DATA

The data was collected from the finance library “yfinance” in python.
We are studying data from 1981 to 2020 and will forecast for 2021.

```
# using Yahoo finance API to get stock data.
stocksdata="AAPL"
df=yf.download(stocksdata,start="1981-1-1",end="2020-12-31")
```

```
df
```

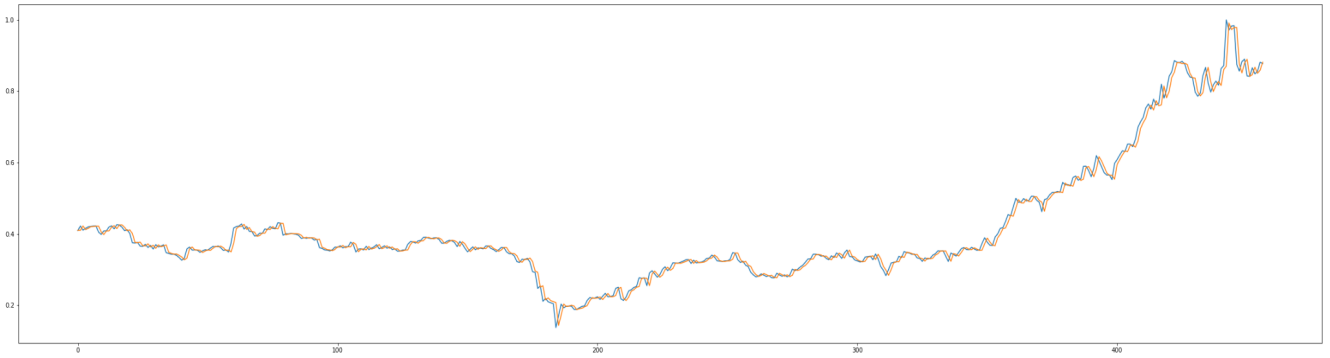
	Date	Open	High	Low	Close	Adj Close	Volume
0	1981-01-02	0.154018	0.155134	0.154018	0.154018	0.119849	21660800
1	1981-01-05	0.151228	0.151228	0.150670	0.150670	0.117244	35728000
2	1981-01-06	0.144531	0.144531	0.143973	0.143973	0.112032	45158400
3	1981-01-07	0.138393	0.138393	0.137835	0.137835	0.107256	55686400
4	1981-01-08	0.135603	0.135603	0.135045	0.135045	0.105085	39827200
...	...	...	...	...	...	...	...
10081	2020-12-23	132.160004	132.429993	130.779999	130.960007	129.406555	88223700
10082	2020-12-24	131.320007	133.460007	131.100006	131.970001	130.404572	54930100
10083	2020-12-28	133.990005	137.339996	133.509995	136.690002	135.068588	124486200
10084	2020-12-29	138.050003	138.789993	134.339996	134.869995	133.270187	121047300
10085	2020-12-30	135.580002	135.990005	133.399994	133.720001	132.133820	96452100

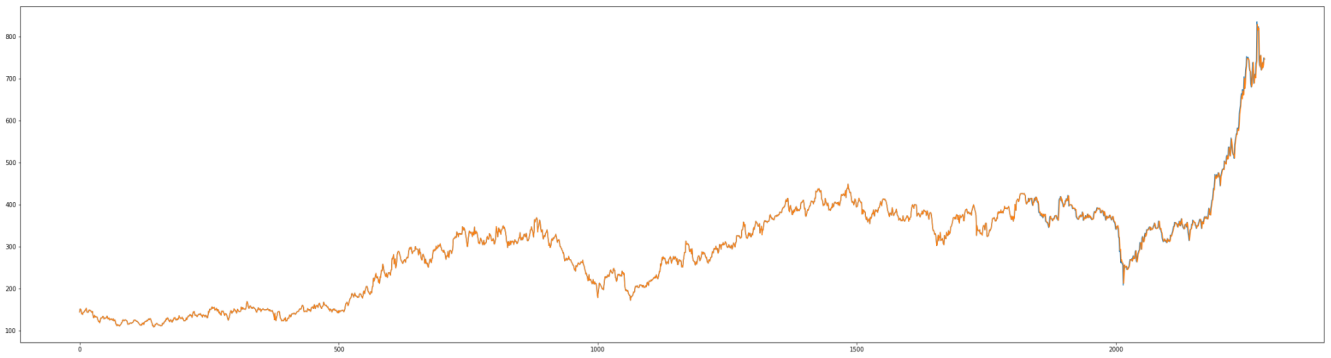
10086 rows × 7 columns

Predictions using LR:

[Lr of sf adaniport.ipynb - Colaboratory (google.com)]

R2-score: `0.9913189150766206`





Predictions Using LSTM:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 50)           10400

 lstm_1 (LSTM)               (None, 100, 50)           20200

 lstm_2 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

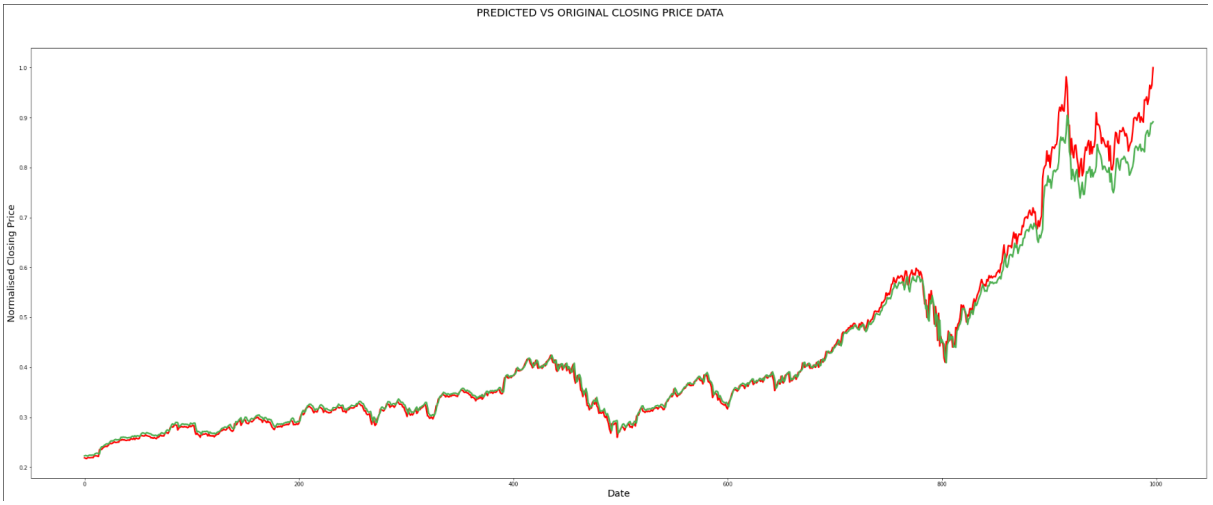
LSTM ARCHITECTURE:

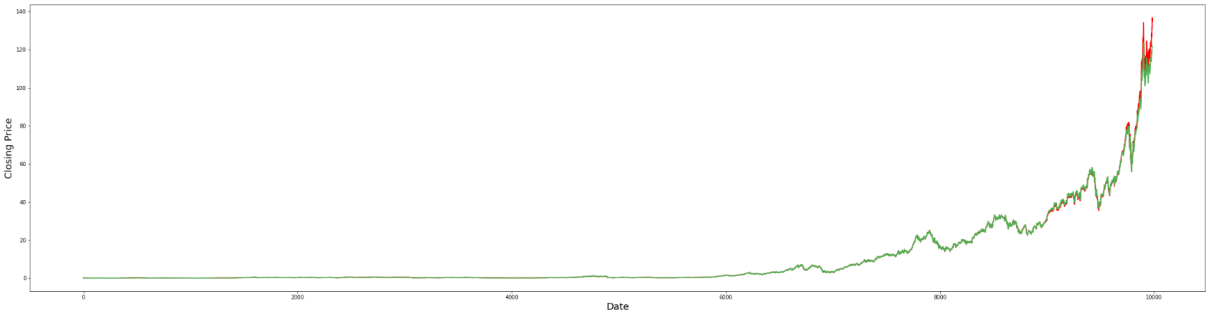
```
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(X_train.shape[1],1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1,activation='linear'))
model.compile(loss='mean_squared_error',optimizer='adam')
```

APPLE

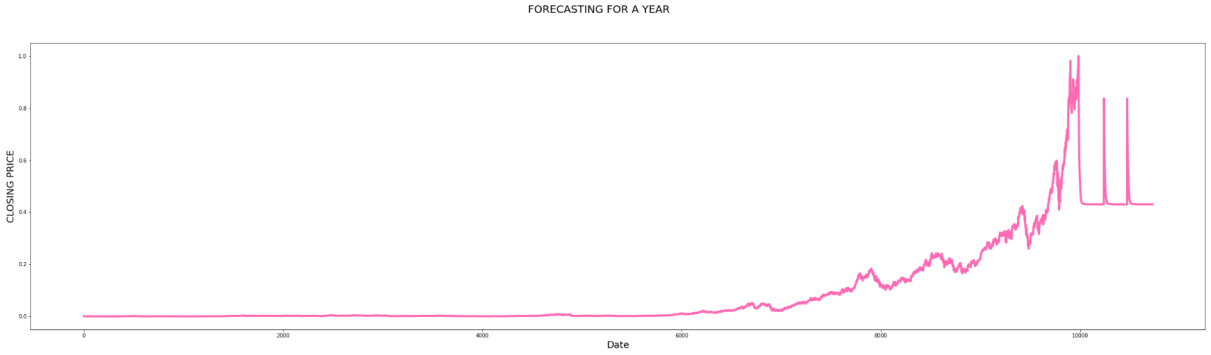
```
R2 score :  0.9820609540002779
MSE :  0.06042422772480179
```

Stock forecasting apple.ipynb - Colaboratory (google.com)





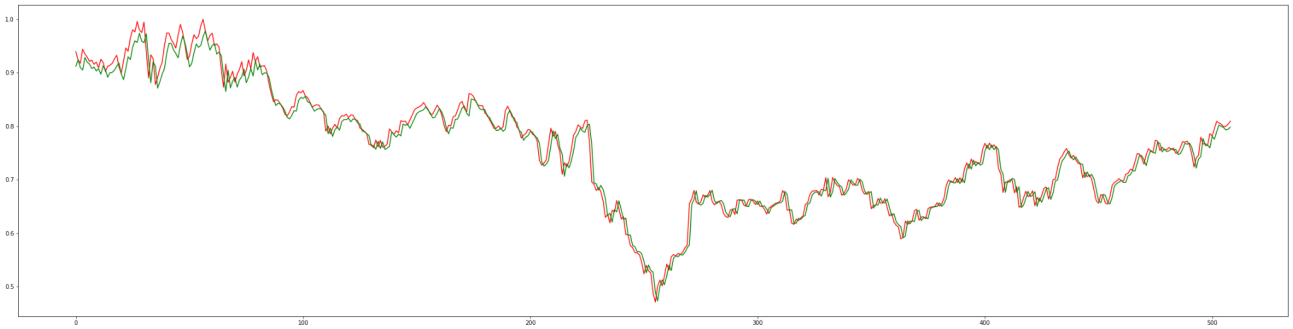
Forecasting USING LSTM:



