Contributors:
NIZAP ΝΤΑΡΟΥΙΣ 1115201800286
ΤΣΙΛΙΦΩΝΗΣ ΑΡΗΣ 1115201700170

Recurrent Neural Networks

General

Exercise A
- Description
- Scaler
- Per set of time series
- Per time series
- Comparing both approaches

Exercise B
- Scaler
- Hyperparameters

Observations

Exercise C
- Scaler
- Model
  - Best Model's Hyperparameters
  - Experiments
- About Encoded curves - Important

Exercise D
- Nearest Neighbours (A)
- Clustering (B)

Some Experiment Images

General

- All the files are colab files (.ipynb), you can change the arguments in the second cell of every exercise.

Disclaimer

For each exercise you can load the models, that have been trained on all the data, instead of training it from scratch:

- **For exercise A the model is named “exercise_a”**
- **For B load the model “exercise_b”**

Exercise A

Description

In this exercise we implemented a recurrent LSTM multi-layer Neural Network to predict price shares.

Two approaches were taken:

1. Training one model with all our data (**Per set of time series**) and testing it on n time series.
2. Training and testing n models on n time series respectively (**Per time series**). In other words each model trained and tested on one unique time series.

Scaler

In each section of the code (“Per set of time series” and “Per time series”) you will find 2 cells. Run the first one if you want to have 1 scaler throughout the execution and the second if you want every time series to have its own scaler. The second approach shows better results, **therefore all the experiments below for exercise A have been done likewise**.

The first approach we mentioned above showed better results on time series with low values and worse results on curves with high values (outliers), on the contrary to the second one (Image 5). This happens because the scaler doesn't do well with outliers.

Per set of time series

For the experiments below we use 10-20 time series for convenience, however the pretrained model named “exercise_a” (offline training) has been trained on all 360 curves.

- This model reaches a loss of ~0.0020. These scores are achieved by the below hyperparameters:
    - I’m using 4 **layers**. Using less would result in worse predictions. On the flip side, using more layers, results in our model overfitting or taking longer to train without any significant improvement. All the **hidden**

layers have 28 units with loss 0.0022 at the last epoch and results as shown (Image 1). Increasing the units more than that (e.g. 56), would make the model train slower but it would make the predicted curve denser, a bit more accurate and loss values are slightly better 0.0015 (Image 2). On the other hand, even though decreasing the number of units (e.g. 16) made the training process faster, it is not worth it because the results were not good enough (Image 3).
    - As for the **number of epochs**, the model converges to its best solution pretty much after the 10-15th epoch. Any more than that, the model would start overfitting and any less would not give the chance for the model to peak.
    - After many experiments, I noticed that the most appropriate **batch size** is 216 (Image 1). Even though using a higher value would offer us faster training, on the other hand we would need to execute more epochs to get the same (good) results. Not to mention that, if we choose a smaller batch size (e.g. 108) the model would take much longer to train without much improvement (Image 4)**.**
    - The ideal Dropout percentage is 0.2 (Image 1). A higher number like 0.5 (Image 3) would result in worse predictions in curves with high values (outliers) because of the fact that time series like these need more training.
    - Lastly, it is good to mention that throughout the experiments the stride number was one. If we increase that the model will be faster but less sharp since we lose some data.
- In conclusion the most appropriate hyperparameters are the following:
    - 4 layers
    - 28 units per layer
    - 10-15 epochs
    - Batch size 216
    - Dropout 0.2

Per time series

For the experiments below we use 5-10 time series for convenience, however you can change that by changing the n value - this value represents the amount of time series that will be trained, tested and shown in the graphs.

- This model reaches a loss of ~0.0020. These scores are achieved by the below hyperparameters:
    - I'm using 4 **layers**. Using less would result in worse predictions. On the flip side, using more layers, results in our model taking longer to train without any significant improvement. All the **hidden layers** have 26 units with loss ~0.0020 at the last epoch and results as shown (Image

6). Increasing the units more than that (e.g. 52), would make the model train slower but it would make the predicted curve denser, a bit more accurate and loss values are slightly better 0.0015 (Image 7). On the other hand, even though decreasing the number of units (e.g. 16) made the training process faster, it is not worth it because the results were not good enough (Image 8).
    - As for the **number of epochs**, the model converges to its best solution pretty much after the 5-10th epoch (Image 6). Any more than that, the model would take too long to train without any improvement, start overfitting and any less would not give the chance for the model to peak.
    - The most appropriate **batch size** is 128 (Image 6). Even though using a higher value would offer us faster training, on the other hand we would need to execute more epochs to get the same (good) results (Image 8). Not to mention that, if we choose a smaller batch size (e.g. 64) the model would take much longer to train without much improvement (Image 7)**.**
    - The ideal Dropout percentage is 0.2. A higher number like 0.5 would result in worse predictions in curves with high values (outliers) because of the fact that time series like these need more training.
- In conclusion the most appropriate hyperparameters are the following:
    - 4 layers
    - 26 units per layer
    - 5-10 epochs
    - Batch size 128
    - Dropout 0.2

Comparing both approaches

Based on our graphs and experiments, the first method (Per set of time series - having one model) is better because training a model using one time series instead of a set of them, if the training set (80%) of the curve does not have sudden spikes, the model will not be able to predict them in the test set (20%) (Image 6). That's because the model needs more examples of all kinds of cases. Not to mention that training n models for n time series (n:n) takes much more time than training one model for all our data (1:n). Additionally, when a model is trained on less data it needs more epochs, a smaller batch size, more layers and more units, which makes the training process slower.

Exercise B

For the experiments below we use the pretrained model, named “exercise_b” (offline training) that has been trained on all 360 curves.

Scaler

Similar to exercise A, the first approach is to have 1 scaler for all our curves and the second one is each time series to have its own scaler. The second approach shows better results, **therefore all the experiments below have been done likewise**.

The first approach we mentioned above showed signs of overfitting. On the second approach (Image 9), we see that at first validation loss is lower than the training loss, which means that the model is not overfitting.

Hyperparameters

Generally, changes to the hyperparameters did not affect the results and the amount of anomaly points as much as the loss. So throughout the experiments we will refer to the loss values and not the anomalies - the anomaly graphs can be found in the notebook.

The ideal hyperparameters are the following:

- 2-3 layers
- 64 units per layer
- 2 epochs
- Batch size 496
- Dropout 0.2

Why we chose these values(?):

- The experiments we did using higher values than the ones listed above showed signs of overfitting (Image 9). Lower values would give us worse results (more anomaly points).
- Using the above hyperparameters we get a balanced model that does not overfit while keeping reasonable results and graphs (Image 10).
- The anomaly graphs can be found in the notebook.

Observations

1. The models find it difficult to predict spikes in time series.

2. It's easy for the models to predict time series with low values, because curves with high values are the minority in our data (outliers).
3. On some occasions, training the model with more data made the predictions worse but the validation loss better.

Exercise C

For the experiments below we use the pretrained models, named "exercise_c_decoded" and "exercise_c_encoded" (offline training) that have been tested on 20% of the data and trained on the rest.

Scaler

Each time series has its own scaler. Similar to the previous exercises we have two approaches. The first one is to break down the curve into curve_length/window pieces and to scale each one separately for all our curves and the second one is to scale the curve as a whole, then break it down into little time series. The first approach is not practical, because after merging the decoded or encoded windows, and inverse transforming each one, the curve will not be like the original one (Image 11). This happens because when scaling the windows to their original values, the scaler limits them to a certain range of values. The second approach fixes the problem. By scaling each time series separately the curve doesn't lose its original shape after the model's prediction, **therefore all the experiments below have been done likewise**.

Model

I will start by describing the best model and then I will go on explaining the experiments we conducted that lead us to the best hyperparameters.

Best Model's Hyperparameters

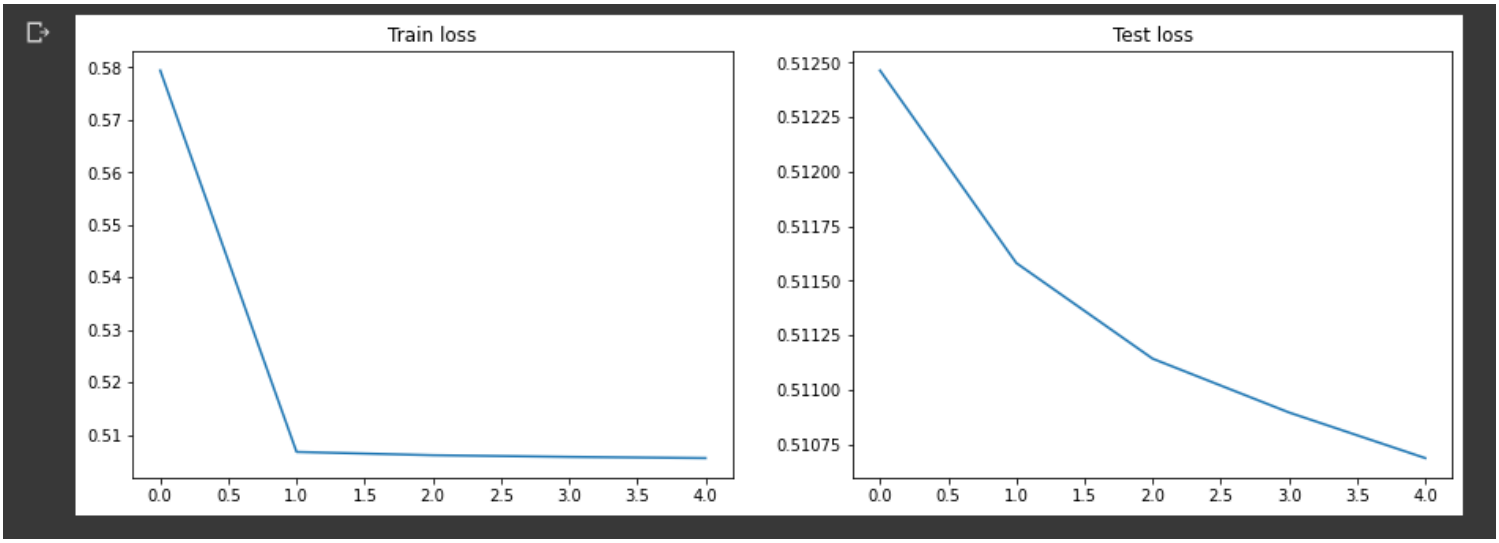
Hyperparameters:

- Number of convolutional layers = 2 encoder + 4 decoder layers
- Number of convolutional filters = 1, 4, 16
- Encoding dimensions = 3
- Epochs = 5
- Window = 50
- Batch = 32

Loss:

As you can see from the below images we see that the model pretty much converges to the best loss after 5 epochs without overfitting.

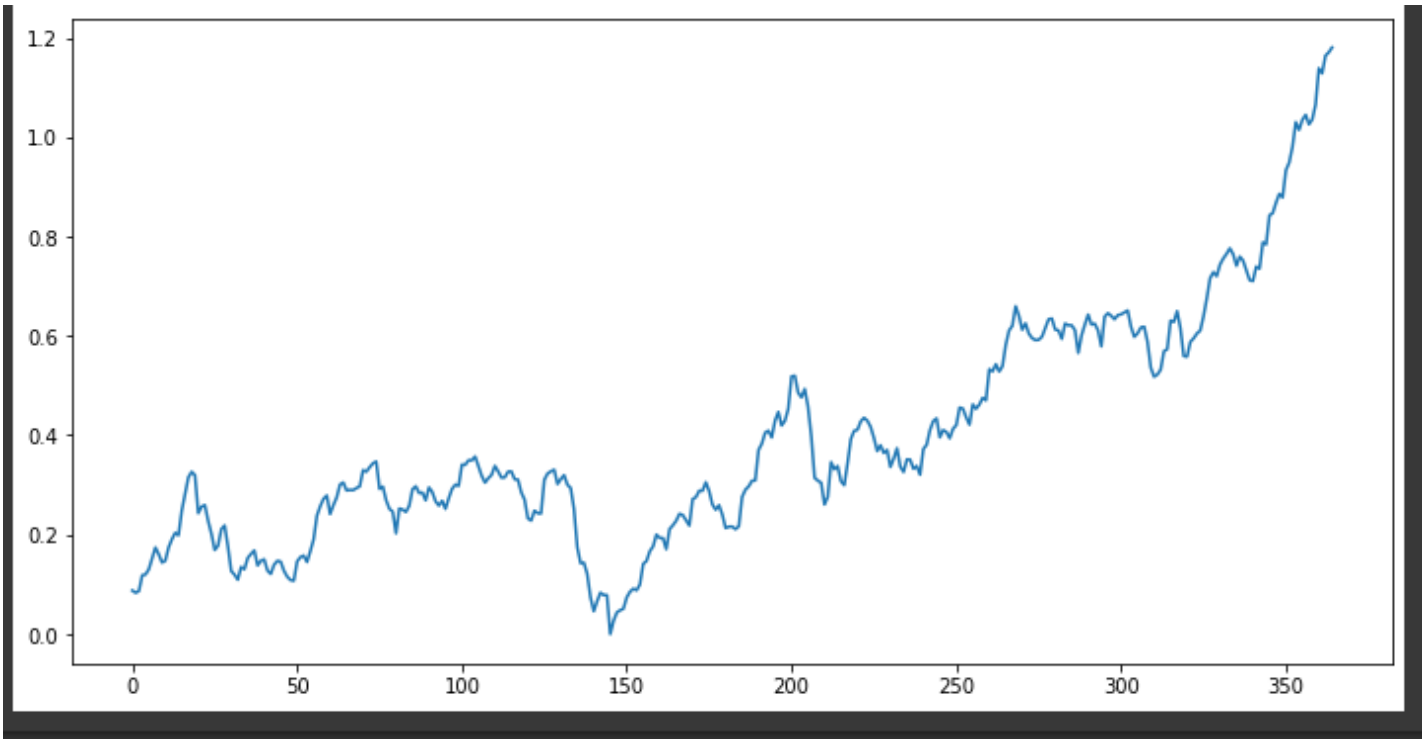
```
Epoch 1/5
657/657 [==============================] - 6s 7ms/step - loss: 0.5651 - val_loss: 0.5144
Epoch 2/5
657/657 [==============================] - 4s 6ms/step - loss: 0.5075 - val_loss: 0.5115
Epoch 3/5
657/657 [==============================] - 4s 7ms/step - loss: 0.5056 - val_loss: 0.5106
Epoch 4/5
657/657 [==============================] - 4s 6ms/step - loss: 0.5050 - val_loss: 0.5102
Epoch 5/5
657/657 [==============================] - 4s 6ms/step - loss: 0.5047 - val_loss: 0.5099
```

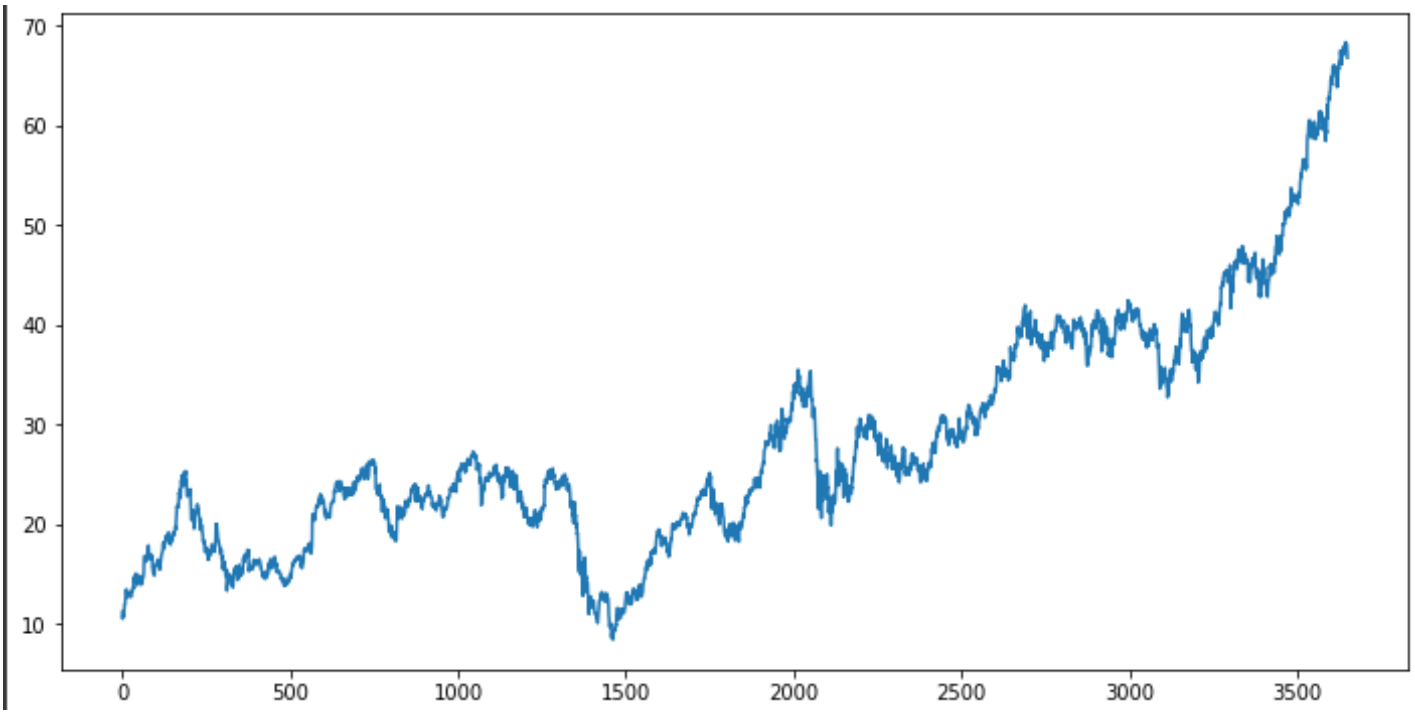


Comparing Decoded with Original curve:

The model does a really good job predicting the original curve. The noise (vertical lines) you can see is where each window connects, basically the last and first value of every window. ***More on that later**

Comparing Encoded with Original curve:





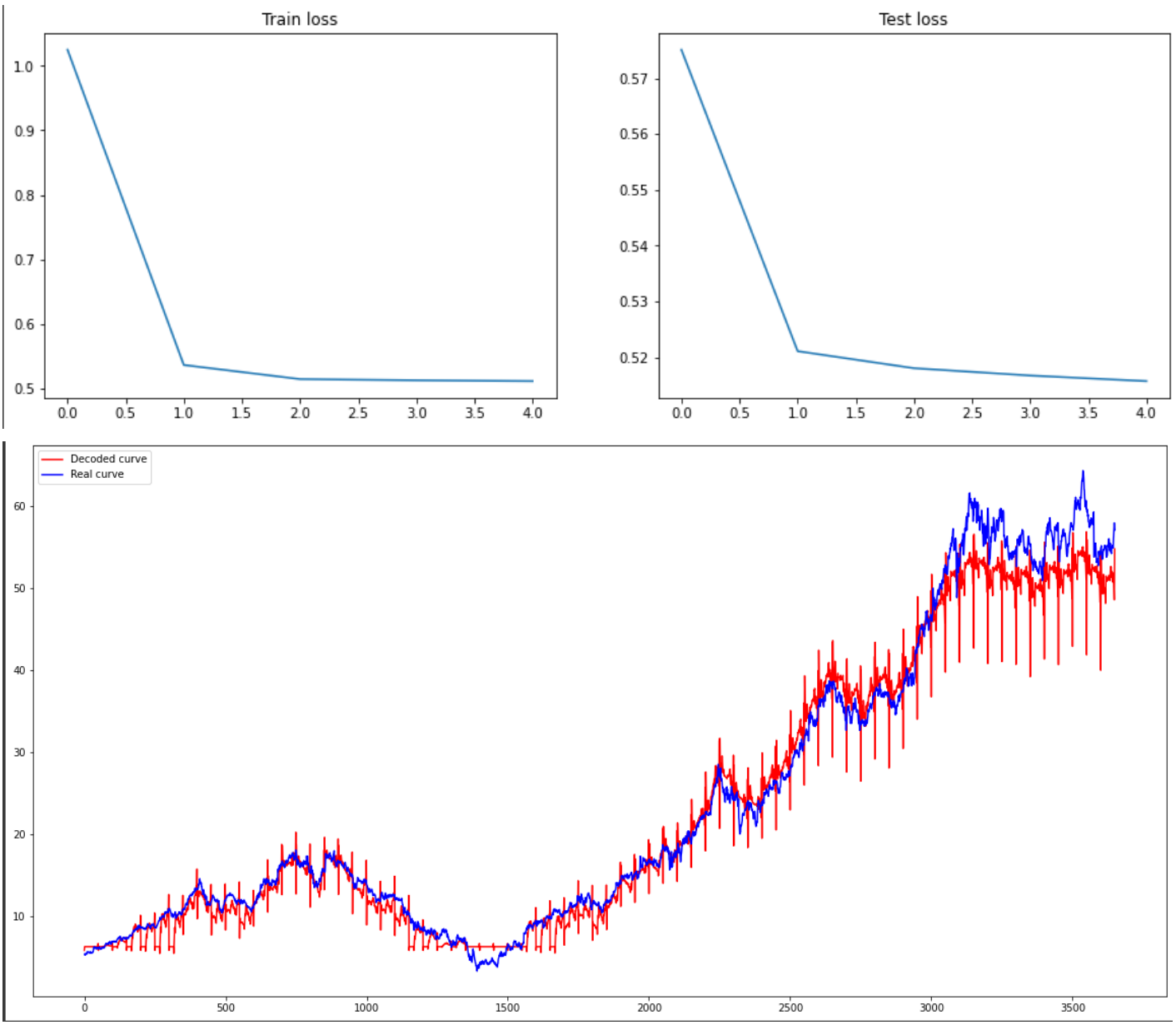
The complexity of the encoded curve is way smaller than the original, while maintaining pretty much the same shape, this is important for the data clustering in exercise four.

Experiments

Experimented with the below hyperparameters:

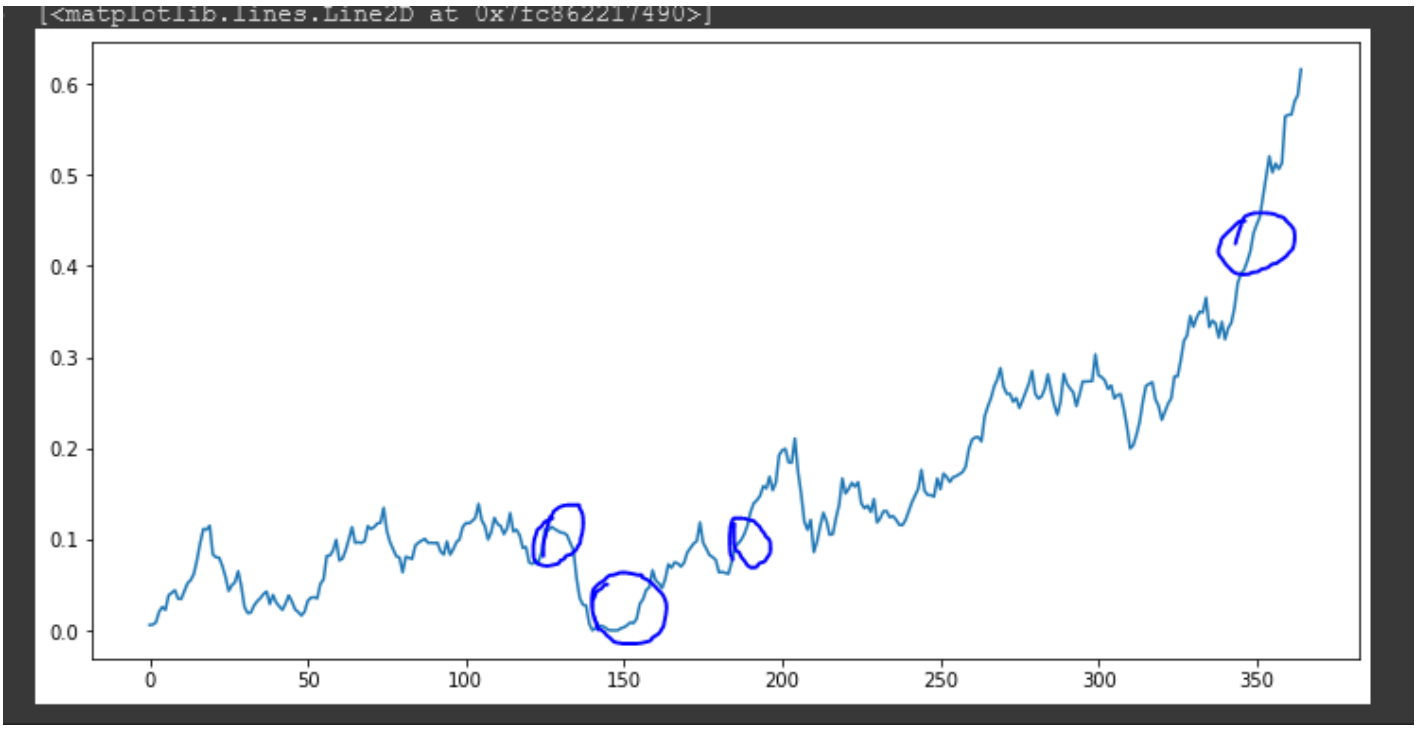
- With 3 encoder + 5 decoder layers. Surprisingly, the model did not overfit, however the decoded and encoded curves were not as good (as shown below).

```
Epoch 1/5
83/83 [==============================] - 4s 30ms/step - loss: 1.0250 - val_loss: 0.5751
Epoch 2/5
83/83 [==============================] - 2s 27ms/step - loss: 0.5362 - val_loss: 0.5211
Epoch 3/5
83/83 [==============================] - 2s 27ms/step - loss: 0.5143 - val_loss: 0.5180
Epoch 4/5
83/83 [==============================] - 2s 27ms/step - loss: 0.5124 - val_loss: 0.5167
Epoch 5/5
83/83 [==============================] - 2s 27ms/step - loss: 0.5112 - val_loss: 0.5157
```

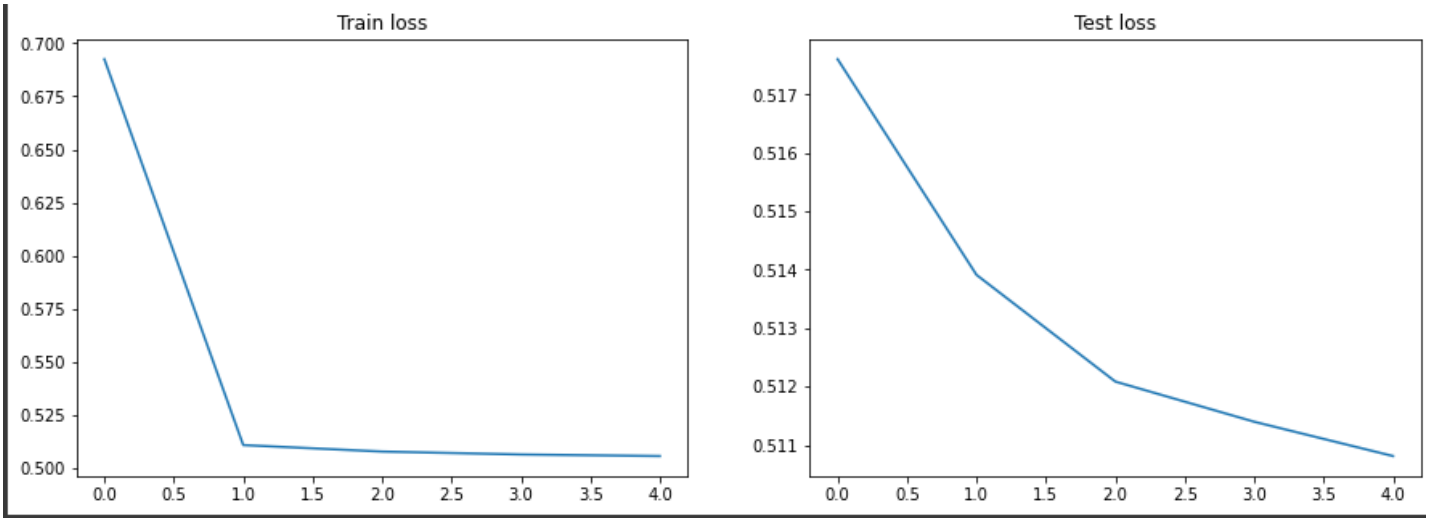


- With less than 2 encoder + 4 decoder layers would need more epochs to converge to the ideal loss and training state.

- With a bigger batch size than 32 (e.g. 128) the loss is pretty much the same, but the encoded curve loses some accuracy and detail, in comparison our best model.



- The number of filters is a tricky tweak. When trying less filters the loss is higher and the model starts to show signs of overfitting.



- On the other hand, there is no point trying more filters because after a certain point the results don't change.
- When it comes to encoding dimensions putting more than 3 (e.g. 6) the mode would start to overfit. With lower values than 3 the predictions are not as good.

About Encoded curves - Important

Notice that in our encoded curve we don't have the noise we talked about earlier. That's because originally the size of the encoded curve was 511 not 365.
I removed the last and first value of every window and as we will see later this improved our results in exercise four. Our algorithms had better clustering and identified the right n-Nearest Neighbours.

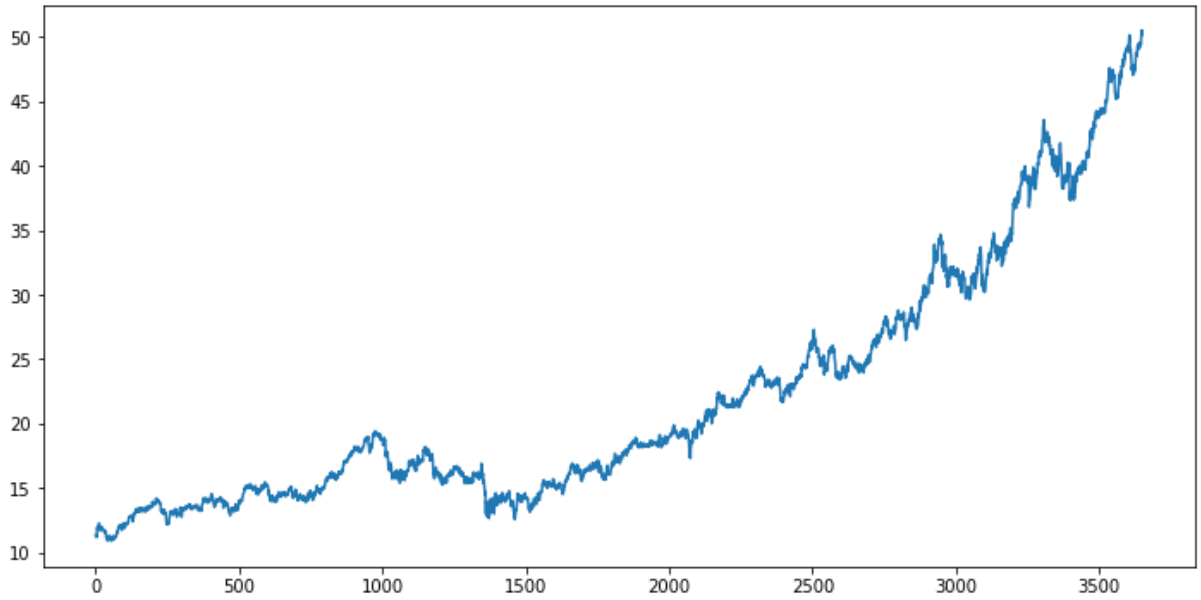
Exercise D

1. **We use the last 11 times series of the file query_nasdaq2007_17.csv for queries and the rest for input**
2. **The encoded curve complexity is 365**

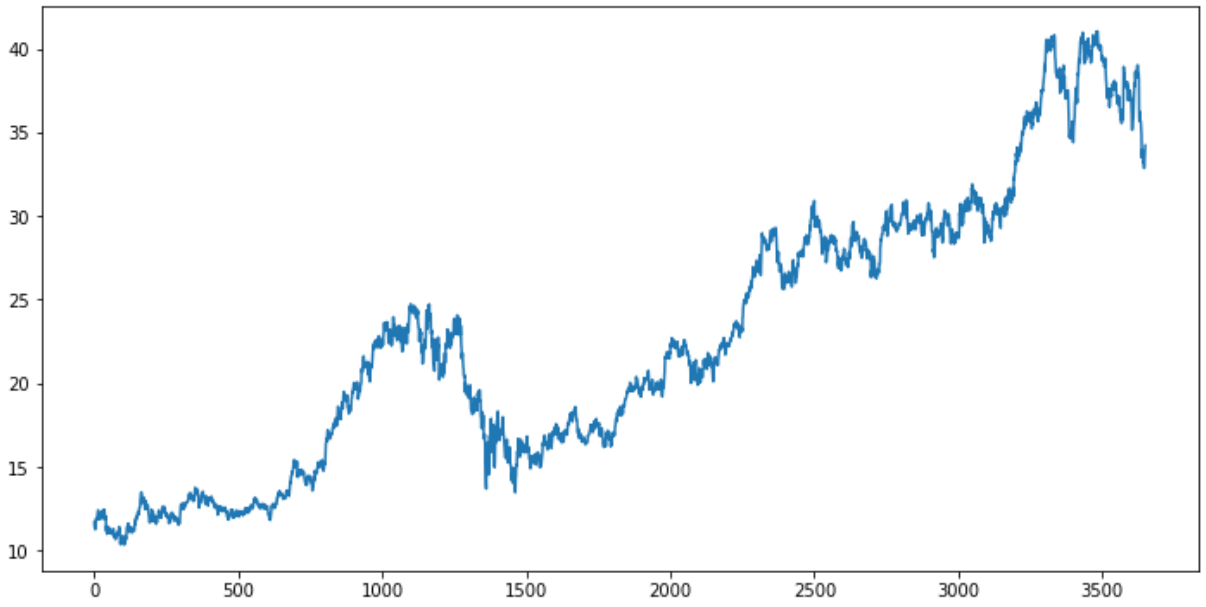
Nearest Neighbours (A)

I. You can find the results in the files out_encoded_LSH, out_encoded_Hypercube for the encoded time series and out_LSH, out_Hypercube for the original ones. When running LSH and Hypercube we noticed that the algorithms find the same nearest neighbour for both the encoded and the original time series except for two queries (xel and xrx). For the encoded xel curve the nearest curve is cms and for the original xel curve it's t.

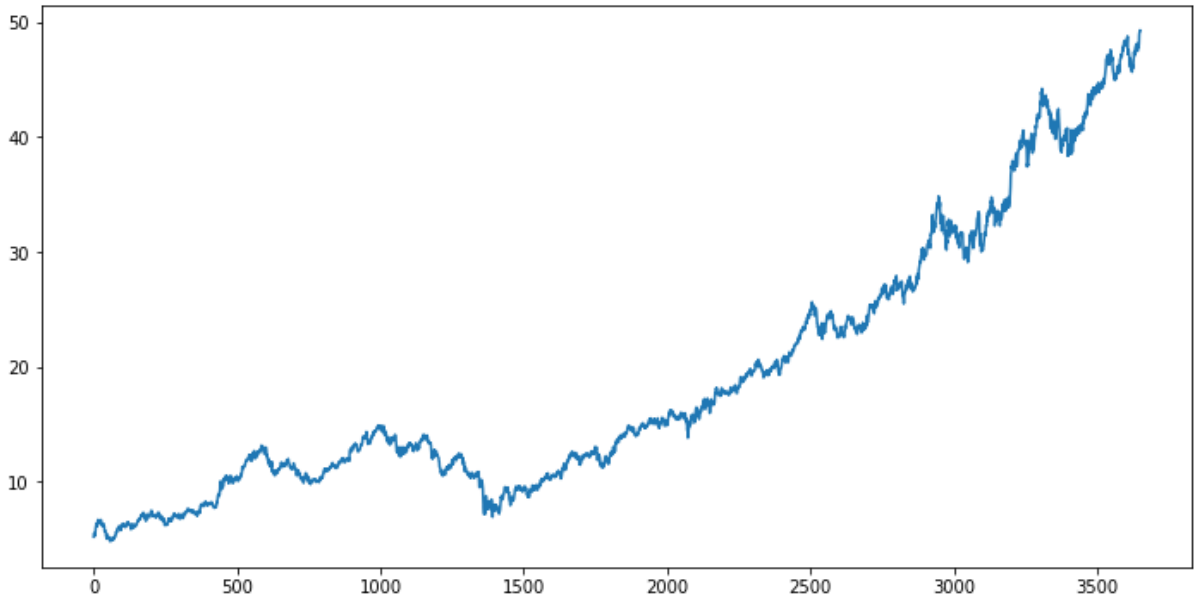
CURVE XEL



CURVE T



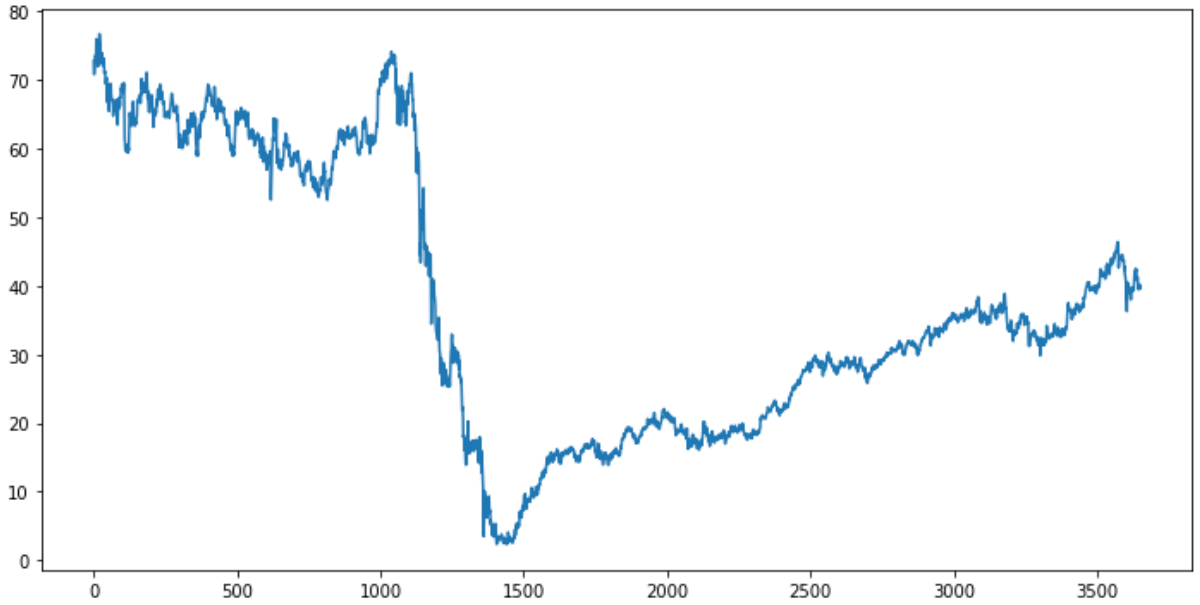
CURVE CMS



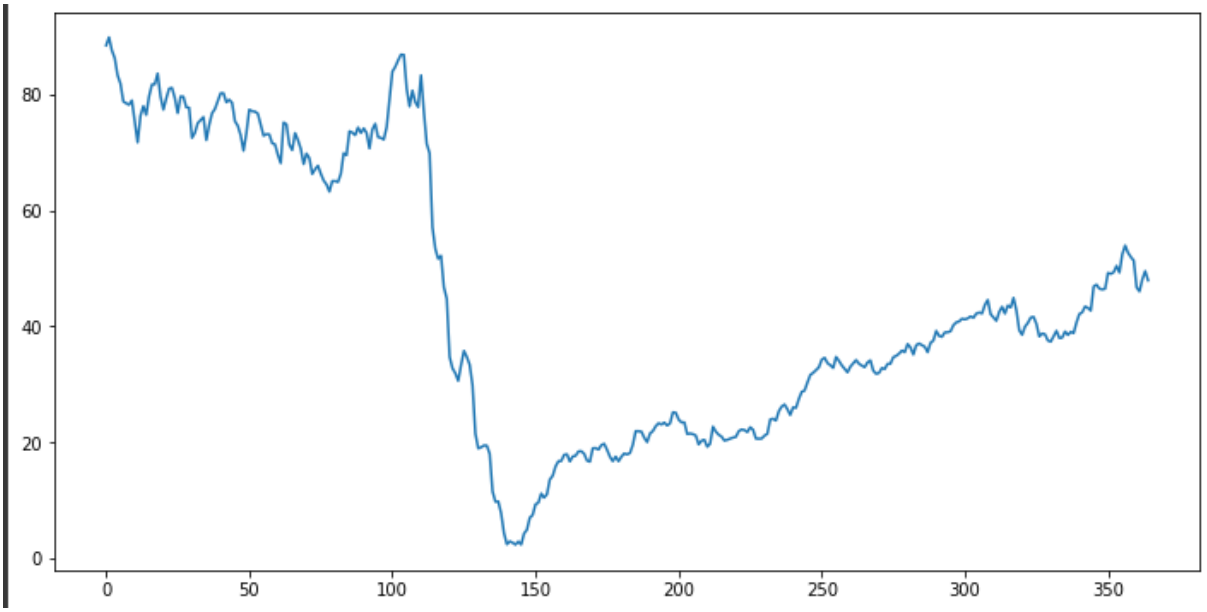
As you can see XEL is closer to CMS than T, therefore with the encoded curves the result is more accurate. We see this pattern in the following algorithms as well.

II. You can find the results in the files out_encoded_Discrete for the encoded time series and out_Discrete for the original ones. Similar to the previous one when running LSH Discrete we noticed that the algorithm found the same nearest neighbour for both the encoded and the original time series except for two queries (xl and yum). For the encoded yum curve the nearest curve is snv and for the original xl curve it's sti.

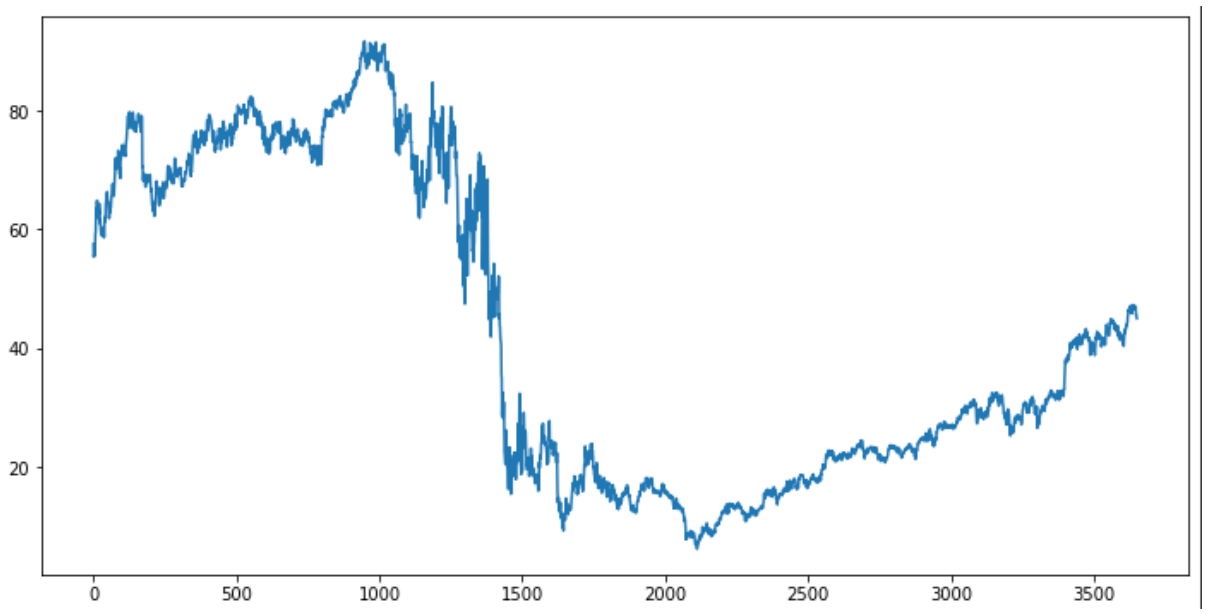
CURVE XL



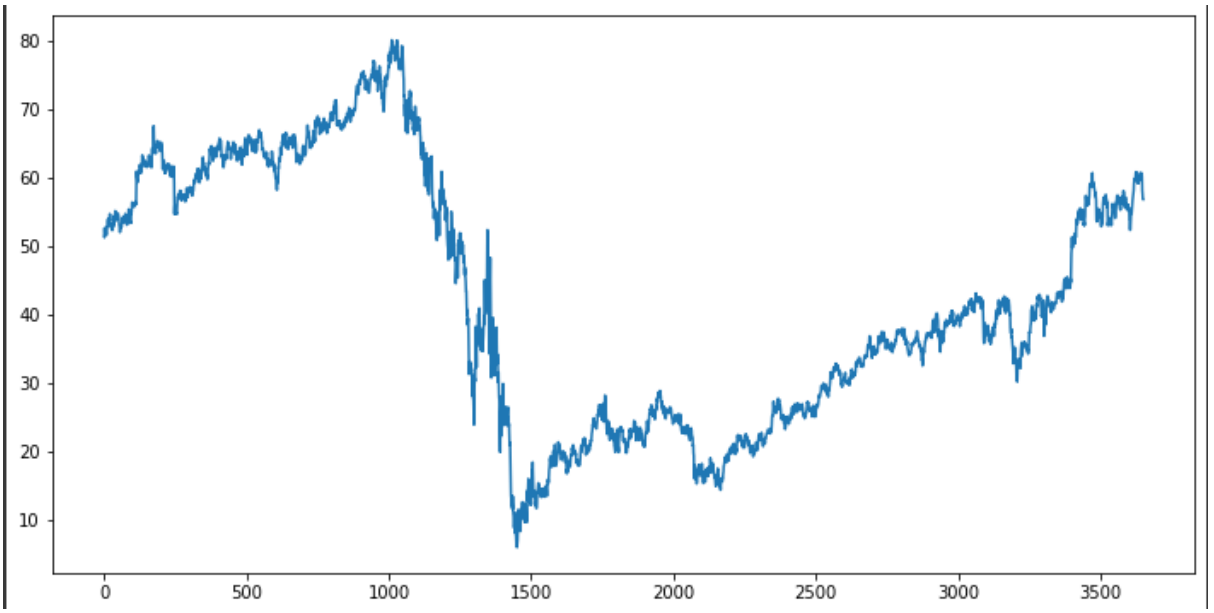
ENCODED CURVE XL



CURVE SNV



CURVE STI



<u>As you can see XL is close to both of them, therefore any answer would be correct.</u>

III. You can find the results in the files out_encoded_Continuous for the encoded time series and out_Continuous for the original ones. The algorithm LSH Continuous found the 4/11 same nearest neighbour for both the encoded and the original time series. The differences between the curves, as in the example of the discrete, were very close - so to not fill the page with screenshots we will not show the diagrams as above.

Clustering (B)

With 4 number of clusters:

The clusters that have 1 curve (outlier curves that have a much higher value than the rest of the data) have a silhouette value of 1.

1. Lloyd - Discrete distance (Frechet): (files out_encoded_Frechet_Lloyd and out_Frechet_Lloyd)

   As you can see from the results below the improvement in our clustering time while maintaining pretty much the same clustering.

   a. Encoded Data:
   cluster_time: 5.62128
   Silhouette: [0.986551,1,1,0.104862,0.772853]

   b. Original Data:
   cluster_time: 287.272
   Silhouette: [0.986205,1,1,0.125996,0.77805]

2. Lloyd - L2 distance (Mean_Vector): (files out_encoded_Vector_Lloyd and out_Vector_Lloyd)

   As you can see from the results below the improvement in our clustering time while maintaining pretty much the same clustering.

   a. Encoded Data:
   cluster_time: 0.0042066
   Silhouette: [1,-0.311419,-0.0819857,1,0.401649]

   b. Original Data:
   cluster_time: 0.0383419
   Silhouette: [1,-0.299396,-0.0699128,1,0.407673]

3. LSH Frechet: (files out_encoded_Frechet_LSH and out_Frechet_LSH)

   As you can see from the results below the clustering time is much lower than the original, while improving the silhouette score as well.

   a. Encoded Data:
   cluster_time: 10.9069
   Silhouette: [0.979425,1,1,0.153679,0.783276]

b. Original Data:
cluster_time: 490.225
Silhouette: [0.970047,1,1,-0.0473775,0.730667]

4. LSH L2 distance: (files out_encoded_Vector_LSH and out_Vector_LSH)

   As you can see from the results below the improvement in our clustering time while maintaining pretty much the same clustering.

   a. Encoded Data:
   cluster_time: 0.132709
   Silhouette: [0.854071,1,1,1,0.963518]

   b. Original Data:
   cluster_time: 1.27344
   Silhouette: [0.860451,1,1,1,0.965113]

5. Hypercube: (files out_encoded_Cluster_Hypercube and out_Cluster_Hypercube)

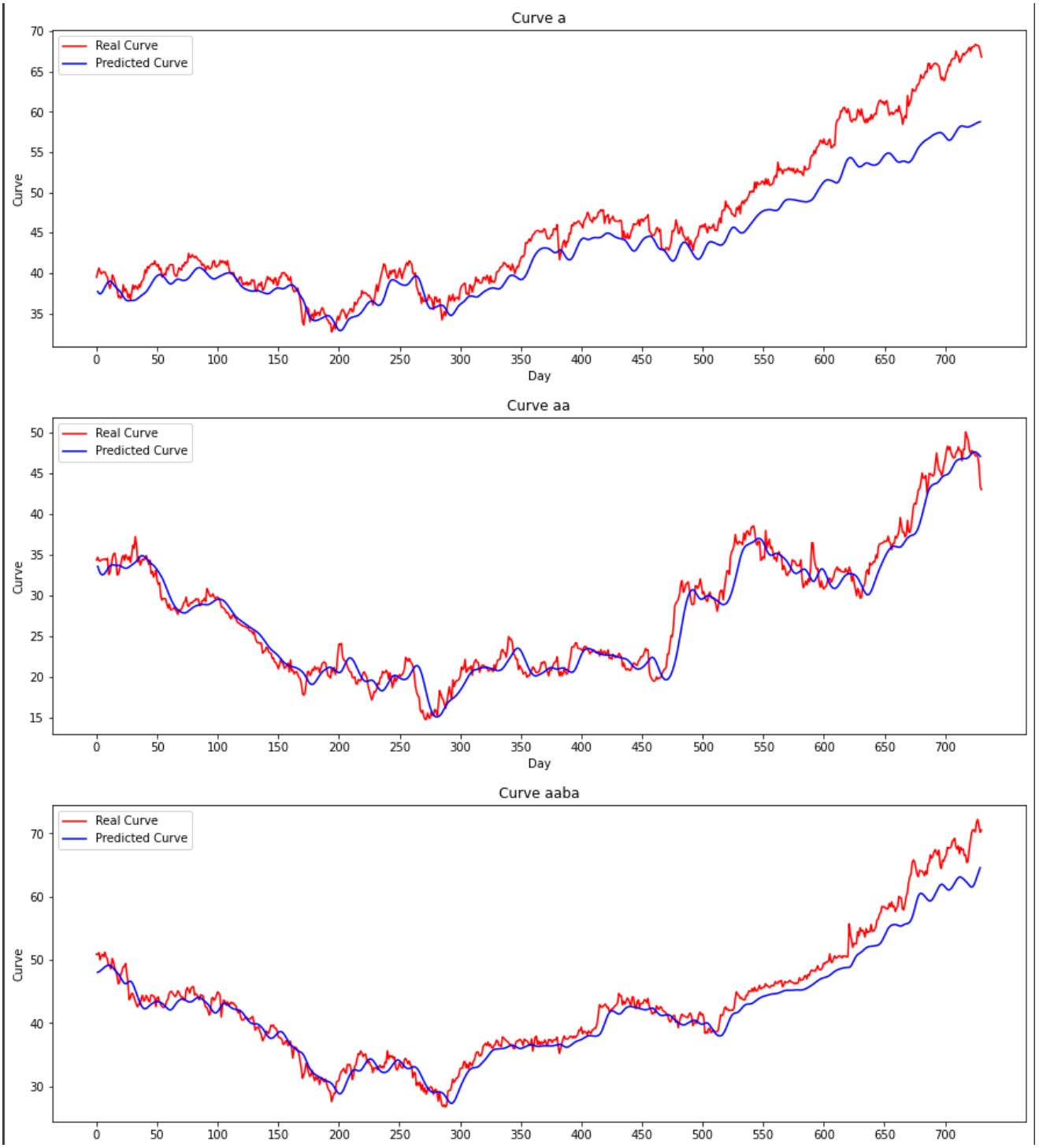
   As you can see from the results below the improvement in our clustering time while maintaining pretty much the same clustering.

   a. Encoded Data:
   cluster_time: 0.0130864
   Silhouette: [0.821787,1,1,1,0.955447]

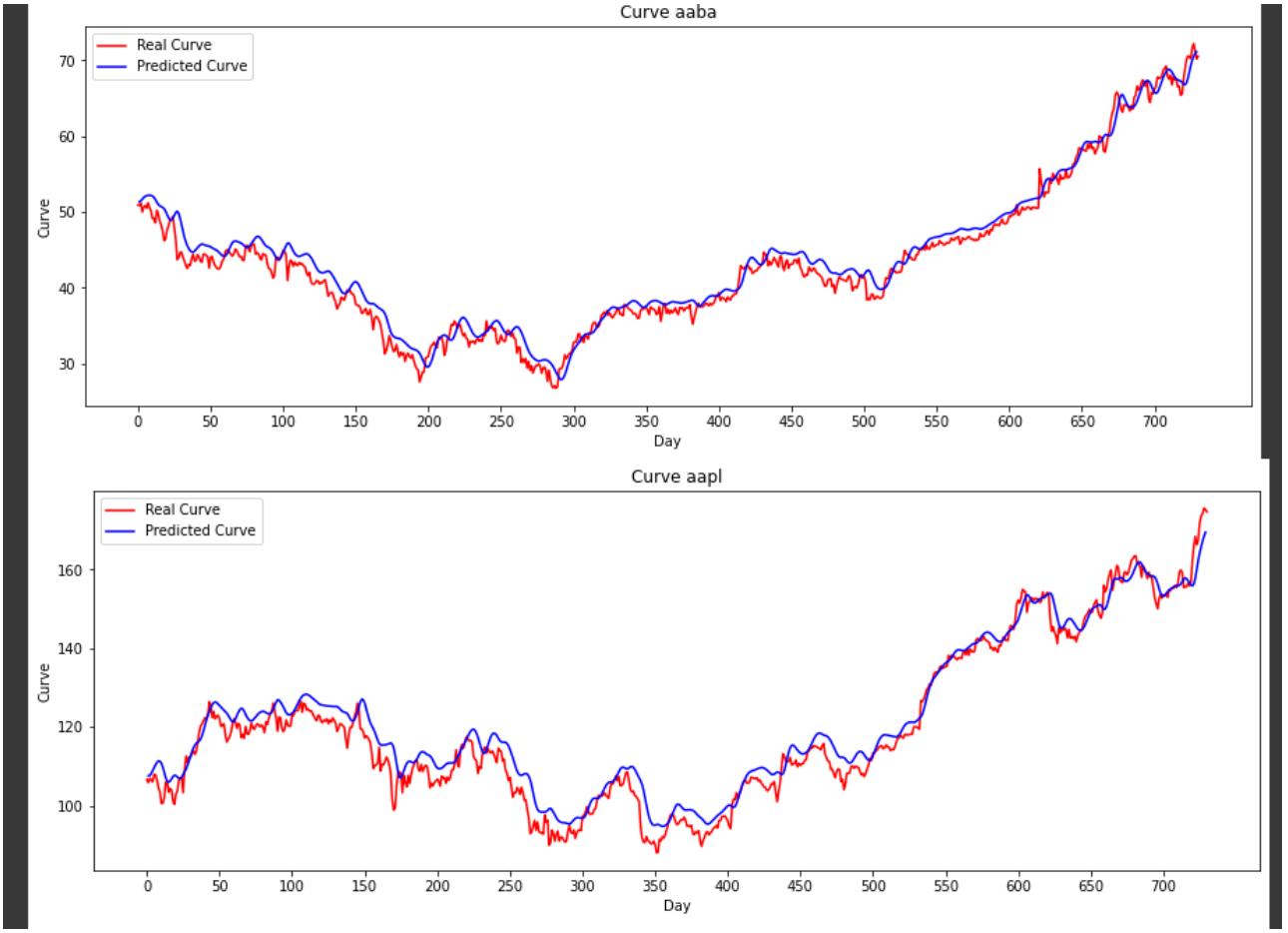
   b. Original Data:
   cluster_time: 0.113866
   Silhouette: [0.828886,1,1,1,0.957222]

Some Experiment Images

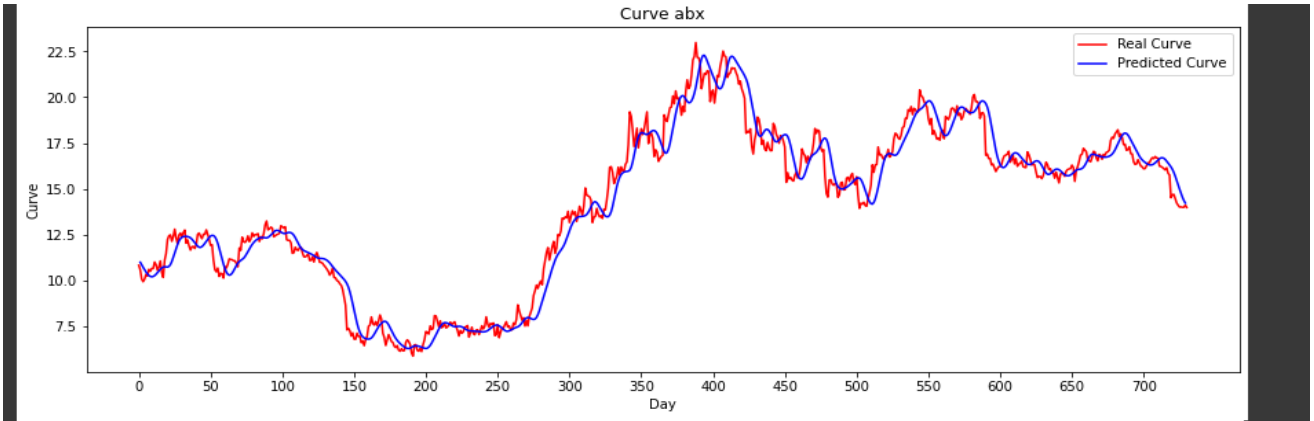
1)

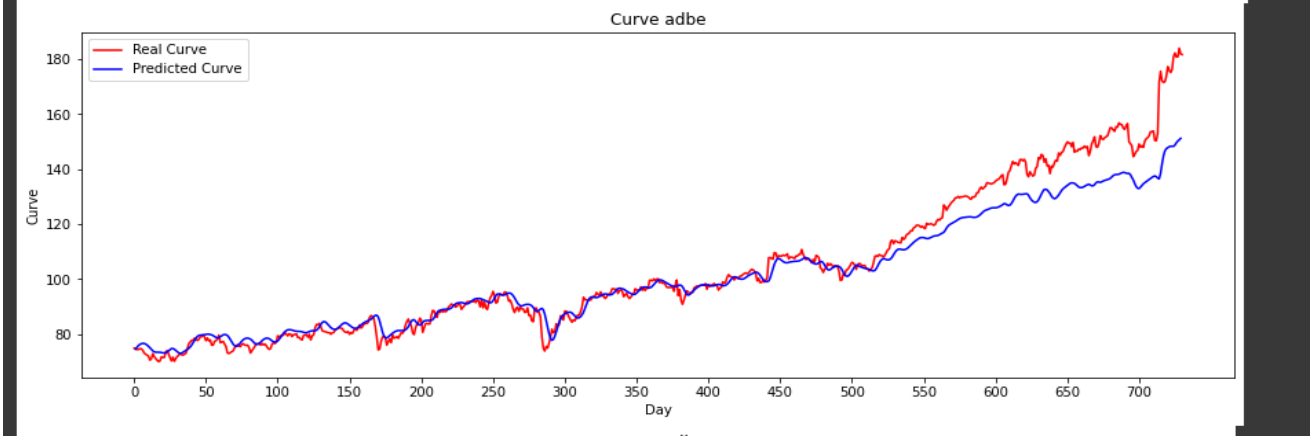


2)

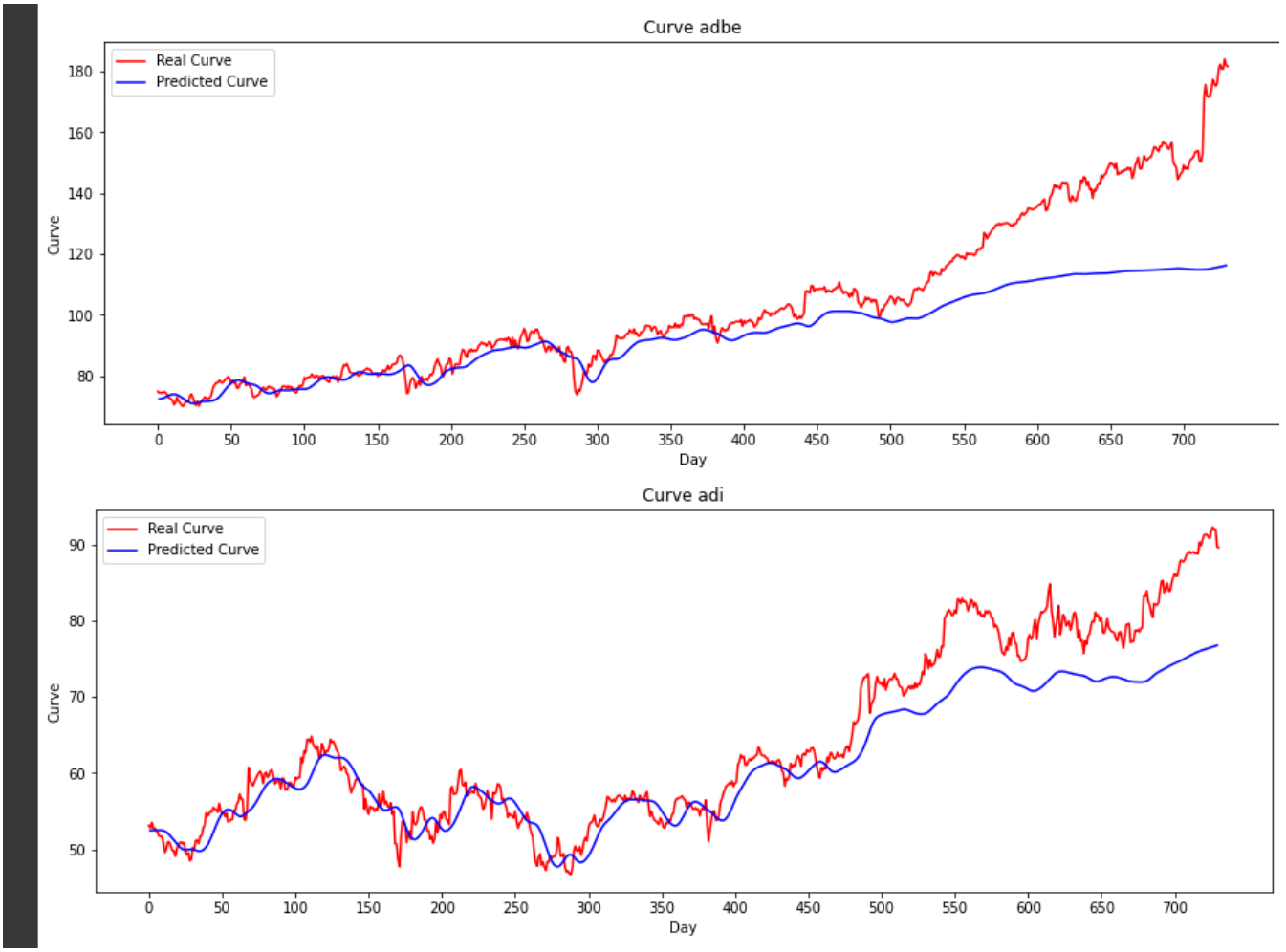

Curve aaba
Real Curve
Predicted Curve
Curve
Day

Curve aapl
Real Curve
Predicted Curve
Curve
Day

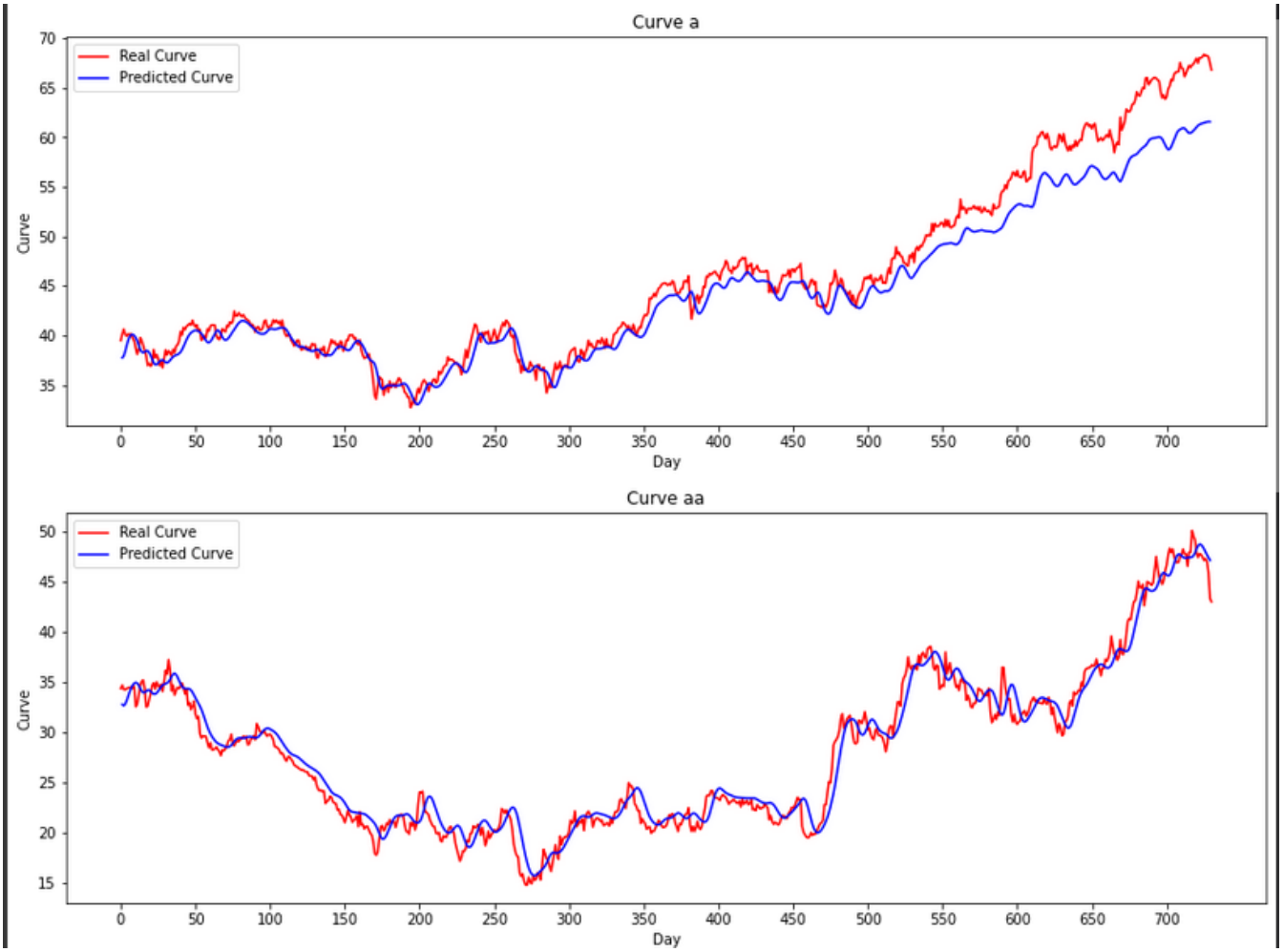

Curve abx
Real Curve
Predicted Curve
Curve
Day

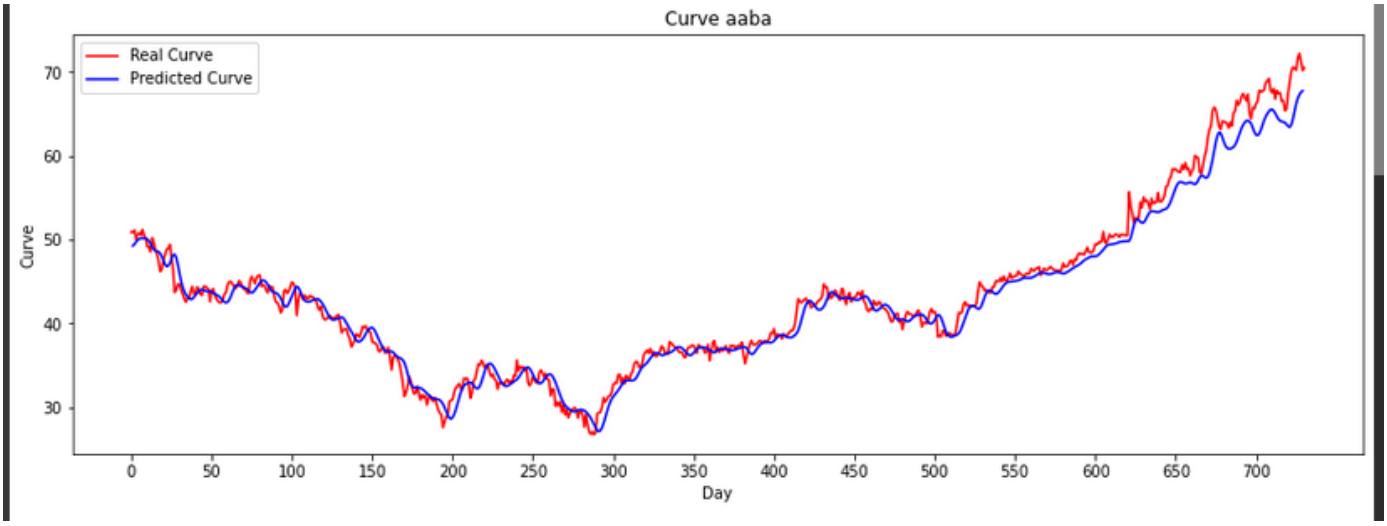

Curve adbe
Real Curve
Predicted Curve
Curve
Day

3)

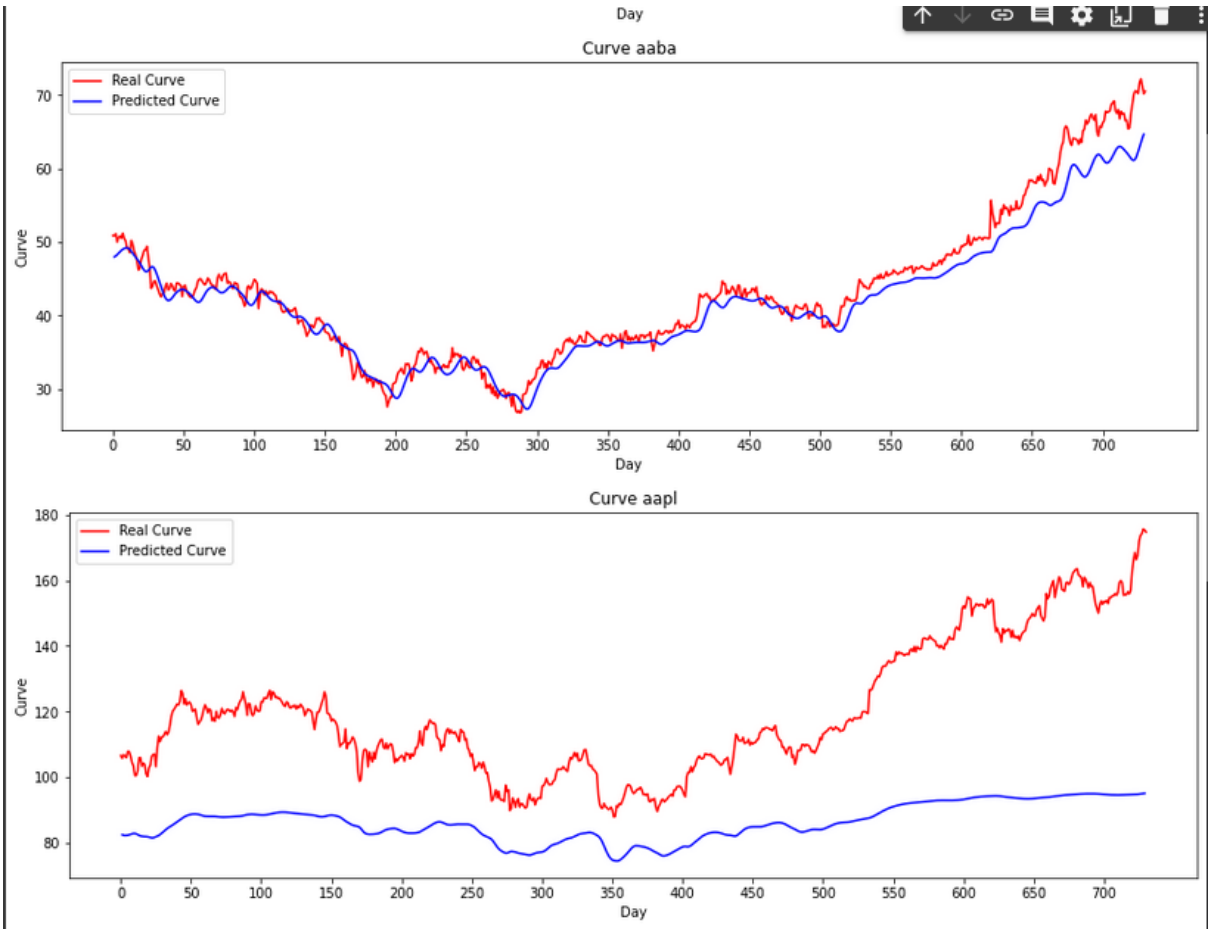

Curve adbe
Real Curve
Predicted Curve
Curve
Day
Curve adi
Real Curve
Predicted Curve
Curve
Day

4)

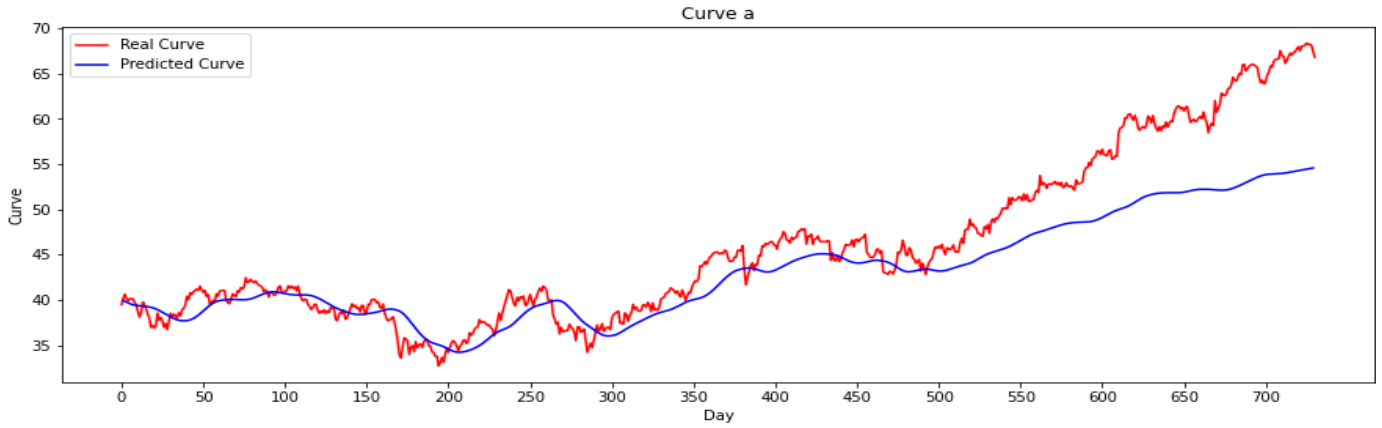

Curve a
Real Curve
Predicted Curve
Curve
Day
Curve aa
Real Curve
Predicted Curve
Curve
Day

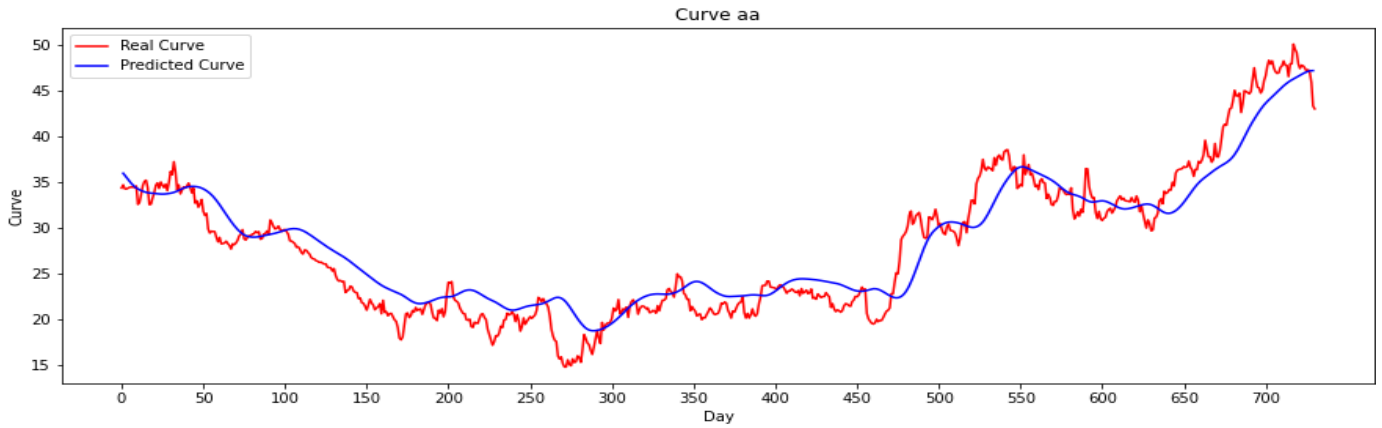

Curve aaba
Real Curve
Predicted Curve
Curve
Day

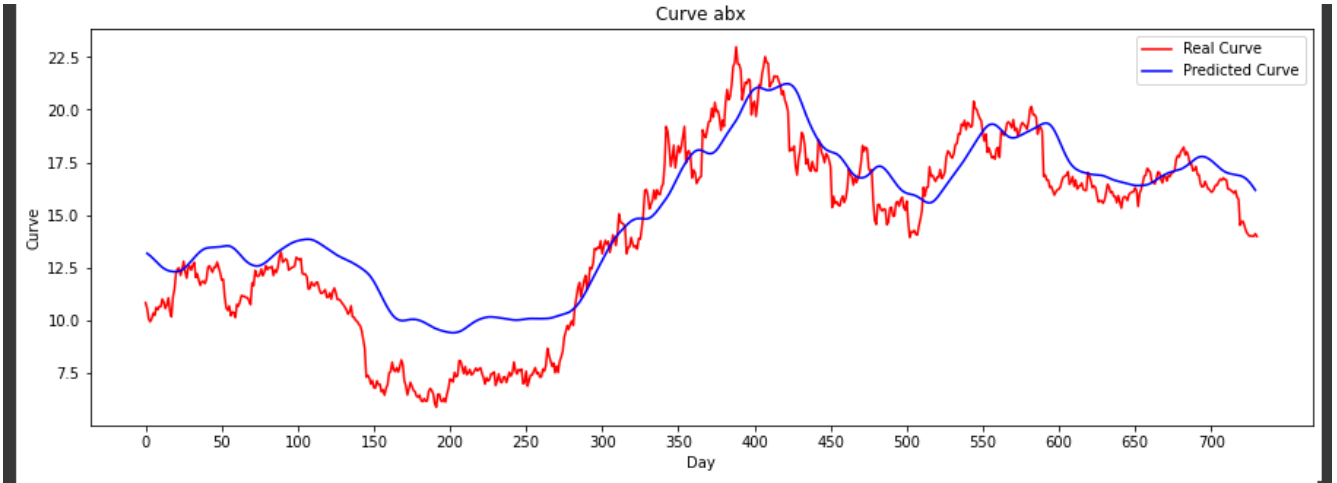
5)

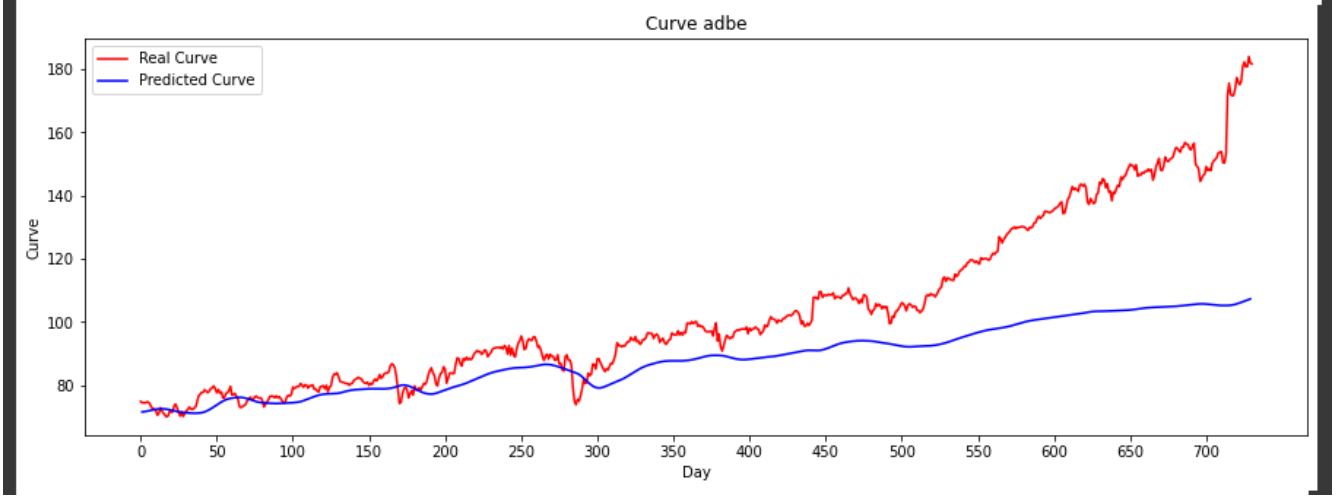

Day
Curve aaba
Real Curve
Predicted Curve
Curve
Day
Curve aapl
Real Curve
Predicted Curve
Curve
Day

6)

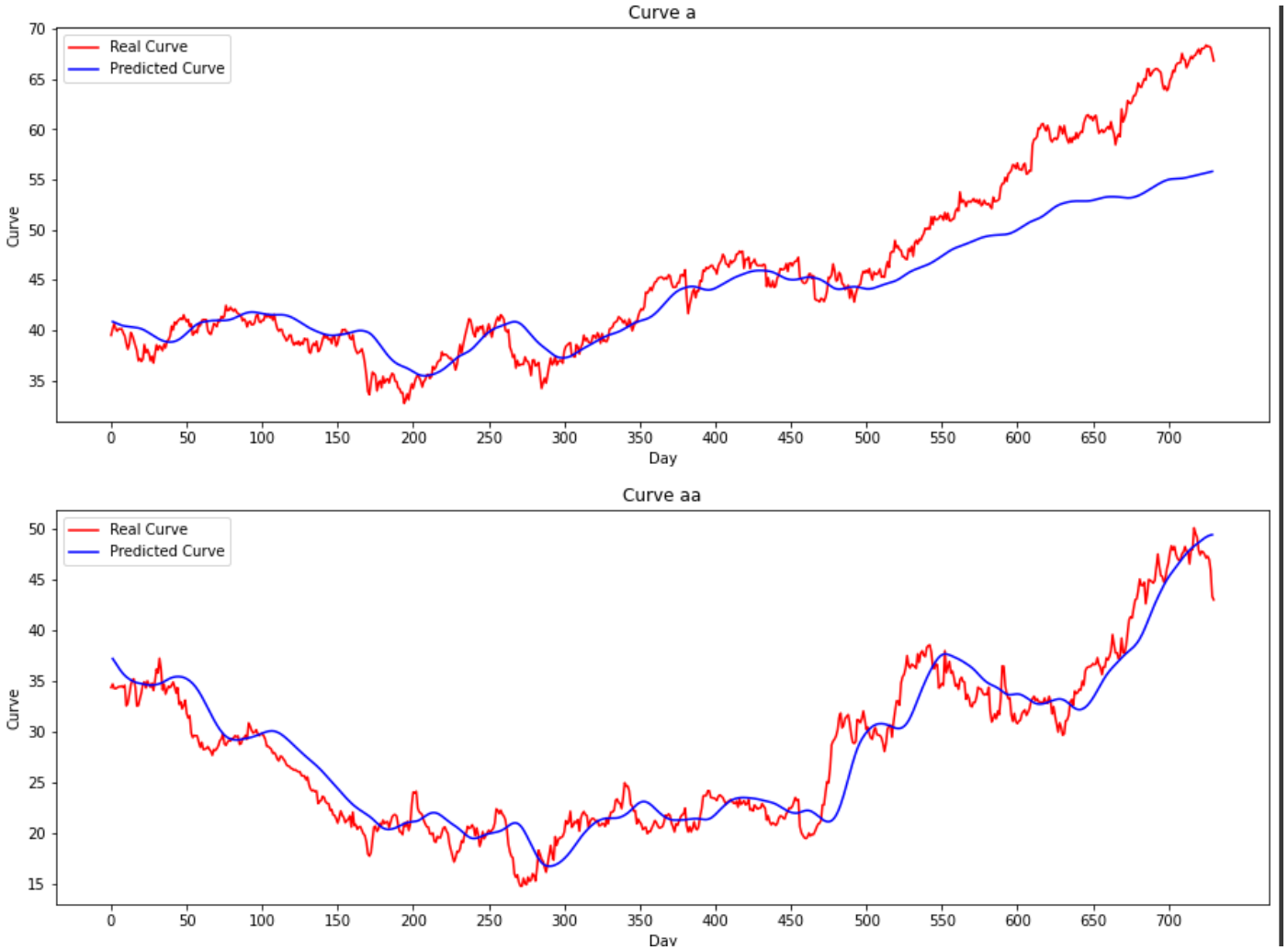

Curve a
Real Curve
Predicted Curve
Curve
Day


Curve aa
Real Curve
Predicted Curve
Curve
Day

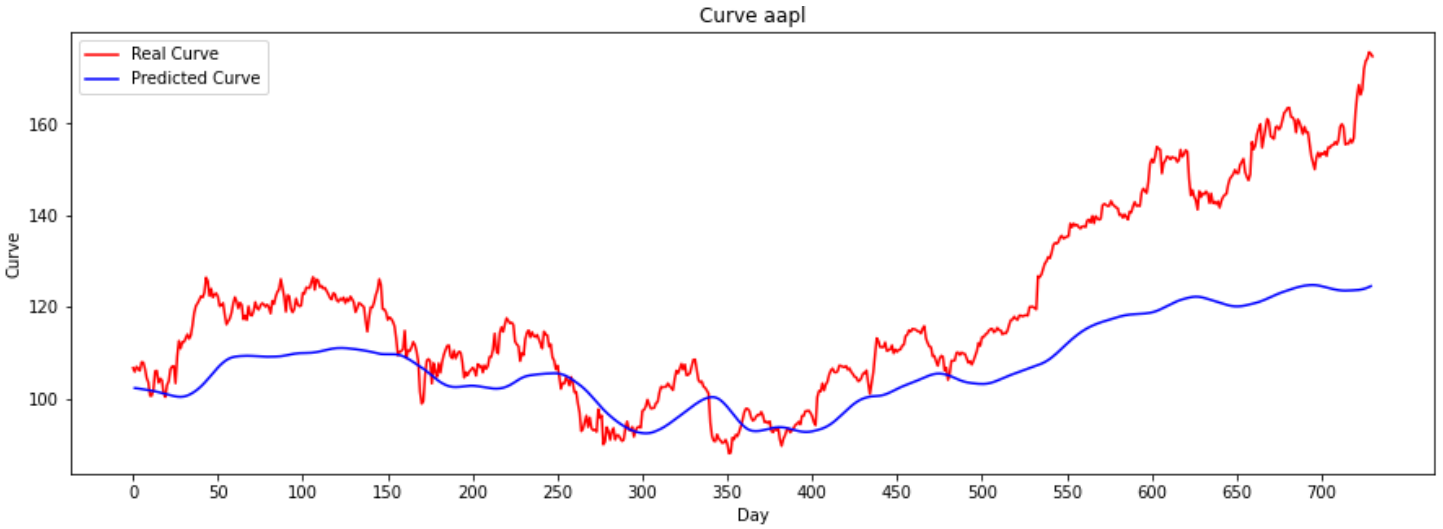

Curve abx
Real Curve
Predicted Curve
Curve
Day

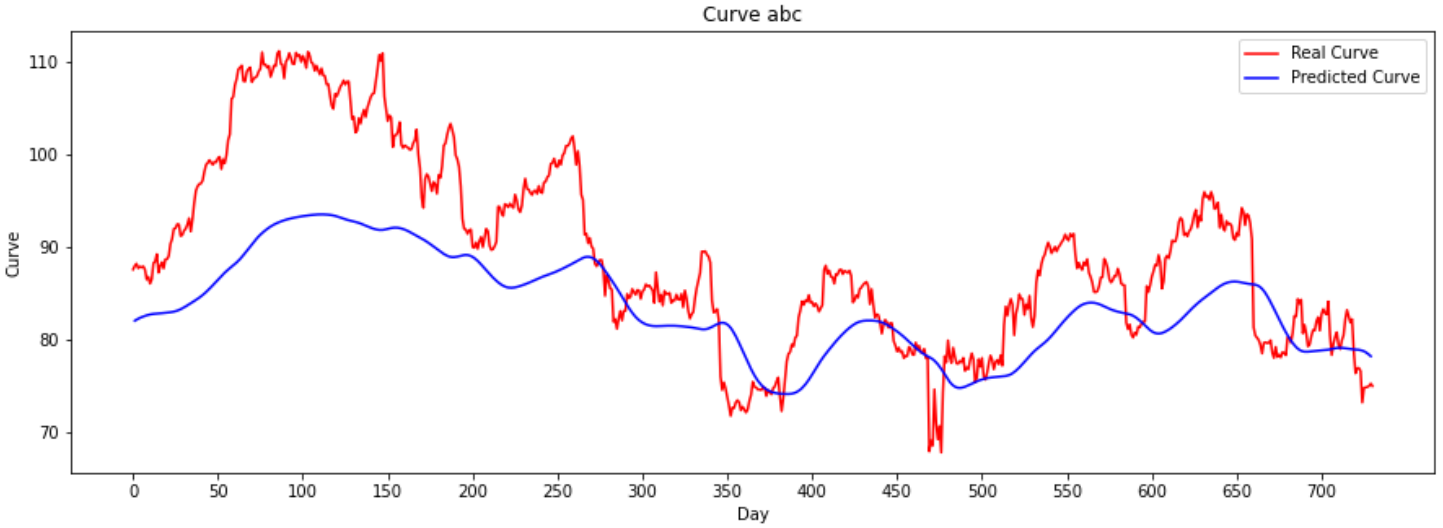

Curve adbe
Real Curve
Predicted Curve
Curve
Day

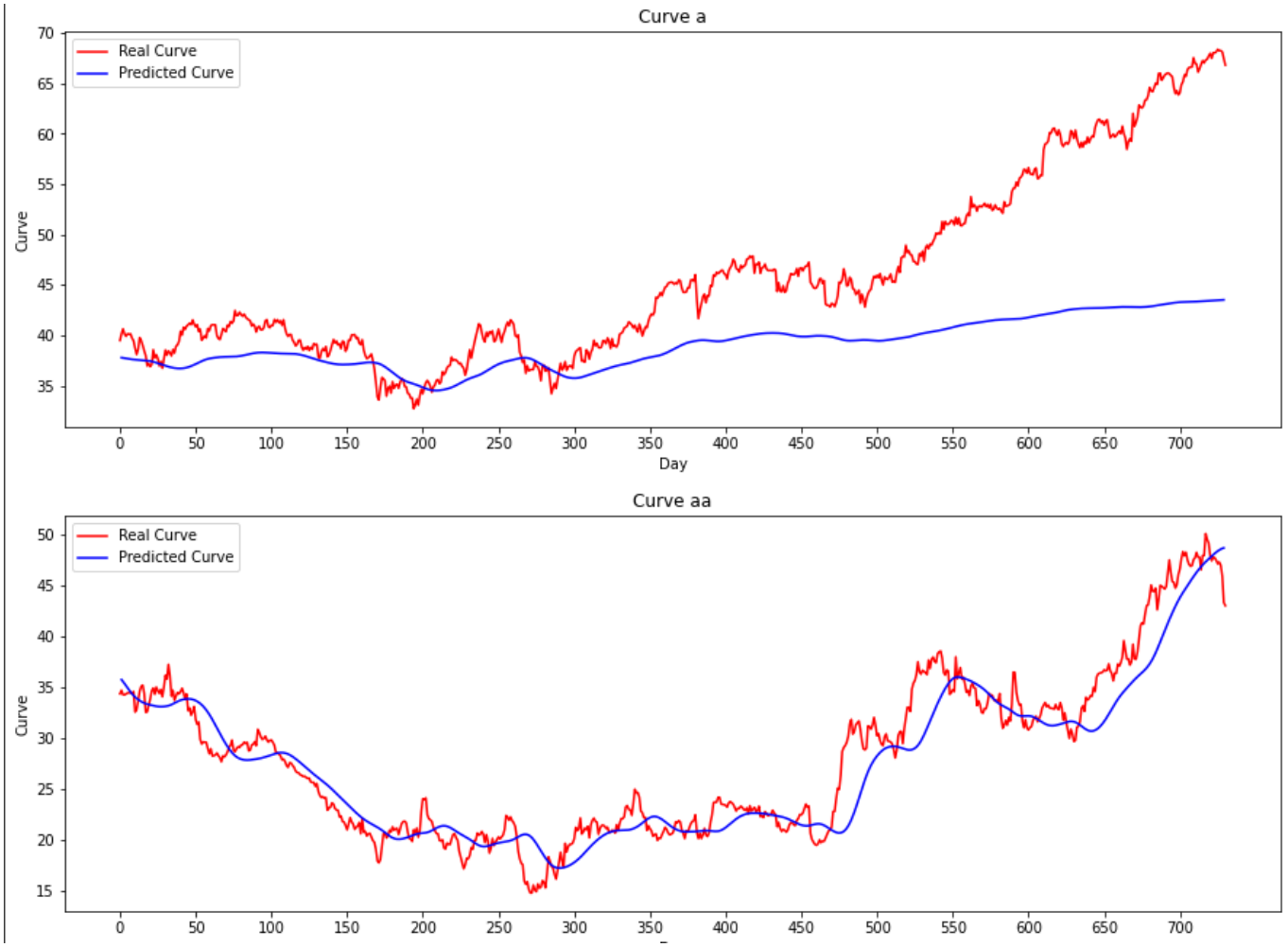
7)


Curve a
Real Curve
Predicted Curve
Curve
Day
Curve aa
Real Curve
Predicted Curve
Curve
Day

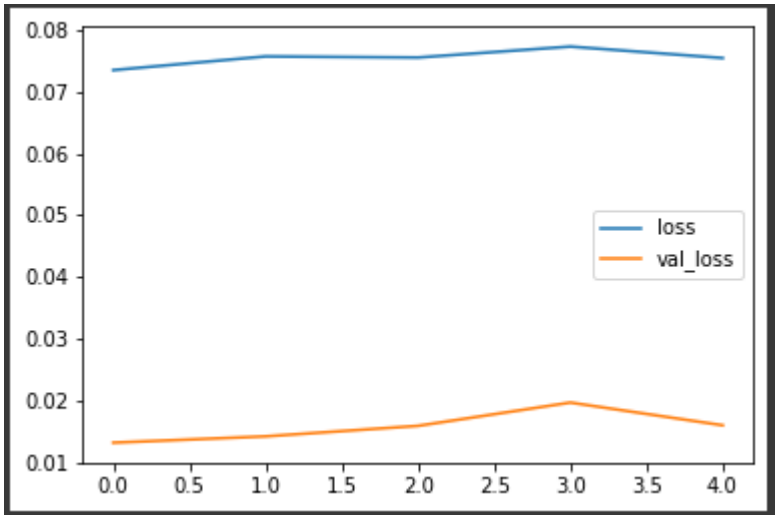
8)


Curve aapl
Real Curve
Predicted Curve
Curve
Day
100
120
140
160
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700

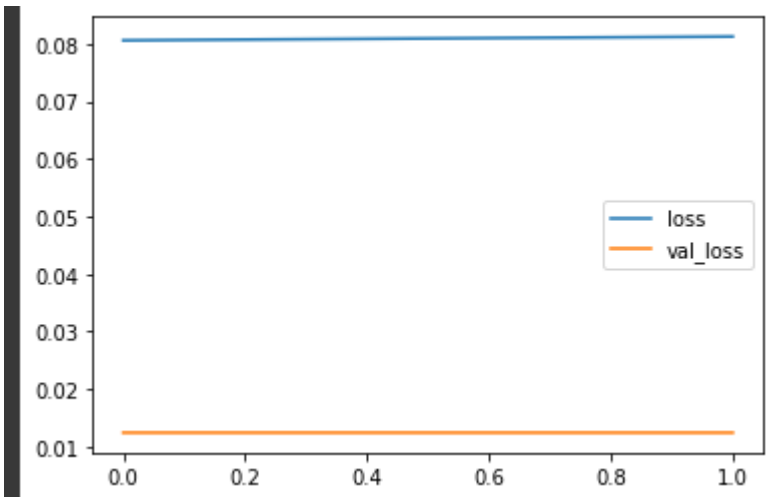

Curve abc
Real Curve
Predicted Curve
Curve
Day
70
80
90
100
110
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700


Curve a
Real Curve
Predicted Curve
Curve
Day
Curve aa
Real Curve
Predicted Curve
Curve

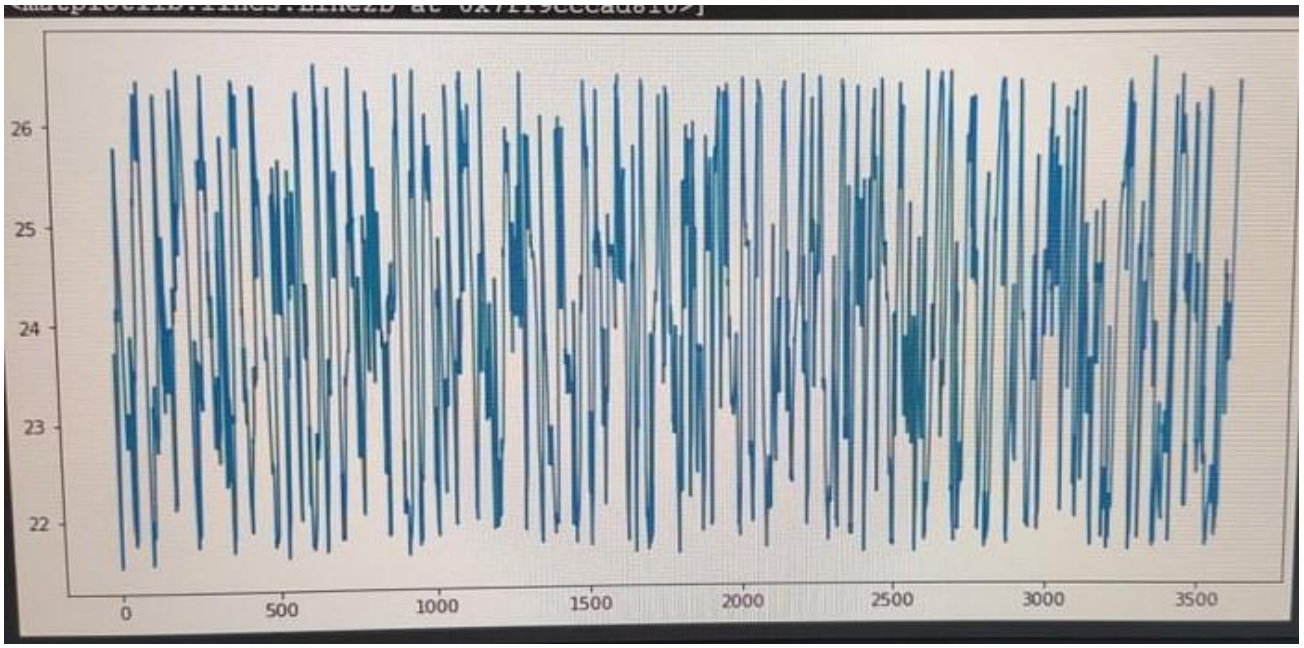
9)


loss
val_loss

10)


0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.0
0.2
0.4
0.6
0.8
1.0
loss
val_loss

11)


26
25
24
23
22
0
500
1000
1500
2000
2500
3000
3500