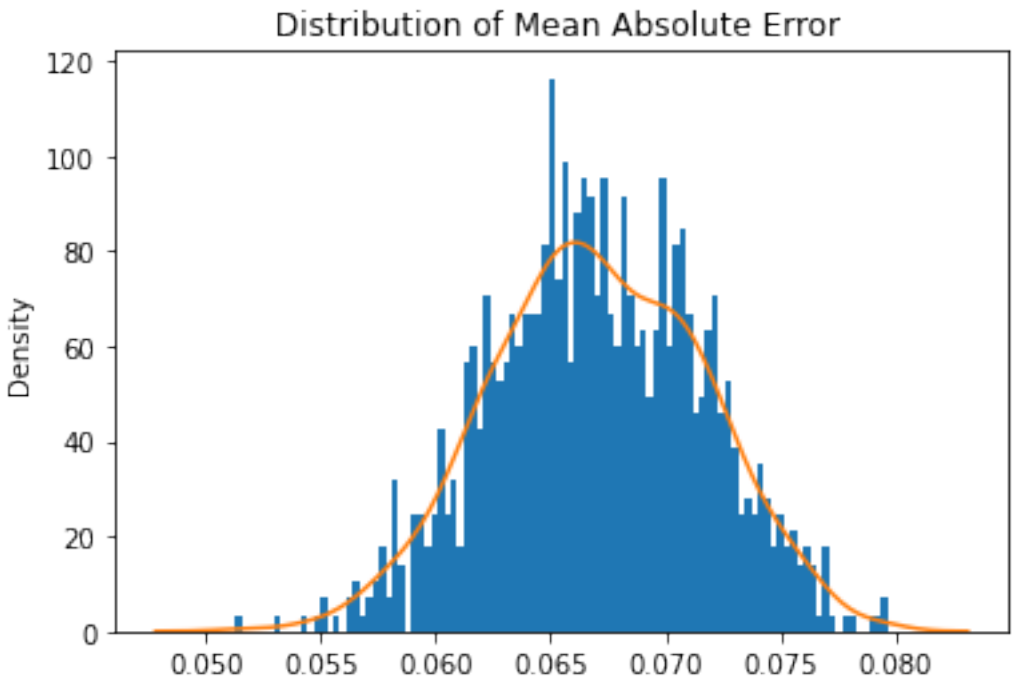

Y_Pred
Y
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Y_Pred
Y
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

