dl-bnb

April 24, 2024

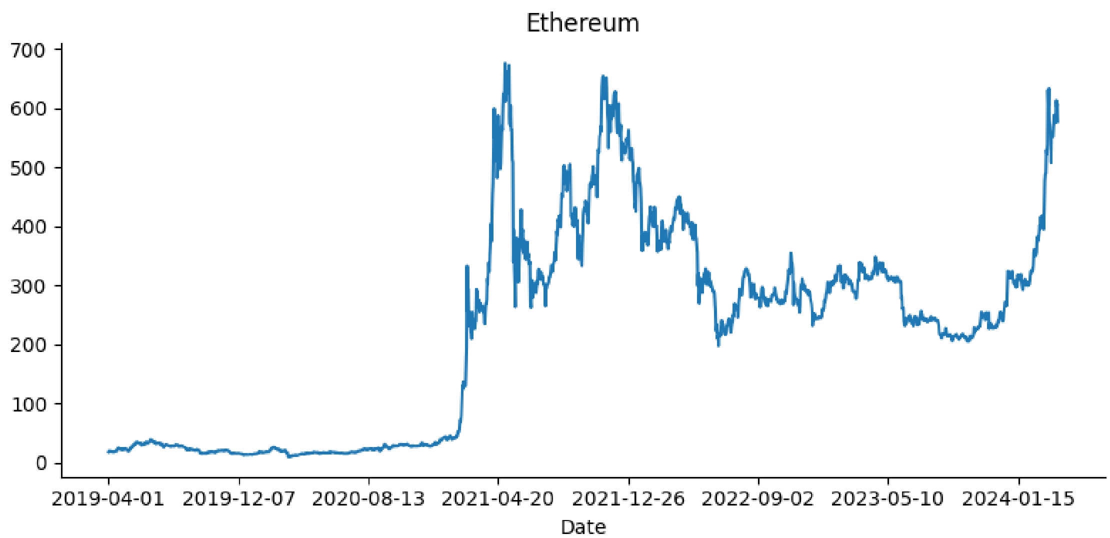
[19]:
```python
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
import torch.nn.functional as F
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import math
```

[20]:
```python
binance = pd.read_csv('/kaggle/input/binance-2/BNB-USD-2.csv', index_col='Date')
binance
```

[20]:
```
                  Open        High         Low       Close   Adj Close  \
Date
2019-04-01   17.410639   18.058975   17.291075   17.950502   17.950502
2019-04-02   17.960625   19.921143   17.691793   19.789783   19.789783
2019-04-03   19.799784   20.071714   18.429638   18.753063   18.753063
2019-04-04   18.743467   19.545368   18.447649   19.142689   19.142689
2019-04-05   19.139215   19.530912   18.941456   19.453466   19.453466
…                  …           …           …           …           …
2024-03-28  574.511597  591.314636  574.201782  583.270874  583.270874
2024-03-29  583.286743  619.972595  582.469360  612.657959  612.657959
2024-03-30  612.660156  612.994934  597.629822  601.016357  601.016357
2024-03-31  601.005127  608.664246  600.879272  606.908630  606.908630
2024-04-01  606.908691  607.645569  570.053162  576.396667  576.396667

                Volume
Date
2019-04-01   202977286
2019-04-02   264406306
2019-04-03   254862222
2019-04-04   214630296
2019-04-05   181774781
…                    …
2024-03-28  1986567688
```

```
2024-03-29  2788931743
2024-03-30  1750650703
2024-03-31  1456592924
2024-04-01  2020241864

[1828 rows x 6 columns]
```

[21]:
```python
binance['Close'].plot(kind='line', figsize=(8, 4), title='Ethereum')
plt.tight_layout()
plt.gca().spines[['top', 'right']].set_visible(False)
```



[22]:
```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
```

[23]:
```
binance_train_x, binance_test_x, binance_train_y, binance_test_y =␣
 ↪Dataset(binance)
print(f'trainX: {binance_train_x.shape} trainY: {binance_train_y.shape}')
print(f'testX: {binance_test_x.shape} testY: {binance_test_y.shape}')
```

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

[24]:
```
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
```

[25]:
```
train_loader = DataLoader(TensorDataset(torch.from_numpy(binance_train_x).
 ↪float()), batch_size = 128, shuffle = False)
model = VAE([6, 400, 400, 400, 10], 10)
```

[26]:
```
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())
```

```python
        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: 322.5194482803345
[2/300] Loss: 303.8097791671753
[3/300] Loss: 286.5582799911499
[4/300] Loss: 271.2115249633789
[5/300] Loss: 257.0912265777588
[6/300] Loss: 241.4587984085083
[7/300] Loss: 222.1250925064087
[8/300] Loss: 202.71923065185547
[9/300] Loss: 189.39505577087402
[10/300] Loss: 189.961980342865
[11/300] Loss: 202.60717964172363
[12/300] Loss: 213.53272533416748
[13/300] Loss: 212.90704250335693
[14/300] Loss: 203.08952808380127
[15/300] Loss: 185.47548484802246
[16/300] Loss: 166.08263206481934
[17/300] Loss: 156.74502754211426
[18/300] Loss: 160.18627738952637
[19/300] Loss: 170.6508240699768
[20/300] Loss: 181.29750776290894
[21/300] Loss: 188.0122847557068
[22/300] Loss: 189.4911298751831
[23/300] Loss: 186.18890047073364
[24/300] Loss: 179.4037046432495
[25/300] Loss: 170.4535837173462
[26/300] Loss: 160.95874404907227
[27/300] Loss: 152.3553342819214
[28/300] Loss: 145.63802528381348
[29/300] Loss: 141.15626907348633
[30/300] Loss: 138.18528175354004
[31/300] Loss: 135.20142889022827
[32/300] Loss: 131.48801851272583
[33/300] Loss: 128.38127422332764
[34/300] Loss: 128.62106561660767
[35/300] Loss: 133.8542776107788
[36/300] Loss: 142.55132913589478
[37/300] Loss: 144.72242975234985
```

```
[38/300] Loss: 140.2861647605896
[39/300] Loss: 130.6321988105774
[40/300] Loss: 119.00174236297607
[41/300] Loss: 109.51393556594849
[42/300] Loss: 104.74511003494263
[43/300] Loss: 104.58333778381348
[44/300] Loss: 107.5797848701477
[45/300] Loss: 111.6252064704895
[46/300] Loss: 115.41031455993652
[47/300] Loss: 118.0535569190979
[48/300] Loss: 119.12130689620972
[49/300] Loss: 118.5458550453186
[50/300] Loss: 116.25270509719849
[51/300] Loss: 112.43357515335083
[52/300] Loss: 107.3369369506836
[53/300] Loss: 101.30474185943604
[54/300] Loss: 94.75376224517822
[55/300] Loss: 88.31412816047668
[56/300] Loss: 82.60129976272583
[57/300] Loss: 78.2289628982544
[58/300] Loss: 75.98947715759277
[59/300] Loss: 76.26530194282532
[60/300] Loss: 79.21202421188354
[61/300] Loss: 84.1648781299591
[62/300] Loss: 89.85629892349243
[63/300] Loss: 94.39847469329834
[64/300] Loss: 96.12443208694458
[65/300] Loss: 93.78761291503906
[66/300] Loss: 87.55085182189941
[67/300] Loss: 79.03337907791138
[68/300] Loss: 70.44909691810608
[69/300] Loss: 63.7007577419281
[70/300] Loss: 59.737417459487915
[71/300] Loss: 58.58195924758911
[72/300] Loss: 59.733168840408325
[73/300] Loss: 62.434800148010254
[74/300] Loss: 65.76910877227783
[75/300] Loss: 69.02642750740051
[76/300] Loss: 71.77041006088257
[77/300] Loss: 73.74871301651001
[78/300] Loss: 74.74751329421997
[79/300] Loss: 74.66927814483643
[80/300] Loss: 73.47808575630188
[81/300] Loss: 71.28221559524536
[82/300] Loss: 68.2193021774292
[83/300] Loss: 64.41043376922607
[84/300] Loss: 60.07807970046997
[85/300] Loss: 55.53090786933899
```

```
[86/300] Loss: 51.10667967796326
[87/300] Loss: 47.2332603931427
[88/300] Loss: 44.26588821411133
[89/300] Loss: 42.54536581039429
[90/300] Loss: 42.226481437683105
[91/300] Loss: 43.35333585739136
[92/300] Loss: 45.48945713043213
[93/300] Loss: 48.23714017868042
[94/300] Loss: 50.82144808769226
[95/300] Loss: 52.67081427574158
[96/300] Loss: 53.34408903121948
[97/300] Loss: 52.70270538330078
[98/300] Loss: 50.83440089225769
[99/300] Loss: 48.03227186203003
[100/300] Loss: 44.78186011314392
[101/300] Loss: 41.57022321224213
[102/300] Loss: 38.79637336730957
[103/300] Loss: 36.6818745136261
[104/300] Loss: 35.28153419494629
[105/300] Loss: 34.57606101036072
[106/300] Loss: 34.37330901622772
[107/300] Loss: 34.53371715545654
[108/300] Loss: 34.840972900390625
[109/300] Loss: 35.18018388748169
[110/300] Loss: 35.43798780441284
[111/300] Loss: 35.57672190666199
[112/300] Loss: 35.53991460800171
[113/300] Loss: 35.37435531616211
[114/300] Loss: 35.034353494644165
[115/300] Loss: 34.53846716880798
[116/300] Loss: 33.903762459754944
[117/300] Loss: 33.17631983757019
[118/300] Loss: 32.39758825302124
[119/300] Loss: 31.588250160217285
[120/300] Loss: 30.762446403503418
[121/300] Loss: 29.95168375968933
[122/300] Loss: 29.196985721588135
[123/300] Loss: 28.510313868522644
[124/300] Loss: 27.90014088153839
[125/300] Loss: 27.428735494613647
[126/300] Loss: 27.094842195510864
[127/300] Loss: 26.9251389503479
[128/300] Loss: 26.910765886306763
[129/300] Loss: 26.983790278434753
[130/300] Loss: 27.064926862716675
[131/300] Loss: 27.106892824172974
[132/300] Loss: 27.055555820465088
[133/300] Loss: 26.908159136772156
```

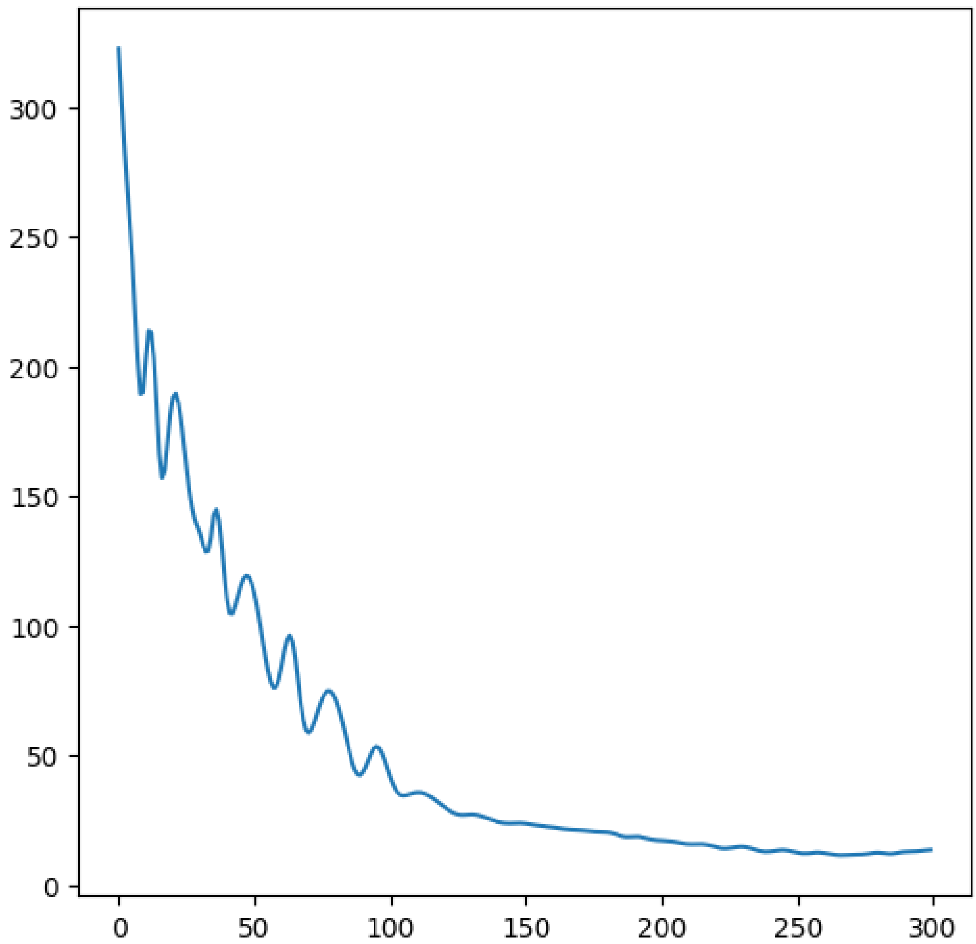
```
[134/300] Loss: 26.64013159275055
[135/300] Loss: 26.278625011444092
[136/300] Loss: 25.87162208557129
[137/300] Loss: 25.491195678710938
[138/300] Loss: 25.11386287212372
[139/300] Loss: 24.753126859664917
[140/300] Loss: 24.415254592895508
[141/300] Loss: 24.13285458087921
[142/300] Loss: 23.903281211853027
[143/300] Loss: 23.742698192596436
[144/300] Loss: 23.652142763137817
[145/300] Loss: 23.640012502670288
[146/300] Loss: 23.672537803649902
[147/300] Loss: 23.728273034095764
[148/300] Loss: 23.773908495903015
[149/300] Loss: 23.770217299461365
[150/300] Loss: 23.707326531410217
[151/300] Loss: 23.581494212150574
[152/300] Loss: 23.40187430381775
[153/300] Loss: 23.220226526260376
[154/300] Loss: 23.043824911117554
[155/300] Loss: 22.890330910682678
[156/300] Loss: 22.76840341091156
[157/300] Loss: 22.666019320487976
[158/300] Loss: 22.566425561904907
[159/300] Loss: 22.456832766532898
[160/300] Loss: 22.328912615776062
[161/300] Loss: 22.170762181282043
[162/300] Loss: 21.985108137130737
[163/300] Loss: 21.78873383998871
[164/300] Loss: 21.619561910629272
[165/300] Loss: 21.455514788627625
[166/300] Loss: 21.33460509777069
[167/300] Loss: 21.26902449131012
[168/300] Loss: 21.219729900360107
[169/300] Loss: 21.180370211601257
[170/300] Loss: 21.134389519691467
[171/300] Loss: 21.04200303554535
[172/300] Loss: 20.943037509918213
[173/300] Loss: 20.823561310768127
[174/300] Loss: 20.70341384410858
[175/300] Loss: 20.581169724464417
[176/300] Loss: 20.506584882736206
[177/300] Loss: 20.45439338684082
[178/300] Loss: 20.422407507896423
[179/300] Loss: 20.38403630256653
[180/300] Loss: 20.3402601480484
[181/300] Loss: 20.25636315345764
```

```
[182/300] Loss: 20.09506845474243
[183/300] Loss: 19.841726660728455
[184/300] Loss: 19.52416694164276
[185/300] Loss: 19.15642535686493
[186/300] Loss: 18.830745339393616
[187/300] Loss: 18.553630113601685
[188/300] Loss: 18.395488381385803
[189/300] Loss: 18.382903695106506
[190/300] Loss: 18.443796753883362
[191/300] Loss: 18.516160249710083
[192/300] Loss: 18.53534162044525
[193/300] Loss: 18.43068027496338
[194/300] Loss: 18.242023587226868
[195/300] Loss: 17.982628345489502
[196/300] Loss: 17.717705249786377
[197/300] Loss: 17.47985601425171
[198/300] Loss: 17.288691759109497
[199/300] Loss: 17.166346430778503
[200/300] Loss: 17.092057943344116
[201/300] Loss: 17.05155634880066
[202/300] Loss: 16.99992299079895
[203/300] Loss: 16.934886813163757
[204/300] Loss: 16.82175236940384
[205/300] Loss: 16.67082542181015
[206/300] Loss: 16.469095945358276
[207/300] Loss: 16.24656105041504
[208/300] Loss: 16.023524284362793
[209/300] Loss: 15.830095112323761
[210/300] Loss: 15.684193968772888
[211/300] Loss: 15.612771928310394
[212/300] Loss: 15.595455408096313
[213/300] Loss: 15.629431009292603
[214/300] Loss: 15.667024493217468
[215/300] Loss: 15.689244151115417
[216/300] Loss: 15.663196504116058
[217/300] Loss: 15.55372154712677
[218/300] Loss: 15.368080258369446
[219/300] Loss: 15.097471237182617
[220/300] Loss: 14.789879024028778
[221/300] Loss: 14.49136483669281
[222/300] Loss: 14.23601758480072
[223/300] Loss: 14.053084969520569
[224/300] Loss: 13.981616377830505
[225/300] Loss: 14.00132954120636
[226/300] Loss: 14.107130110263824
[227/300] Loss: 14.265160083770752
[228/300] Loss: 14.437694668769836
[229/300] Loss: 14.589403748512268
```

```
[230/300] Loss: 14.667521357536316
[231/300] Loss: 14.653615653514862
[232/300] Loss: 14.525996565818787
[233/300] Loss: 14.306551337242126
[234/300] Loss: 14.009086966514587
[235/300] Loss: 13.679604828357697
[236/300] Loss: 13.328081607818604
[237/300] Loss: 13.031077146530151
[238/300] Loss: 12.844057500362396
[239/300] Loss: 12.720197200775146
[240/300] Loss: 12.721324682235718
[241/300] Loss: 12.803742170333862
[242/300] Loss: 12.953623116016388
[243/300] Loss: 13.117996394634247
[244/300] Loss: 13.263349652290344
[245/300] Loss: 13.360530316829681
[246/300] Loss: 13.356760382652283
[247/300] Loss: 13.257929682731628
[248/300] Loss: 13.078242361545563
[249/300] Loss: 12.841771066188812
[250/300] Loss: 12.58338713645935
[251/300] Loss: 12.348599255084991
[252/300] Loss: 12.160124063491821
[253/300] Loss: 12.03171056509018
[254/300] Loss: 12.008434116840363
[255/300] Loss: 12.065412282943726
[256/300] Loss: 12.204639077186584
[257/300] Loss: 12.32778286933899
[258/300] Loss: 12.406437873840332
[259/300] Loss: 12.406391978263855
[260/300] Loss: 12.313592255115509
[261/300] Loss: 12.155773937702179
[262/300] Loss: 11.964878857135773
[263/300] Loss: 11.763239324092865
[264/300] Loss: 11.578374803066254
[265/300] Loss: 11.441133260726929
[266/300] Loss: 11.354487001895905
[267/300] Loss: 11.316992342472076
[268/300] Loss: 11.32954716682434
[269/300] Loss: 11.367399513721466
[270/300] Loss: 11.418692648410797
[271/300] Loss: 11.479357182979584
[272/300] Loss: 11.527125298976898
[273/300] Loss: 11.560405671596527
[274/300] Loss: 11.57755035161972
[275/300] Loss: 11.615749478340149
[276/300] Loss: 11.711136519908905
[277/300] Loss: 11.866506814956665
```

```
[278/300] Loss: 12.052727162837982
[279/300] Loss: 12.22130835056305
[280/300] Loss: 12.32024073600769
[281/300] Loss: 12.3339564204216
[282/300] Loss: 12.24927532672882
[283/300] Loss: 12.110510051250458
[284/300] Loss: 11.962328612804413
[285/300] Loss: 11.879091739654541
[286/300] Loss: 11.909885585308075
[287/300] Loss: 12.053935527801514
[288/300] Loss: 12.27372533082962
[289/300] Loss: 12.482215881347656
[290/300] Loss: 12.633005380630493
[291/300] Loss: 12.716813027858734
[292/300] Loss: 12.758142709732056
[293/300] Loss: 12.788620710372925
[294/300] Loss: 12.827677726745605
[295/300] Loss: 12.890933513641357
[296/300] Loss: 12.985733270645142
[297/300] Loss: 13.105046689510345
[298/300] Loss: 13.217972934246063
[299/300] Loss: 13.327046811580658
[300/300] Loss: 13.428720533847809
```

[26]:
```
[<matplotlib.lines.Line2D at 0x7b9884194070>]
```



[27]:
```
model.eval()
_, VAE_train_x, train_x_mu, train_x_var = model(torch.
 ↪from_numpy(binance_train_x).float().to(device))
_, VAE_test_x, test_x_mu, test_x_var = model(torch.from_numpy(binance_test_x).
 ↪float().to(device))
```

[28]:
```
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
```

[29]:
```
binance_train_x = np.concatenate((binance_train_x, VAE_train_x.cpu().detach().
 ↪numpy()), axis = 1)
binance_test_x = np.concatenate((binance_test_x, VAE_test_x.cpu().detach().
 ↪numpy()), axis = 1)
```

[30]:
```
b_train_x_slide, b_train_y_slide, b_train_y_gan =␣
 ↪sliding_window(binance_train_x, binance_train_y, 3)
b_test_x_slide, b_test_y_slide, b_test_y_gan = sliding_window(binance_test_x,␣
 ↪binance_test_y, 3)
print(f'train_x: {b_train_x_slide.shape} train_y: {b_train_y_slide.shape}␣
 ↪train_y_gan: {b_train_y_gan.shape}')
print(f'test_x: {b_test_x_slide.shape} test_y: {b_test_y_slide.shape}␣
 ↪test_y_gan: {b_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])
```

[31]:
```
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
```

[32]:
```
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(b_train_x_slide, b_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)
```

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

    if flag == 1:
      break
```

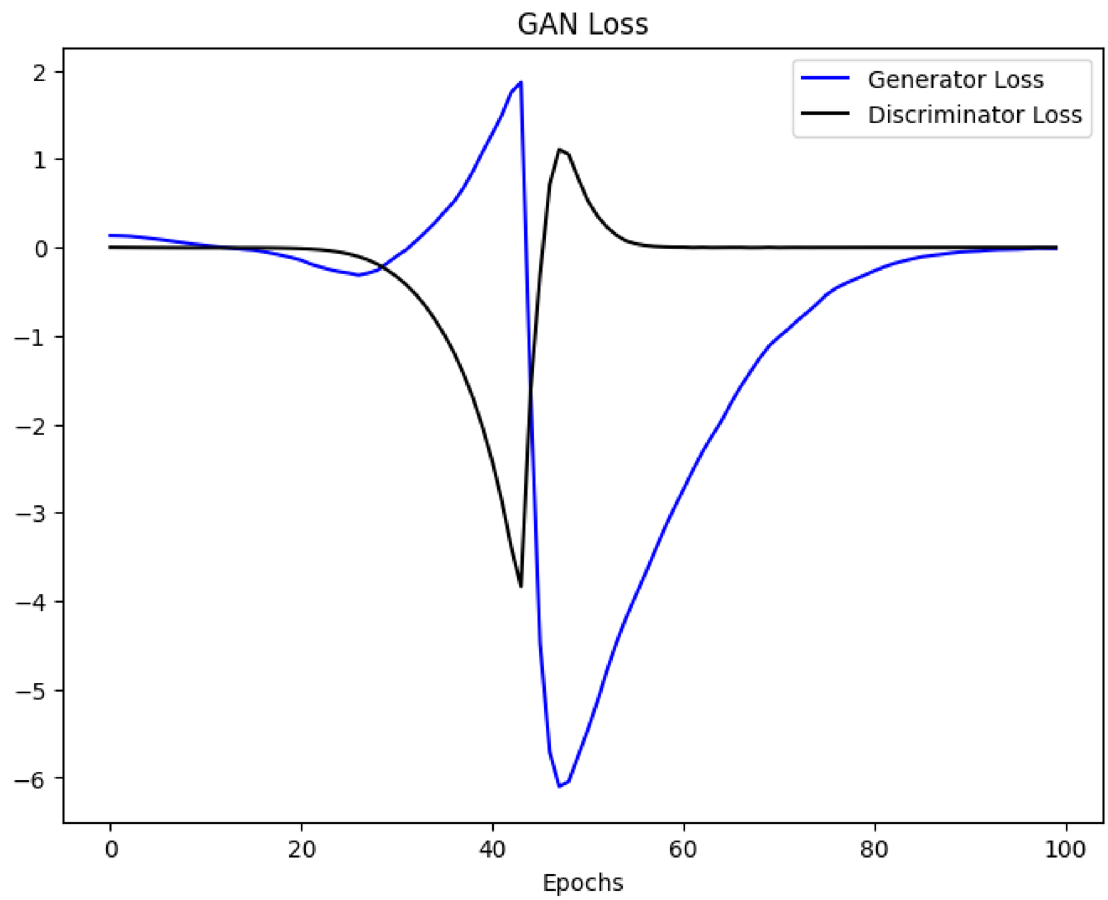
```
[1/100] LossD: -0.0005726367235183716 LossG:0.13258466962724924
[2/100] LossD: -0.0014142664149403572 LossG:0.13018302246928215
[3/100] LossD: -0.0021189767867326736 LossG:0.12523345462977886
[4/100] LossD: -0.0028218189254403114 LossG:0.11503317300230265
[5/100] LossD: -0.003508306574076414 LossG:0.1029013954102993
[6/100] LossD: -0.004041616339236498 LossG:0.09010222740471363
[7/100] LossD: -0.004523737821727991 LossG:0.07540043955668807
[8/100] LossD: -0.005029510939493775 LossG:0.06110945716500282
[9/100] LossD: -0.005434327176772058 LossG:0.04617511201649904
[10/100] LossD: -0.005636921210680157 LossG:0.032166033051908016
[11/100] LossD: -0.0057392934104427695 LossG:0.019798148132395
[12/100] LossD: -0.005779724044259638 LossG:0.007296099865925498
[13/100] LossD: -0.005774194723926485 LossG:-0.003327940750750713
[14/100] LossD: -0.005728086456656456 LossG:-0.013302053775987588
[15/100] LossD: -0.005816066724946722 LossG:-0.023007275827694684
[16/100] LossD: -0.005973355728201568 LossG:-0.03115100529976189
[17/100] LossD: -0.00652541546151042 LossG:-0.048382948618382215
[18/100] LossD: -0.007504618493840098 LossG:-0.06947861914522946
[19/100] LossD: -0.009020860306918621 LossG:-0.09219179814681411
[20/100] LossD: -0.011469208169728518 LossG:-0.11749297706410289
[21/100] LossD: -0.015067805536091328 LossG:-0.14657482132315636
[22/100] LossD: -0.02032331097871065 LossG:-0.19017885625362396
[23/100] LossD: -0.028067192994058132 LossG:-0.22368385456502438
[24/100] LossD: -0.039216929115355015 LossG:-0.25273824483156204
[25/100] LossD: -0.05464316811412573 LossG:-0.27616875246167183
[26/100] LossD: -0.07609176635742188 LossG:-0.2924239570274949
[27/100] LossD: -0.10534885246306658 LossG:-0.31496303249150515
[28/100] LossD: -0.14389842748641968 LossG:-0.2915585292503238
[29/100] LossD: -0.1931863371282816 LossG:-0.2588189421221614
[30/100] LossD: -0.2558847516775131 LossG:-0.18047714745625854
[31/100] LossD: -0.3330715522170067 LossG:-0.0984134441241622
[32/100] LossD: -0.4241458037868142 LossG:-0.026945197954773903
[33/100] LossD: -0.5344686880707741 LossG:0.06896561430767179
[34/100] LossD: -0.6645063776522875 LossG:0.17149692890234292
[35/100] LossD: -0.8174161873757839 LossG:0.27853740022692364
[36/100] LossD: -0.993200208991766 LossG:0.40108645940199494
[37/100] LossD: -1.1945392936468124 LossG:0.5180910236667842
[38/100] LossD: -1.4327703341841698 LossG:0.6726625435985625
[39/100] LossD: -1.7118279486894608 LossG:0.8670613472349942
[40/100] LossD: -2.039786282926798 LossG:1.0812110546976328
[41/100] LossD: -2.422262541949749 LossG:1.2856236174702644
```

```
[42/100] LossD: -2.8712549209594727 LossG:1.4971776232123375
[43/100] LossD: -3.39692559838295 LossG:1.7556678652763367
[44/100] LossD: -3.8346847891807556 LossG:1.865262396633625
[45/100] LossD: -1.6314373910427094 LossG:-1.6000610813498497
[46/100] LossD: -0.31471535563468933 LossG:-4.472530107945204
[47/100] LossD: 0.7081254422664642 LossG:-5.704853534698486
[48/100] LossD: 1.106851041316986 LossG:-6.099211752414703
[49/100] LossD: 1.0508266389369965 LossG:-6.044095486402512
[50/100] LossD: 0.7768899202346802 LossG:-5.748934835195541
[51/100] LossD: 0.5283607542514801 LossG:-5.460351824760437
[52/100] LossD: 0.3539104163646698 LossG:-5.137943238019943
[53/100] LossD: 0.22877919673919678 LossG:-4.788393974304199
[54/100] LossD: 0.1370420753955841 LossG:-4.474872976541519
[55/100] LossD: 0.07134106755256653 LossG:-4.1922246515750885
[56/100] LossD: 0.03976845741271973 LossG:-3.9435914158821106
[57/100] LossD: 0.020494580268859863 LossG:-3.6999606490135193
[58/100] LossD: 0.009737163782119751 LossG:-3.441811591386795
[59/100] LossD: 0.004481852054595947 LossG:-3.190454363822937
[60/100] LossD: 0.0013116896152496338 LossG:-2.9567607045173645
[61/100] LossD: 0.000645563006401062 LossG:-2.7420085221529007
[62/100] LossD: -0.0033166706562042236 LossG:-2.52374529838562
[63/100] LossD: -0.0012475401163101196 LossG:-2.314177379012108
[64/100] LossD: -0.00359676783164978 LossG:-2.134771540760994
[65/100] LossD: -0.0028075426816940308 LossG:-1.9643270075321198
[66/100] LossD: -0.0019732266664505005 LossG:-1.7601068615913391
[67/100] LossD: -0.0019053071737289429 LossG:-1.5774430483579636
[68/100] LossD: -0.0045341625809669495 LossG:-1.41351068764925
[69/100] LossD: -0.003808245062828064 LossG:-1.2523281052708626
[70/100] LossD: -0.0012299269437789917 LossG:-1.1155113950371742
[71/100] LossD: -0.003949366509914398 LossG:-1.0132554396986961
[72/100] LossD: -0.00277043879032135 LossG:-0.9244905114173889
[73/100] LossD: -0.0024683624505996704 LossG:-0.8227146565914154
[74/100] LossD: -0.002814825624227524 LossG:-0.7350933961570263
[75/100] LossD: -0.0028173401951789856 LossG:-0.6392869427800179
[76/100] LossD: -0.002578228712081909 LossG:-0.536181665956974
[77/100] LossD: -0.0026359111070632935 LossG:-0.4608824923634529
[78/100] LossD: -0.002321045845746994 LossG:-0.40826427191495895
[79/100] LossD: -0.0022981632500886917 LossG:-0.36014322377741337
[80/100] LossD: -0.002252025529742241 LossG:-0.3142757248133421
[81/100] LossD: -0.0020618438720703125 LossG:-0.26687365025281906
[82/100] LossD: -0.002082815393805504 LossG:-0.22303833439946175
[83/100] LossD: -0.0022731702774763107 LossG:-0.18618210032582283
[84/100] LossD: -0.002204473130404949 LossG:-0.1568380119279027
[85/100] LossD: -0.0022319816052913666 LossG:-0.130791530944407
[86/100] LossD: -0.0022971536964178085 LossG:-0.10766571015119553
[87/100] LossD: -0.0021225623786449432 LossG:-0.09298095433041453
[88/100] LossD: -0.00217517325654625 9 LossG:-0.08026169613003731
[89/100] LossD: -0.002085884101688862 LossG:-0.06575394002720714
```

```
[90/100] LossD: -0.002339528640732169 LossG:-0.05373803433030844
[91/100] LossD: -0.0022305273450911045 LossG:-0.04672659281641245
[92/100] LossD: -0.0021315470803529024 LossG:-0.041410623118281364
[93/100] LossD: -0.001948951161466539 LossG:-0.03253774659242481
[94/100] LossD: -0.0021049558417871594 LossG:-0.026776340208016336
[95/100] LossD: -0.0024509874638170004 LossG:-0.024757537292316556
[96/100] LossD: -0.002236974658444524 LossG:-0.02232795226154849
[97/100] LossD: -0.002241090638563037 LossG:-0.017118316201958805
[98/100] LossD: -0.0024480974243488163 LossG:-0.009834985889028758
[99/100] LossD: -0.0024186512746382505 LossG:-0.011079219395469408
[100/100] LossD: -0.002538612869102508 LossG:-0.012261279916856438
```

[1]:
```python
plt.figure(figsize = (8, 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')
```

[1]: <matplotlib.legend.Legend at 0x7a4edde68520>

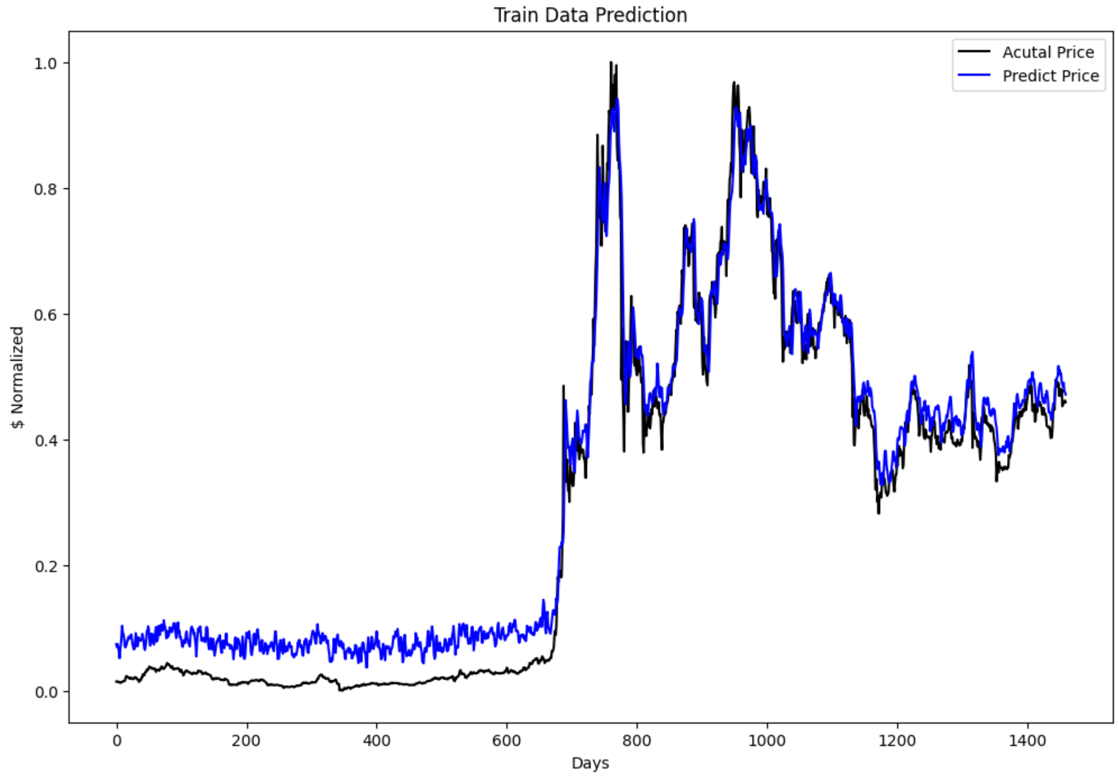


[34]:
```
y_scaler = MinMaxScaler(feature_range = (0, 1))
dummy = y_scaler.fit_transform(b_train_y_slide.reshape(-1, 1))

modelG.eval()
pred_y_train = modelG(b_train_x_slide.to(device))
pred_y_test = modelG(b_test_x_slide.to(device))

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

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

[35]:
```
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
```

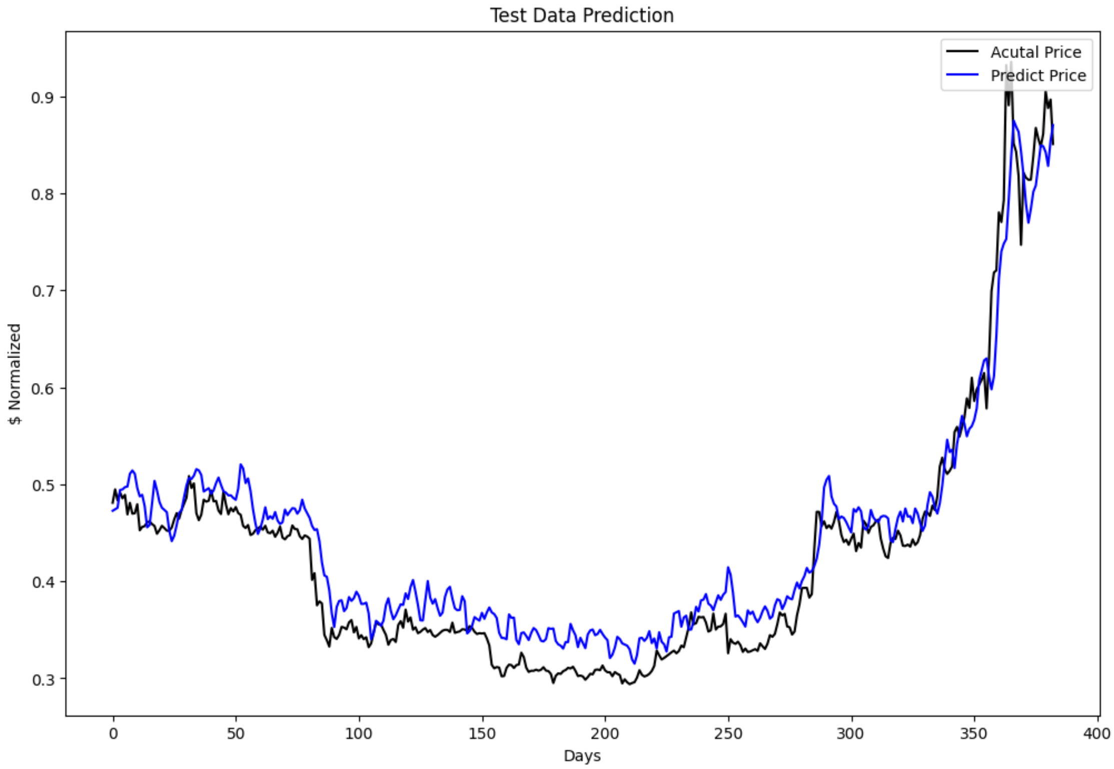
[36]:
```
plt.figure(figsize=(12, 8))
plt.plot(y_train_true, color = 'black', label = 'Acutal Price')
plt.plot(y_train_pred, color = 'blue', label = 'Predict Price')
plt.title('Train Data Prediction')
plt.ylabel('$ Normalized')
plt.xlabel('Days')
plt.legend(loc = 'upper right')

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:0.0519245401109849
Training dataset MAPE:48.85249384825739
```



[37]:
```
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:0.03393868037953706
Testing dataset MAPE:0.07072657746421177
```


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