dl-eth

April 24, 2024

[1]:

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
import torch.nn.functional as F
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import math
```

[2]:

```
ethereum = pd.read_csv('/kaggle/input/ethereum-2/ETH-USD-2.csv',␣
 ↪index_col='Date')
ethereum
```

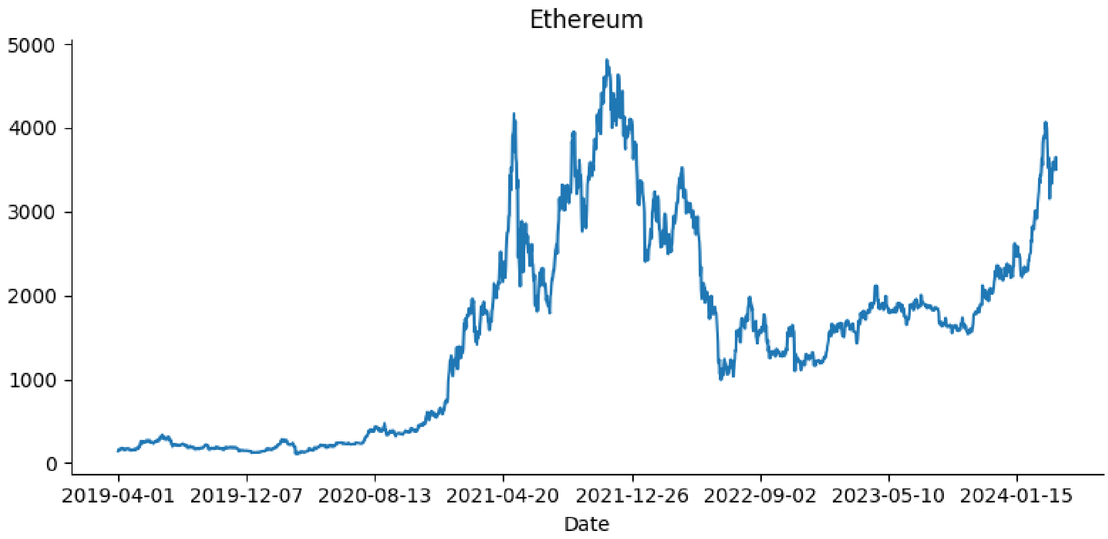
[2]:

```
                   Open         High          Low        Close    Adj Close  \
Date
2019-04-01   141.465485   142.733994   140.737564   141.830322   141.830322
2019-04-02   141.839523   165.226822   141.636459   163.961746   163.961746
2019-04-03   164.008636   178.322052   157.322144   161.458801   161.458801
2019-04-04   161.431763   164.929214   155.241104   158.052536   158.052536
2019-04-05   158.020004   167.220383   157.443954   165.514847   165.514847
…                     …           …           …           …           …
2024-03-28  3500.216064  3609.705322  3465.332275  3561.293945  3561.293945
2024-03-29  3561.011719  3583.701416  3475.725586  3511.806152  3511.806152
2024-03-30  3511.827637  3566.084473  3489.902100  3507.944336  3507.944336
2024-03-31  3507.951660  3655.218994  3507.242676  3647.856445  3647.856445
2024-04-01  3647.819580  3648.129150  3418.695313  3505.030029  3505.030029

                 Volume
Date
2019-04-01   4611999536
2019-04-02   9826645698
2019-04-03  10622456246
2019-04-04   7953123529
2019-04-05   7531316908
…                     …
```

```
2024-03-28  16419674157
2024-03-29  12712701619
2024-03-30   9389066783
2024-03-31  10499881424
2024-04-01  16002098681

[1828 rows x 6 columns]
```

[3]:

```python
ethereum['Close'].plot(kind='line', figsize=(8, 4), title='Ethereum')
plt.tight_layout()
plt.gca().spines[['top', 'right']].set_visible(False)
```



[4]:

```python
def Dataset(data, split=0.8):
    """Function to split the data"""

    data['y'] = data['Close']

    x = data.iloc[:, :6].values
    y = data.iloc[:, 6].values

    split = int(data.shape[0]* split)
    train_x, test_x = x[: split, :], x[split - 20:, :]
    train_y, test_y = y[: split, ], y[split - 20: , ]

    # print(f'trainX: {train_x.shape} trainY: {train_y.shape}')
    # print(f'testX: {test_x.shape} testY: {test_y.shape}')

    x_scaler = MinMaxScaler(feature_range = (0, 1))
    y_scaler = MinMaxScaler(feature_range = (0, 1))

    train_x = x_scaler.fit_transform(train_x)
    test_x = x_scaler.transform(test_x)

    train_y = y_scaler.fit_transform(train_y.reshape(-1, 1))
    test_y = y_scaler.transform(test_y.reshape(-1, 1))

    return train_x, test_x, train_y, test_y
```

```
[5]: ethereum_train_x, ethereum_test_x, ethereum_train_y, ethereum_test_y =␣
       ↪Dataset(ethereum)
     print(f'trainX: {ethereum_train_x.shape} trainY: {ethereum_train_y.shape}')
     print(f'testX: {ethereum_test_x.shape} testY: {ethereum_test_y.shape}')
```

```
trainX: (1462, 6) trainY: (1462, 1)
testX: (386, 6) testY: (386, 1)
```

```
[6]: class VAE(nn.Module):
         def __init__(self, config, latent_dim):
             super().__init__()

             modules = []
             for i in range(1, len(config)):
                 modules.append(
                     nn.Sequential(
                         nn.Linear(config[i - 1], config[i]),
                         nn.ReLU()
                     )
                 )

             self.encoder = nn.Sequential(*modules)
             self.fc_mu = nn.Linear(config[-1], latent_dim)
             self.fc_var = nn.Linear(config[-1], latent_dim)

             modules = []
             self.decoder_input = nn.Linear(latent_dim, config[-1])

             for i in range(len(config) - 1, 1, -1):
                 modules.append(
                     nn.Sequential(
                         nn.Linear(config[i], config[i - 1]),
                         nn.ReLU()
                     )
                 )
             modules.append(
                 nn.Sequential(
                     nn.Linear(config[1], config[0]),
                     nn.Sigmoid()
                 )
             )

             self.decoder = nn.Sequential(*modules)

         def encode(self, x):
             result = self.encoder(x)
             mu = self.fc_mu(result)
             logVar = self.fc_var(result)
             return mu, logVar

         def decode(self, x):
             result = self.decoder(x)
             return result

         def reparameterize(self, mu, logVar):
             std = torch.exp(0.5* logVar)
             eps = torch.randn_like(std)
             return eps * std + mu

         def forward(self, x):
             mu, logVar = self.encode(x)
             z = self.reparameterize(mu, logVar)
             output = self.decode(z)
             return output, z, mu, logVar
```

[7]:

```
train_loader = DataLoader(TensorDataset(torch.from_numpy(ethereum_train_x).
 ↪float()), batch_size = 128, shuffle = False)
model = VAE([6, 400, 400, 400, 10], 10)
```

[8]:

```
use_cuda = 1
device = torch.device("cuda" if (torch.cuda.is_available() & use_cuda) else␣
 ↪"cpu")
num_epochs = 300
learning_rate = 0.00003
model = model.to(device)
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)

hist = np.zeros(num_epochs)
for epoch in range(num_epochs):
    total_loss = 0
    loss_ = []
    for (x, ) in train_loader:
        x = x.to(device)
        output, z, mu, logVar = model(x)
        kl_divergence = 0.5* torch.sum(-1 - logVar + mu.pow(2) + logVar.exp())
        loss = F.binary_cross_entropy(output, x) + kl_divergence
        loss.backward()
        optimizer.step()
        loss_.append(loss.item())
    hist[epoch] = sum(loss_)
    print('[{}/{}] Loss:'.format(epoch+1, num_epochs), sum(loss_))

plt.figure(figsize=(6, 6))
plt.plot(hist)
```

```
[1/300] Loss: 417.9039430618286
[2/300] Loss: 390.82844829559326
[3/300] Loss: 365.83587646484375
[4/300] Loss: 344.05071544647217
[5/300] Loss: 328.8708257675171
[6/300] Loss: 323.54253005981445
[7/300] Loss: 326.9771919250488
[8/300] Loss: 330.2842073440552
[9/300] Loss: 324.8374195098877
[10/300] Loss: 313.1388740539551
[11/300] Loss: 302.32001876831055
[12/300] Loss: 295.58713817596436
[13/300] Loss: 292.1310787200928
[14/300] Loss: 290.243688583374
[15/300] Loss: 288.55703258514404
[16/300] Loss: 286.1344356536865
[17/300] Loss: 282.4678649902344
[18/300] Loss: 277.53676319122314
[19/300] Loss: 272.1165027618408
[20/300] Loss: 267.23609828948975
[21/300] Loss: 263.64473056793213
[22/300] Loss: 261.14090061187744
[23/300] Loss: 258.60572147369385
[24/300] Loss: 254.53058052062988
[25/300] Loss: 248.42494678497314
[26/300] Loss: 241.3482484817505
[27/300] Loss: 234.929594039917
[28/300] Loss: 230.66348838806152
[29/300] Loss: 229.0065402984619
[30/300] Loss: 229.01826667785645
[31/300] Loss: 229.21364879608154
[32/300] Loss: 228.66716480255127
[33/300] Loss: 226.5842523574829
[34/300] Loss: 222.65360069274902
[35/300] Loss: 216.81375217437744
[36/300] Loss: 209.5296812057495
```

```
[37/300] Loss: 201.61739253997803
[38/300] Loss: 194.09895133972168
[39/300] Loss: 188.2283205986023
[40/300] Loss: 184.85946226119995
[41/300] Loss: 184.38644313812256
[42/300] Loss: 185.90603351593018
[43/300] Loss: 187.74462938308716
[44/300] Loss: 187.83210945129395
[45/300] Loss: 184.99164199829102
[46/300] Loss: 179.64986896514893
[47/300] Loss: 173.6965913772583
[48/300] Loss: 168.970383644104
[49/300] Loss: 166.35696744918823
[50/300] Loss: 165.60309505462646
[51/300] Loss: 165.7787618637085
[52/300] Loss: 165.76652336120605
[53/300] Loss: 164.77618837356567
[54/300] Loss: 162.35105657577515
[55/300] Loss: 158.48567867279053
[56/300] Loss: 153.46345233917236
[57/300] Loss: 147.94287109375
[58/300] Loss: 142.9292917251587
[59/300] Loss: 139.4278860092163
[60/300] Loss: 138.2641019821167
[61/300] Loss: 139.7170753479004
[62/300] Loss: 143.18951797485352
[63/300] Loss: 146.92874336242676
[64/300] Loss: 148.67775630950928
[65/300] Loss: 146.56606340408325
[66/300] Loss: 140.32100200653076
[67/300] Loss: 131.84946632385254
[68/300] Loss: 123.66675758361816
[69/300] Loss: 117.87572860717773
[70/300] Loss: 113.87204456329346
[71/300] Loss: 111.4940595626831
[72/300] Loss: 110.80289888381958
[73/300] Loss: 111.32046556472778
[74/300] Loss: 112.05092573165894
[75/300] Loss: 112.45068645477295
[76/300] Loss: 112.12041902542114
[77/300] Loss: 110.78953886032104
[78/300] Loss: 108.44825553894043
[79/300] Loss: 105.35074996948242
[80/300] Loss: 101.81970691680908
[81/300] Loss: 98.16007614135742
[82/300] Loss: 94.63558149337769
[83/300] Loss: 91.57139205932617
[84/300] Loss: 88.99548959732056
```

```
[85/300] Loss: 86.82885432243347
[86/300] Loss: 84.84273290634155
[87/300] Loss: 82.8149688243866
[88/300] Loss: 80.56214332580566
[89/300] Loss: 78.0407612323761
[90/300] Loss: 75.35360312461853
[91/300] Loss: 72.64293622970581
[92/300] Loss: 70.12079572677612
[93/300] Loss: 68.04391717910767
[94/300] Loss: 66.5478765964508
[95/300] Loss: 65.72392511367798
[96/300] Loss: 65.42858624458313
[97/300] Loss: 65.60278749465942
[98/300] Loss: 65.9159779548645
[99/300] Loss: 66.18312215805054
[100/300] Loss: 66.15885877609253
[101/300] Loss: 65.61669874191284
[102/300] Loss: 64.46716523170471
[103/300] Loss: 62.68437838554382
[104/300] Loss: 60.35617208480835
[105/300] Loss: 57.589810848236084
[106/300] Loss: 54.53690147399902
[107/300] Loss: 51.43278646469116
[108/300] Loss: 48.54409313201904
[109/300] Loss: 46.091654777526855
[110/300] Loss: 44.16968512535095
[111/300] Loss: 43.00642538070679
[112/300] Loss: 42.54198718070984
[113/300] Loss: 42.588077545166016
[114/300] Loss: 42.86305117607117
[115/300] Loss: 43.066354274749756
[116/300] Loss: 42.8084716796875
[117/300] Loss: 41.96073126792908
[118/300] Loss: 40.50636434555054
[119/300] Loss: 38.67799949645996
[120/300] Loss: 36.69609475135803
[121/300] Loss: 34.90401291847229
[122/300] Loss: 33.49766802787781
[123/300] Loss: 32.57578766345978
[124/300] Loss: 32.102619647979736
[125/300] Loss: 32.0319344997406
[126/300] Loss: 32.207504749298096
[127/300] Loss: 32.452343344688416
[128/300] Loss: 32.63554549217224
[129/300] Loss: 32.647231101989746
[130/300] Loss: 32.39835059642792
[131/300] Loss: 31.845906257629395
[132/300] Loss: 30.953301668167114
```

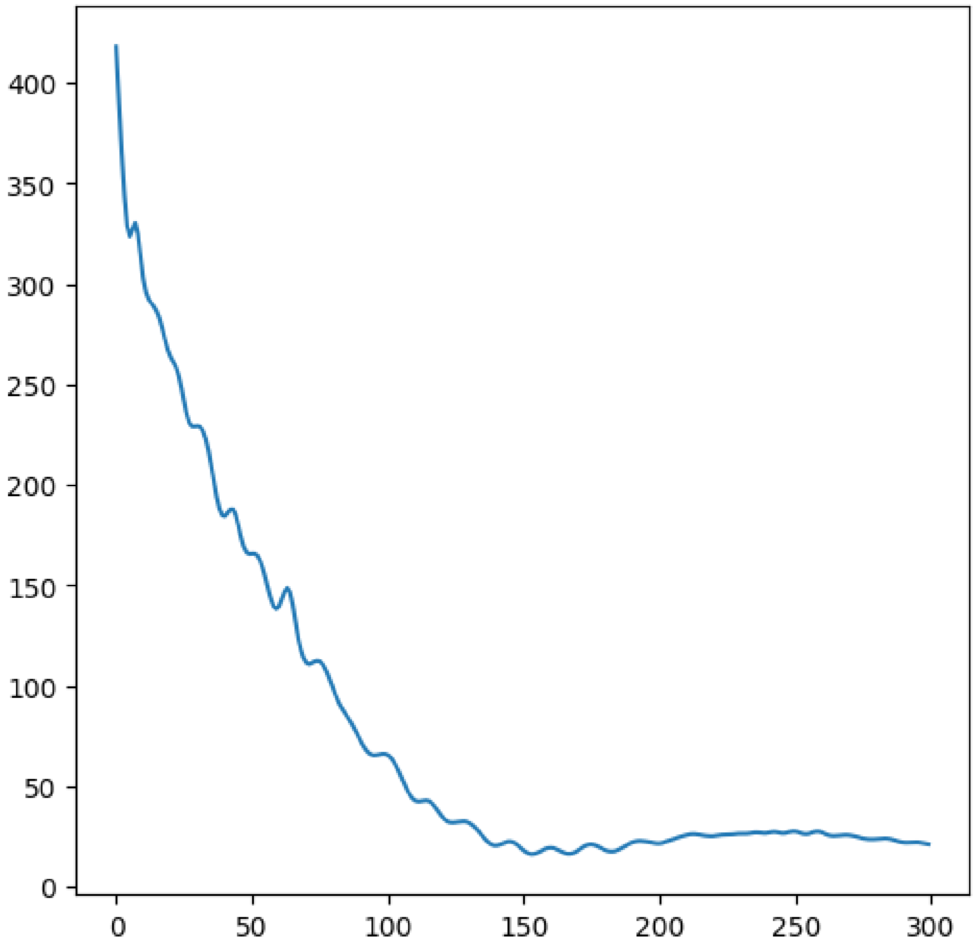
```
[133/300] Loss: 29.751924514770508
[134/300] Loss: 28.30819320678711
[135/300] Loss: 26.70137393474579
[136/300] Loss: 25.06140625476837
[137/300] Loss: 23.50759470462799
[138/300] Loss: 22.194298148155212
[139/300] Loss: 21.221676111221313
[140/300] Loss: 20.69800055027008
[141/300] Loss: 20.630918860435486
[142/300] Loss: 20.94594633579254
[143/300] Loss: 21.489707827568054
[144/300] Loss: 22.054504990577698
[145/300] Loss: 22.48168182373047
[146/300] Loss: 22.58911955356598
[147/300] Loss: 22.289634585380554
[148/300] Loss: 21.591177940368652
[149/300] Loss: 20.566193222999573
[150/300] Loss: 19.347059726715088
[151/300] Loss: 18.165254950523376
[152/300] Loss: 17.158498525619507
[153/300] Loss: 16.498219192028046
[154/300] Loss: 16.24907898902893
[155/300] Loss: 16.39651083946228
[156/300] Loss: 16.893636107444763
[157/300] Loss: 17.579067707061768
[158/300] Loss: 18.325476944446564
[159/300] Loss: 18.97124147415161
[160/300] Loss: 19.427056074142456
[161/300] Loss: 19.540920615196228
[162/300] Loss: 19.388068735599518
[163/300] Loss: 18.869509160518646
[164/300] Loss: 18.201465845108032
[165/300] Loss: 17.45719015598297
[166/300] Loss: 16.84767174720764
[167/300] Loss: 16.46464204788208
[168/300] Loss: 16.400920033454895
[169/300] Loss: 16.64945936203003
[170/300] Loss: 17.25240731239319
[171/300] Loss: 18.0911226272583
[172/300] Loss: 19.053759455680847
[173/300] Loss: 19.974342823028564
[174/300] Loss: 20.708818435668945
[175/300] Loss: 21.148772597312927
[176/300] Loss: 21.241952776908875
[177/300] Loss: 20.965591430664062
[178/300] Loss: 20.41844952106476
[179/300] Loss: 19.73277246952057
[180/300] Loss: 19.005130529403687
```

```
[181/300] Loss: 18.34029257297516
[182/300] Loss: 17.851243257522583
[183/300] Loss: 17.614379048347473
[184/300] Loss: 17.637797951698303
[185/300] Loss: 17.984108924865723
[186/300] Loss: 18.525052547454834
[187/300] Loss: 19.227921962738037
[188/300] Loss: 20.04885196685791
[189/300] Loss: 20.854984045028687
[190/300] Loss: 21.570236444473267
[191/300] Loss: 22.154615879058838
[192/300] Loss: 22.580048084259033
[193/300] Loss: 22.83323574066162
[194/300] Loss: 22.885164499282837
[195/300] Loss: 22.786263704299927
[196/300] Loss: 22.581645250320435
[197/300] Loss: 22.33285892009735
[198/300] Loss: 22.06628978252411
[199/300] Loss: 21.826918601989746
[200/300] Loss: 21.667649507522583
[201/300] Loss: 21.65074396133423
[202/300] Loss: 21.87996232509613
[203/300] Loss: 22.34664273262024
[204/300] Loss: 22.90319335460663
[205/300] Loss: 23.388299345970154
[206/300] Loss: 23.814812898635864
[207/300] Loss: 24.217304706573486
[208/300] Loss: 24.634546160697937
[209/300] Loss: 25.054832458496094
[210/300] Loss: 25.476171016693115
[211/300] Loss: 25.84227967262268
[212/300] Loss: 26.150442361831665
[213/300] Loss: 26.293853044509888
[214/300] Loss: 26.294299125671387
[215/300] Loss: 26.15122377872467
[216/300] Loss: 25.933248043060303
[217/300] Loss: 25.68055272102356
[218/300] Loss: 25.451624393463135
[219/300] Loss: 25.295194506645203
[220/300] Loss: 25.222488045692444
[221/300] Loss: 25.3159157037735
[222/300] Loss: 25.55912721157074
[223/300] Loss: 25.810415029525757
[224/300] Loss: 25.94995617866516
[225/300] Loss: 26.006983757019043
[226/300] Loss: 26.024738550186157
[227/300] Loss: 26.072431206703186
[228/300] Loss: 26.185575485229492
```

```
[229/300] Loss: 26.345511317253113
[230/300] Loss: 26.492823719978333
[231/300] Loss: 26.55469846725464
[232/300] Loss: 26.53150761127472
[233/300] Loss: 26.530428647994995
[234/300] Loss: 26.646130084991455
[235/300] Loss: 26.85608434677124
[236/300] Loss: 27.081844329833984
[237/300] Loss: 27.096721291542053
[238/300] Loss: 26.956592082977295
[239/300] Loss: 26.794275164604187
[240/300] Loss: 26.765549421310425
[241/300] Loss: 26.94442903995514
[242/300] Loss: 27.228275299072266
[243/300] Loss: 27.42932963371277
[244/300] Loss: 27.335834980010986
[245/300] Loss: 27.022144317626953
[246/300] Loss: 26.772724270820618
[247/300] Loss: 26.772239089012146
[248/300] Loss: 27.06317663192749
[249/300] Loss: 27.4270259141922
[250/300] Loss: 27.66939377784729
[251/300] Loss: 27.633597373962402
[252/300] Loss: 27.30612564086914
[253/300] Loss: 26.835982084274292
[254/300] Loss: 26.445947885513306
[255/300] Loss: 26.31533920764923
[256/300] Loss: 26.512107372283936
[257/300] Loss: 26.941953420639038
[258/300] Loss: 27.398836135864258
[259/300] Loss: 27.634369134902954
[260/300] Loss: 27.49129319190979
[261/300] Loss: 27.0028418302536
[262/300] Loss: 26.3871808052063
[263/300] Loss: 25.855555772781372
[264/300] Loss: 25.51343584060669
[265/300] Loss: 25.37414824962616
[266/300] Loss: 25.410558700561523
[267/300] Loss: 25.542433857917786
[268/300] Loss: 25.70699954032898
[269/300] Loss: 25.822691559791565
[270/300] Loss: 25.8378164768219
[271/300] Loss: 25.733574271202087
[272/300] Loss: 25.508376240730286
[273/300] Loss: 25.180721879005432
[274/300] Loss: 24.80367410182953
[275/300] Loss: 24.42489743232727
[276/300] Loss: 24.094566822052002
```

```
[277/300] Loss: 23.849444150924683
[278/300] Loss: 23.71059226989746
[279/300] Loss: 23.663193345069885
[280/300] Loss: 23.705860018730164
[281/300] Loss: 23.802362084388733
[282/300] Loss: 23.93418252468109
[283/300] Loss: 24.055481910705566
[284/300] Loss: 24.10930907726288
[285/300] Loss: 24.042463302612305
[286/300] Loss: 23.813526511192322
[287/300] Loss: 23.464398503303528
[288/300] Loss: 23.03538191318512
[289/300] Loss: 22.619827151298523
[290/300] Loss: 22.29394233226776
[291/300] Loss: 22.102742552757263
[292/300] Loss: 22.039245128631592
[293/300] Loss: 22.097647190093994
[294/300] Loss: 22.19353151321411
[295/300] Loss: 22.263108611106873
[296/300] Loss: 22.224143147468567
[297/300] Loss: 22.06233775615692
[298/300] Loss: 21.82223868370056
[299/300] Loss: 21.54167926311493
[300/300] Loss: 21.275021076202393
```

```
[8]: [<matplotlib.lines.Line2D at 0x785a9a9d3ee0>]
```



[9]:

```
model.eval()
_, VAE_train_x, train_x_mu, train_x_var = model(torch.from_numpy(ethereum_train_x).float().to(device))
_, VAE_test_x, test_x_mu, test_x_var = model(torch.from_numpy(ethereum_test_x).float().to(device))
```

[10]:

```
def sliding_window(x, y, window):
    x_ = []
    y_ = []
    y_gan = []
    for i in range(window, x.shape[0]):
        tmp_x = x[i - window: i, :]
        tmp_y = y[i]
        tmp_y_gan = y[i - window: i + 1]
        x_.append(tmp_x)
        y_.append(tmp_y)
        y_gan.append(tmp_y_gan)
    x_ = torch.from_numpy(np.array(x_)).float()
    y_ = torch.from_numpy(np.array(y_)).float()
    y_gan = torch.from_numpy(np.array(y_gan)).float()
    return x_, y_, y_gan
```

[11]:

```
ethereum_train_x = np.concatenate((ethereum_train_x, VAE_train_x.cpu().detach().
 ↪numpy()), axis = 1)
ethereum_test_x = np.concatenate((ethereum_test_x, VAE_test_x.cpu().detach().
 ↪numpy()), axis = 1)
```

[12]:

```
e_train_x_slide, e_train_y_slide, e_train_y_gan =␣
 ↪sliding_window(ethereum_train_x, ethereum_train_y, 3)
e_test_x_slide, e_test_y_slide, e_test_y_gan = sliding_window(ethereum_test_x,␣
 ↪ethereum_test_y, 3)
print(f'train_x: {e_train_x_slide.shape} train_y: {e_train_y_slide.shape}␣
 ↪train_y_gan: {e_train_y_gan.shape}')
print(f'test_x: {e_test_x_slide.shape} test_y: {e_test_y_slide.shape}␣
 ↪test_y_gan: {e_test_y_gan.shape}')
```

```
train_x: torch.Size([1459, 3, 16]) train_y: torch.Size([1459, 1]) train_y_gan:
torch.Size([1459, 4, 1])
test_x: torch.Size([383, 3, 16]) test_y: torch.Size([383, 1]) test_y_gan:
torch.Size([383, 4, 1])
```

[13]:

```
class Generator(nn.Module):
    def __init__(self, input_size):
        super().__init__()
        self.gru_1 = nn.GRU(input_size, 1024, batch_first = True)
        self.gru_2 = nn.GRU(1024, 512, batch_first = True)
        self.gru_3 = nn.GRU(512, 256, batch_first = True)
        self.linear_1 = nn.Linear(256, 128)
        self.linear_2 = nn.Linear(128, 64)
        self.linear_3 = nn.Linear(64, 1)
        self.dropout = nn.Dropout(0.2)


    def forward(self, x):
        use_cuda = 1
        device = torch.device("cuda" if (torch.cuda.is_available() & use_cuda)␣
 ↪else "cpu")
        h0 = torch.zeros(1, x.size(0), 1024).to(device)
        out_1, _ = self.gru_1(x, h0)
        out_1 = self.dropout(out_1)
        h1 = torch.zeros(1, x.size(0), 512).to(device)
        out_2, _ = self.gru_2(out_1, h1)
        out_2 = self.dropout(out_2)
        h2 = torch.zeros(1, x.size(0), 256).to(device)
        out_3, _ = self.gru_3(out_2, h2)
        out_3 = self.dropout(out_3)
        out_4 = self.linear_1(out_3[:, -1, :])
        out_5 = self.linear_2(out_4)
        out_6 = self.linear_3(out_5)
        return out_6

class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv1d(4, 32, kernel_size = 5, stride = 1, padding =␣
 ↪'same')
        self.conv2 = nn.Conv1d(32, 64, kernel_size = 5, stride = 1, padding =␣
 ↪'same')
        self.conv3 = nn.Conv1d(64, 128, kernel_size = 5, stride = 1, padding =␣
 ↪'same')
        self.linear1 = nn.Linear(128, 220)
        self.linear2 = nn.Linear(220, 220)
        self.linear3 = nn.Linear(220, 1)
        self.leaky = nn.LeakyReLU(0.01)
        self.relu = nn.ReLU()

    def forward(self, x):
        conv1 = self.conv1(x)
        conv1 = self.leaky(conv1)
        conv2 = self.conv2(conv1)
        conv2 = self.leaky(conv2)
        conv3 = self.conv3(conv2)
        conv3 = self.leaky(conv3)
        flatten_x =  conv3.reshape(conv3.shape[0], conv3.shape[1])
        out_1 = self.linear1(flatten_x)
        out_1 = self.leaky(out_1)
        out_2 = self.linear2(out_1)
        out_2 = self.relu(out_2)
        out = self.linear3(out_2)
        return out
```

[14]:

```
from ast import Break
use_cuda = 1
device = torch.device("cuda" if (torch.cuda.is_available() & use_cuda) else␣
 ↪"cpu")

batch_size = 128
learning_rate = 0.000115
num_epochs = 100
critic_iterations = 5
weight_clip = 0.01
```

```
trainDataloader = DataLoader(TensorDataset(e_train_x_slide, e_train_y_gan),␣
 ↪batch_size = batch_size, shuffle = False)

modelG = Generator(16).to(device)
modelD = Discriminator().to(device)

optimizerG = torch.optim.Adam(modelG.parameters(), lr = learning_rate, betas =␣
 ↪(0.0, 0.9), weight_decay = 1e-3)
optimizerD = torch.optim.Adam(modelD.parameters(), lr = learning_rate, betas =␣
 ↪(0.0, 0.9), weight_decay = 1e-3)

histG = np.zeros(num_epochs)
histD = np.zeros(num_epochs)
count = 0
flag = 0
for epoch in range(num_epochs):
    loss_G = []
    loss_D = []
    for (x, y) in trainDataloader:
        x = x.to(device)
        y = y.to(device)

        fake_data = modelG(x)
        fake_data = torch.cat([y[:, :3, :], fake_data.reshape(-1, 1, 1)], axis␣
 ↪= 1)
        critic_real = modelD(y)
        critic_fake = modelD(fake_data)
        lossD = -(torch.mean(critic_real) - torch.mean(critic_fake))
        modelD.zero_grad()
        lossD.backward(retain_graph = True)
        optimizerD.step()

        output_fake = modelD(fake_data)
        lossG = -torch.mean(output_fake)
        modelG.zero_grad()
        lossG.backward()
        optimizerG.step()

        loss_D.append(lossD.item())
        loss_G.append(lossG.item())

        if np.abs(lossD.item()) < 1e-9 or np.abs(lossG.item()) < 1e-9:
          flag = 1
          break

    histG[epoch] = sum(loss_G)
    histD[epoch] = sum(loss_D)
    print(f'[{epoch+1}/{num_epochs}] LossD: {sum(loss_D)} LossG:{sum(loss_G)}')

    if flag == 1:
      break
```

```
[1/100] LossD: -0.000666031613945961 LossG:-0.2005807738751173
[2/100] LossD: -0.0026543578132987022 LossG:-0.18202272150665522
[3/100] LossD: -0.004352514632046223 LossG:-0.15882108639925718
[4/100] LossD: -0.0055821677669882774 LossG:-0.1327707851305604
[5/100] LossD: -0.006835516542196274 LossG:-0.10793979372829199
[6/100] LossD: -0.007679302711039782 LossG:-0.08314626850187778
[7/100] LossD: -0.007753927959129214 LossG:-0.05606777290813625
[8/100] LossD: -0.007311774184927344 LossG:-0.03079394856467843
[9/100] LossD: -0.005936992587521672 LossG:-0.01187271805247292
[10/100] LossD: -0.0038584147405344993 LossG:0.005194780904275831
[11/100] LossD: -0.0016011727275326848 LossG:0.019176848349161446
[12/100] LossD: 0.00016296422109007835 LossG:0.03551756986416876
[13/100] LossD: 0.0012317553628236055 LossG:0.04306064988486469
[14/100] LossD: 0.0016872880514711142 LossG:0.05197926424443722
[15/100] LossD: 0.0012846579775214195 LossG:0.05509061040356755
[16/100] LossD: 0.0006416963879019022 LossG:0.056602475233376026
[17/100] LossD: 4.5159365981817245e-05 LossG:0.055771195795387O3
[18/100] LossD: -0.0005900934338569641 LossG:0.05493453564122319
[19/100] LossD: -0.0010178268421441317 LossG:0.05674670822918415
[20/100] LossD: -0.001553150126710534 LossG:0.05881395284086466
[21/100] LossD: -0.001962540205568075 LossG:0.06151173822581768
[22/100] LossD: -0.0021944460459053516 LossG:0.0640257210470736
[23/100] LossD: -0.002586481161415577 LossG:0.06436203420162201
[24/100] LossD: -0.002832704223692417 LossG:0.06537824356928468
[25/100] LossD: -0.0032971701584756374 LossG:0.06462931493297219
[26/100] LossD: -0.0036046630702912807 LossG:0.06738066580146551
[27/100] LossD: -0.004094256553798914 LossG:0.06365822814404964
[28/100] LossD: -0.004420840181410313 LossG:0.05663383658975363
[29/100] LossD: -0.0050626215524971485 LossG:0.05169633007608354
[30/100] LossD: -0.005913021974265575 LossG:0.054023105185478926
[31/100] LossD: -0.007024796679615974 LossG:0.056679457891732454
[32/100] LossD: -0.008519726572558284 LossG:0.059527989476919174
[33/100] LossD: -0.01047378615476191 LossG:0.05468072183430195
[34/100] LossD: -0.012964809546247125 LossG:0.05618627252988517
[35/100] LossD: -0.016494292998686433 LossG:0.05486194184049964
[36/100] LossD: -0.02133050188422203 LossG:0.06419895496219397
[37/100] LossD: -0.028063883539289236 LossG:0.06623653555288911
[38/100] LossD: -0.03721790760755539 LossG:0.07994594285264611
[39/100] LossD: -0.04967968026176095 LossG:0.10433801636099815
[40/100] LossD: -0.06664252653717995 LossG:0.12247380707412958
[41/100] LossD: -0.08865693444386125 LossG:0.1309313978999853
```

```
[42/100] LossD: -0.11632354184985161 LossG:0.18046584632247686
[43/100] LossD: -0.15203545847907662 LossG:0.23149354103952646
[44/100] LossD: -0.1963244299404323 LossG:0.24078052397817373
[45/100] LossD: -0.24980038218200207 LossG:0.32884669210761786
[46/100] LossD: -0.3153291689231992 LossG:0.410181013867259
[47/100] LossD: -0.3946838341653347 LossG:0.44887533970177174
[48/100] LossD: -0.4911243040114641 LossG:0.5632819291204214
[49/100] LossD: -0.6106976810842752 LossG:0.6773185133934021
[50/100] LossD: -0.7567421700805426 LossG:0.7766121067106724
[51/100] LossD: -0.9332034457474947 LossG:0.8944320231676102
[52/100] LossD: -1.1349964551627636 LossG:1.060188353061676
[53/100] LossD: -1.3842711001634598 LossG:1.3036746457219124
[54/100] LossD: -1.6629563719034195 LossG:1.4084900990128517
[55/100] LossD: -1.9921568147838116 LossG:1.5428821220993996
[56/100] LossD: -2.366080466657877 LossG:1.6618676111102104
[57/100] LossD: -2.826912097632885 LossG:2.0947841964662075
[58/100] LossD: -3.3621279262006283 LossG:2.408921469002962
[59/100] LossD: -3.9962663874030113 LossG:2.6032233610749245
[60/100] LossD: -4.697783894836903 LossG:2.7893417850136757
[61/100] LossD: -5.521539997309446 LossG:3.272246077656746
[62/100] LossD: -6.453829325735569 LossG:3.611471012234688
[63/100] LossD: -7.53415459394455 LossG:4.316265732049942
[64/100] LossD: -8.781210079789162 LossG:5.22690512239933
[65/100] LossD: -10.148784138262272 LossG:5.467879615724087
[66/100] LossD: -11.784536600112915 LossG:6.5210171192884445
[67/100] LossD: -13.461864978075027 LossG:7.413811773061752
[68/100] LossD: -15.352959156036377 LossG:8.439712077379227
[69/100] LossD: -17.420637220144272 LossG:9.588287770748138
[70/100] LossD: -19.708322823047638 LossG:10.885059967637062
[71/100] LossD: -22.258541077375412 LossG:12.214263141155243
[72/100] LossD: -25.122201204299927 LossG:13.540394097566605
[73/100] LossD: -28.384537756443024 LossG:15.453543990850449
[74/100] LossD: -32.02360075712204 LossG:17.60704904794693
[75/100] LossD: -36.097813576459885 LossG:19.811986088752747
[76/100] LossD: -40.34210389852524 LossG:21.982374399900436
[77/100] LossD: -41.60353794693947 LossG:23.444043666124344
[78/100] LossD: -26.46922144293785 LossG:4.842031747102737
[79/100] LossD: -30.576063692569733 LossG:2.3243316262960434
[80/100] LossD: -32.58367830514908 LossG:2.614806119352579
[81/100] LossD: -32.171064496040344 LossG:2.0376336574554443
[82/100] LossD: -28.331584692001343 LossG:-2.2493875324726105
[83/100] LossD: -17.3640878200531 LossG:-13.744630577042699
[84/100] LossD: -1.2913246154785156 LossG:-37.55781841278076
[85/100] LossD: 2.9801316261291504 LossG:-48.47538876533508
[86/100] LossD: 2.981811046600342 LossG:-49.20903038978577
[87/100] LossD: 2.076793909072876 LossG:-49.296222448349
[88/100] LossD: 1.3314731121063232 LossG:-48.46582889556885
[89/100] LossD: 0.7617216110229492 LossG:-46.97056698799133
```

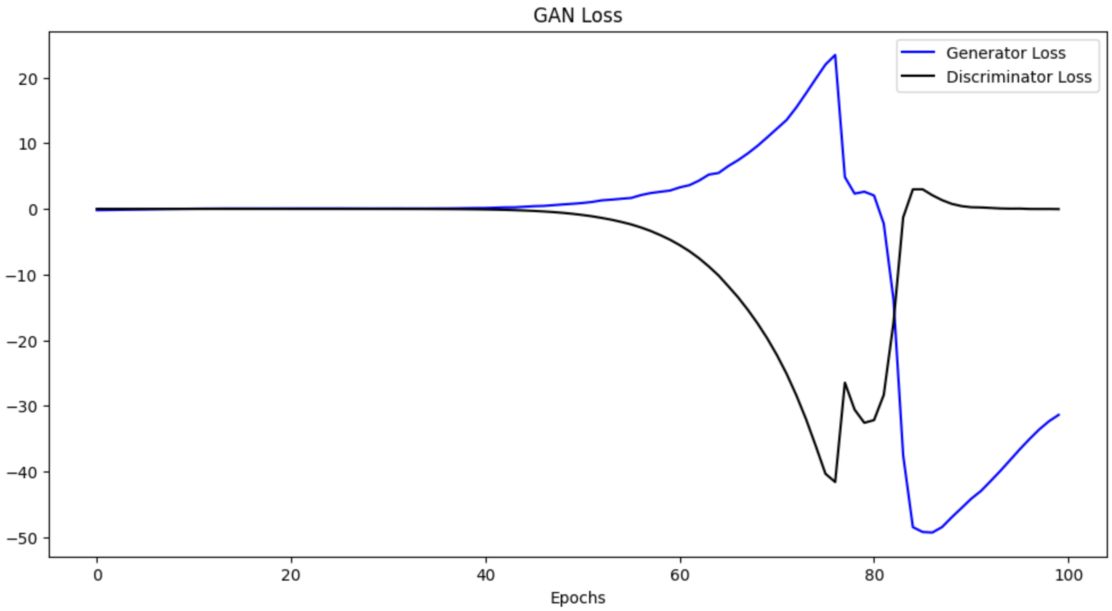
```
[90/100] LossD: 0.4215528964996338 LossG:-45.57131814956665
[91/100] LossD: 0.2506091594696045 LossG:-44.15081787109375
[92/100] LossD: 0.2180633544921875 LossG:-42.971333742141724
[93/100] LossD: 0.1446666717529297 LossG:-41.45828938484192
[94/100] LossD: 0.0650186538696289 LossG:-39.9096245765686
[95/100] LossD: 0.032111167907714844 LossG:-38.27099871635437
[96/100] LossD: 0.04394936561584473 LossG:-36.62075686454773
[97/100] LossD: -0.012690544128417969 LossG:-35.05781054496765
[98/100] LossD: -0.017923831939697266 LossG:-33.597312211990356
[99/100] LossD: -0.01578378677368164 LossG:-32.35325288772583
[100/100] LossD: -0.034819841384887695 LossG:-31.37304377555847
```

[2]:

```
plt.figure(figsize = (12, 6))
plt.plot(histG, color = 'blue', label = 'Generator Loss')
plt.plot(histD, color = 'black', label = 'Discriminator Loss')
plt.title('GAN Loss')
plt.xlabel('Epochs')
plt.legend(loc = 'upper right')
```

[2]:

```
<matplotlib.legend.Legend at 0x78d339c8b2b0>
```



[16]:

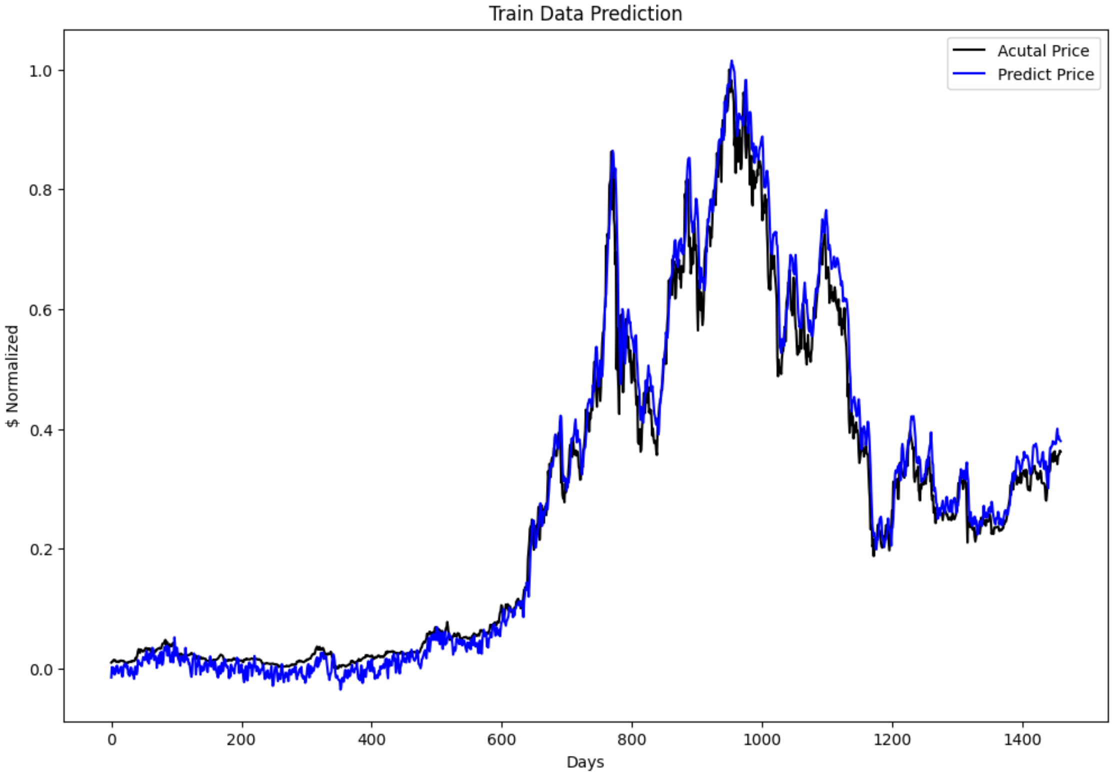
```
y_scaler = MinMaxScaler(feature_range = (0, 1))
dummy = y_scaler.fit_transform(e_train_y_slide.reshape(-1, 1))

modelG.eval()
pred_y_train = modelG(e_train_x_slide.to(device))
pred_y_test = modelG(e_test_x_slide.to(device))
```

```
y_train_true = y_scaler.inverse_transform(e_train_y_slide)
y_train_pred = y_scaler.inverse_transform(pred_y_train.cpu().detach().numpy())

y_test_true = y_scaler.inverse_transform(e_test_y_slide)
y_test_pred = y_scaler.inverse_transform(pred_y_test.cpu().detach().numpy())
```

[17]:
```
y_train_true = y_train_true + 0.000001
y_train_pred = y_train_pred + 0.000001
y_test_true = y_test_true + 0.000001
y_test_pred = y_test_pred + 0.000001
```

[18]:
```
plt.figure(figsize=(12, 8))
plt.plot(y_train_true, color = 'black', label = 'Acutal Price')
plt.plot(y_train_pred, color = 'blue', label = 'Predict Price')
plt.title('Train Data Prediction')
plt.ylabel('$ Normalized')
plt.xlabel('Days')
plt.legend(loc = 'upper right')

MSE = mean_squared_error(y_train_true, y_train_pred)
RMSE = math.sqrt(MSE)
print(f'Training dataset RMSE:{RMSE}')
mape = np.mean(np.abs(y_train_pred - y_train_true)/np.abs(y_train_true))
print(f'Training dataset MAPE:{mape}')
```

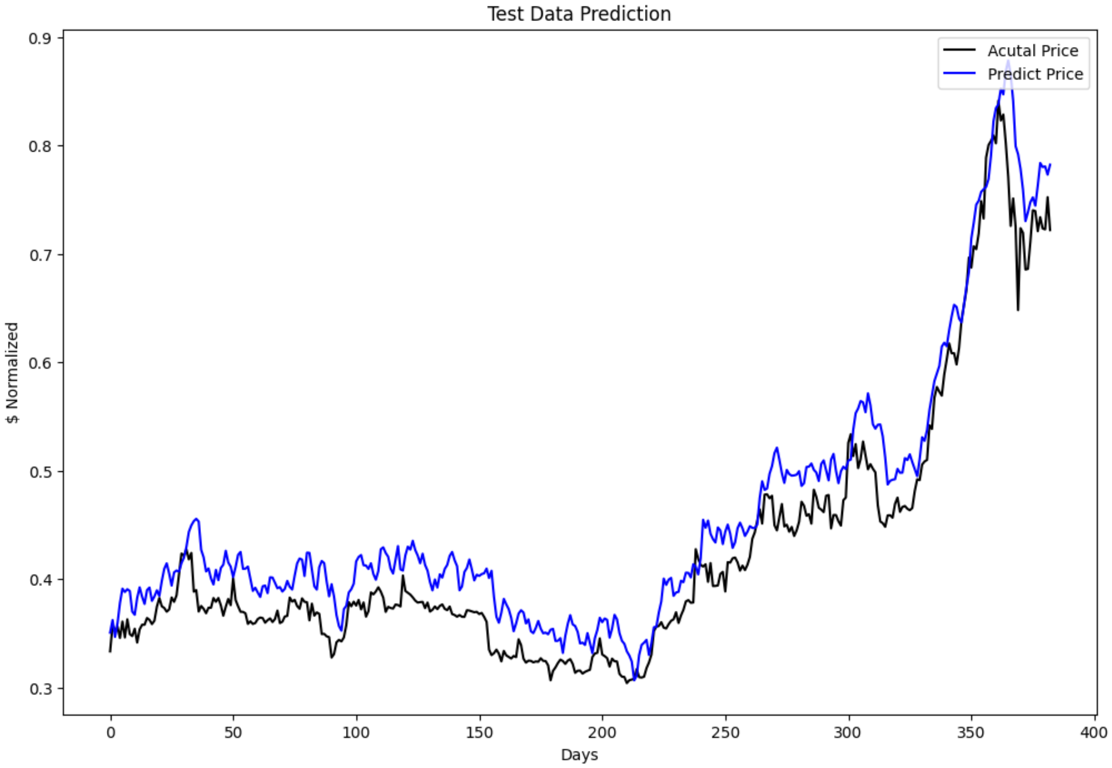
```
Training dataset RMSE:0.039176202123907794
Training dataset MAPE:8.793037802421868
```



[19]:

```
plt.figure(figsize=(12, 8))
plt.plot(y_test_true, color = 'black', label = 'Acutal Price')
plt.plot(y_test_pred, color = 'blue', label = 'Predict Price')
plt.title('Test Data Prediction')
plt.ylabel('$ Normalized')
plt.xlabel('Days')
plt.legend(loc = 'upper right')

MSE = mean_squared_error(y_test_true, y_test_pred)
RMSE = math.sqrt(MSE)
print(f'Testing dataset RMSE:{RMSE}')
mape = np.mean(np.abs(y_test_pred - y_test_true)/np.abs(y_test_true))
print(f'Testing dataset MAPE:{mape}')
```

```
Testing dataset RMSE:0.03825794573067061
Testing dataset MAPE:0.08205526294958074
```


Test Data Prediction
Acutal Price
Predict Price
$ Normalized
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Days
0
50
100
150
200
250
300
350
400