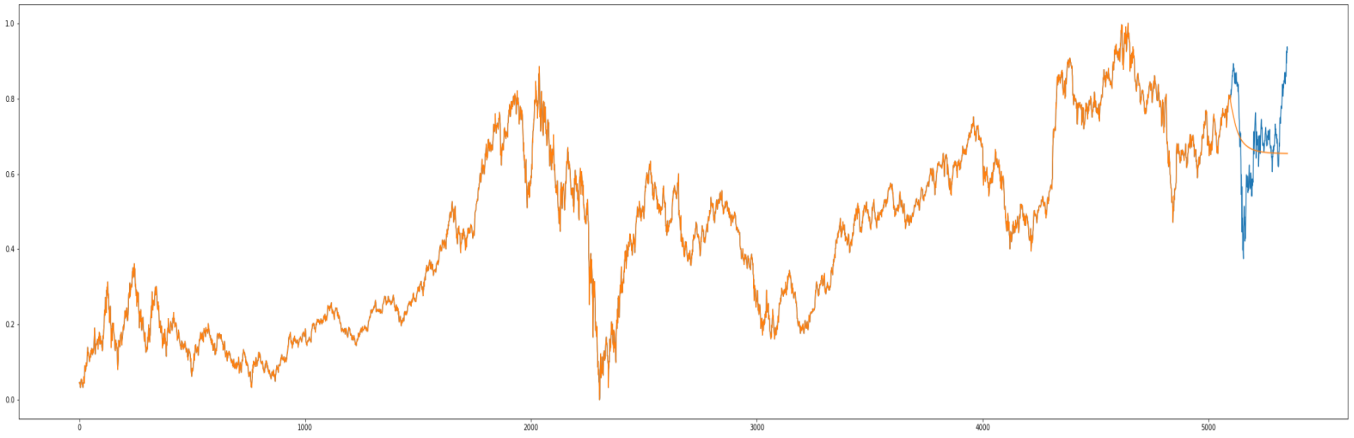
GOLDMAN SACHS

R2-Score - `0.9762633409794264`

[Stock forcasting GS.ipynb - Colaboratory (google.com)]



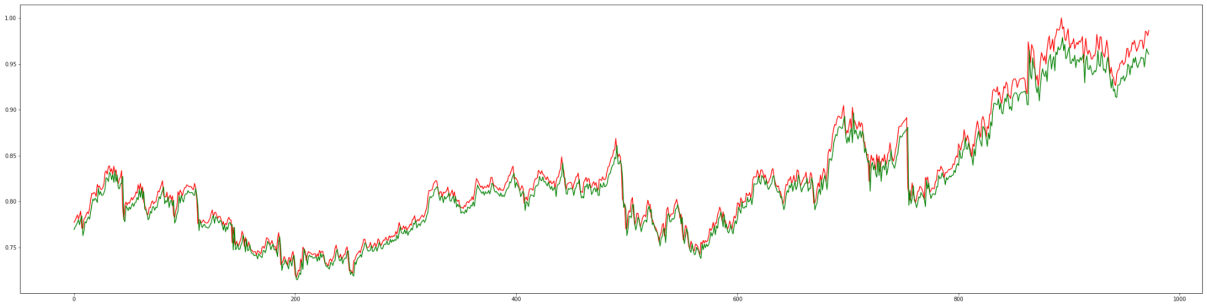
Forecasting USING LSTM:

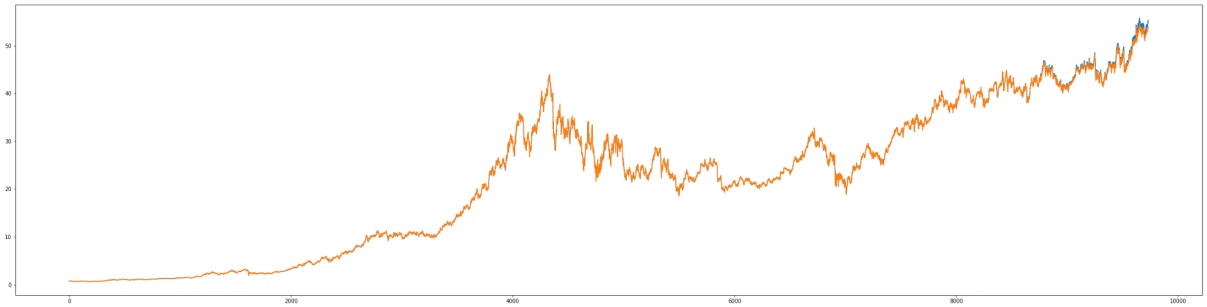


COCA-COLA

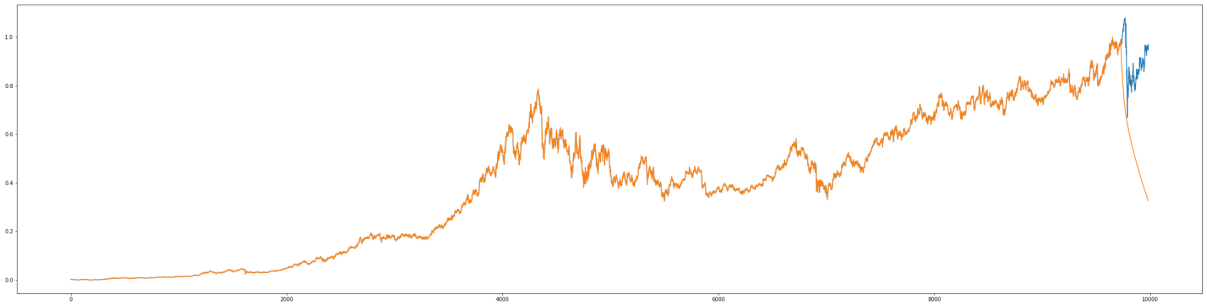
R2-Score-0.9630954002841426

Stock forecasting coca-cola.ipynb - Colaboratory (google.com)





