Testing Enviornment

In [1]:
```
import pandas as pd
import numpy as np
```

Running tests

The following section unit-tests most of the code written for the proof of concept prototype.

In [76]:
```
%run ../test/test_algorithms.py
```

```
..................
----------------------------------------------------------------------
Ran 17 tests in 0.218s

OK
```

In [77]:
```
%run ../test/test_dataloader.py
```

```
...............
----------------------------------------------------------------------
Ran 15 tests in 0.030s

OK
```

In [78]:
```
%run ../test/test_predictorsI.py
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.003s

OK
```

In [79]:
```
%run ../test/test_predictorsII.py
```

```
.
----------------------------------------------------------------------
Ran 1 test in 0.004s

OK
```

Running example of the system

In [6]:
```
%run ../tools/algorithms.py
```

In [7]:
```
%run ../tools/dataloader.py
```

In [8]:
```
test = DataLoader('aapl', '2009-01-01', '2010-02-10')
```

In [9]:
```
prices = test.get_close()
```

In [10]:
```
prices
```

Out[10]:

Date	Close
2009-01-02	3.241071
2009-01-05	3.377857
2009-01-06	3.322143
2009-01-07	3.250357
2009-01-08	3.310714
...	...
2010-02-04	6.858929
2010-02-05	6.980714
2010-02-08	6.932857
2010-02-09	7.006786
2010-02-10	6.968571

279 rows × 1 columns

In [11]:
```
#prices = np.array(prices)
#len(prices)
```


In [12]:
```
prices
```

Out[12]:

Date	Close
2009-01-02	3.241071
2009-01-05	3.377857
2009-01-06	3.322143
2009-01-07	3.250357
2009-01-08	3.310714
...	...
2010-02-04	6.858929
2010-02-05	6.980714
2010-02-08	6.932857
2010-02-09	7.006786
2010-02-10	6.968571

279 rows × 1 columns

In [13]:
```
%run ../tools/predictorsI.py
```

In [14]:
```
op0 = BasicUnivariatePredictor(prices, 25, 7)
op1 = BasicUnivariatePredictor(prices, 25, 7)
op2 = BasicUnivariatePredictor(prices, 25, 7)
op3 = BasicUnivariatePredictor(prices, 25, 7)
```

In [15]:
```
op0.create_bilstm()
```

In [16]:
```
op0.model_blueprint()
```

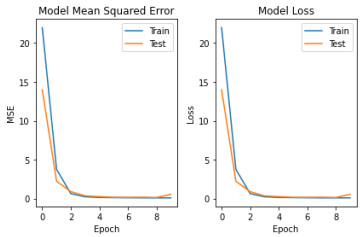
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectional (None, 25, 100)           20800
_________________________________________________________________
lstm_1 (LSTM)                (None, 50)                30200
_________________________________________________________________
dense (Dense)                (None, 7)                 357
=================================================================
Total params: 51,357
Trainable params: 51,357
Non-trainable params: 0
_________________________________________________________________
```

In [17]:
```
op0.fit_model(10)
```

