Stock Market Prediction And Forecasting Using Hybrid QML Model(Quantum-Machine Learning!)

In [1]:
```
### Keras and Tensorflow >2.0
```

In [2]:
```
### Data Collection
import pandas_datareader as pdr
import yfinance as yf

df = yf.Ticker("AAPL")

# get historical market data
df = df.history(period="max")
```

In [3]:
```
df
```

Out[3]:

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
1980-12-12	0.100453	0.100890	0.100453	0.100453	469033600	0.0	0.0
1980-12-15	0.095649	0.095649	0.095213	0.095213	175884800	0.0	0.0
1980-12-16	0.088661	0.088661	0.088224	0.088224	105728000	0.0	0.0
1980-12-17	0.090408	0.090845	0.090408	0.090408	86441600	0.0	0.0
1980-12-18	0.093029	0.093466	0.093029	0.093029	73449600	0.0	0.0
...	...	...	...	...	...	...	...
2021-11-15	150.369995	151.880005	149.429993	150.000000	59222800	0.0	0.0
2021-11-16	149.940002	151.490005	149.339996	151.000000	59256200	0.0	0.0
2021-11-17	151.000000	155.000000	150.990005	153.490005	88807000	0.0	0.0
2021-11-18	153.710007	158.669998	153.050003	157.869995	137827700	0.0	0.0
2021-11-19	157.649994	161.020004	156.529999	160.550003	117147500	0.0	0.0

10324 rows × 7 columns

In [4]:
```
df.to_csv('AAPL.csv')
```

In [5]:
```
import pandas as pd
```

In [6]:
```
df=pd.read_csv('AAPL.csv')
```

In [7]:
```
df.head()
```

Out[7]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	1980-12-12	0.100453	0.100890	0.100453	0.100453	469033600	0.0	0.0
1	1980-12-15	0.095649	0.095649	0.095213	0.095213	175884800	0.0	0.0
2	1980-12-16	0.088661	0.088661	0.088224	0.088224	105728000	0.0	0.0
3	1980-12-17	0.090408	0.090845	0.090408	0.090408	86441600	0.0	0.0
4	1980-12-18	0.093029	0.093466	0.093029	0.093029	73449600	0.0	0.0

In [8]:
```
df.tail()
```

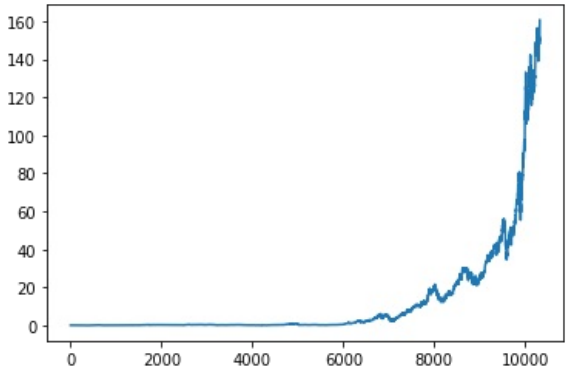
Out[8]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
10319	2021-11-15	150.369995	151.880005	149.429993	150.000000	59222800	0.0	0.0
10320	2021-11-16	149.940002	151.490005	149.339996	151.000000	59256200	0.0	0.0
10321	2021-11-17	151.000000	155.000000	150.990005	153.490005	88807000	0.0	0.0
10322	2021-11-18	153.710007	158.669998	153.050003	157.869995	137827700	0.0	0.0
10323	2021-11-19	157.649994	161.020004	156.529999	160.550003	117147500	0.0	0.0

In [9]:
```
df1=df.reset_index()['Close']
```

In [10]:
```
import matplotlib.pyplot as plt
plt.plot(df1)
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x7ffac802f1f0>]
```



In [11]:
```
### LSTM are sensitive to the scale of the data. so we apply MinMax scaler
```