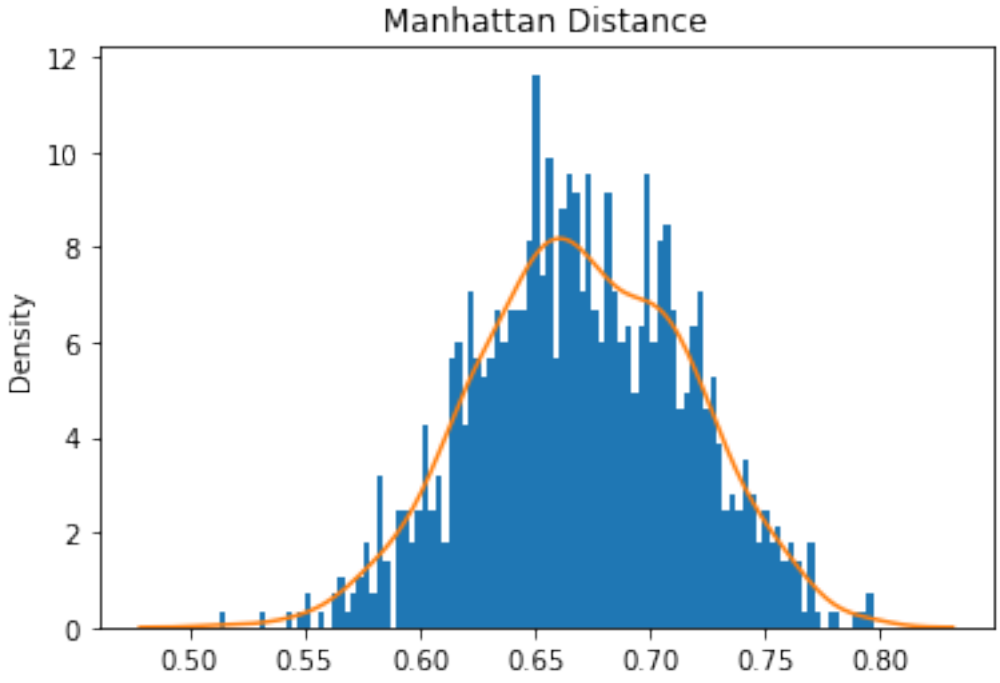




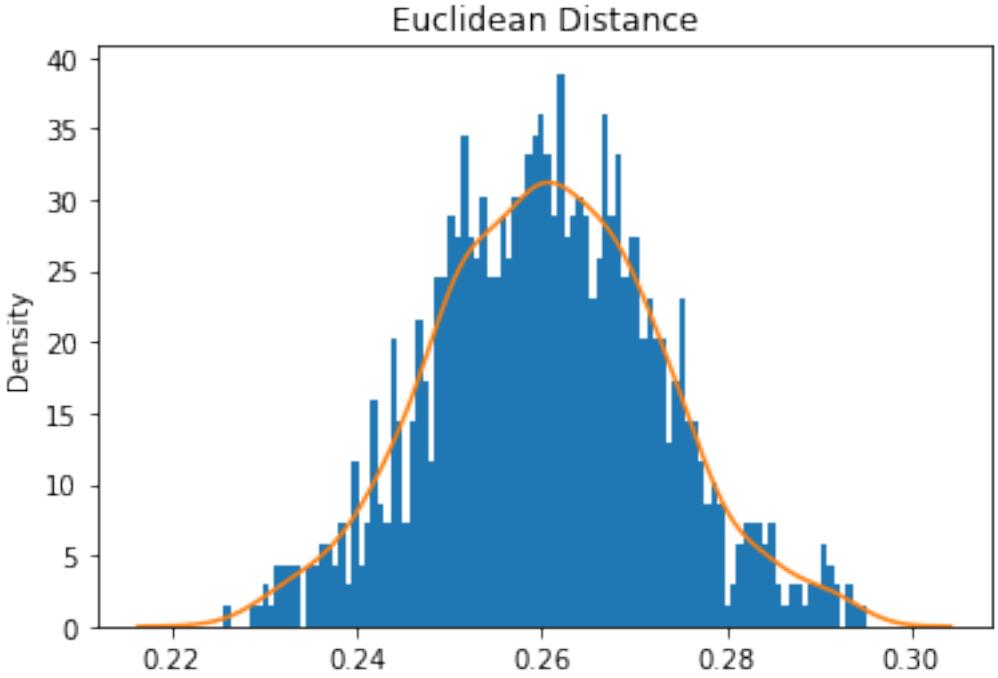

```
Mean Square Error: 0.006804091479603544
```




```
Mean Absolute Error: 0.06702379017174244
Mean Manhattan Distance: 0.6702379017174244
```

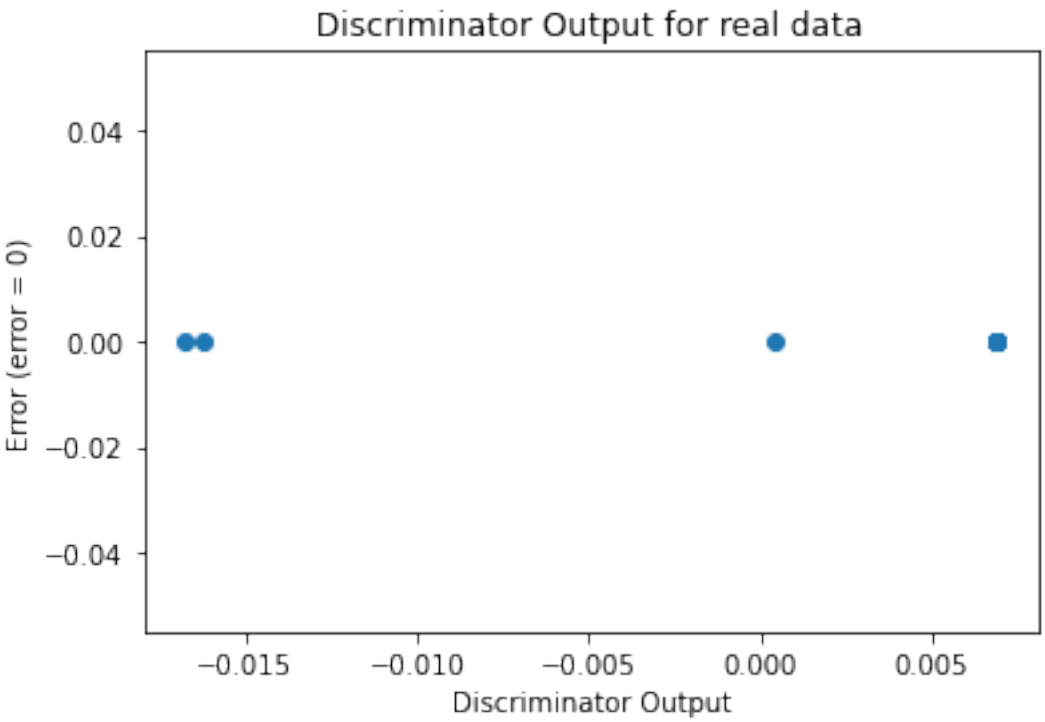


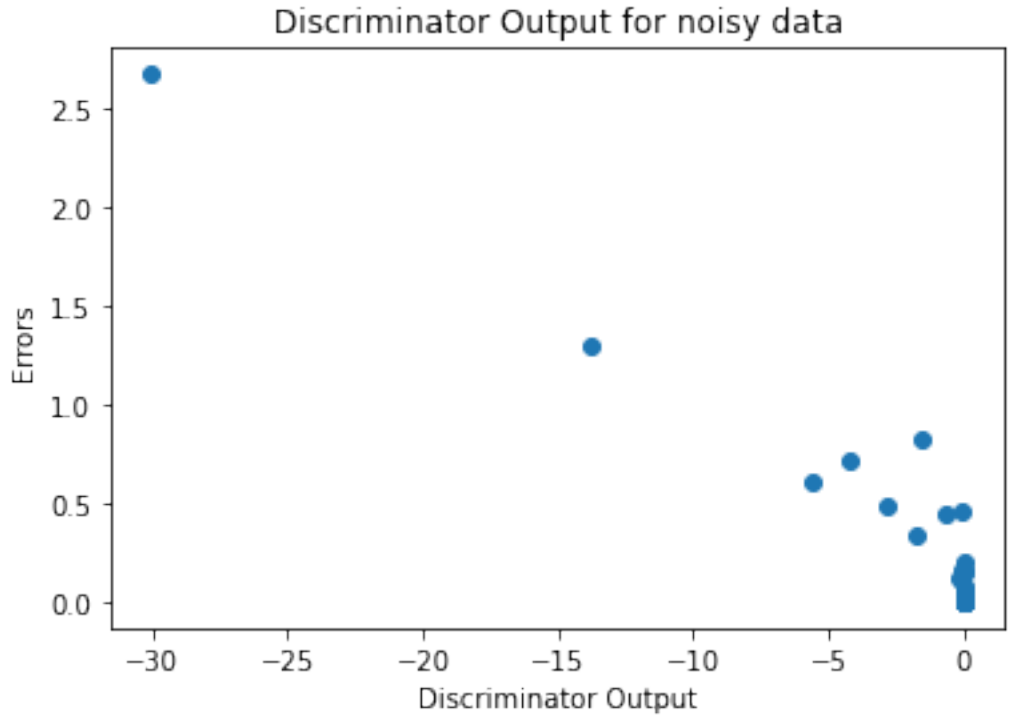
```
Mean Euclidean Distance: 0.2605583816660528
```



Sanity Checks

[20]:
```
sanityChecks.discProbVsError(real_dataset,disc,device)
```


Discriminator Output for real data
Error (error = 0)
0.04
0.02
0.00
−0.02
−0.04
−0.015
−0.010
−0.005
0.000
0.005
Discriminator Output

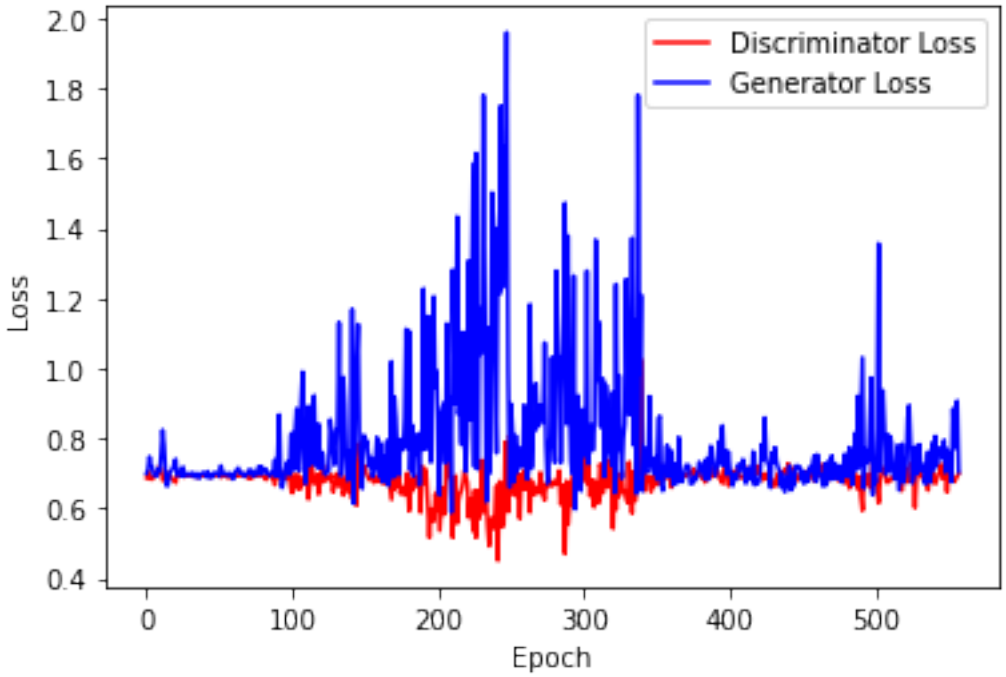

Discriminator Output for noisy data
Errors
2.5
2.0
1.5
1.0
0.5
0.0
−30
−25
−20
−15
−10
−5
0
Discriminator Output

Training GAN until mse of y_pred is > 0.1 or n_epochs < 30000

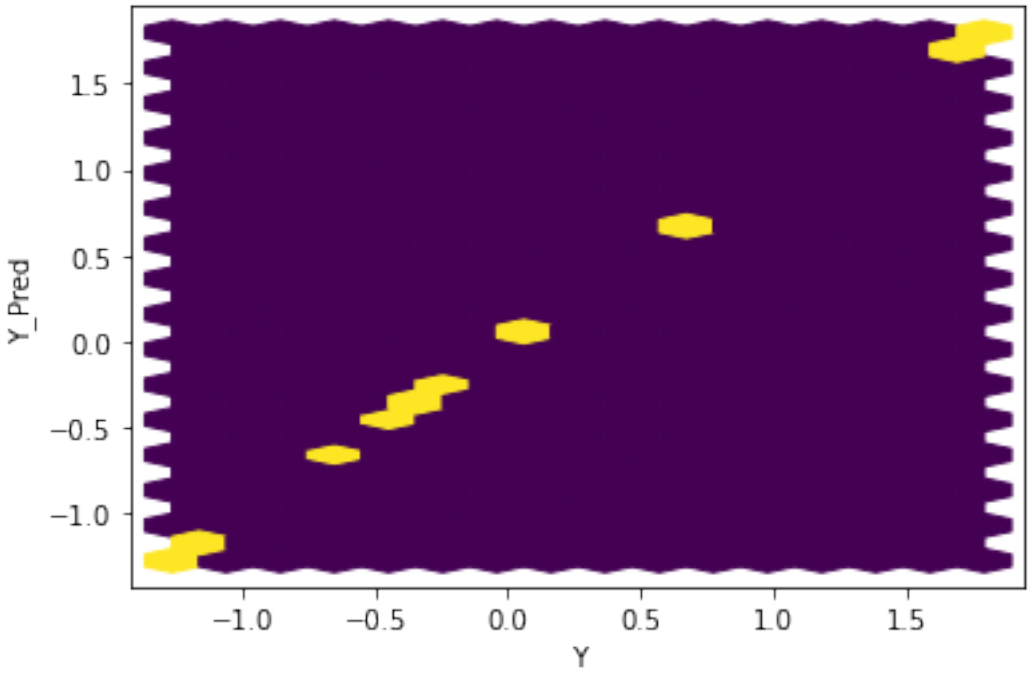
[21]:
```
gen = network.Generator(n_features+2)
disc = network.Discriminator(n_features+2)

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))
```

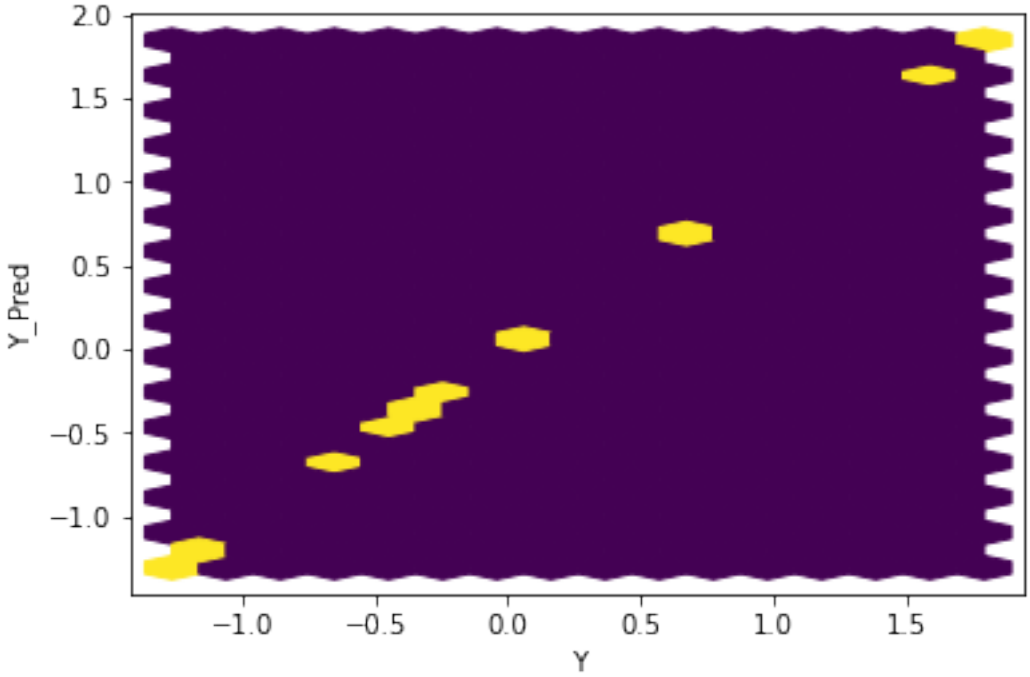
[22]:
```
ABC_train_test.
↪training_GAN_2(disc,gen,disc_opt,gen_opt,real_dataset,batch_size,␣
↪error,criterion,coeff,mean,variance,device)
```

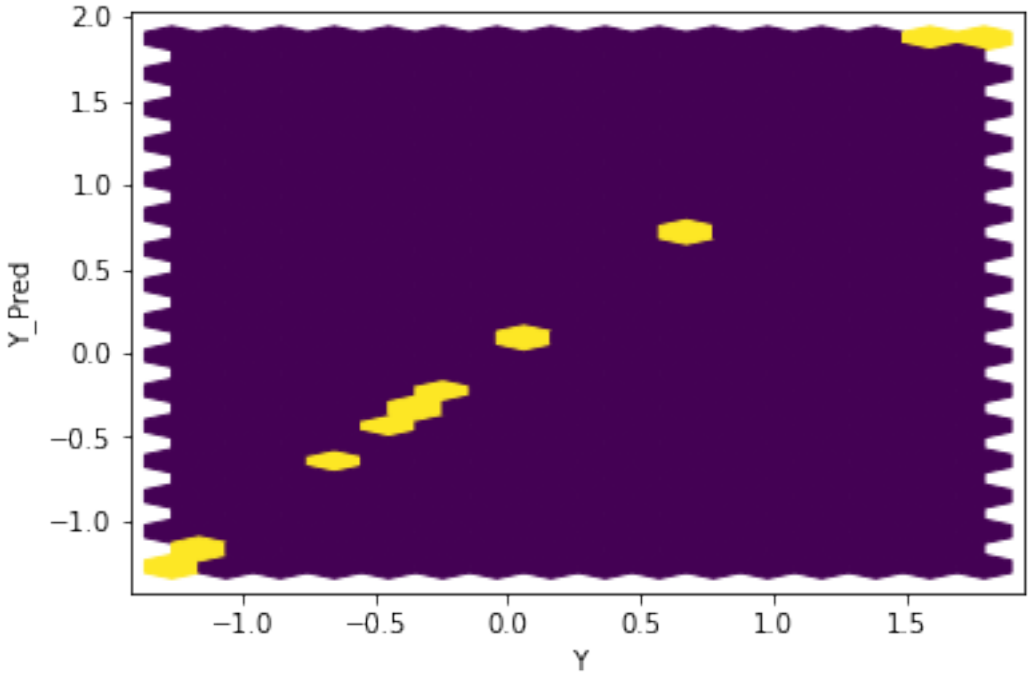
```
Number of epochs 279
```

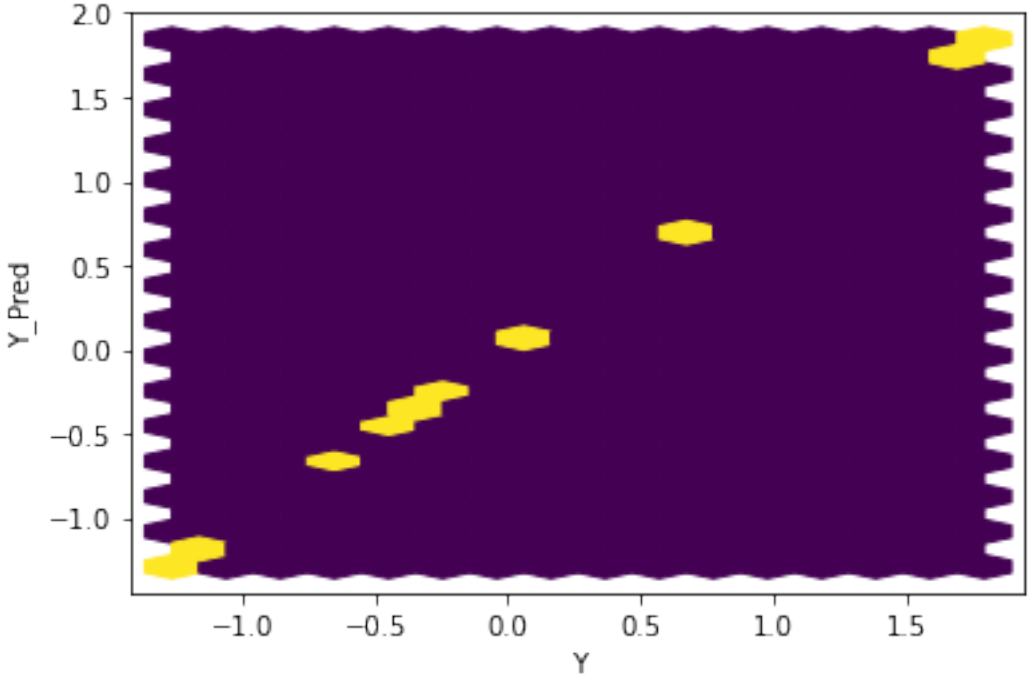


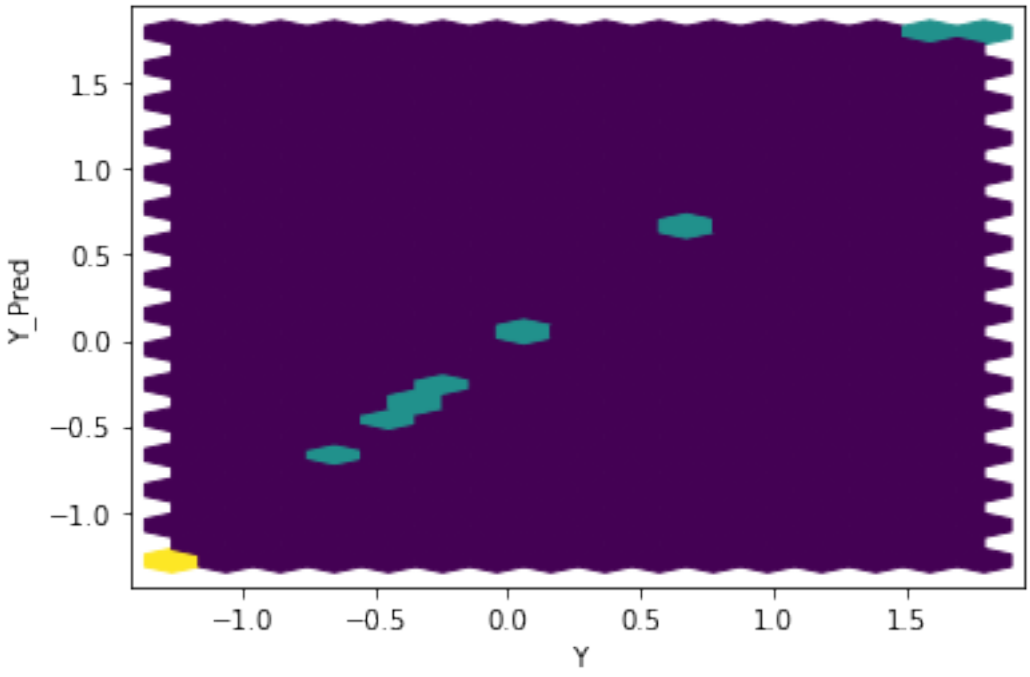
[23]:
```
ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

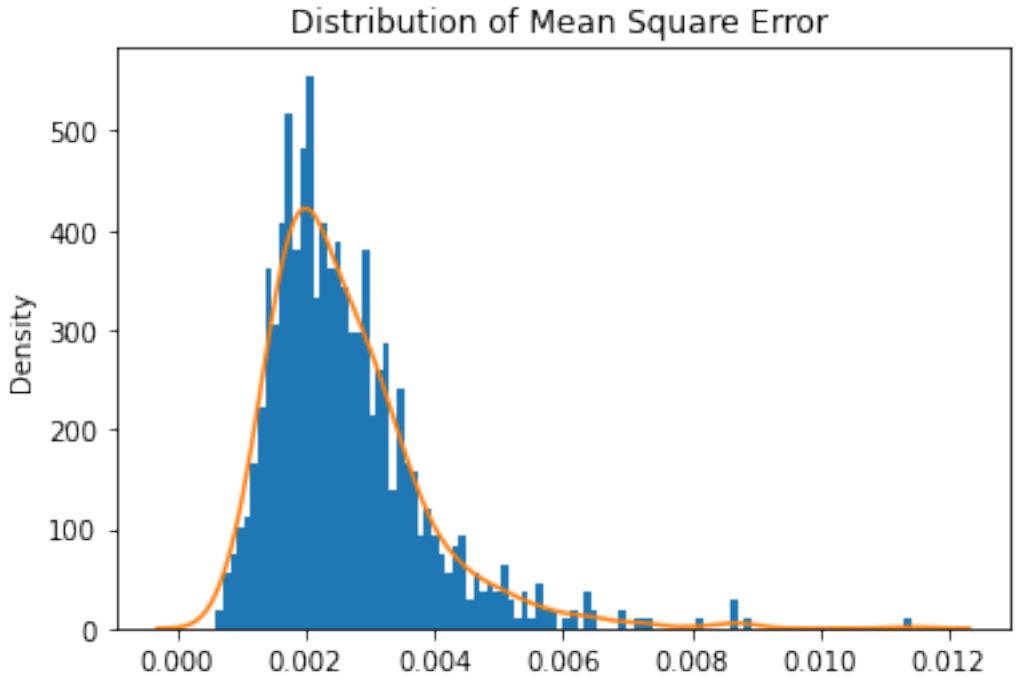

Y_Pred
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5
Y


Y_Pred
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5
Y

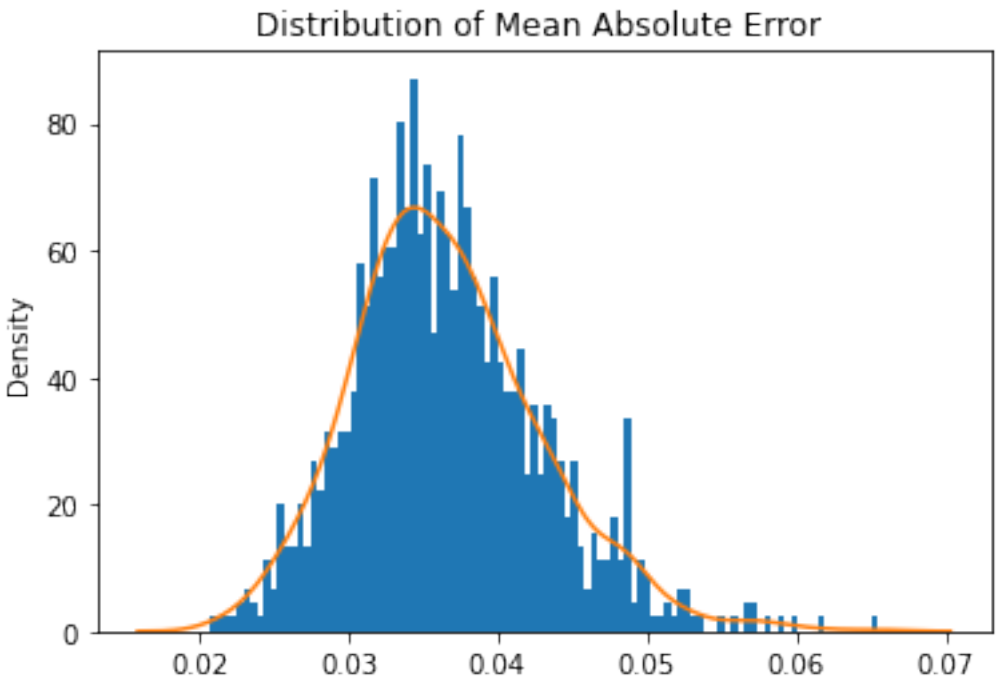

2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
Y_Pred
−1.0
−0.5
0.0
0.5
1.0
1.5
Y


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
Y_Pred
−1.0
−0.5
0.0
0.5
1.0
1.5
Y

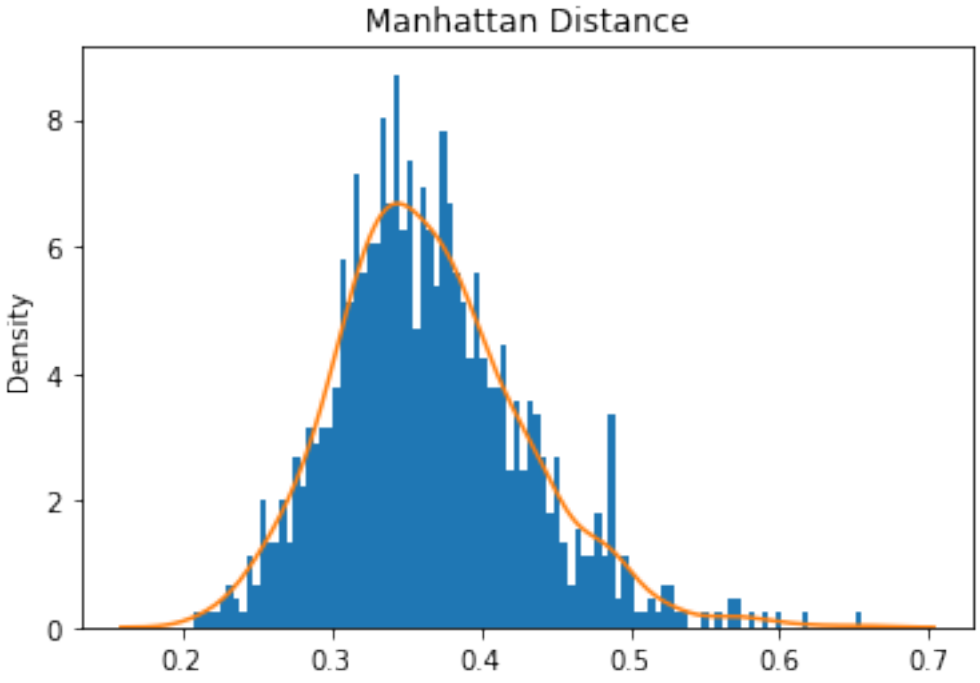




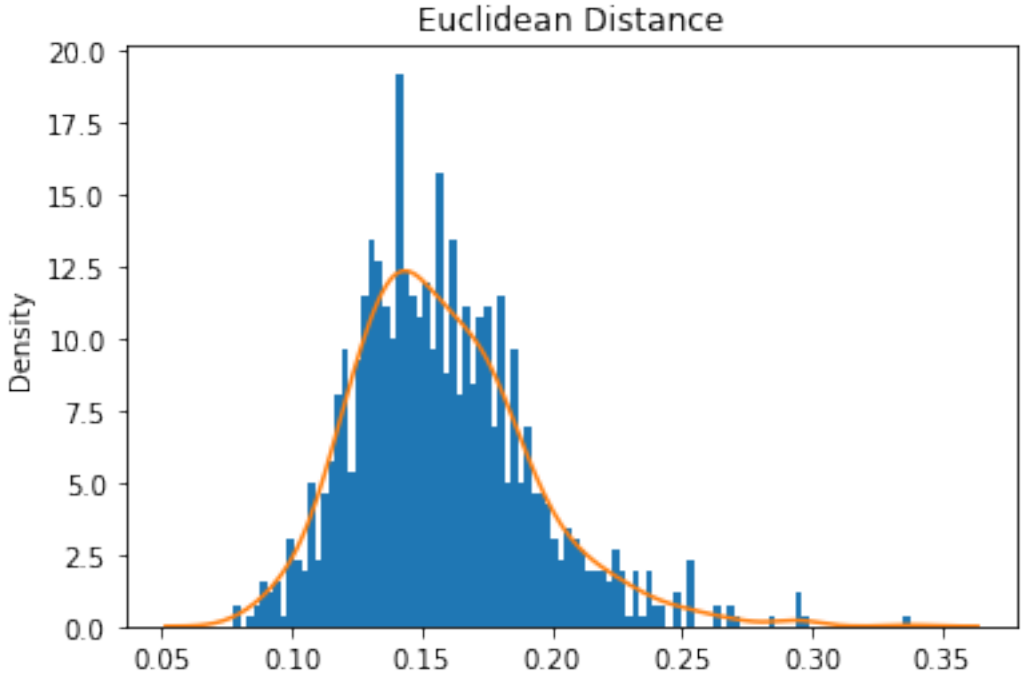

```
Mean Square Error: 0.0026144783373992057
```




```
Mean Absolute Error: 0.03659585256949067
Mean Manhattan Distance: 0.3659585256949067
```



```
Mean Euclidean Distance: 0.15795714514742182
```



[ ]: