# Dataset1-Regression_output_1

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. $\sigma^*$ 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 $\beta^*$ 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 = 0.1
```

### 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 -0.540385 -0.490080 -1.447188 -0.919648  0.040996  1.057338 -0.392487
1  0.923025 -0.953074  1.615658 -0.435705  0.641171  2.435204  1.535587
2  0.211138 -0.888743  0.418675  0.689261 -1.252928 -0.208687 -0.661367
3 -0.540528 -1.069342 -1.354002  0.661209  0.007159  1.475346 -0.404701
4  2.058156 -0.207390  0.264763  0.328278 -0.338119 -0.484247  0.380594

         X8        X9       X10           Y
```

```
0 -0.388198 -0.848136 -0.103359 -210.295427
1  0.489707 -0.143183  2.226057  596.371441
2 -0.616962  0.225813  1.055791   74.753093
3  0.783811  0.845635 -1.055352  -22.720015
4  0.622670 -1.179701  0.793570   65.298452
```

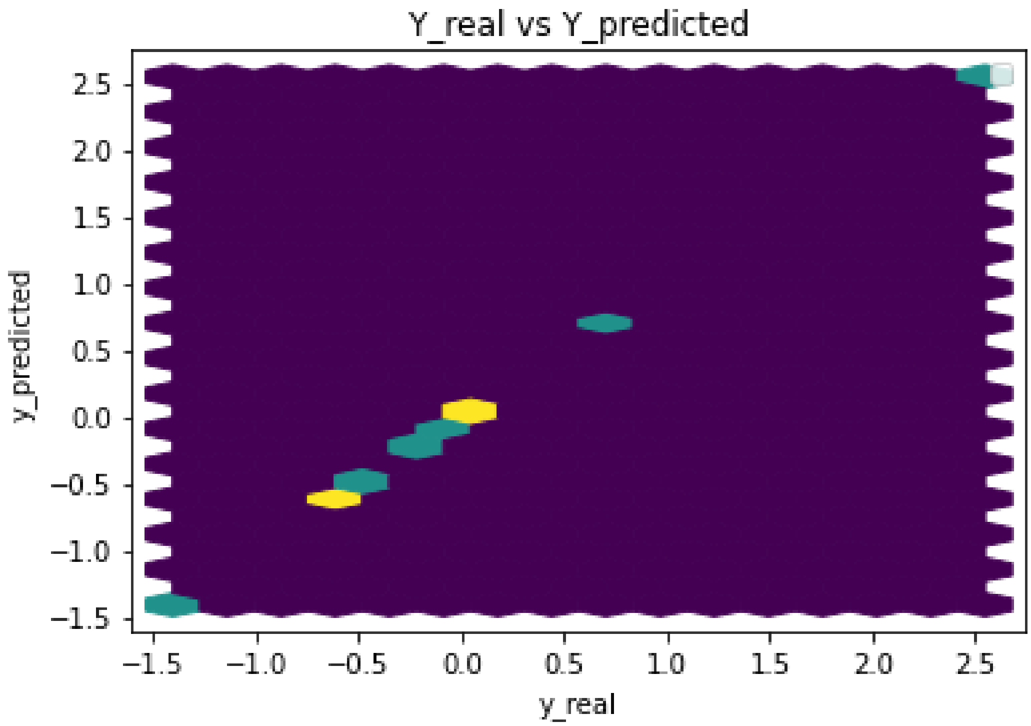
## 1.5 Stats Model

[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:                        22:39:12   Log-Likelihood:                 323.21
No. Observations:                  10   AIC:                            -626.4
Df Residuals:                       0   BIC:                            -623.4
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -5.551e-17        inf         -0        nan         nan         nan
x1             0.0658        inf          0        nan         nan         nan
x2             0.1942        inf          0        nan         nan         nan
x3             0.4064        inf          0        nan         nan         nan
x4             0.2812        inf          0        nan         nan         nan
x5             0.0621        inf          0        nan         nan         nan
x6             0.4371        inf          0        nan         nan         nan
x7             0.0844        inf          0        nan         nan         nan
x8             0.2782        inf          0        nan         nan         nan
x9             0.2085        inf          0        nan         nan         nan
x10            0.4970        inf          0        nan         nan         nan
==============================================================================
Omnibus:                        1.486   Durbin-Watson:                   2.514
Prob(Omnibus):                  0.476   Jarque-Bera (JB):                0.736
Skew:                          -0.067   Prob(JB):                        0.692
Kurtosis:                       1.678   Cond. No.                         8.81
==============================================================================

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   -5.551115e-17
```

```
x1       6.578735e-02
x2       1.942104e-01
x3       4.063667e-01
x4       2.812406e-01
x5       6.214164e-02
x6       4.371153e-01
x7       8.435049e-02
x8       2.781725e-01
x9       2.085464e-01
x10      4.970263e-01
dtype: float64
```



```
Performance Metrics
Mean Squared Error: 4.939611880596015e-30
Mean Absolute Error: 2.029279522197669e-15
Manhattan distance: 2.029279522197669e-14
Euclidean distance: 7.02823724741561e-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))
```

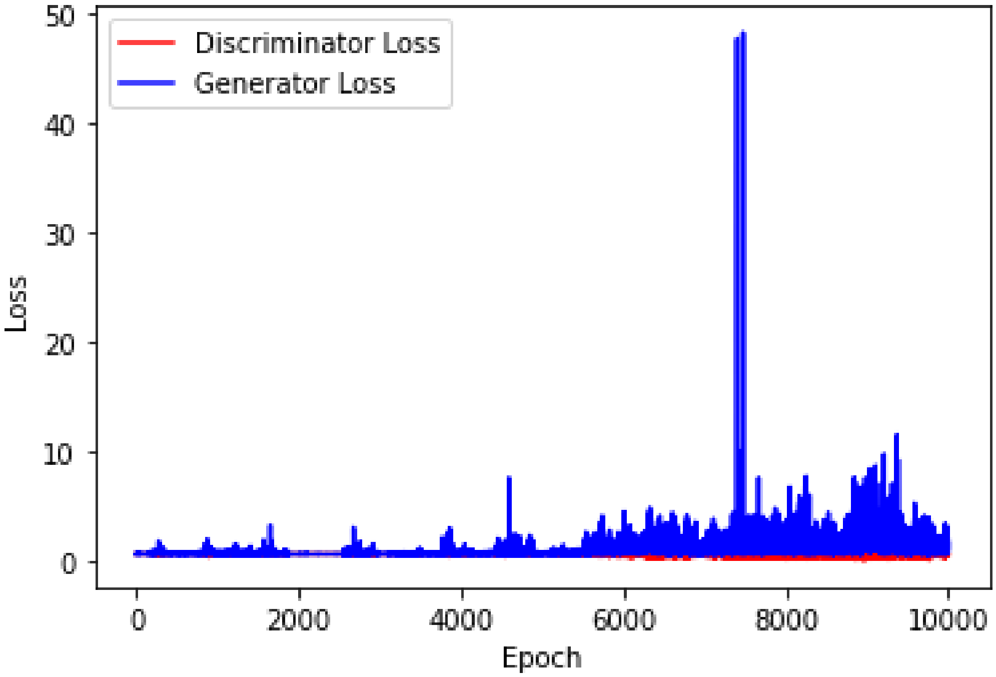
[10]:

```
print(generator)
print(discriminator)
```

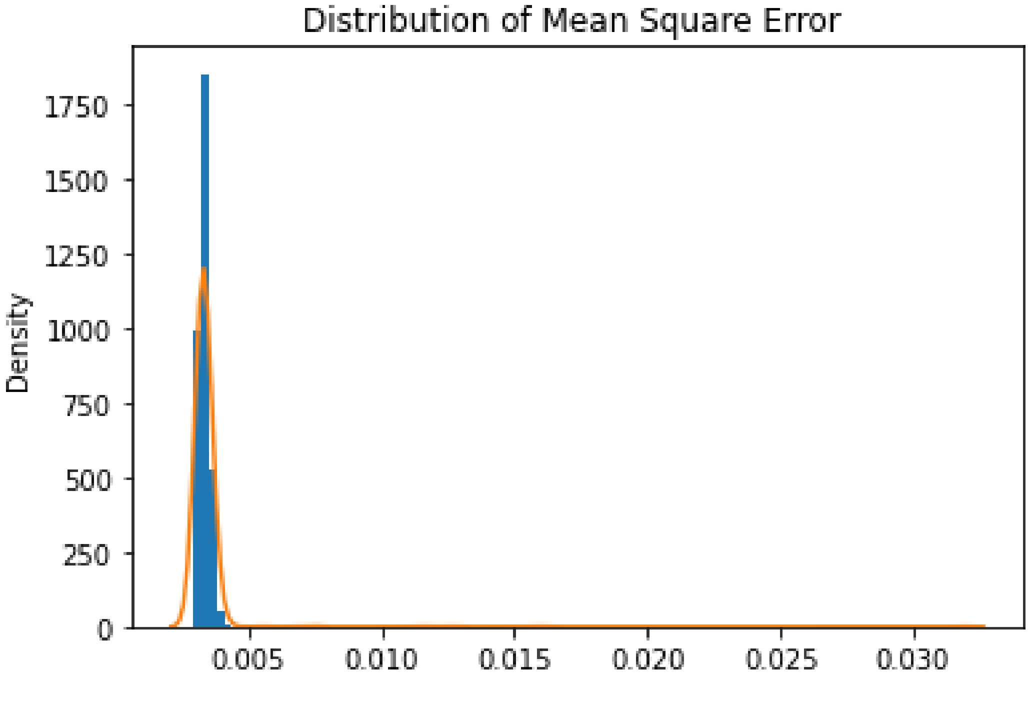
```
Generator(
  (hidden1): Linear(in_features=12, out_features=100, bias=True)
  (hidden2): Linear(in_features=100, out_features=100, bias=True)
  (output): Linear(in_features=100, out_features=1, bias=True)
  (relu): ReLU()
)
Discriminator(
  (hidden1): Linear(in_features=12, out_features=25, bias=True)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

[11]:

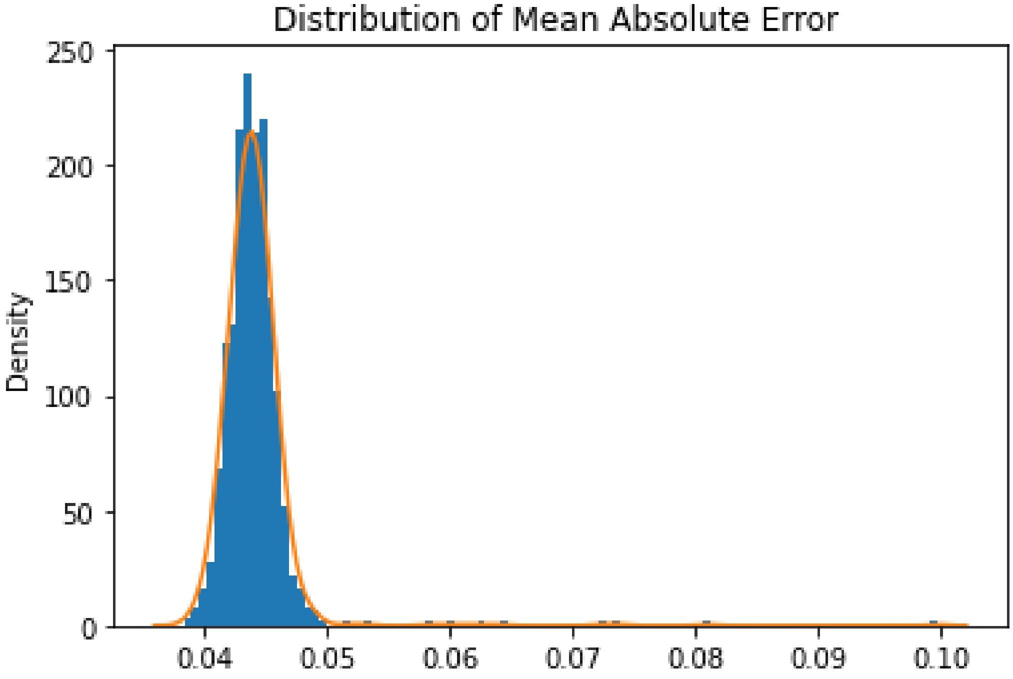
```
train_test.
 ↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,␣
 ↪n_epochs,criterion,device)
```



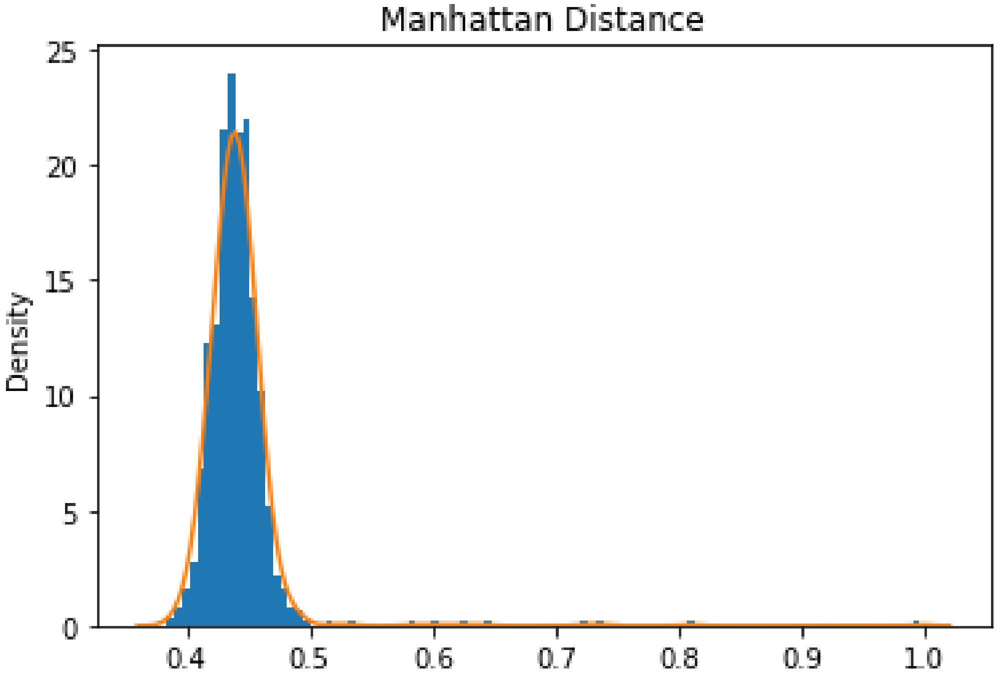
[12]:
```
train_test.test_generator(generator,real_dataset,device)
```



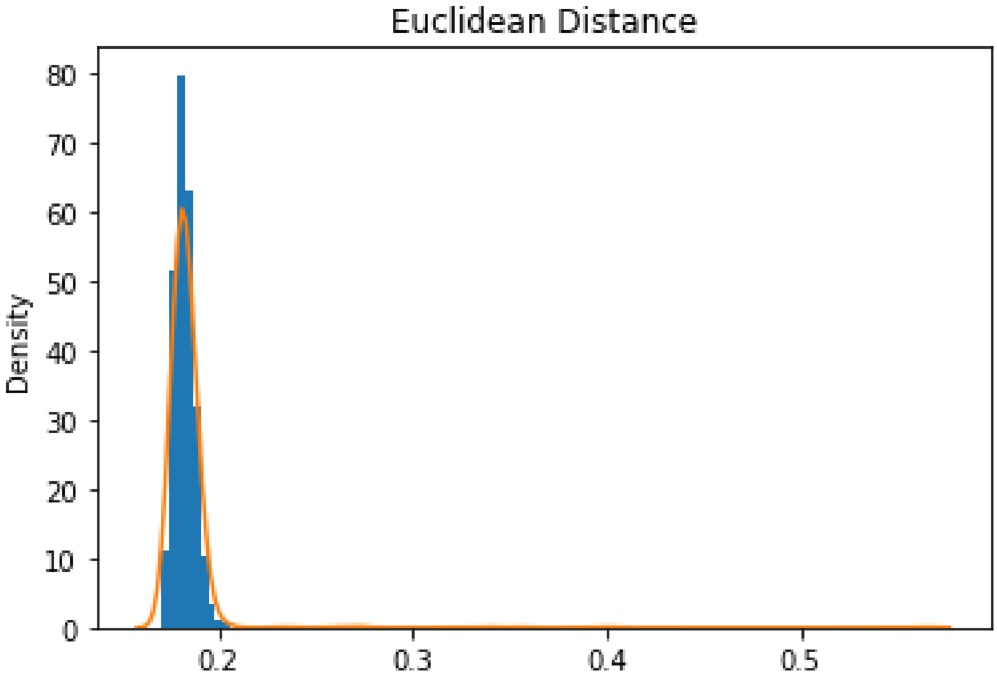
```
Mean Square Error: 0.003392256482045935
```



```
Mean Absolute Error: 0.04409617279022932
```

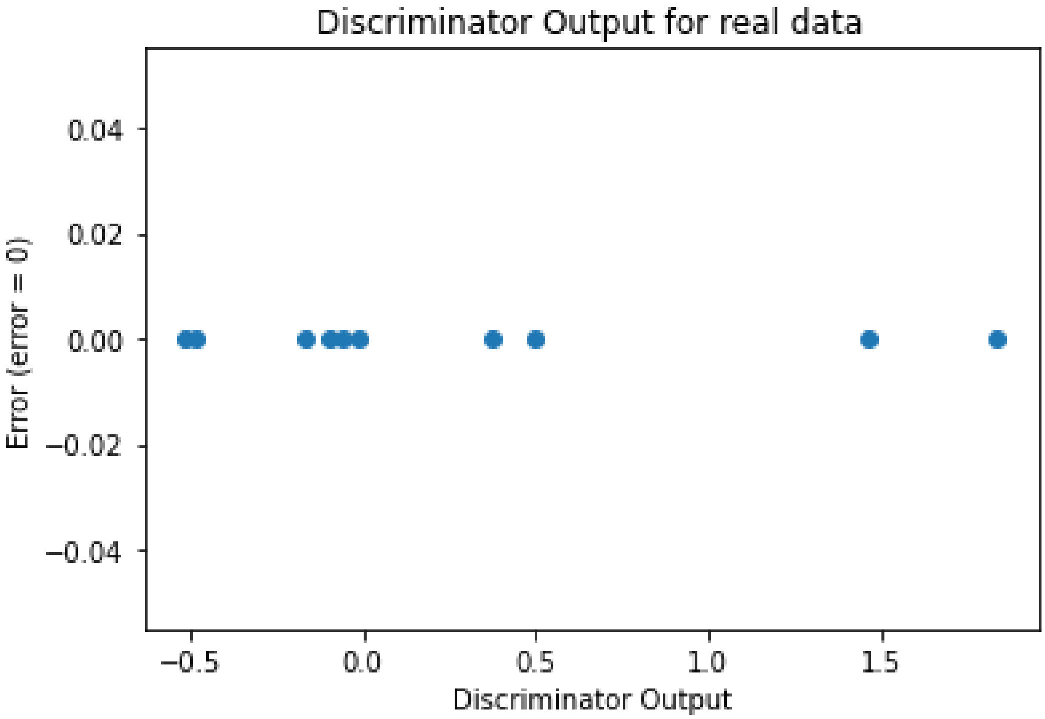


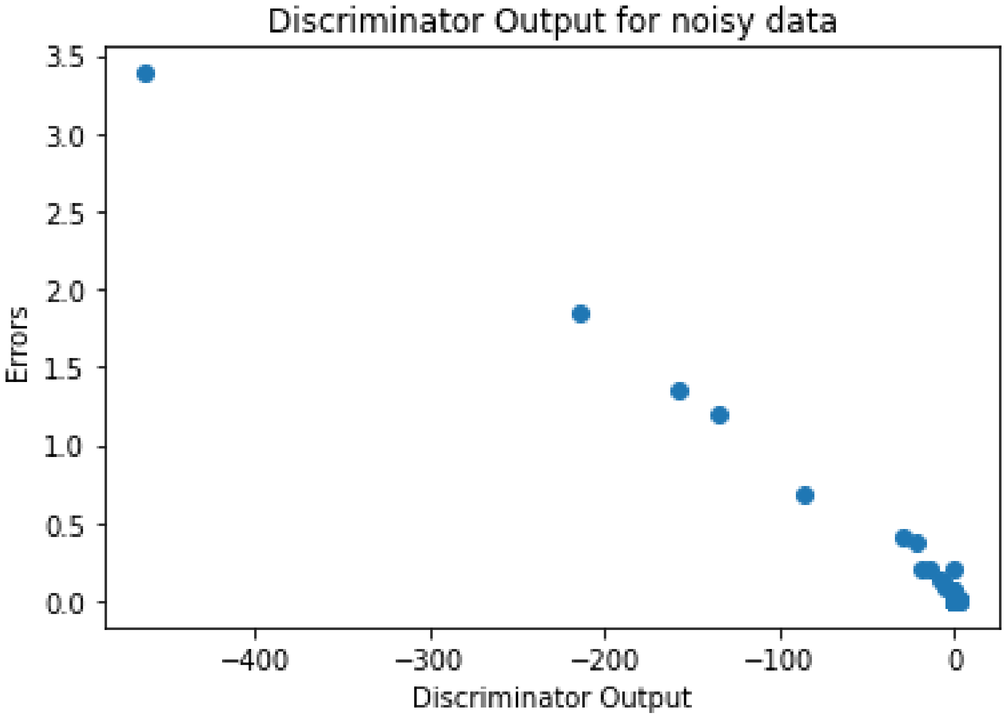
```
Mean Manhattan Distance: 0.4409617279022932
```



```
Mean Euclidean Distance: 0.18337509418847248
```

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


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

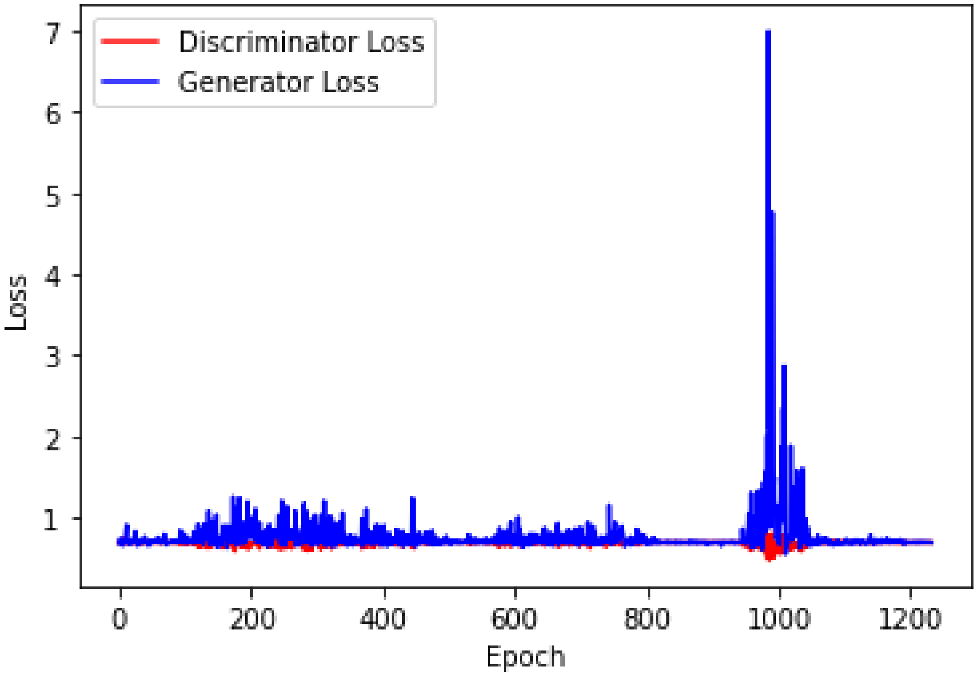

Discriminator Output for noisy data
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Errors
−400
−300
−200
−100
0
Discriminator Output

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

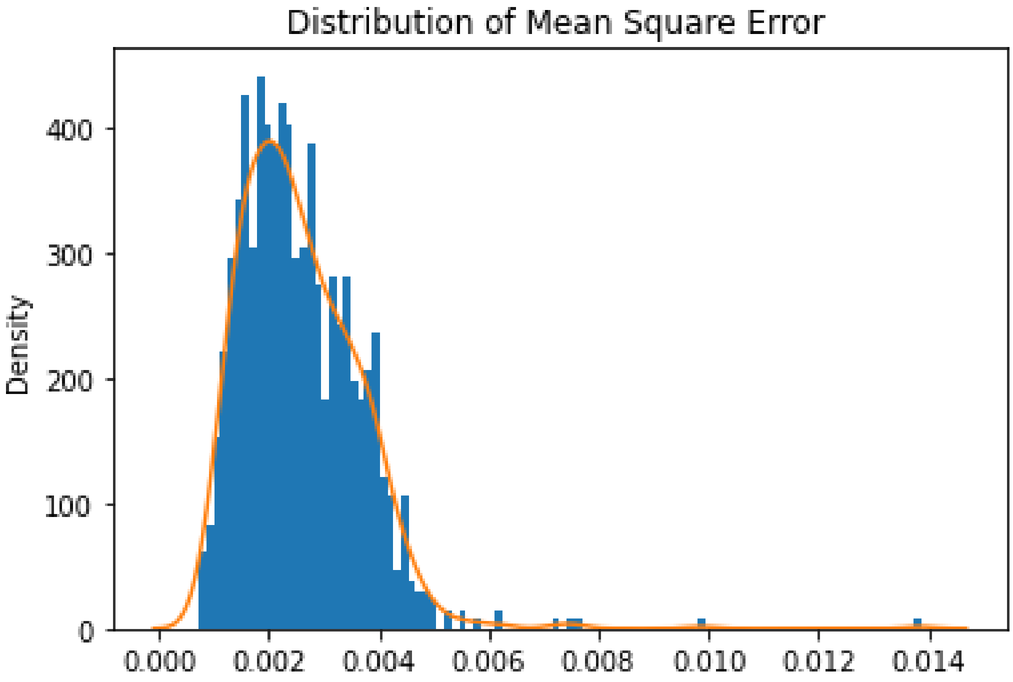
[14]:
```
generator = network.Generator(n_features+2)
discriminator = network.Discriminator(n_features+2)
criterion = torch.nn.BCEWithLogitsLoss()
gen_opt = torch.optim.Adam(generator.parameters(), lr=0.01, betas=(0.5, 0.999))
disc_opt = torch.optim.Adam(discriminator.parameters(), lr=0.01, betas=(0.5,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

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

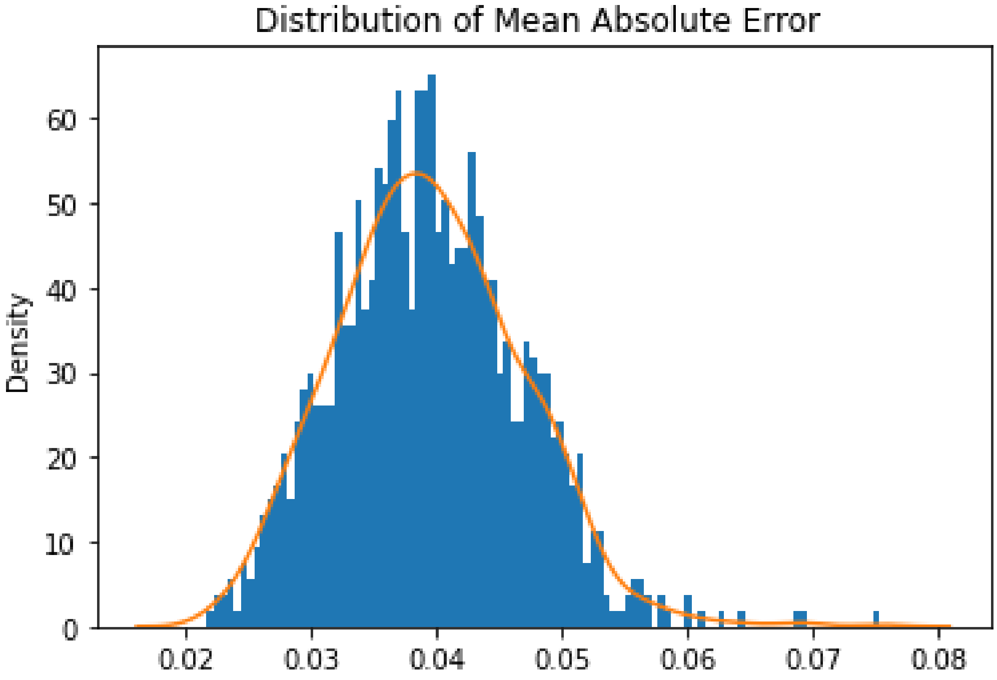
```
Number of epochs needed 616
```



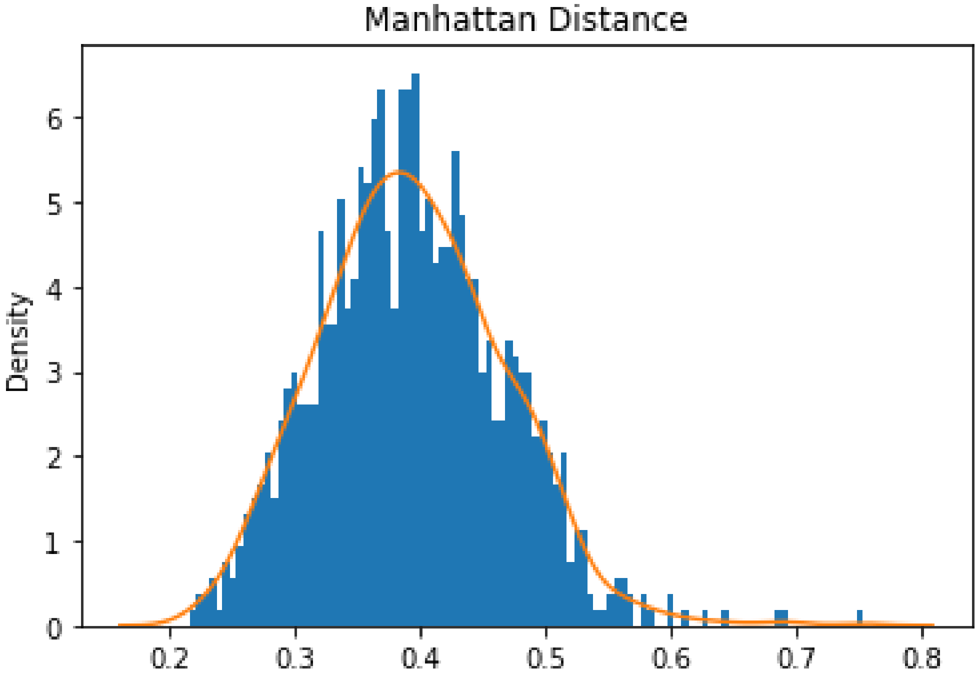
[16]:
```
train_test.test_generator(generator,real_dataset,device)
```



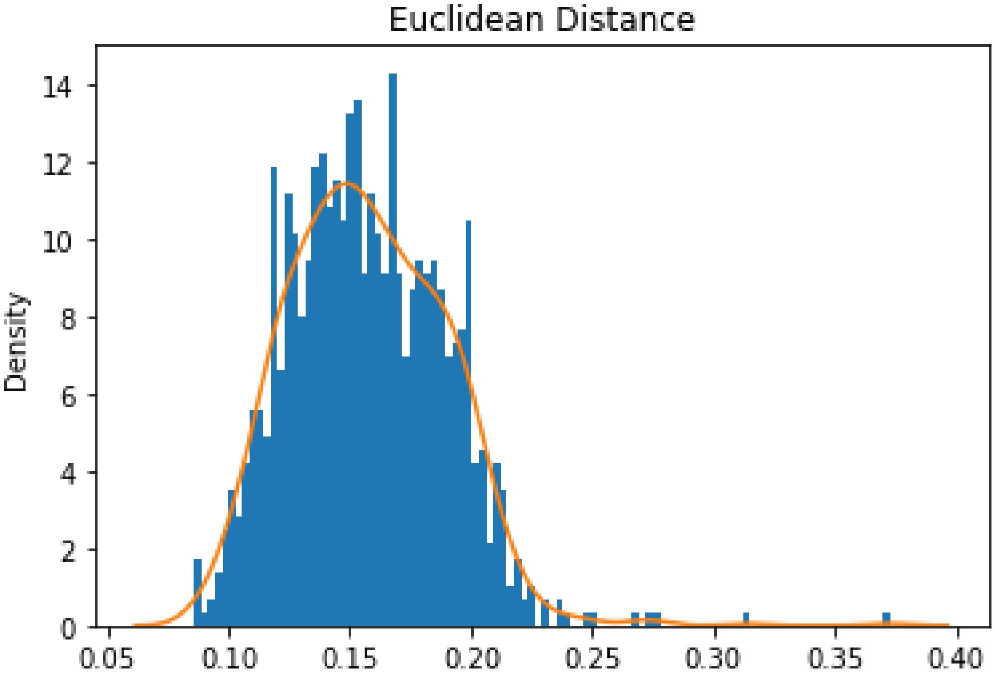
```
Mean Square Error: 0.0025733542154698008
```



```
Mean Absolute Error: 0.039422291231621054
```



```
Mean Manhattan Distance: 0.3942229123162106
```



```
Mean Euclidean Distance: 0.15716514936342832
```

## 2 ABC GAN Model

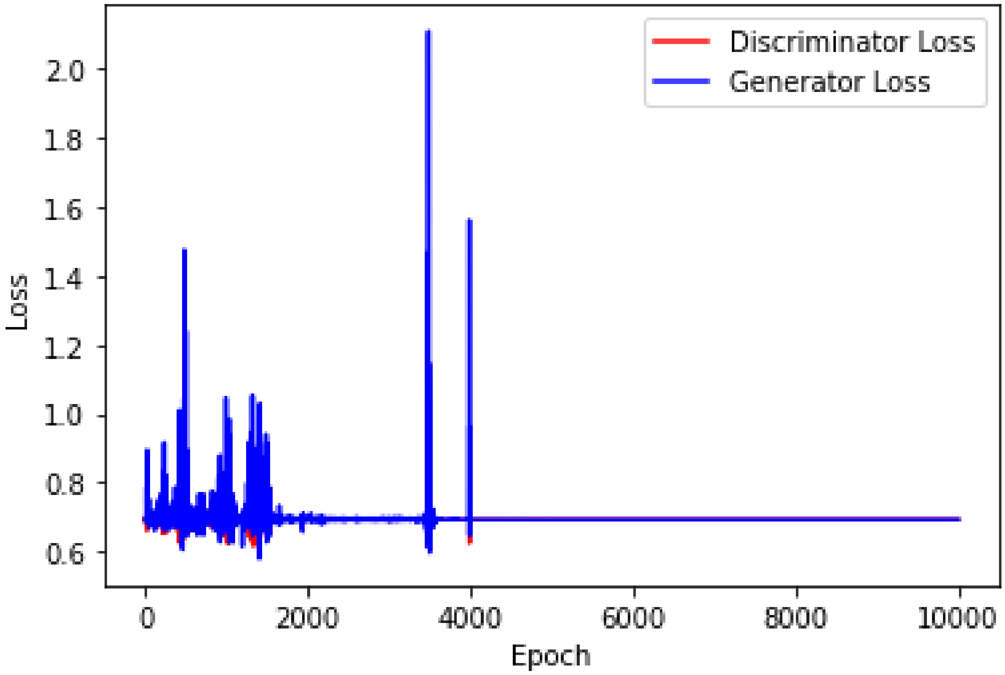
#### 2.0.1 Training the network

**Training ABC-GAN for n_epochs number of epochs**

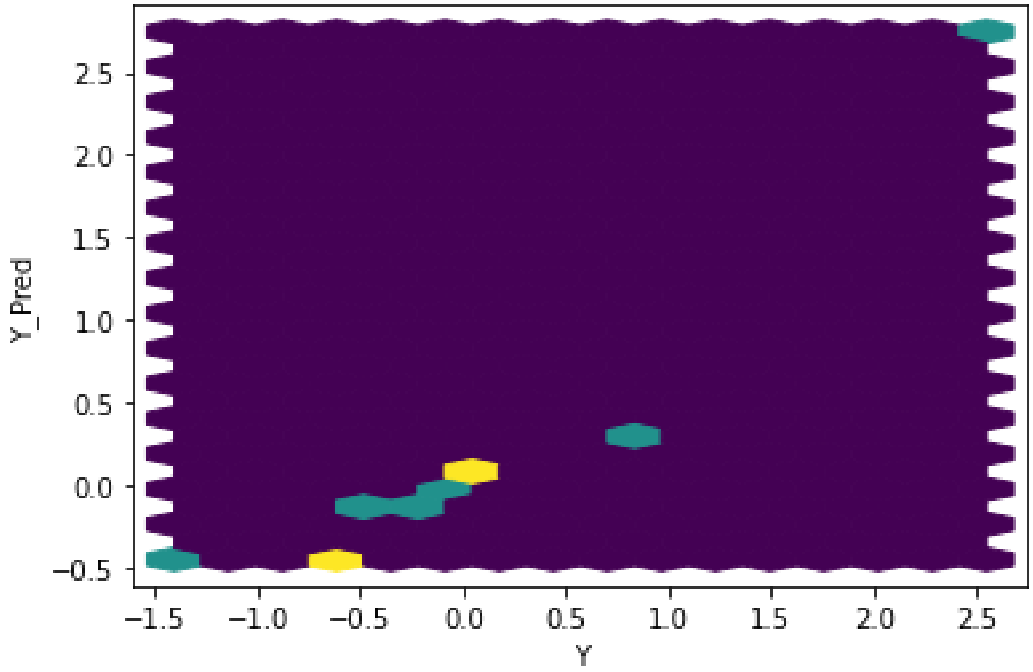
```
[17]: gen = network.Generator(n_features+2)
      disc = network.Discriminator(n_features+2)

      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))
```

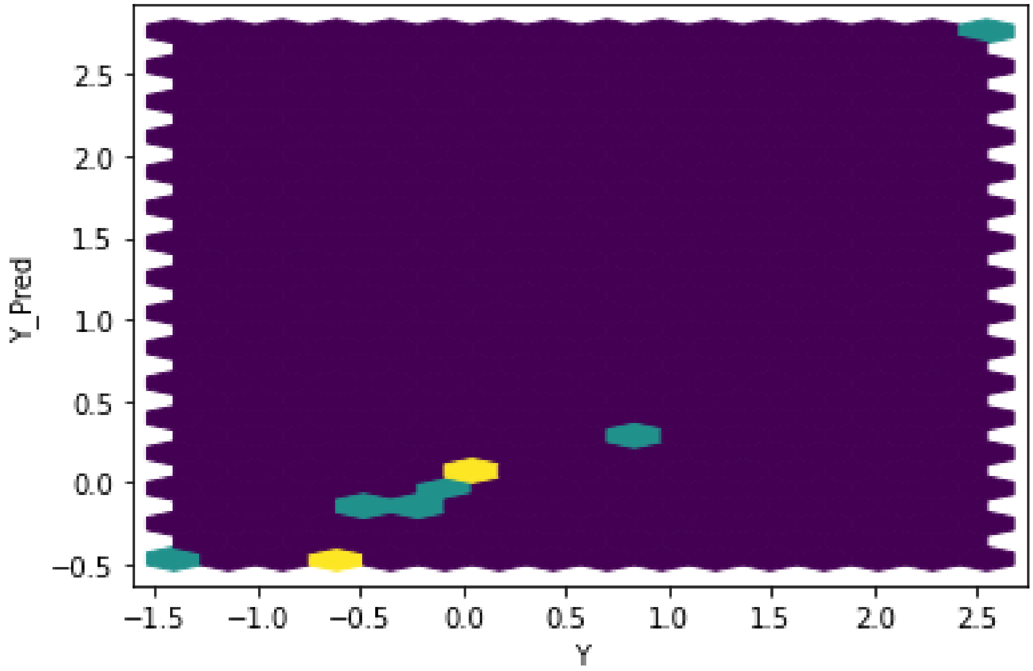
```
[18]: ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,␣
       ↪batch_size, n_epochs,criterion,coeff,mean,variance,device)
```



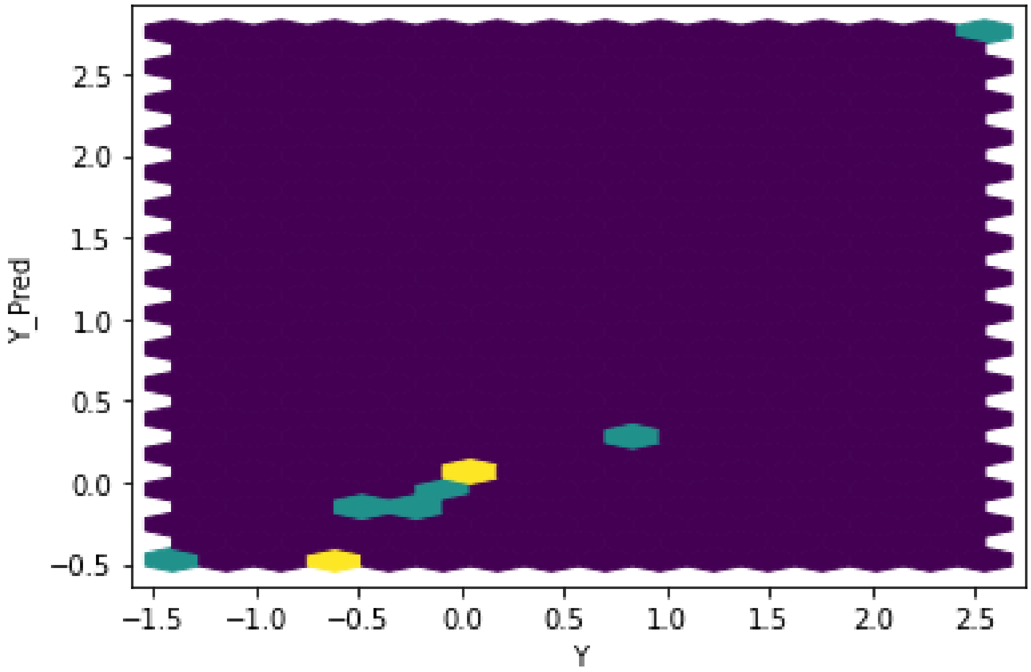
```
[19]: ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

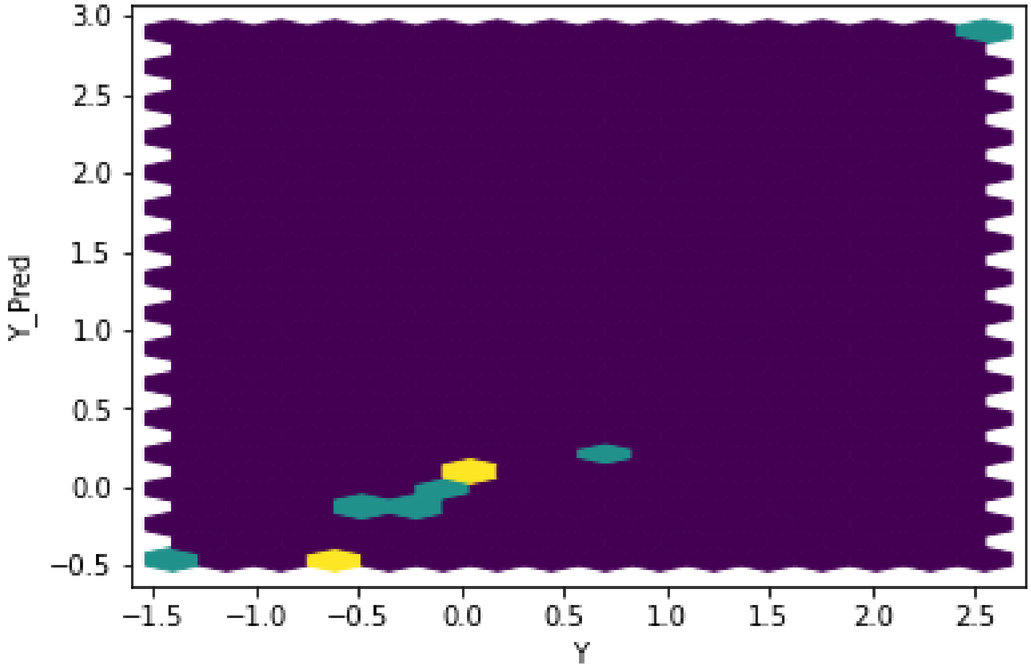


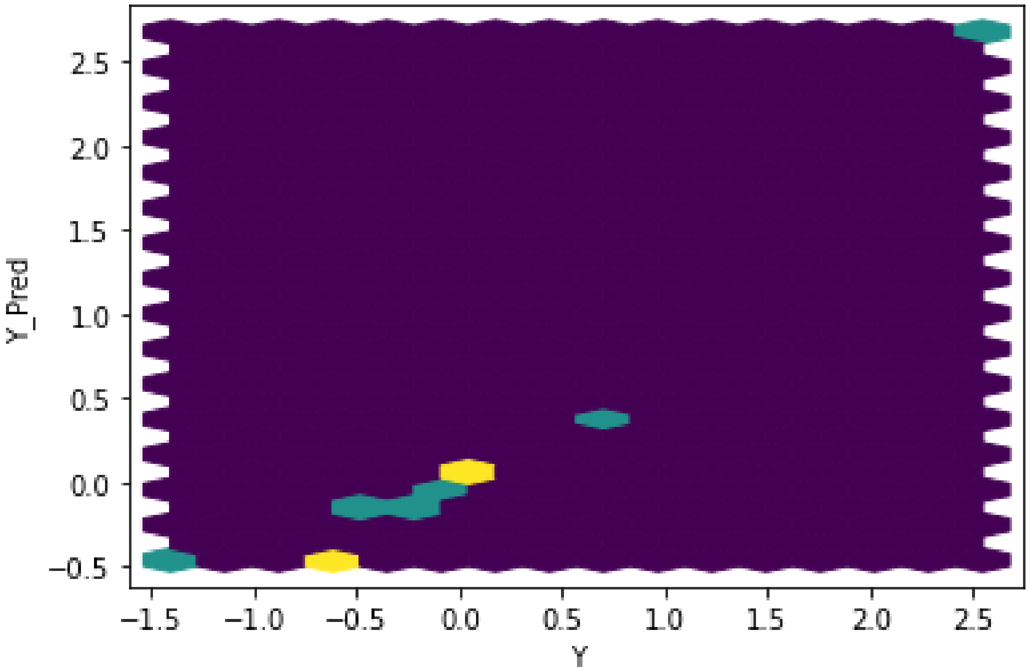
Y_Pred
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Y

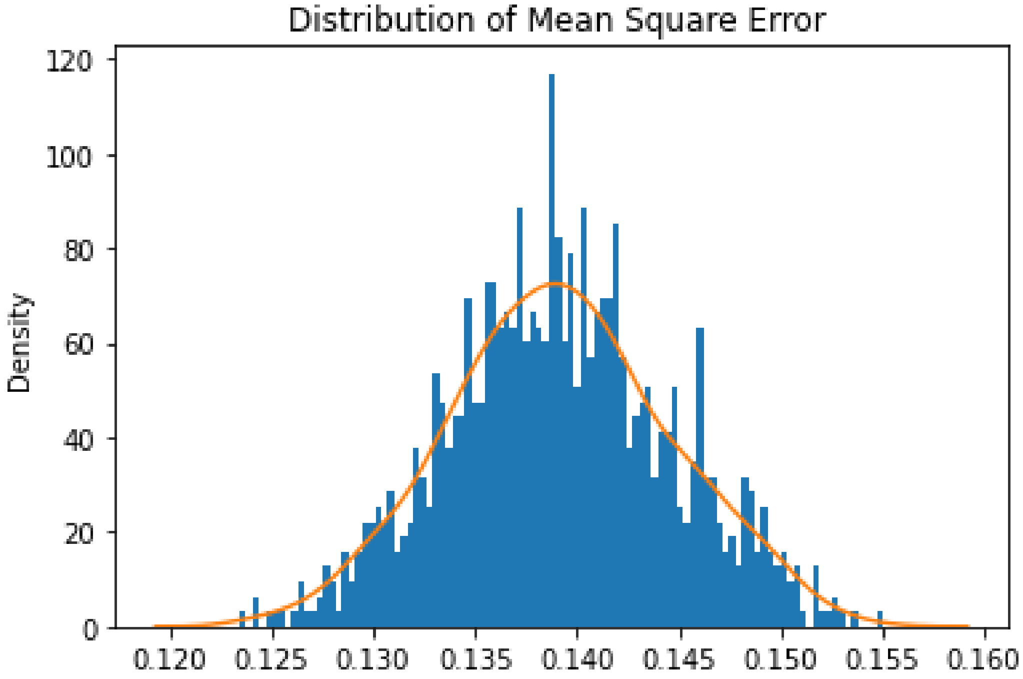


Y_Pred
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Y

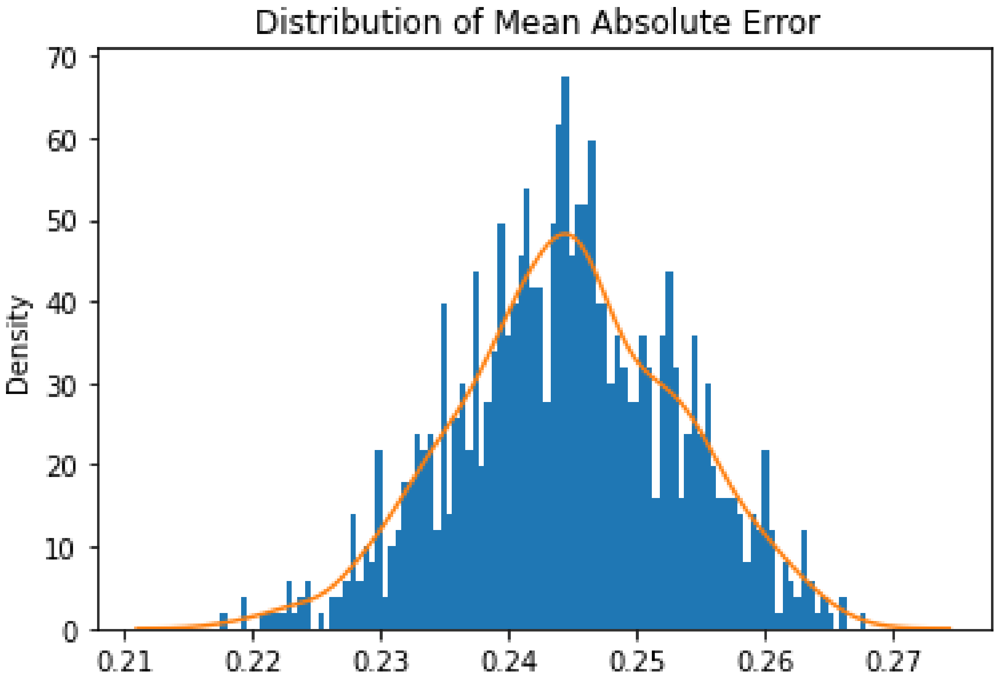

Y_Pred
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Y


Y_Pred
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Y

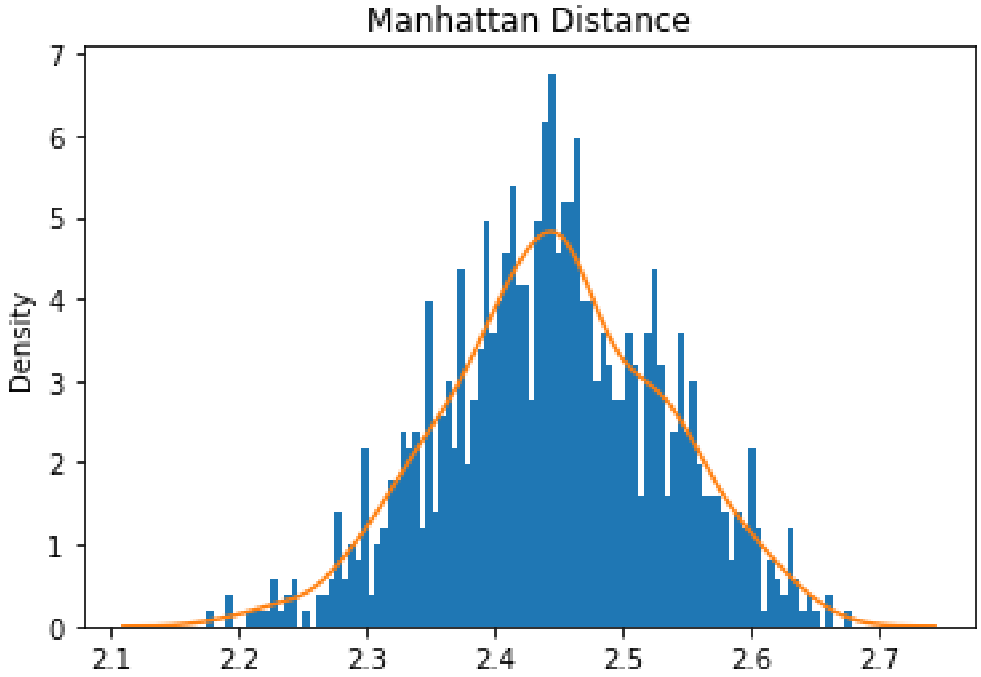




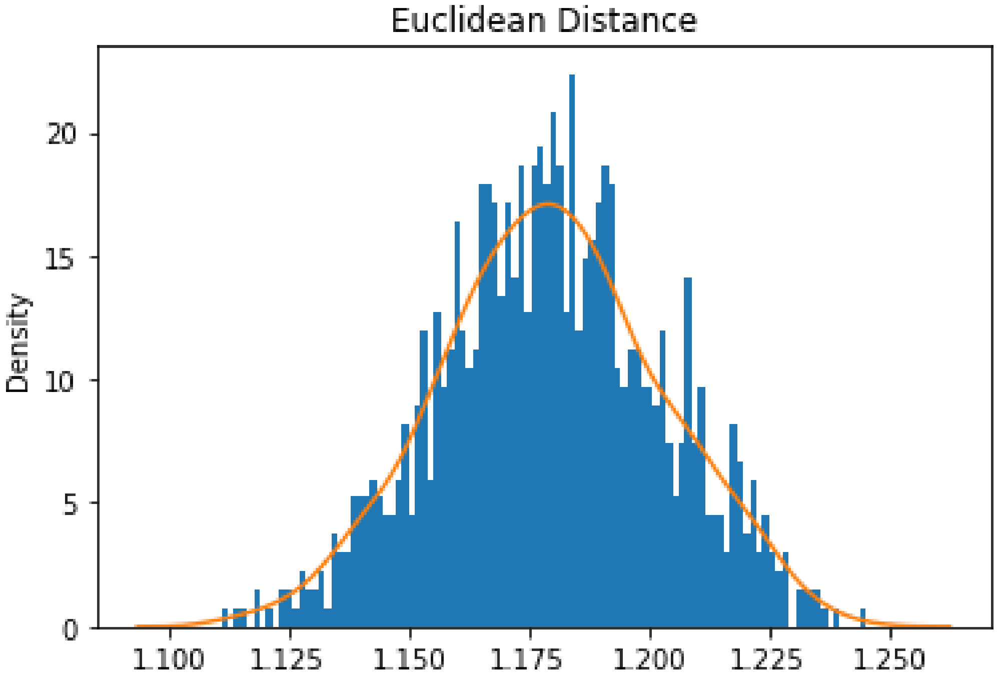
```
Mean Square Error: 0.13916747055912587
```



```
Mean Absolute Error: 0.2445710836187005
Mean Manhattan Distance: 2.445710836187005
```

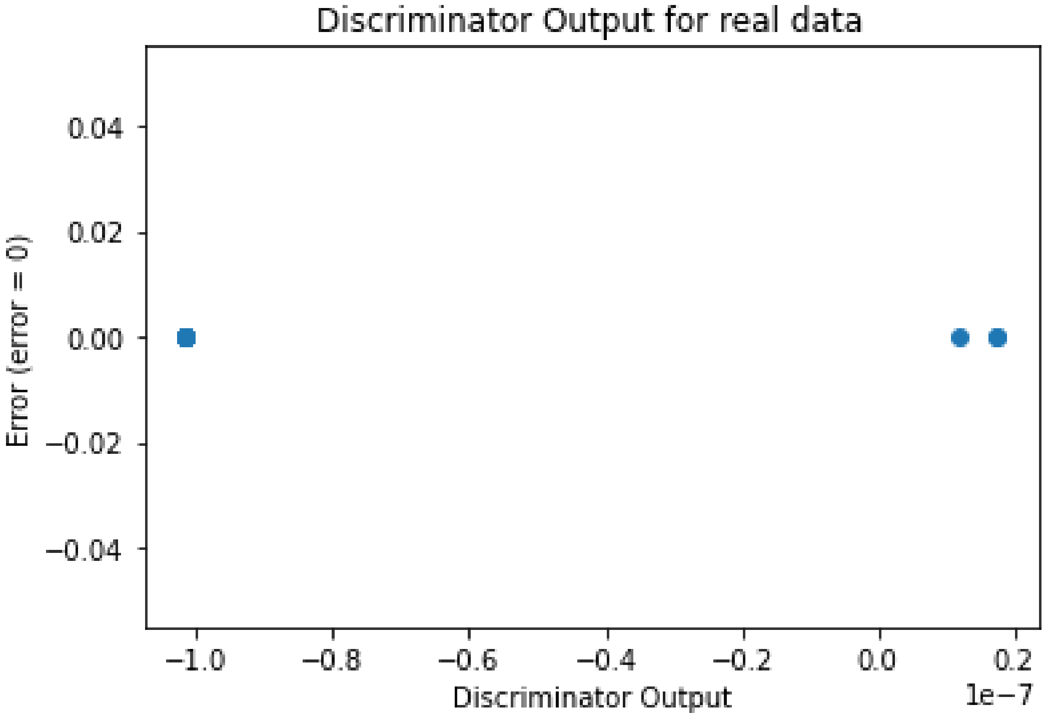


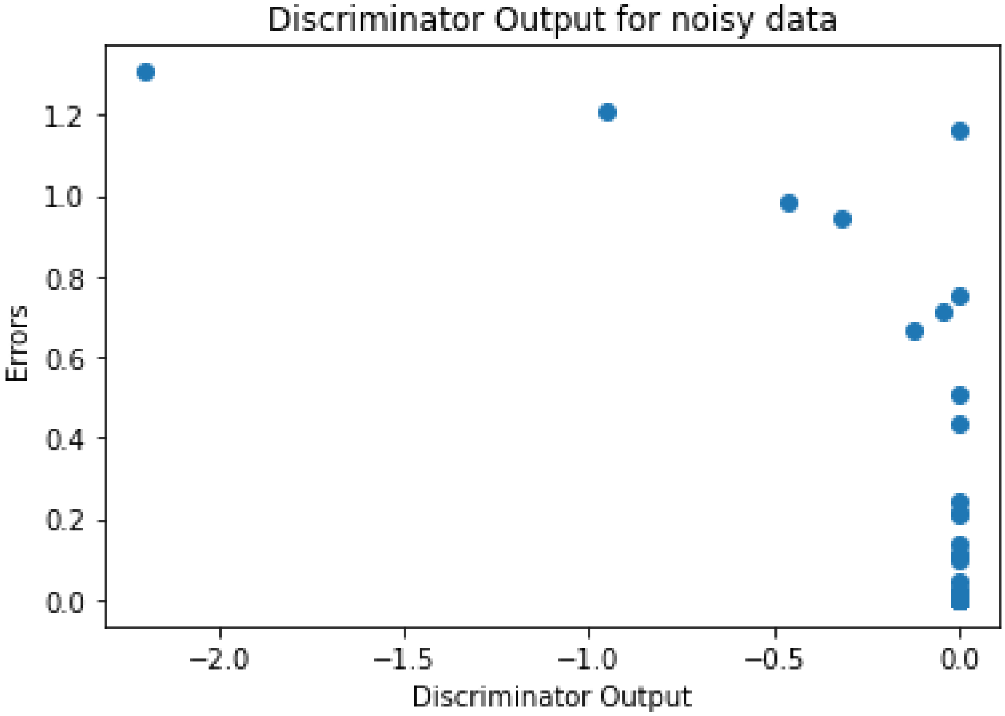
```
Mean Euclidean Distance: 1.1794656171425117
```



**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
−1.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2
1e−7
Discriminator Output


Discriminator Output for noisy data
Errors
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−2.0
−1.5
−1.0
−0.5
0.0
Discriminator Output

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

[21]:

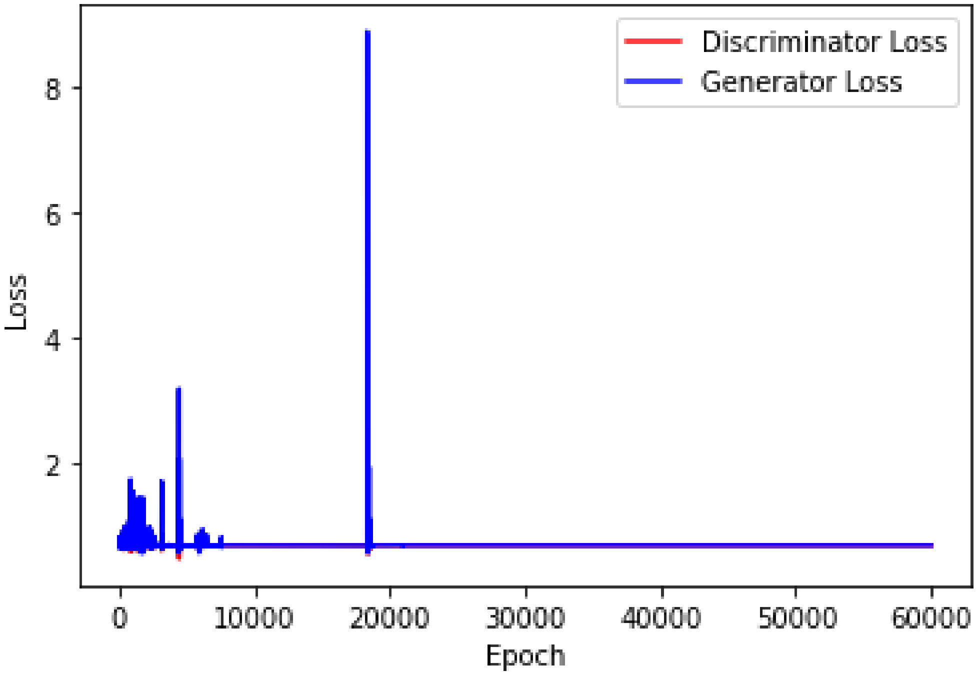
```
gen = network.Generator(n_features+2)
disc = network.Discriminator(n_features+2)

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]:

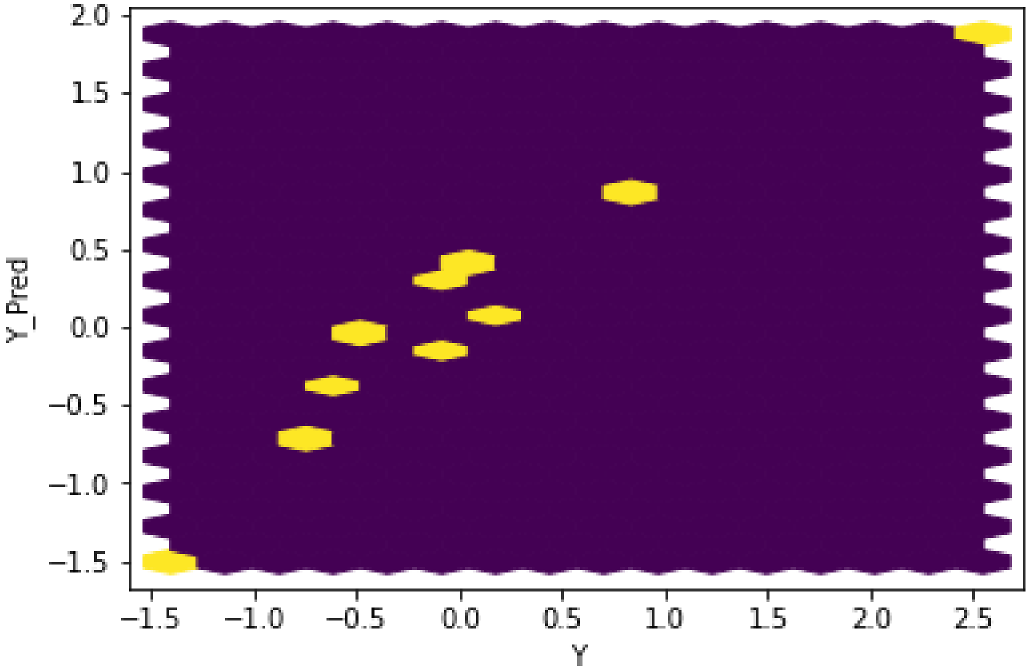
```
ABC_train_test.
 ↪training_GAN_2(disc,gen,disc_opt,gen_opt,real_dataset,batch_size,␣
 ↪error,criterion,coeff,mean,variance,device)
```

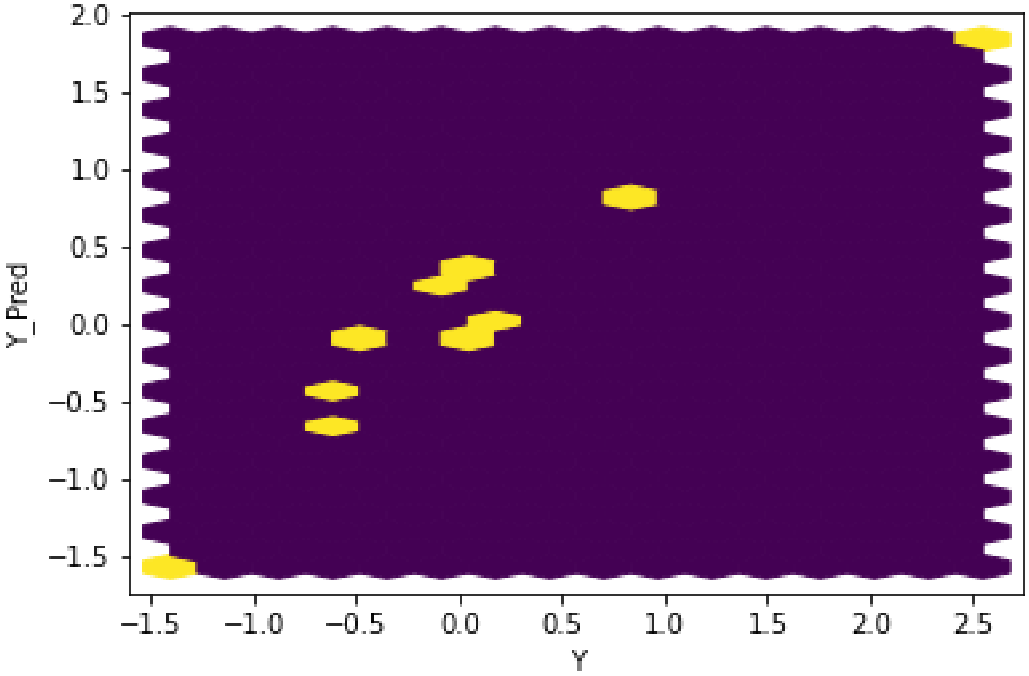
```
Number of epochs 30000
```

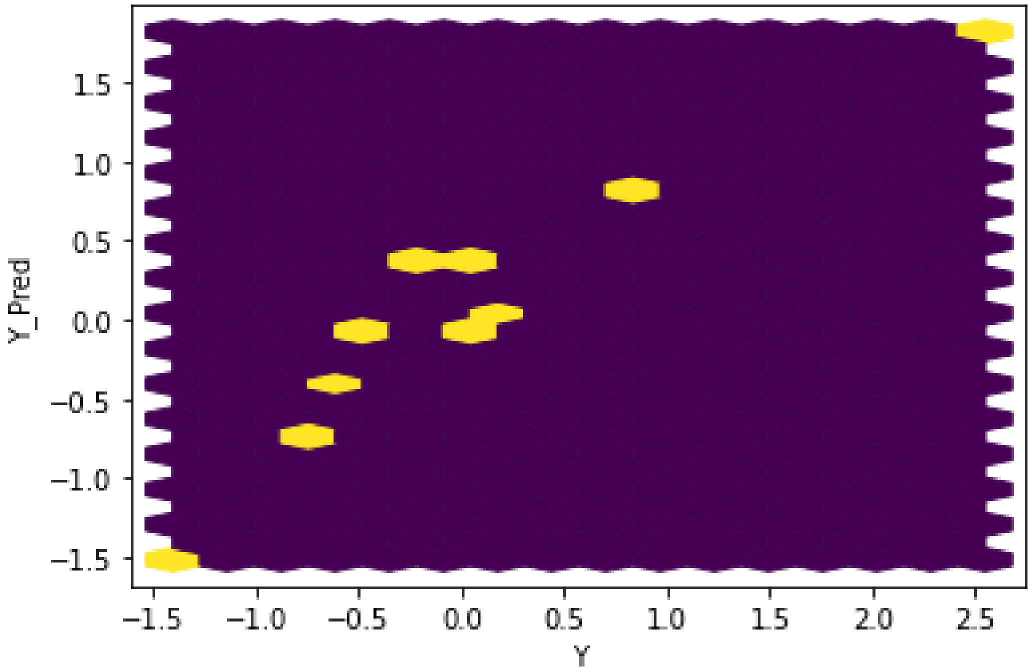


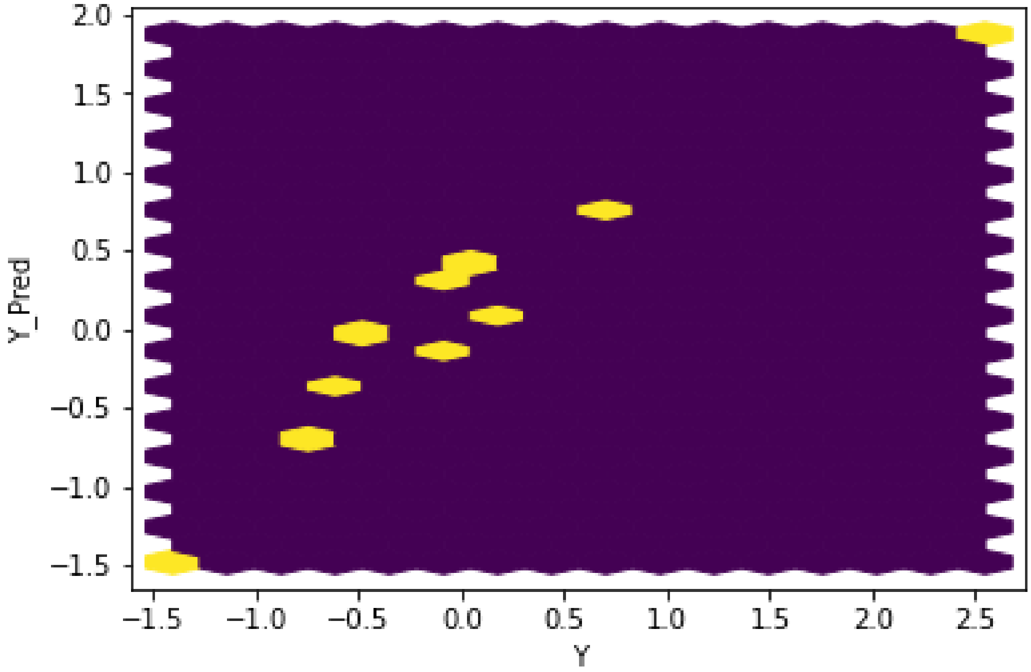
[23]:

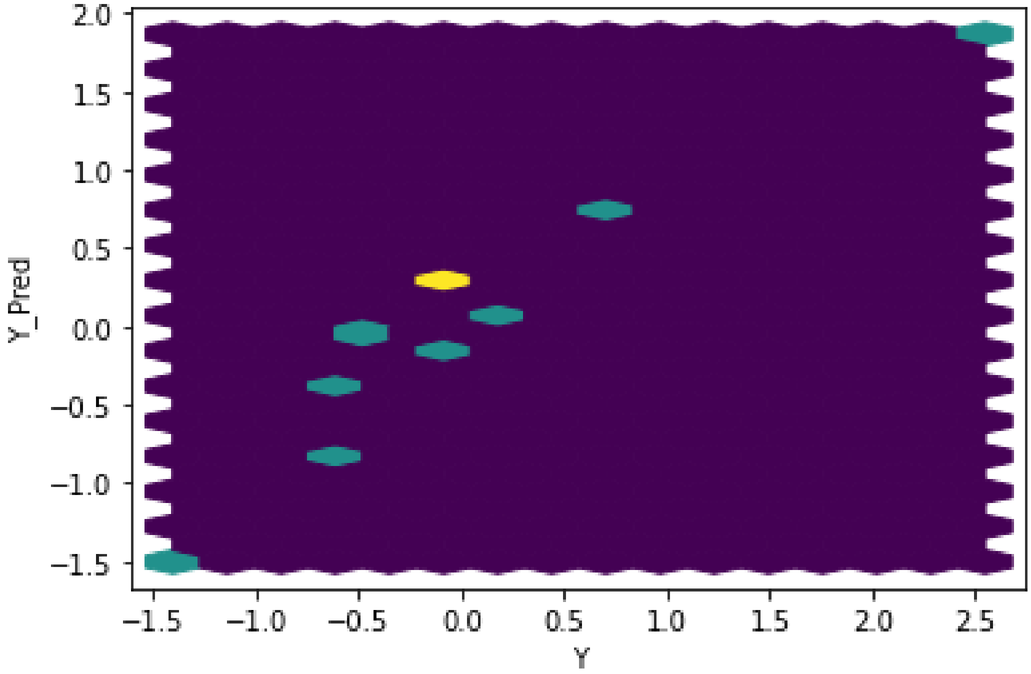
```
ABC_train_test.test_generator(gen,real_dataset,coeff,mean,variance,device)
```

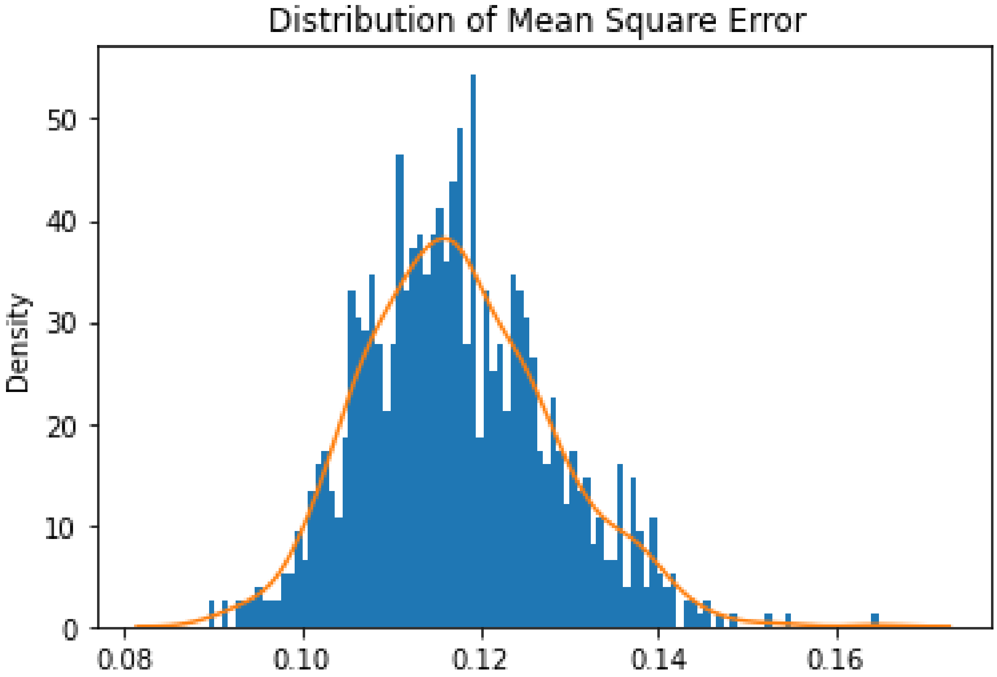

2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Y_Pred
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Y


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Y_Pred
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Y

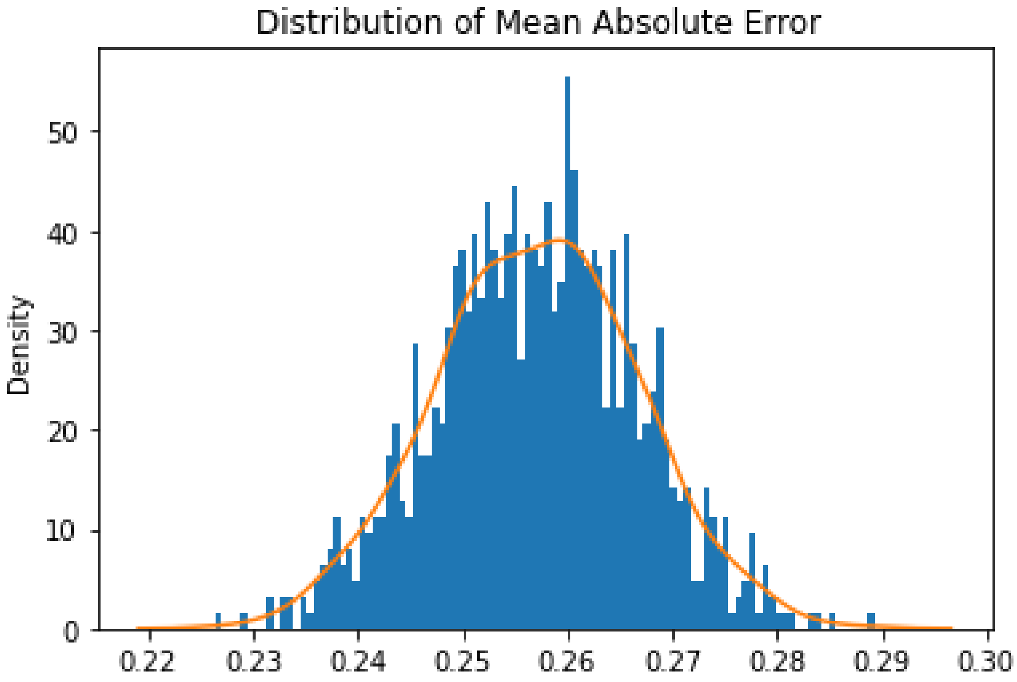

1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Y_Pred
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Y


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Y_Pred
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Y

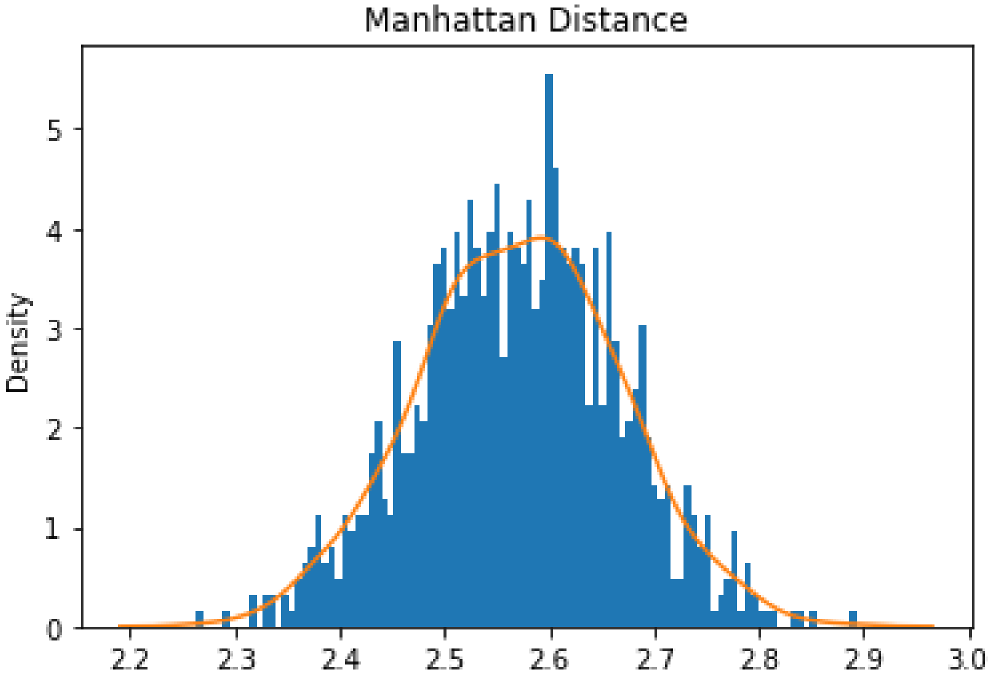




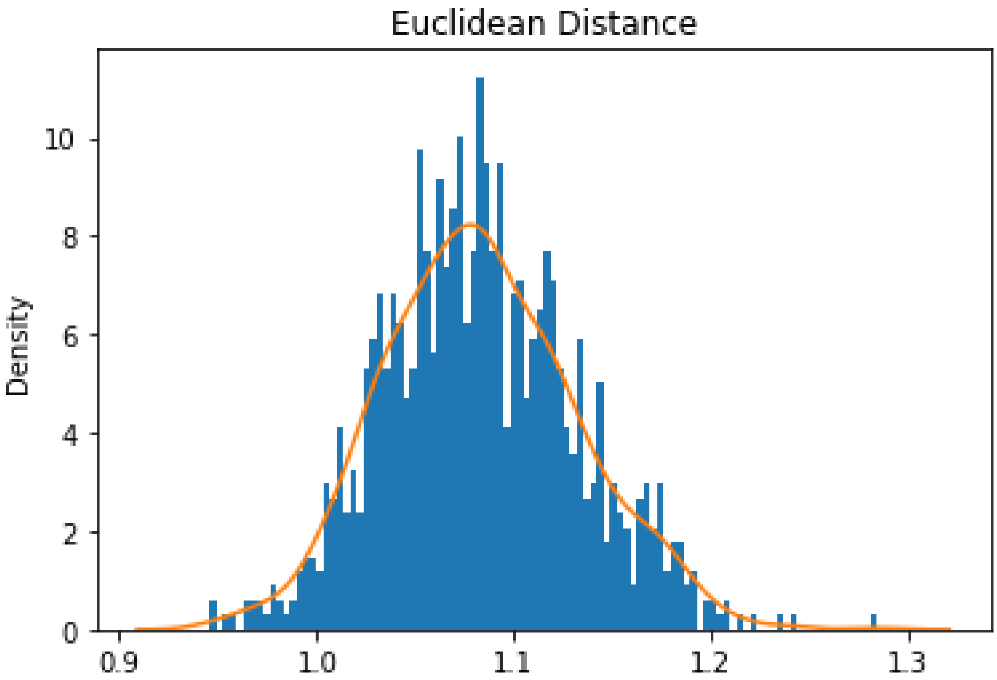
```
Mean Square Error: 0.11749825835970663
```



```
Mean Absolute Error: 0.25708277566358445
Mean Manhattan Distance: 2.5708277566358446
```



```
Mean Euclidean Distance: 1.0828667676284571
```



```
[ ]:
```