In [12]:
```
import numpy as np
```

In [13]:
```
df1
```

Out[13]:
```
0          0.100453
1          0.095213
2          0.088224
3          0.090408
4          0.093029
            ...
10319    150.000000
10320    151.000000
10321    153.490005
10322    157.869995
10323    160.550003
Name: Close, Length: 10324, dtype: float64
```

In [14]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))
```

In [15]:
```
print(df1)
```

```
[[3.86384327e-04]
 [3.53734058e-04]
 [3.10195539e-04]
 ...
 [9.56015647e-01]
 [9.83303335e-01]
 [1.00000000e+00]]
```

In [16]:
```
##splitting dataset into train and test split
training_size=int(len(df1)*0.65)
test_size=len(df1)-training_size
train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]
```

In [17]:
```
training_size,test_size
```

Out[17]:
```
(6710, 3614)
```

In [18]:
```
train_data
```

Out[18]:
```
array([[0.00038638],
       [0.00035373],
       [0.0003102 ],
       ...,
       [0.02603478],
       [0.02610536],
       [0.02625989]])
```

In [19]:
```
import numpy

# This is like feature matrix.
def create_dataset(dataset, time_step=1):
```

```
        dataX, dataY = [], []
        for i in range(len(dataset)-time_step-1):
                a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
                dataX.append(a)
                dataY.append(dataset[i + time_step, 0])
        return numpy.array(dataX), numpy.array(dataY)
```

In [20]:
```
# reshape into X=t,t+1,t+2,t+3 and Y=t+4
time_step = 100 # this is like X = [t,t+1,t+2,t+3,t+4,t+5] and Y=[t+6]
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)
```

In [21]:
```
print(X_train.shape), print(y_train.shape)
```

Out[21]:
```
(6609, 100)
(6609,)
(None, None)
```

In [22]:
```
X_test.shape,ytest.shape
```

Out[22]:
```
((3513, 100), (3513,))
```

In [23]:
```
# OK so this reshape is required, because we are sending it to LSTM! This is just reshape for now! :D


# reshape input to be [samples, time steps, features] which is required for LSTM
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)
X_train.shape,X_test.shape

# So here we have 6609 recorda(Rows) and 100 Columns(aka feature-Vectors) What is 1?
```

Out[23]:
```
((6609, 100, 1), (3513, 100, 1))
```

In [24]:
```
### Create the Stacked LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

In [25]:
```
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(4))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

```
2021-11-21 16:09:21.757088: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-11-21 16:09:21.762396: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-11-21 16:09:21.762950: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-11-21 16:09:21.763723: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimi
zed with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critica
l operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-11-21 16:09:21.765135: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-11-21 16:09:21.765572: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-11-21 16:09:21.765900: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-11-21 16:09:22.150949: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-11-21 16:09:22.151242: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-11-21 16:09:22.151493: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-11-21 16:09:22.151729: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device /job:localhos
t/replica:0/task:0/device:GPU:0 with 3493 MB memory:  -> device: 0, name: NVIDIA GeForce GTX 1660 Ti with Max-Q D
esign, pci bus id: 0000:01:00.0, compute capability: 7.5
```

In [26]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 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, 4)                 204

dense_1 (Dense)              (None, 1)                 5

=================================================================
Total params: 51,009
Trainable params: 51,009
Non-trainable params: 0
_________________________________________________________________
```