```
Epoch 1/10
20/20 [==============================] - 1s 46ms/step - loss: 21.9913 - mean_squared_error: 21.9913 - val_loss: 13.9966 - val_mean_squa
red_error: 13.9966
Epoch 2/10
20/20 [==============================] - 0s 18ms/step - loss: 3.7463 - mean_squared_error: 3.7463 - val_loss: 2.2233 - val_mean_squared
_error: 2.2233
Epoch 3/10
20/20 [==============================] - 0s 17ms/step - loss: 0.6251 - mean_squared_error: 0.6251 - val_loss: 0.8816 - val_mean_squared
_error: 0.8816
Epoch 4/10
20/20 [==============================] - 0s 16ms/step - loss: 0.2216 - mean_squared_error: 0.2216 - val_loss: 0.3390 - val_mean_squared
_error: 0.3390
Epoch 5/10
20/20 [==============================] - 0s 16ms/step - loss: 0.1360 - mean_squared_error: 0.1360 - val_loss: 0.2437 - val_mean_squared
_error: 0.2437
Epoch 6/10
20/20 [==============================] - 0s 17ms/step - loss: 0.1148 - mean_squared_error: 0.1148 - val_loss: 0.1558 - val_mean_squared
_error: 0.1558
Epoch 7/10
20/20 [==============================] - 0s 16ms/step - loss: 0.0982 - mean_squared_error: 0.0982 - val_loss: 0.1503 - val_mean_squared
_error: 0.1503
Epoch 8/10
20/20 [==============================] - 0s 18ms/step - loss: 0.0824 - mean_squared_error: 0.0824 - val_loss: 0.1696 - val_mean_squared
_error: 0.1696
Epoch 9/10
20/20 [==============================] - 0s 18ms/step - loss: 0.0788 - mean_squared_error: 0.0788 - val_loss: 0.1324 - val_mean_squared
_error: 0.1324
Epoch 10/10
20/20 [==============================] - 0s 17ms/step - loss: 0.0806 - mean_squared_error: 0.0806 - val_loss: 0.5354 - val_mean_squared
_error: 0.5354
```

Out[17]: <tensorflow.python.keras.callbacks.History at 0x1950b89fd60>

In [18]:
```
op0.show_performance()
```



In [19]:
```
oyea = prices[-26:-1]
#oyea = X[-1]
#oyea
```

In [20]:
```
nice = op0.predict(oyea)
nice
```

Out[20]:

	Bidirectional LSTM
0	6.981127
1	7.293480
2	6.425726
3	7.495827
4	7.578663
5	6.643592
6	8.182841

In [21]:
```
op1.create_lstm()
```

In [22]:
```
op1.model_blueprint()
```

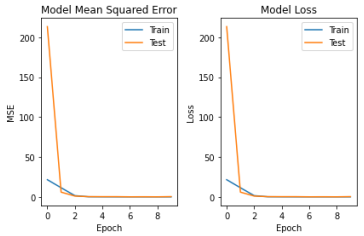
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 25, 40)            6720
_________________________________________________________________
lstm_3 (LSTM)                (None, 25, 50)            18200
_________________________________________________________________
lstm_4 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 357
=================================================================
Total params: 45,477
Trainable params: 45,477
Non-trainable params: 0
_________________________________________________________________
```


In [23]:
```
op1.fit_model(10)
```

```
Epoch 1/10
20/20 [==============================] - 1s 46ms/step - loss: 21.5897 - mean_squared_error: 21.5897 - val_loss: 213.2412 - val_mean_squared_error: 213.2412
Epoch 2/10
20/20 [==============================] - 1s 25ms/step - loss: 11.5143 - mean_squared_error: 11.5143 - val_loss: 6.0122 - val_mean_squared_error: 6.0122
Epoch 3/10
20/20 [==============================] - 0s 22ms/step - loss: 1.6485 - mean_squared_error: 1.6485 - val_loss: 1.1164 - val_mean_squared_error: 1.1164
Epoch 4/10
20/20 [==============================] - 0s 19ms/step - loss: 0.2219 - mean_squared_error: 0.2219 - val_loss: 0.3703 - val_mean_squared_error: 0.3703
Epoch 5/10
20/20 [==============================] - 0s 20ms/step - loss: 0.1322 - mean_squared_error: 0.1322 - val_loss: 0.3340 - val_mean_squared_error: 0.3340
Epoch 6/10
20/20 [==============================] - 0s 21ms/step - loss: 0.0992 - mean_squared_error: 0.0992 - val_loss: 0.3363 - val_mean_squared_error: 0.3363
Epoch 7/10
20/20 [==============================] - 0s 21ms/step - loss: 0.0885 - mean_squared_error: 0.0885 - val_loss: 0.1426 - val_mean_squared_error: 0.1426
Epoch 8/10
20/20 [==============================] - 0s 20ms/step - loss: 0.0785 - mean_squared_error: 0.0785 - val_loss: 0.2126 - val_mean_squared_error: 0.2126
Epoch 9/10
20/20 [==============================] - 0s 20ms/step - loss: 0.0811 - mean_squared_error: 0.0811 - val_loss: 0.1371 - val_mean_squared_error: 0.1371
Epoch 10/10
20/20 [==============================] - 0s 20ms/step - loss: 0.0775 - mean_squared_error: 0.0775 - val_loss: 0.4050 - val_mean_squared_error: 0.4050
```

Out[23]:
```
<tensorflow.python.keras.callbacks.History at 0x195110898e0>
```

In [24]:
```
op1.show_performance()
```



In [25]:
```
nice = op1.predict(oyea)
nice
```

Out[25]:

	LSTM
0	7.444017
1	7.485474
2	7.953871
3	7.617049
4	7.955699
5	8.169379
6	7.753589

In [26]:
```
op2.create_cnn()
```

In [27]:
```
op2.model_blueprint()
```

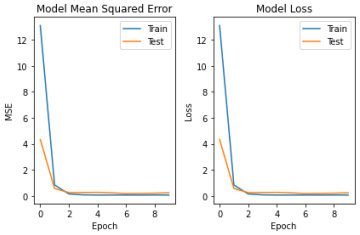
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 24, 64)            192
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 23, 32)            4128
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 11, 32)            0
_________________________________________________________________
flatten (Flatten)            (None, 352)               0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                17650
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 357
=================================================================
Total params: 22,327
Trainable params: 22,327
Non-trainable params: 0
_________________________________________________________________
```

In [28]:
```
op2.fit_model(10)
```

```
Epoch 1/10
20/20 [==============================] - 0s 8ms/step - loss: 13.1028 - mean_squared_error: 13.1028 - val_loss: 4.3357 - val_mean_square
d_error: 4.3357
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.8406 - mean_squared_error: 0.8406 - val_loss: 0.5708 - val_mean_squared_
error: 0.5708
Epoch 3/10
20/20 [==============================] - 0s 3ms/step - loss: 0.1509 - mean_squared_error: 0.1509 - val_loss: 0.2541 - val_mean_squared_
error: 0.2541
Epoch 4/10
20/20 [==============================] - 0s 3ms/step - loss: 0.0810 - mean_squared_error: 0.0810 - val_loss: 0.2608 - val_mean_squared_
error: 0.2608
Epoch 5/10
20/20 [==============================] - 0s 3ms/step - loss: 0.0656 - mean_squared_error: 0.0656 - val_loss: 0.2700 - val_mean_squared_
error: 0.2700
Epoch 6/10
20/20 [==============================] - 0s 3ms/step - loss: 0.0655 - mean_squared_error: 0.0655 - val_loss: 0.2363 - val_mean_squared_
error: 0.2363
Epoch 7/10
20/20 [==============================] - 0s 3ms/step - loss: 0.0663 - mean_squared_error: 0.0663 - val_loss: 0.1798 - val_mean_squared_
error: 0.1798
Epoch 8/10
20/20 [==============================] - 0s 3ms/step - loss: 0.0643 - mean_squared_error: 0.0643 - val_loss: 0.1912 - val_mean_squared_
error: 0.1912
Epoch 9/10
20/20 [==============================] - 0s 3ms/step - loss: 0.0697 - mean_squared_error: 0.0697 - val_loss: 0.2065 - val_mean_squared_
error: 0.2065
Epoch 10/10
20/20 [==============================] - 0s 3ms/step - loss: 0.0646 - mean_squared_error: 0.0646 - val_loss: 0.2393 - val_mean_squared_
error: 0.2393
```

Out[28]:
```
<tensorflow.python.keras.callbacks.History at 0x195187baaf0>
```

In [29]:
```
op2.show_performance()
```



In [30]:
```
nice = op2.predict(oyea)
nice
```

Out[30]:

	CNN
0	7.571640
1	7.636959
2	7.629258
3	7.693495
4	7.731886
5	7.737085
6	7.691160

In [31]:
```
op3.create_mlp()
```

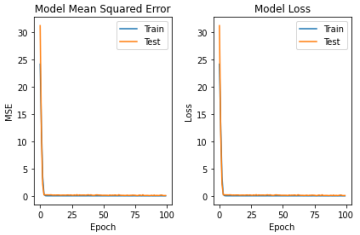

In [32]:
```
op3.model_blueprint()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 50)                1300
_________________________________________________________________
dense_5 (Dense)              (None, 25)                1275
_________________________________________________________________
dense_6 (Dense)              (None, 25)                650
_________________________________________________________________
dense_7 (Dense)              (None, 7)                 182
=================================================================
Total params: 3,407
Trainable params: 3,407
Non-trainable params: 0
_________________________________________________________________
```

In [33]:
```
op3.fit_model(100)
```

```
ared_error: 0.1549
Epoch 51/100
```

In [34]:
```
op3.show_performance()
```



In [35]:
```
oyea = prices[-26:-1]
#oyea = X[-1]
#oyea
```

In [36]:
```
nice = op3.predict(oyea)
nice
```

Out[36]:

	MLP
0	7.016368
1	7.353357
2	7.350390
3	7.333137
4	7.241974
5	7.493763
6	7.504691

In [37]:
```
%run ../tools/predictorsII.py
```

In [38]:
```
oo = UnivariatePredictorII(prices, 7)
```

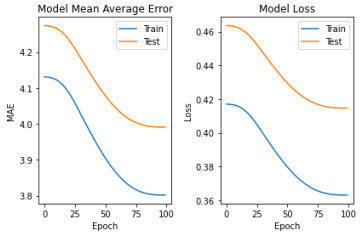
In [39]:
```
oo.fit_neural_model(100,"D")
```

```
INFO: nprophet.utils - set_auto_seasonalities: Disabling yearly seasonality. Run NeuralProphet with yearly_seasonality=True to override
this.
INFO: nprophet.utils - set_auto_seasonalities: Disabling daily seasonality. Run NeuralProphet with daily_seasonality=True to override t
his.
INFO: nprophet.config - set_auto_batch_epoch: Auto-set batch_size to 8

87%                                                  87/100 [00:01<00:00, 97.96it/s]

INFO: nprophet - _lr_range_test: learning rate range test found optimal lr: 3.51E-05
Epoch[100/100]: 100%|| 100/100 [00:04<00:00, 23.52it/s, SmoothL1Loss=0.363, MAE=3.8, RegLoss=0, MAE_val=3.99, SmoothL1
```

In [40]:
```
oo.show_performance_neural()
```



In [41]:
```
oo.predict_neural()
```

Out[41]:

	Neural Prophet
0	7.858598
1	11.582884
2	6.680636
3	14.030499
4	5.648793
5	-0.428853
6	2.138511

In [42]:
```
oo.fit_prophet_model()
```

```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

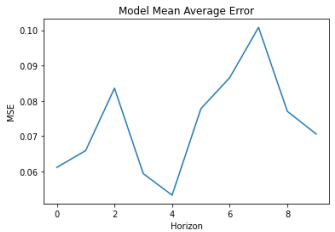
In [43]:
```
oo.show_performance_prophet()
```

```
INFO:fbprophet:Making 50 forecasts with cutoffs between 2009-08-15 12:00:00 and 2010-02-03 00:00:00

100%                                                  50/50 [01:10<00:00, 1.35s/it]
```

Out[43]:

	horizon	mse	rmse	mae	mape	mdape	coverage
0	1 days 00:00:00	0.061154	0.247293	0.202087	0.029285	0.030000	0.625000
1	1 days 12:00:00	0.065897	0.256705	0.198447	0.028946	0.022455	0.609375
2	2 days 00:00:00	0.083575	0.289093	0.234182	0.033957	0.034854	0.483696
3	2 days 12:00:00	0.059378	0.243676	0.180051	0.026026	0.018797	0.720000
4	3 days 12:00:00	0.053277	0.230818	0.185698	0.027001	0.024332	0.640000
5	4 days 12:00:00	0.077778	0.278887	0.225152	0.032715	0.027796	0.583333
6	5 days 00:00:00	0.086568	0.294225	0.229526	0.033036	0.026833	0.607639
7	5 days 12:00:00	0.100804	0.317496	0.257112	0.037308	0.036535	0.483696
8	6 days 00:00:00	0.077016	0.277518	0.209196	0.030107	0.023650	0.640000
9	7 days 00:00:00	0.070612	0.265729	0.215752	0.031163	0.027964	0.600000



In [44]:
```
oo.predict_prophet()
```

Out[44]:

	Prophet
0	7.309689
1	7.288514
2	7.412138
3	7.413944
4	7.306175
5	7.295620
6	7.317085

In [45]:
```
%run ../tools/predictorsIII.py
```

In [46]:
```
len(prices)
```

Out[46]: 279

In [47]:
```
op4 = HybridUnivariatePredictor(prices,2, 24, 7)
```

In [48]:
```
op4.create_cnnlstm()
```

In [49]:
```
op4.model_blueprint()
```

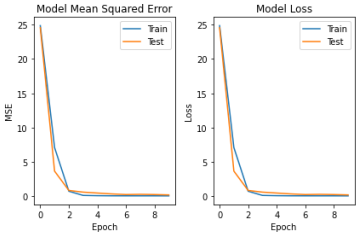
```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed (TimeDistri (None, None, 11, 64)      192
_________________________________________________________________
time_distributed_1 (TimeDist (None, None, 10, 32)      4128
_________________________________________________________________
time_distributed_2 (TimeDist (None, None, 5, 32)       0
_________________________________________________________________
time_distributed_3 (TimeDist (None, None, 160)         0
_________________________________________________________________
lstm_5 (LSTM)                (None, None, 50)          42200
_________________________________________________________________
lstm_6 (LSTM)                (None, 25)                7600
_________________________________________________________________
dense_8 (Dense)              (None, 7)                 182
=================================================================
Total params: 54,302
Trainable params: 54,302
Non-trainable params: 0
_________________________________________________________________
```

In [50]:
```
op4.fit_model(10)
```

```
Epoch 1/10
20/20 [==============================] - 0s 22ms/step - loss: 24.8605 - mean_squared_error: 24.8605 - val_loss: 24.5609 - val_mean_squa
red_error: 24.5609
Epoch 2/10
20/20 [==============================] - 0s 6ms/step - loss: 7.1074 - mean_squared_error: 7.1074 - val_loss: 3.6779 - val_mean_squared_
error: 3.6779
Epoch 3/10
20/20 [==============================] - 0s 6ms/step - loss: 0.7514 - mean_squared_error: 0.7514 - val_loss: 0.8648 - val_mean_squared_
error: 0.8648
Epoch 4/10
20/20 [==============================] - 0s 5ms/step - loss: 0.1449 - mean_squared_error: 0.1449 - val_loss: 0.6196 - val_mean_squared_
error: 0.6196
Epoch 5/10
20/20 [==============================] - 0s 6ms/step - loss: 0.1031 - mean_squared_error: 0.1031 - val_loss: 0.4893 - val_mean_squared_
error: 0.4893
Epoch 6/10
20/20 [==============================] - 0s 5ms/step - loss: 0.0849 - mean_squared_error: 0.0849 - val_loss: 0.3605 - val_mean_squared_
error: 0.3605
Epoch 7/10
20/20 [==============================] - 0s 5ms/step - loss: 0.0804 - mean_squared_error: 0.0804 - val_loss: 0.2694 - val_mean_squared_
error: 0.2694
Epoch 8/10
20/20 [==============================] - 0s 5ms/step - loss: 0.0791 - mean_squared_error: 0.0791 - val_loss: 0.3018 - val_mean_squared_
error: 0.3018
Epoch 9/10
20/20 [==============================] - 0s 5ms/step - loss: 0.0787 - mean_squared_error: 0.0787 - val_loss: 0.2691 - val_mean_squared_
error: 0.2691
Epoch 10/10
20/20 [==============================] - 0s 5ms/step - loss: 0.0747 - mean_squared_error: 0.0747 - val_loss: 0.2195 - val_mean_squared_
error: 0.2195
```

Out[50]: <tensorflow.python.keras.callbacks.History at 0x1951a2ad1c0>

In [51]:
```
op4.show_performance()
```



In [52]:
```
oyea = prices[-25:-1]
#oyea = X[-1]
#oyea
```

In [53]:
```
nice = op4.predict(oyea)
nice
#noice = pd.DataFrame(nice, columns=['yea'])
#noice = nice.reshape(20, 1)
#noice = pd.DataFrame(noice, columns=['yea'])
```

WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001952A786940> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args (https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args) and https://www.tensorflow.org/api_docs/python/tf/function (https://www.tensorflow.org/api_docs/python/tf/function) for more details.

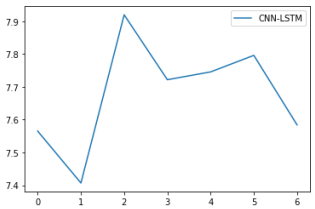
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001952A786940> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args (https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args) and https://www.tensorflow.org/api_docs/python/tf/function (https://www.tensorflow.org/api_docs/python/tf/function) for more details.

Out[53]:

	CNN-LSTM
0	7.565029
1	7.406779
2	7.919835
3	7.721859
4	7.745745
5	7.796144
6	7.584039

In [54]:
```
nice.plot()
```

Out[54]: <AxesSubplot:>



First whole system test - I am alive

In [81]:
```
training = DataLoader('aapl', '2010-01-01', '2010-05-01')
```

In [84]:
```
training = training.get_close()
```


In [91]:
```
predict = DataLoader('aapl', '2010-06-01', '2010-08-01')
```

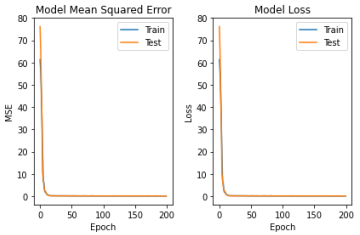
In [92]:
```
predict = predict.get_close()
```

In [101]:
```
predict_req = predict[0:24]
```

In [175]:
```
real = predict[24:34]
```

In [94]:
```
op4 = HybridUnivariatePredictor(training,2, 24, 10)
op4.create_cnnlstm()
op4.fit_model(200)
op4.show_performance()
```

4/4 [==============================] - 0s 7ms/step - loss: 0.0409 - mean_squared_error: 0.0409 - val_loss: 0.0611 - val_mean_squared_error: 0.0611



In [102]:
```
predicted = op4.predict(predict_req)
```

WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001952F071700> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args (https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args) and https://www.tensorflow.org/api_docs/python/tf/function (https://www.tensorflow.org/api_docs/python/tf/function) for more details.

WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001952F071700> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args (https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args) and https://www.tensorflow.org/api_docs/python/tf/function (https://www.tensorflow.org/api_docs/python/tf/function) for more details.

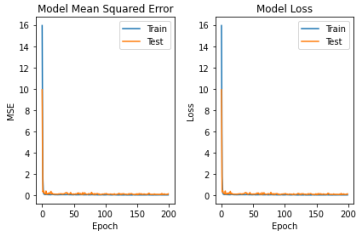
In [123]:
```
predicted
```

Out[123]:

	CNN-LSTM
0	9.672359
1	9.635760
2	9.947772
3	10.329806
4	10.137950
5	10.152438
6	9.962451
7	10.381300
8	10.322065
9	10.027566

In [106]:
```
op0 = BasicUnivariatePredictor(prices, 24, 10)
op0.create_bilstm()
op0.fit_model(200)
op0.show_performance()
```

```
20/20 [==============================] - 0s 14ms/step - loss: 0.0462 - mean_squared_error: 0.0462 - val_loss: 0.1532 - val_mean_sq
uared_error: 0.1532
```



In [107]:
```
predicted2 = op0.predict(predict_req)
```

```
WARNING:tensorflow:7 out of the last 7 calls to <function Model.make_predict_function.<locals>.predict_function at 0x00000195350D04C0>
triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function re
peatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define yo
ur @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that
can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_tens
or_args (https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args) and https://www.tensorflow.org/api_docs/
python/tf/function (https://www.tensorflow.org/api_docs/python/tf/function) for  more details.

WARNING:tensorflow:7 out of the last 7 calls to <function Model.make_predict_function.<locals>.predict_function at 0x00000195350D04C0>
triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function re
peatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define yo
ur @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that
can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_tens
or_args (https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args) and https://www.tensorflow.org/api_docs/
python/tf/function (https://www.tensorflow.org/api_docs/python/tf/function) for  more details.
```

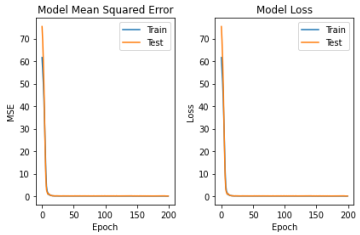
In [108]:
```
predicted2
```

Out[108]:

	Bidirectional LSTM
0	8.873008
1	9.503883
2	9.149117
3	8.483627
4	8.912893
5	9.394436
6	8.264942
7	8.614160
8	9.226923
9	8.695553

In [109]:
```
op1 = BasicUnivariatePredictor(training, 24, 10)
op1.create_cnn()
op1.fit_model(200)
op1.show_performance()
```

```
4/4 [==============================] - 0s 6ms/step - loss: 0.0804 - mean_squared_error: 0.0804 - val_loss: 0.0847 - val_mean_squar
ed_error: 0.0847
```



In [110]:
```
predicted3 = op1.predict(predict_req)
```

```
WARNING:tensorflow:8 out of the last 8 calls to <function Model.make_predict_function.<locals>.predict_function at 0x0000019536597940>
triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function re
peatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define yo
ur @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that
can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_tens
or_args (https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args) and https://www.tensorflow.org/api_docs/
python/tf/function (https://www.tensorflow.org/api_docs/python/tf/function) for  more details.

WARNING:tensorflow:8 out of the last 8 calls to <function Model.make_predict_function.<locals>.predict_function at 0x0000019536597940>
triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function re
peatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define yo
ur @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that
can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_tens
or_args (https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args) and https://www.tensorflow.org/api_docs/
python/tf/function (https://www.tensorflow.org/api_docs/python/tf/function) for  more details.
```

In [111]:
```
predicted3
```

Out[111]:

	CNN
0	9.690488
1	9.948913
2	9.902168
3	9.816916
4	9.872331
5	10.000324
6	9.941864
7	10.087987
8	10.266931
9	10.111290

In [147]:
```
real
```

Out[147]:

Date	Close
2010-07-06	8.879643
2010-07-07	9.238214
2010-07-08	9.217500
2010-07-09	9.272143
2010-07-12	9.188929
2010-07-13	8.992857
2010-07-14	9.026071
2010-07-15	8.980357
2010-07-16	8.925000
2010-07-19	8.770714

In [133]:
```
final_df = pd.concat([predicted, predicted2, predicted3], axis=1)
```

In [143]:
```
final_df
```

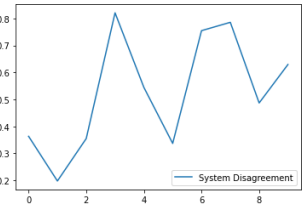
Out[143]:

	CNN-LSTM	Bidirectional LSTM	CNN
0	9.672359	8.873008	9.690488
1	9.635760	9.503883	9.948913
2	9.947772	9.149117	9.902168
3	10.329806	8.483627	9.816916
4	10.137950	8.912893	9.872331
5	10.152438	9.394436	10.000324
6	9.962451	8.264942	9.941864
7	10.381300	8.614160	10.087987
8	10.322065	9.226923	10.266931
9	10.027566	8.695553	10.111290

System Disagreement

In [149]:
```
disagreement(final_df).plot()
```

Out[149]: <AxesSubplot:>



In [150]:
```
predictor_score(final_df)
```

Out[150]:

	0	1	2
0	0.272493	0.538944	0.278536
1	0.148343	0.192302	0.252727
2	0.281420	0.517236	0.266219
3	0.786356	1.059823	0.615393
4	0.496892	0.728165	0.408352
5	0.303372	0.454630	0.252667
6	0.572699	1.124810	0.565836
7	0.686818	1.080323	0.589047
8	0.383426	0.711717	0.365047
9	0.471912	0.915917	0.499820

System consensus

In [160]:
```
average = average_consolidation(final_df)
```

In [161]:
```
nomemory = consolidated_predictions(final_df,real)
```

In [162]:
```
memory = consolidated_predictions_memory(final_df,real)
```

In [170]:
```
anchor = consolidated_predictions_anchor(final_df,real,1.5)
```

In [164]:
```
consensus = pd.DataFrame()
```

In [165]:
```
consensus['Average'] = average
```

In [166]:
```
consensus['NoMemory'] = nomemory
```

In [167]:
```
consensus['Memory'] = memory
```

In [171]:
```
consensus['Anchor'] = anchor
```


In [172]:
```
consensus
```

Out[172]:

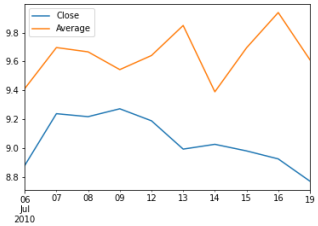
	Average	NoMemory	Memory	Anchor
0	9.411952	9.411952	9.411952	9.441618
1	9.696185	9.647823	9.672004	9.601387
2	9.666352	9.614654	9.606592	9.551840
3	9.543450	9.311197	9.399820	9.494098
4	9.641058	9.624879	9.557972	9.578223
5	9.849066	9.779433	9.794235	9.742893
6	9.389752	9.240135	9.253945	9.219288
7	9.694482	9.659808	9.574268	9.656881
8	9.938640	9.827300	9.856686	9.775970
9	9.611470	9.451074	9.501725	9.419970

In [178]:
```
performance = real.copy()
```

In [179]:
```
performance['Average'] = average
```

In [187]:
```
performance.plot()
```

Out[187]: <AxesSubplot:xlabel='Date'>

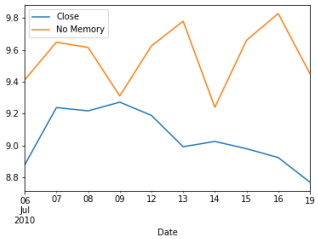


In [188]:
```
performance2 = real.copy()
```

In [190]:
```
performance2['No Memory'] = nomemory
```

In [191]:
```
performance2.plot()
```

Out[191]: <AxesSubplot:xlabel='Date'>

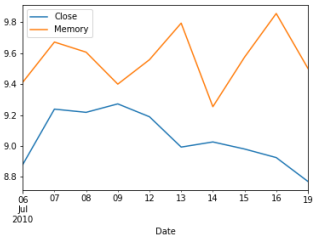


In [196]:
```
performance3 = real.copy()
```

In [198]:
```
performance3['Memory'] = memory
```

In [199]:
```
performance3.plot()
```

Out[199]: <AxesSubplot:xlabel='Date'>



In [204]:
```
performance4 = real.copy()
```

In [205]:
```
performance4['Anchor'] = anchor
```

In [206]:
```
performance4.plot()
```

Out[206]: <AxesSubplot:xlabel='Date'>