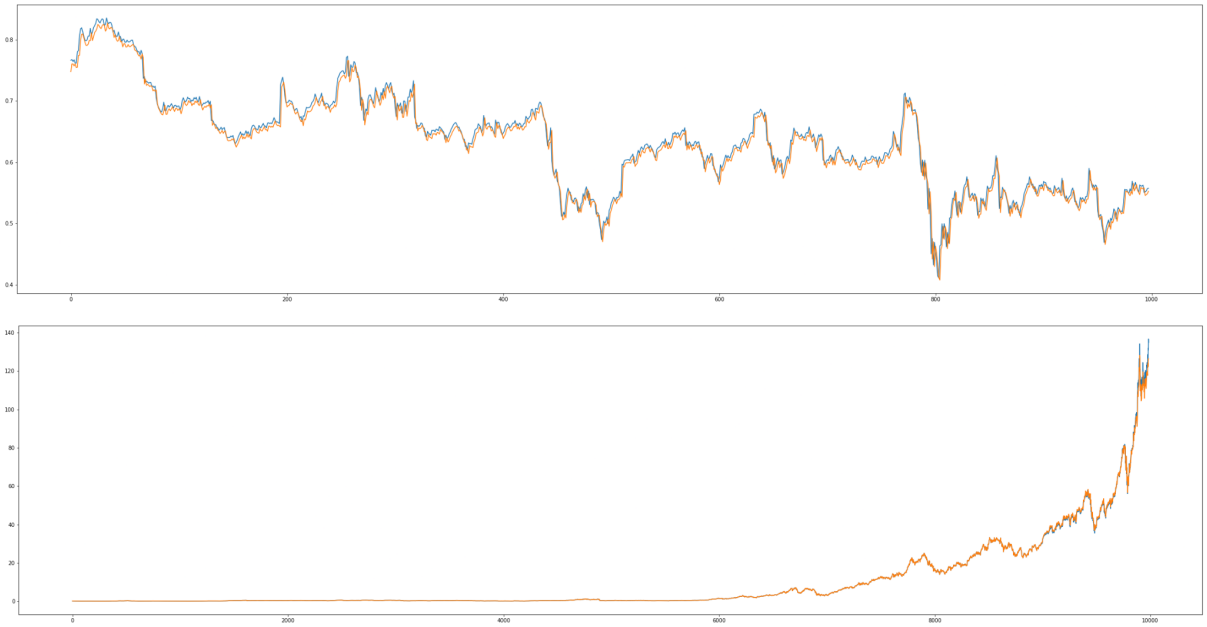
Forecasting USING LSTM:



IBM

R2-Score-0.978463002841426

Stock forecasting IBM.ipynb - Colaboratory (google.com)



USING TRANSFORMERS

APPLE_STOCKS(moving_average).ipynb

Time Embeddings:

Transformers combined with natural language data tend to utilize positional encoding to provide the model with word order. In detail, the positional encoding represents a word's value and position in a sentence, allowing the Transformer to obtain knowledge about a sentence structure and word inter-dependencies, so for the information of time to go through the network, we will use TIME2VEC.

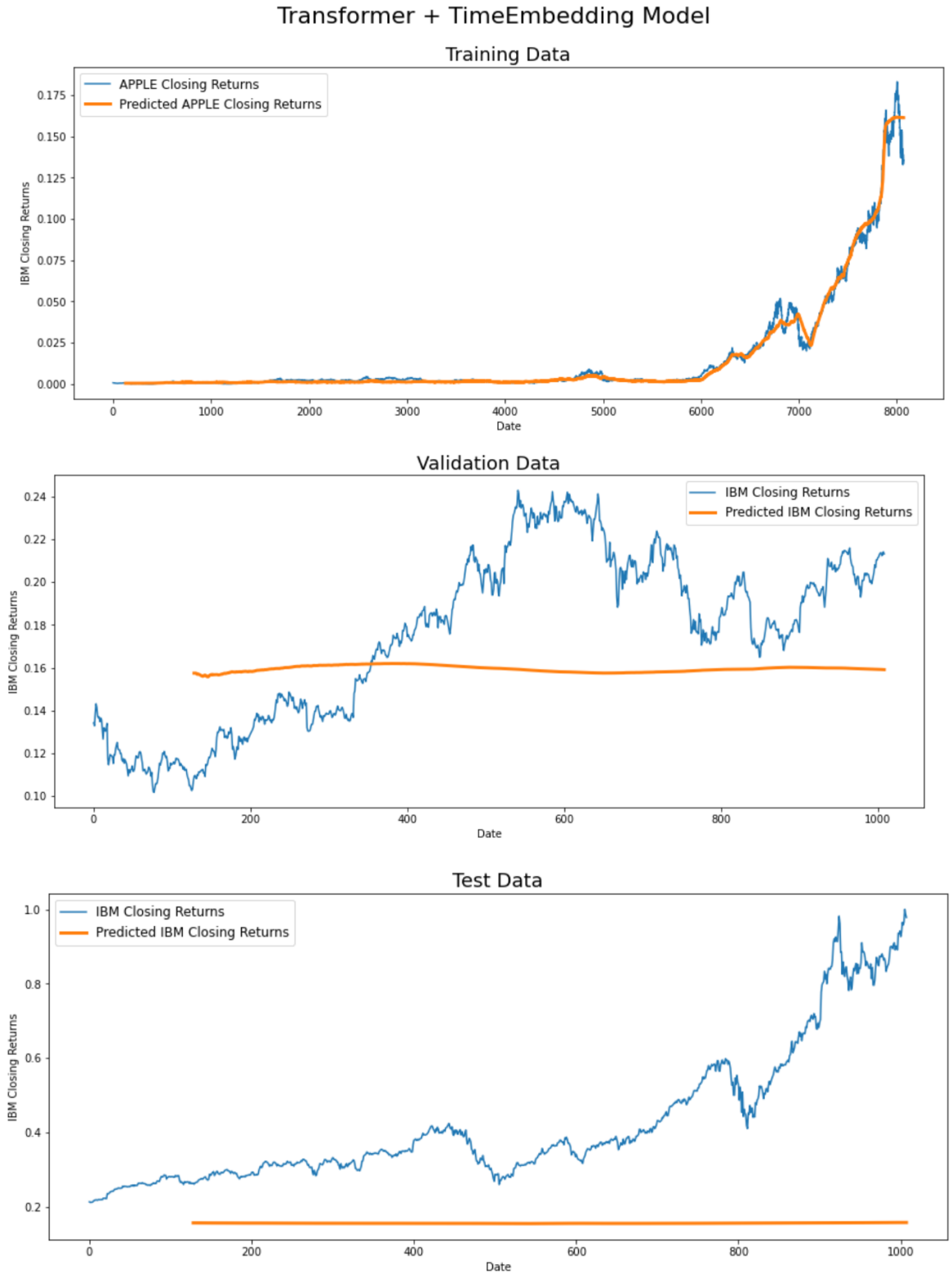
Time to vector:

```
class Time2Vector(Layer):
  def __init__(self, seq_len, **kwargs):
    super(Time2Vector, self).__init__()
    self.seq_len = seq_len

  def build(self, input_shape):
    '''Initialize weights and biases with shape (batch, seq_len)'''
    self.weights_linear = self.add_weight(name='weight_linear',
                                shape=(int(self.seq_len),),
                                initializer='uniform',
                                trainable=True)

    self.bias_linear = self.add_weight(name='bias_linear',
                                shape=(int(self.seq_len),),
                                initializer='uniform',
                                trainable=True)

    self.weights_periodic = self.add_weight(name='weight_periodic',
                                shape=(int(self.seq_len),),
                                initializer='uniform',
                                trainable=True)

    self.bias_periodic = self.add_weight(name='bias_periodic',
                                shape=(int(self.seq_len),),
                                initializer='uniform',
                                trainable=True)

  def call(self, x):
    '''Calculate linear and periodic time features'''
    x = tf.math.reduce_mean(x[:,:,:4], axis=-1)
    time_linear = self.weights_linear * x + self.bias_linear # Linear time feature
    time_linear = tf.expand_dims(time_linear, axis=-1) # Add dimension (batch, seq_len, 1)

    time_periodic = tf.math.sin(tf.multiply(x, self.weights_periodic) + self.bias_periodic)
    time_periodic = tf.expand_dims(time_periodic, axis=-1) # Add dimension (batch, seq_len, 1)
    return tf.concat([time_linear, time_periodic], axis=-1) # shape = (batch, seq_len, 2)

  def get_config(self): # Needed for saving and loading model with custom layer
    config = super().get_config().copy()
    config.update({'seq_len': self.seq_len})
    return config
```

Transformer Architecture:

```
Model: "model_5"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_6 (InputLayer)           [(None, 128, 5)]     0           []

 time2_vector_5 (Time2Vector)   (None, 128, 2)       512         ['input_6[0][0]']

 concatenate_5 (Concatenate)    (None, 128, 7)       0           ['input_6[0][0]',
                                                                  'time2_vector_5[0][0]']

 transformer_encoder_15 (Transf  (None, 128, 7)      99114       ['concatenate_5[0][0]',
 ormerEncoder)                                                    'concatenate_5[0][0]',
                                                                  'concatenate_5[0][0]']

 transformer_encoder_16 (Transf  (None, 128, 7)      99114       ['transformer_encoder_15[0][0]',
 ormerEncoder)                                                    'transformer_encoder_15[0][0]',
                                                                  'transformer_encoder_15[0][0]']

 transformer_encoder_17 (Transf  (None, 128, 7)      99114       ['transformer_encoder_16[0][0]',
 ormerEncoder)                                                    'transformer_encoder_16[0][0]',
                                                                  'transformer_encoder_16[0][0]']

 global_average_pooling1d_5 (Gl  (None, 128)         0           ['transformer_encoder_17[0][0]']
 obalAveragePooling1D)

 dropout_10 (Dropout)           (None, 128)          0           ['global_average_pooling1d_5[0][0
                                                                 ]']

 dense_10 (Dense)               (None, 64)           8256        ['dropout_10[0][0]']

 dropout_11 (Dropout)           (None, 64)           0           ['dense_10[0][0]']

 dense_11 (Dense)               (None, 1)            65          ['dropout_11[0][0]']

==================================================================================================
Total params: 306,175
Trainable params: 306,175
Non-trainable params: 0
```

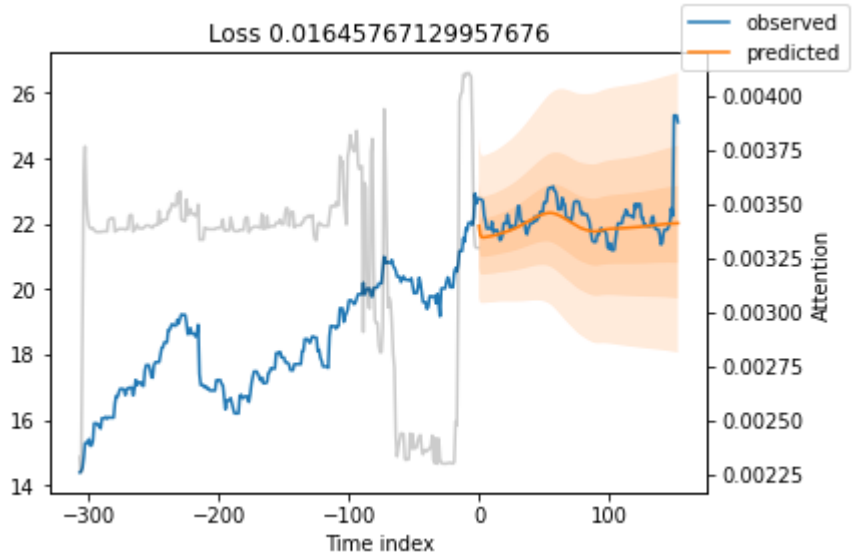
RESULT OF TRANSFORMER MODEL:

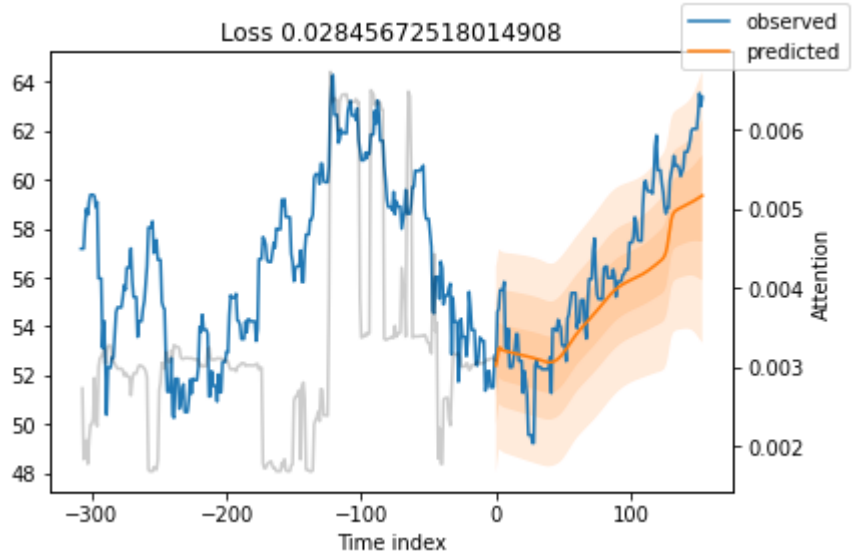


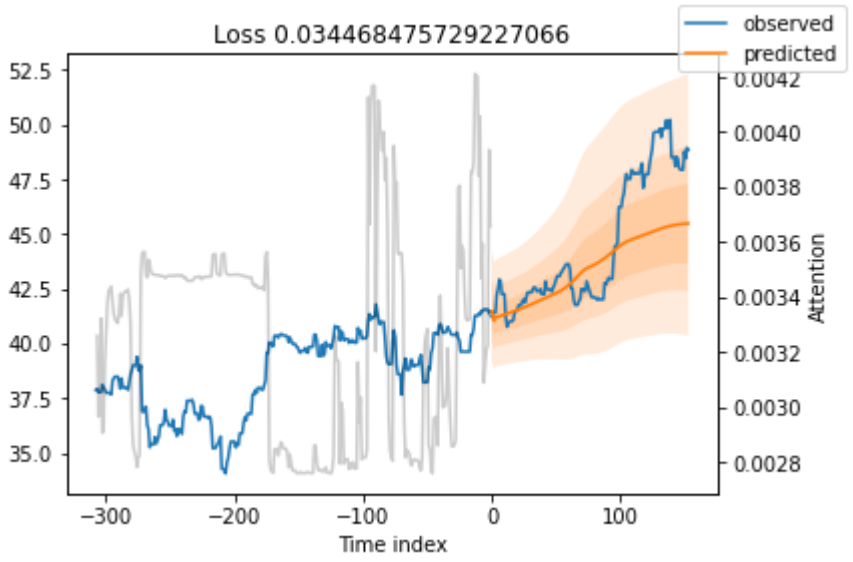
Ig, we can conclude that this transformer model is ***shit***.

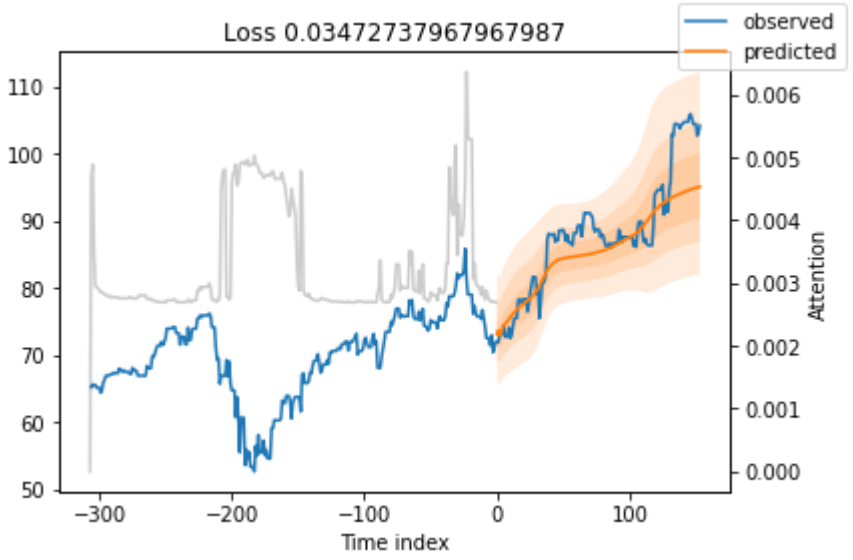
TEMPORAL FUSION TRANSFORMER

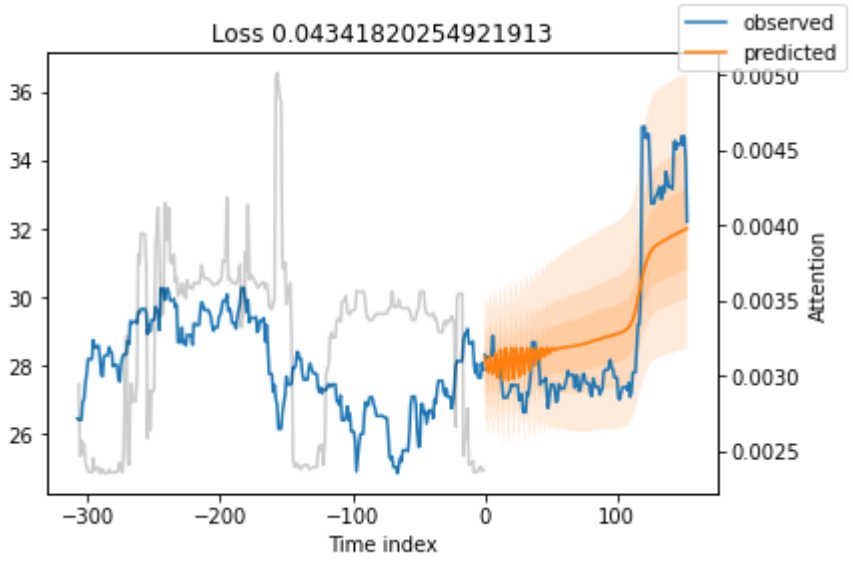
We trained the model to forecast the values of stock closing price for 154 days based on the given context data of 308 days.

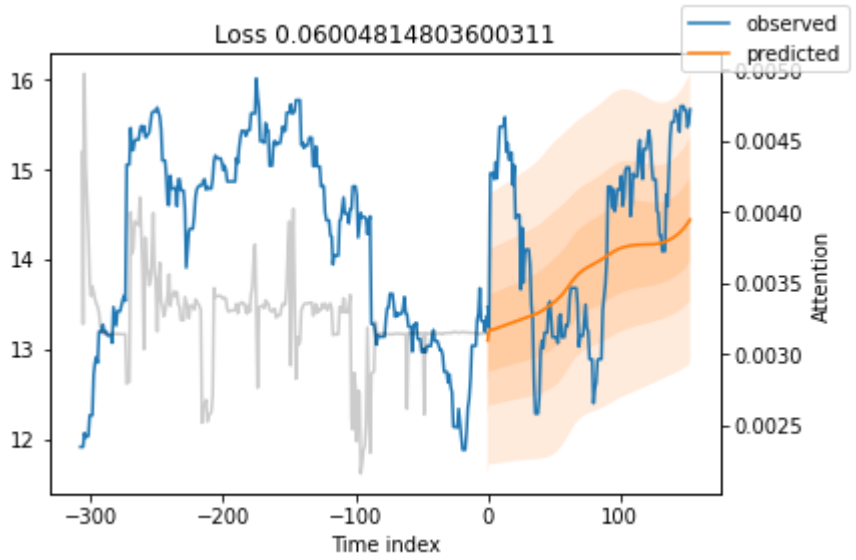


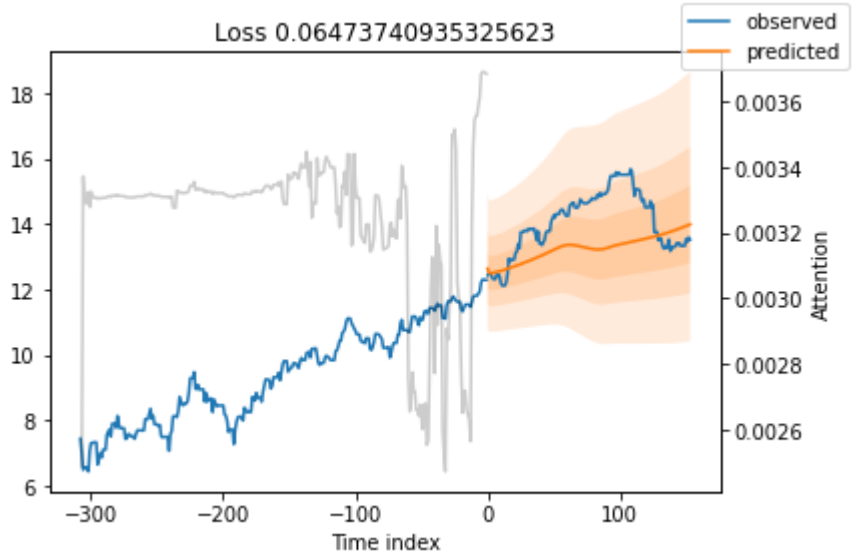


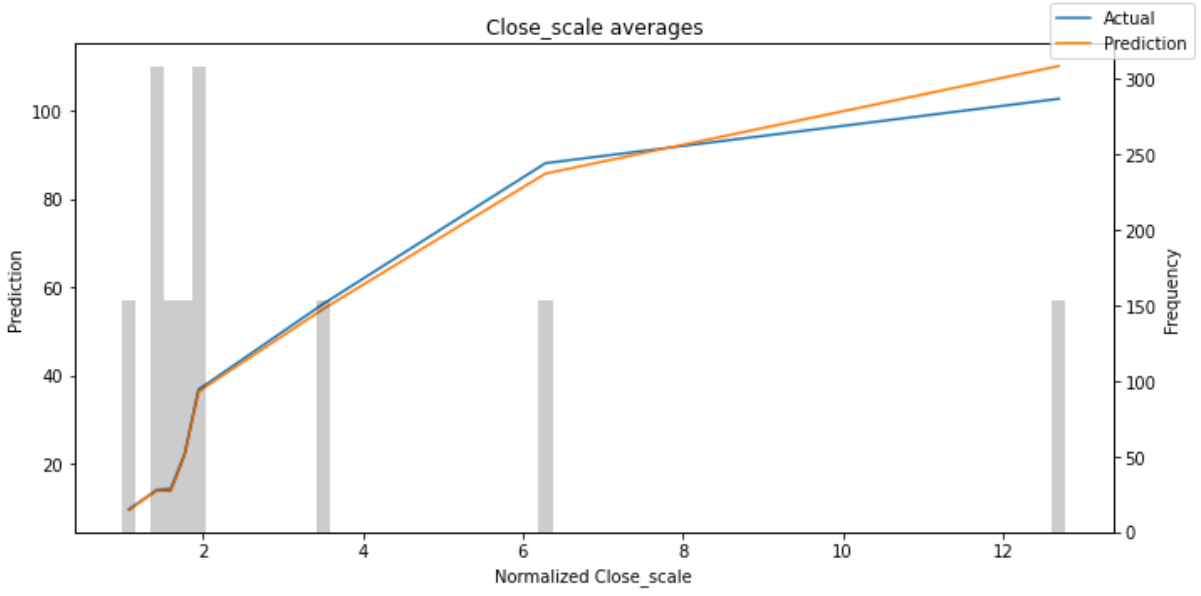

Loss 0.034468475729227066
observed
predicted
52.5
50.0
47.5
45.0
42.5
40.0
37.5
35.0
0.0042
0.0040
0.0038
0.0036
0.0034
0.0032
0.0030
0.0028
Attention
−300
−200
−100
0
100
Time index


Loss 0.03472737967967987
observed
predicted
110
100
90
80
70
60
50
0.006
0.005
0.004
0.003
0.002
0.001
0.000
Attention
−300
−200
−100
0
100
Time index


Loss 0.04341820254921913
observed
predicted
36
34
32
30
28
26
0.0050
0.0045
0.0040
0.0035
0.0030
0.0025
Attention
−300
−200
−100
0
100
Time index


Loss 0.06004814803600311
observed
predicted
Attention
Time index

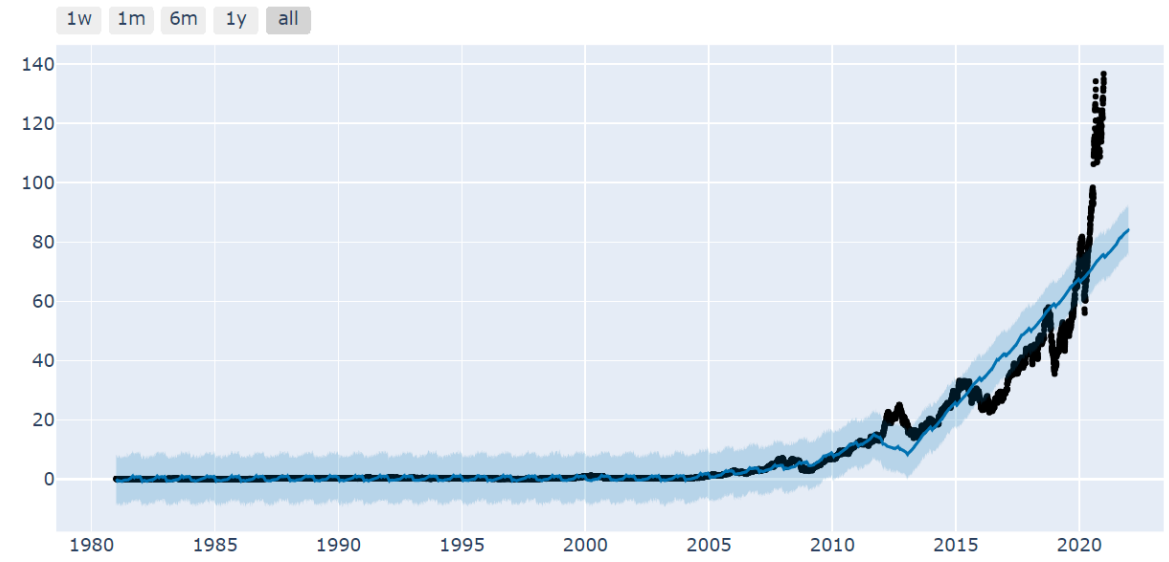

Loss 0.06473740935325623
observed
predicted
Attention
Time index


Close_scale averages
Actual
Prediction
Prediction
Frequency
Normalized Close_scale

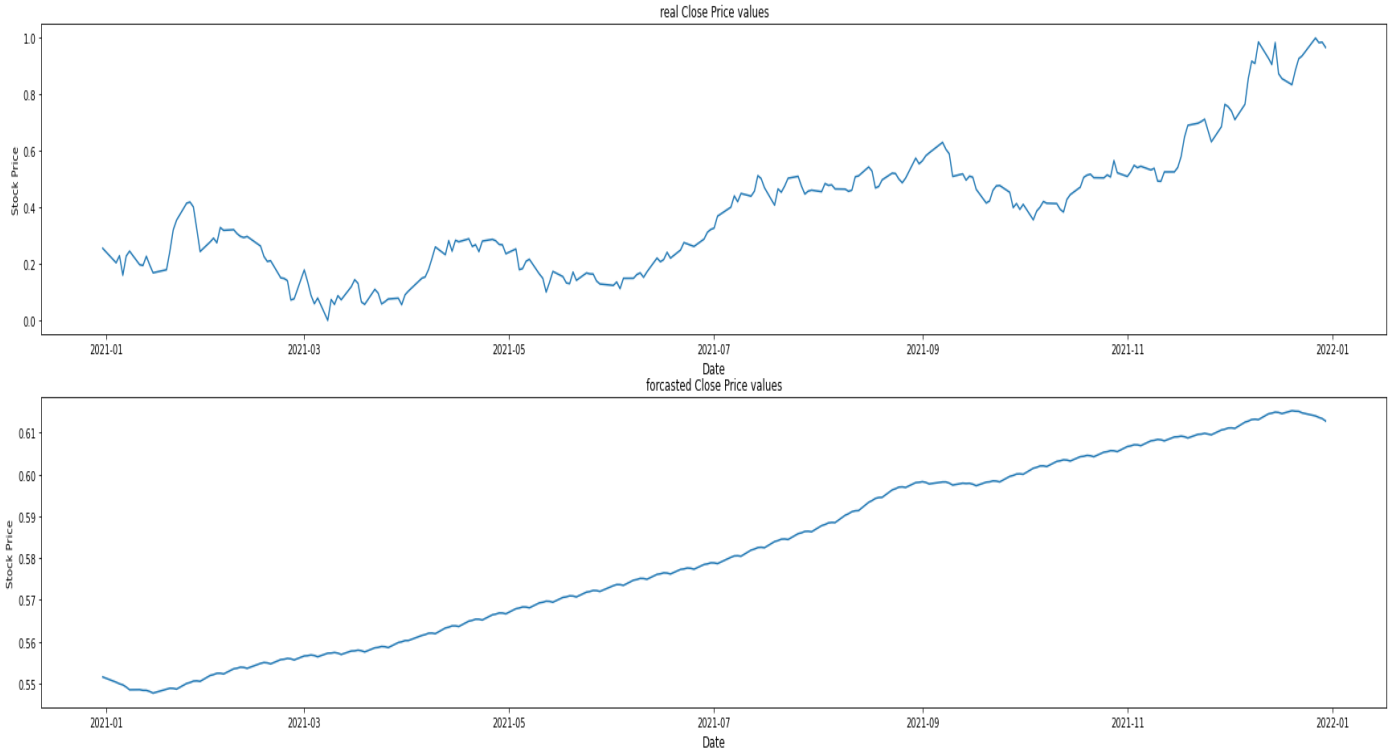
PROPHET - Facebook's Time Series Forecasting model

[forcasting using prophet.ipynb - Colaboratory (google.com)]

We trained Prophet on apples stocks from 1998 to 2020.



We got following results in forecasting for the year 2021.



We used the Base model as it was not showing much improvement on hyperparameter tuning.

```
[ ]  mape(pred1['yhat'],df_real['Close'])

     0.44282468919904

     min(tuning_results['mape'])

     0.43072457760166066
```

Forecasting for One Month.