In [27]:
```
model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=10,batch_size=64,verbose=1)
```

```
Epoch 1/10
2021-11-21 16:09:26.019054: I tensorflow/stream_executor/cuda/cuda_dnn.cc:366] Loaded cuDNN version 8300
104/104 [==============================] - 7s 32ms/step - loss: 3.9244e-05 - val_loss: 0.0190
Epoch 2/10
104/104 [==============================] - 3s 26ms/step - loss: 2.8158e-07 - val_loss: 0.0033
Epoch 3/10
104/104 [==============================] - 3s 25ms/step - loss: 1.5459e-07 - val_loss: 0.0031
Epoch 4/10
104/104 [==============================] - 3s 25ms/step - loss: 1.9859e-07 - val_loss: 0.0029
Epoch 5/10
104/104 [==============================] - 3s 25ms/step - loss: 2.0481e-07 - val_loss: 0.0030
Epoch 6/10
104/104 [==============================] - 3s 25ms/step - loss: 1.6410e-07 - val_loss: 0.0028
Epoch 7/10
104/104 [==============================] - 3s 25ms/step - loss: 2.0121e-07 - val_loss: 0.0023
Epoch 8/10
104/104 [==============================] - 3s 26ms/step - loss: 1.9869e-07 - val_loss: 0.0038
Epoch 9/10
104/104 [==============================] - 3s 26ms/step - loss: 1.7274e-07 - val_loss: 0.0026
Epoch 10/10
104/104 [==============================] - 3s 25ms/step - loss: 1.5691e-07 - val_loss: 0.0025
```

Out[27]:
```
<keras.callbacks.History at 0x7ffa58252bb0>
```

In [28]:
```
### Lets Do the prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

In [29]:
```
##Transformback to original form
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

In [30]:
```
### Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

Out[30]:
```
0.7679865322838249
```

In [31]:
```
### Test Data RMSE
math.sqrt(mean_squared_error(ytest,test_predict))
```

Out[31]:
```
43.025365207938115
```

In [32]:
```
import pennylane as qml

n_qubits = 2
dev = qml.device("default.qubit", wires=n_qubits)

@qml.qnode(dev)
def qnode(inputs, weights):
    qml.AngleEmbedding(inputs, wires=range(n_qubits))
    qml.BasicEntanglerLayers(weights, wires=range(n_qubits))
    return [qml.expval(qml.PauliZ(wires=i)) for i in range(n_qubits)]


n_layers = 6
weight_shapes = {"weights": (n_layers, n_qubits)}

qlayer = qml.qnn.KerasLayer(qnode, weight_shapes, output_dim=n_qubits)



model=Sequential()
```

```
model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
model.add(LSTM(20,return_sequences=True))
model.add(LSTM(10))
model.add(Dense(2))
model.add(qlayer)
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 100, 50)           10400

 lstm_4 (LSTM)               (None, 100, 20)           5680

 lstm_5 (LSTM)               (None, 10)                1240

 dense_2 (Dense)             (None, 2)                 22

 keras_layer (KerasLayer)    (None, 2)                 0 (unused)

 dense_3 (Dense)             (None, 1)                 3

=================================================================
Total params: 17,345
Trainable params: 17,345
Non-trainable params: 0
_________________________________________________________________
```

In [33]:
```
model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=2,batch_size=64,verbose=1)
```

```
Epoch 1/2
104/104 [==============================] - 570s 5s/step - loss: 0.0013 - val_loss: 0.0990
Epoch 2/2
104/104 [==============================] - 610s 6s/step - loss: 1.3523e-05 - val_loss: 0.0893
```

Out[33]:
```
<keras.callbacks.History at 0x7ff9e00925b0>
```

In [37]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 100, 50)           10400

 lstm_4 (LSTM)               (None, 100, 20)           5680

 lstm_5 (LSTM)               (None, 10)                1240

 dense_2 (Dense)             (None, 2)                 22

 keras_layer (KerasLayer)    (None, 2)                 12

 dense_3 (Dense)             (None, 1)                 3

=================================================================
Total params: 17,357
Trainable params: 17,357
Non-trainable params: 0
_________________________________________________________________
```

In [34]:
```
### Lets Do the prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

In [35]:
```
##Transformback to original form
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)

### Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

Out[35]:
```
0.48442422883130176
```

In [36]:
```
# Calculate RMSE performance on Test Metrics!
math.sqrt(mean_squared_error(ytest,test_predict))
```

Out[36]:
```
1.7449557868671541
```

In [ ]: