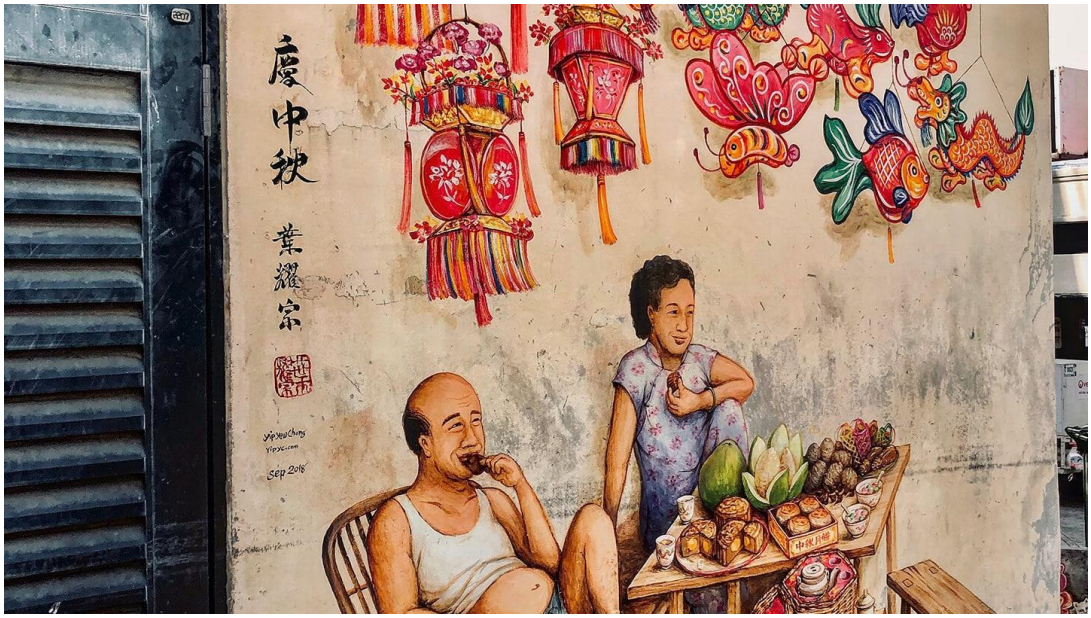
PREDICTING DBS STOCK PRICE WITH RNN

USING TENSORFLOW

DR. ALVIN ANG

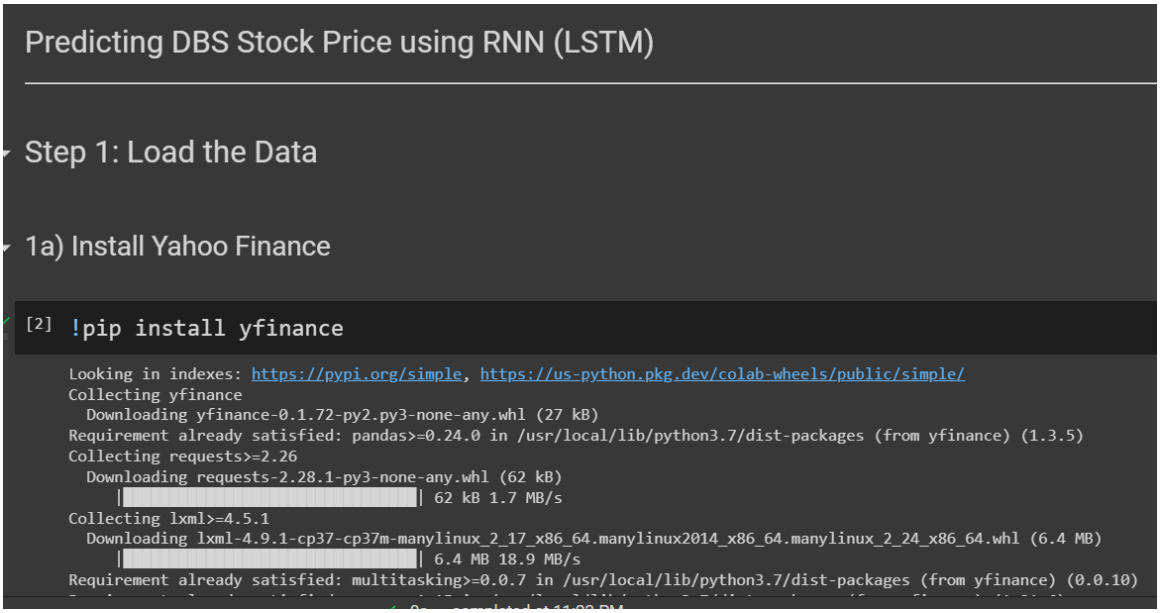


CONTENTS

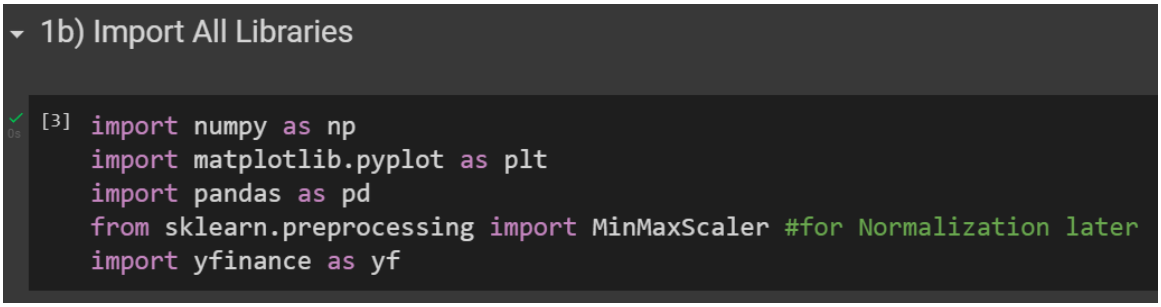
STEP 1: LOAD THE DATA

https://www.alvinang.sg/s/IBF_Day_4_Predicting_DBS_Stock_Price_using_LSTMRNN_by_Dr_Alvin_Ang.ipynb

A. INSTALL YAHOO FINANCE



B. IMPORT ALL LIBRARIES



C. CHOOSING START AND END DATES

1c) Choosing Start and End Dates

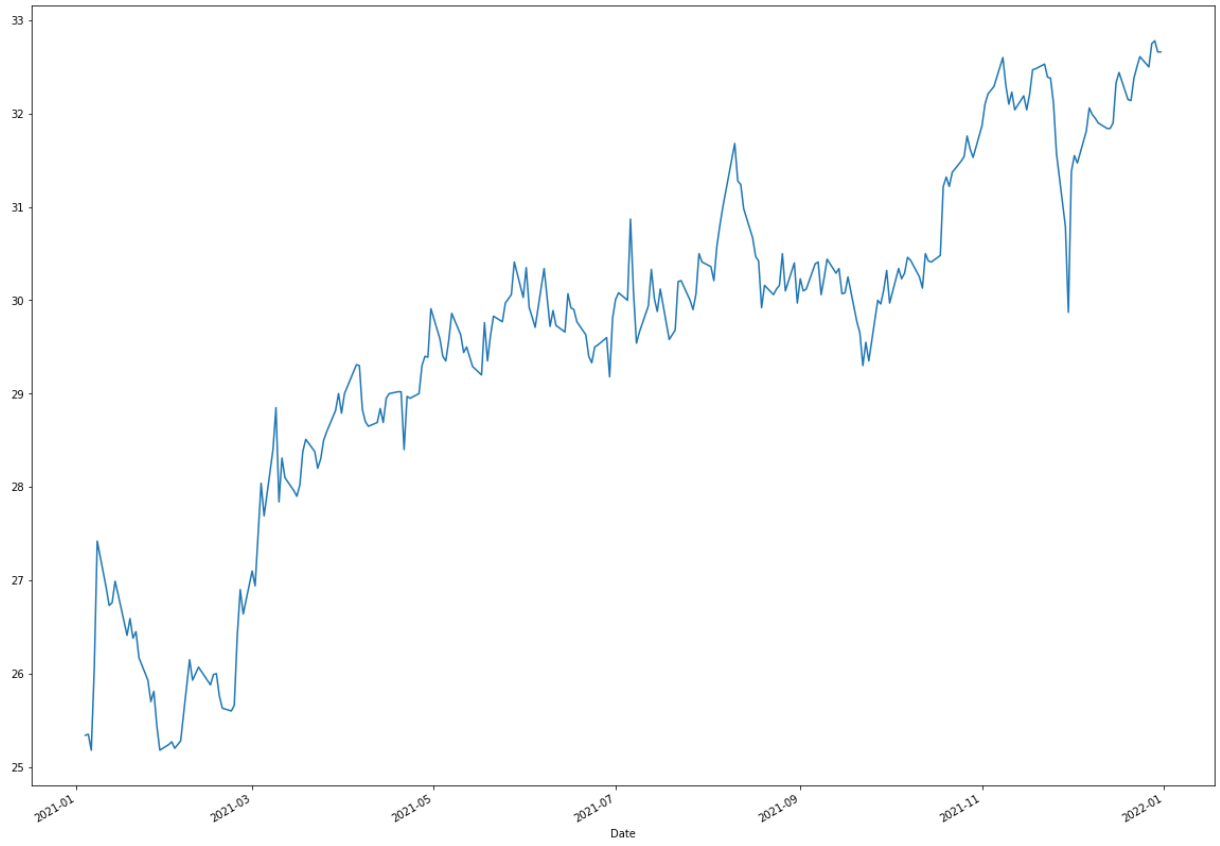
```
[4] start_date = '2021-01-01'
    end_date = '2022-01-01'
    ticker = 'D05.SI' # DBS Bank
    stock_dataset = yf.download(ticker, start=start_date, end=end_date)

    display(stock_dataset.head(10)) #display first 10 days
```

```
[**********100%**********]  1 of 1 completed
```

Date	Open	High	Low	Close	Adj Close	Volume
2021-01-04	25.129999	25.340000	25.010000	25.340000	23.995502	2156614
2021-01-05	25.219999	25.350000	25.090000	25.350000	24.004972	2076800
2021-01-06	25.299999	25.350000	25.070000	25.180000	23.843990	3807600
2021-01-07	25.680000	26.180000	25.629999	26.049999	24.667831	11938600
2021-01-08	26.549999	27.469999	26.500000	27.420000	25.965141	14510800
2021-01-11	27.389999	27.389999	26.920000	26.920000	25.491671	5528700
2021-01-12	26.790001	26.860001	26.520000	26.730000	25.311750	4847200
2021-01-13	27.000000	27.100000	26.719999	26.760000	25.340158	3740200
2021-01-14	26.900000	26.990000	26.770000	26.990000	25.557957	2521200
2021-01-15	27.000000	27.020000	26.790001	26.850000	25.425386	3097600

D. VISUALIZE THE DATA

```
1d) Visualize the Data

[ ] stock_dataset['Close'].plot(figsize = (20, 15))
```



E. EXTRACT THE 'CLOSE' PRICE'

1e) Extract the 'Close' Price

```
[5] import tensorflow as tf

    dataset = stock_dataset['Close'].values
```

```
[6] dataset
```

```
array([25.34000015, 25.35000038, 25.18000031, 26.04999924, 27.42000008,
       26.92000008, 26.72999954, 26.76000023, 26.98999977, 26.85000038,
       26.40999985, 26.59000015, 26.37999916, 26.45000076, 26.17000008,
       25.93000031, 25.70000076, 25.80999947, 25.44000053, 25.18000031,
       25.23999977, 25.27000046, 25.20000076, 25.23999977, 25.28000069,
       26.14999962, 25.93000031, 26.        , 26.06999969, 25.87999916,
       25.98999977, 26.        , 25.76000023, 25.62999916, 25.60000038,
       25.65999985, 26.39999962, 26.89999962, 26.63999939, 27.10000038,
       26.94000053, 27.47999954, 28.04000092, 27.69000053, 28.39999962,
       28.85000038, 27.84000015, 28.30999947, 28.10000038, 27.95999908,
       27.89999962, 28.02000046, 28.37999916, 28.51000023, 28.37999916,
       28.20000076, 28.29999924, 28.5       , 28.59000015, 28.81999969,
       29.        , 28.79000092, 29.        , 29.30999947, 29.29999924,
       28.82999992, 28.70000076, 28.64999962, 28.69000053, 28.84000015,
```

```
[7] dataset.shape
    # 253 'Close' prices... meaning 253 days of data...

    #1 year = 365 days
    #1 month = 4 weekends = 4x2 = 8 non-market opening days

    #1 year = 365 - (8x12) = 269 WORKING days
    #but there are public holidays too (market is not open)
    #and febrauary month might have 5 weeks
    #that's how we derive 253 days of market opening / "working" days..
```

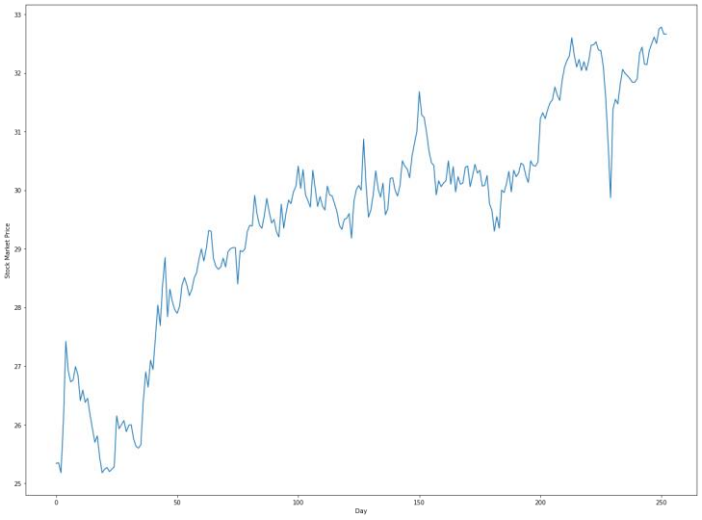
```
(253,)
```

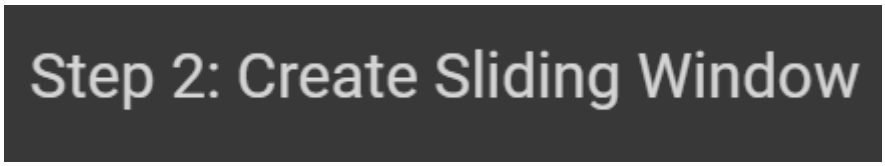
```
[8] dataset = dataset[..., tf.newaxis]
    #convert entire dataset to 1 column
```

```
dataset.shape
#253 rows x 1 column

(253, 1)
```

F. PLOT THE DATASET

```
1f) Plot the Dataset

plt.figure(figsize=(20,15))
plt.plot(dataset)

plt.xlabel('Day')
plt.ylabel('Stock Market Price')
plt.show()
```





This is what sliding window does...
window slides down 1 row each step
timestep = 4 means capture 4 rows of data each time

	Date	Open	High	Low	Close	Adj Close	Volume
Day 1	2021-01-04	25.129999	25.340000	25.010000	25.340000	23.995502	2156614
Day 2	2021-01-05	25.219999	25.350000	25.090000	25.350000	24.004972	2076800
Day 3	2021-01-06	25.299999	25.350000	25.070000	25.180000	23.843892	3807600
Day 4	2021-01-07	25.680000	26.180000	25.629999	26.049999	24.667830	11938600
Day 5	2021-01-08	26.549999	27.469999	26.500000	27.420000	25.965139	14510800
Day 6	2021-01-11	27.389999	27.389999	26.920000	26.920000	25.491671	5528700
Day 7	2021-01-12	26.790001	26.860001	26.520000	26.730000	25.311750	4847200
Day 8	2021-01-13	27.000000	27.100000	26.719999	26.760000	25.340160	3740200
Day 9	2021-01-14	26.900000	26.990000	26.770000	26.990000	25.557955	2521200
Day 10	2021-01-15	27.000000	27.020000	26.790001	26.850000	25.425386	3097600

X[0]
X[1]
X[2]
X[3]

A. DEFINING THE SLIDING WINDOW FUNCTION

2a) Defining the Sliding Window Function

```
[10] # Define sliding window
     #kindly ignore the details inside the "def" and "for" loop....

     def sliding_window(data, seq_length):
         x = [] #create an empty array to put data inside later
         y = []


         for i in range(len(data)-seq_length-1):
             _x = data[i:(i+seq_length)] #day 1 to day 5
             _y = data[i+seq_length] #next day -> day 6
             x.append(_x) #appending (put inside) the x empty array
             y.append(_y)

         return np.array(x), np.array(y)
```

B. NORMALIZE THE DATA

2b) Normalize the Data

```
[11] # Normalization
     sc = MinMaxScaler()
     dataset = sc.fit_transform(dataset)

     #we normalize because Machine Learning likes to use small numbers...
     #not large numbers... harder to crunch....
```

C. DEFINING TIMESTEPS

2c) Defining Timesteps

```
[12] timesteps = 4 #4 is the window size
     X, y = sliding_window(dataset, timesteps)
```

```
[13] X
     #you can see each window is capturing 4 row (days) of close price data
     #and the subsequent window is the next sliding window
```

```
array([[[0.02105262],
        [0.02236844],
        [0.        ],
        [0.11447357]],

       [[0.02236844],
        [0.        ],
        [0.11447357],
        [0.29473687]],

       [[0.        ],
        [0.11447357],
        [0.29473687],
        [0.22894738]],
```

```
[14] X.shape
     #248 rows of data (working days)...hmm seems like some days got chopped off
     #after scaling and sliding window... coz initially was 253...

     #4 rows or days per 1 slicing window

     (248, 4, 1)

     X[0] #first window -> day 1, 2, 3, 4

     array([[0.02105262],
            [0.02236844],
            [0.        ],
            [0.11447357]])

[16] X[1] #second window -> day 2, 3, 4, 5

     array([[0.02236844],
            [0.        ],
            [0.11447357],
            [0.29473687]])

[17] y[0]
     # y is next day price, so is day 5
     #i'm using the past 4 days to predict the next day price, day 5
     #or rather, day 5's actual price is used as the "learning" price

     array([0.29473687])
```

	Date	Open	High	Low	Close	Adj Close	Volume
Day 1	2021-01-04	25.129999	25.340000	25.010000	25.340000	23.995502	2156614
Day 2	2021-01-05	25.219999	25.350000	25.090000	25.350000	24.004972	2076800
Day 3	2021-01-06	25.299999	25.350000	25.070000	25.180000	23.843992	3807600
Day 4	2021-01-07	25.680000	26.180000	25.629999	26.049999	24.667330	11928800
Day 5	2021-01-08	26.549999	27.469999	26.500000	27.420000	25.965139	14510800
Day 6	2021-01-11	27.389999	27.389999	26.920000	26.920000	25.491671	5528700
Day 7	2021-01-12	26.790001	26.860001	26.520000	26.730000	25.311750	4847200
Day 8	2021-01-13	27.000000	27.100000	26.719999	26.760000	25.340160	3740200
Day 9	2021-01-14	26.900000	26.990000	26.770000	26.990000	25.557955	2521200
Day 10	2021-01-15	27.000000	27.020000	26.790001	26.850000	25.425386	3097600

X[0]

this is the price prediction for the previous window

III. STEP 3: TRAIN TEST SPLIT

A. DEFINING THE SIZE FOR TRAINING AND TESTING (USING THE LENGTH OF Y)

Step 3: Train Test Split

3a) Defining the Size for Training and Testing (using the length of y)

```
[18] train_size = int(len(y) * 0.67)
     #training size is 2/3 of the dataset

     test_size = int(len(y) - train_size)
     #testing size is 1/3 of the dataset
```

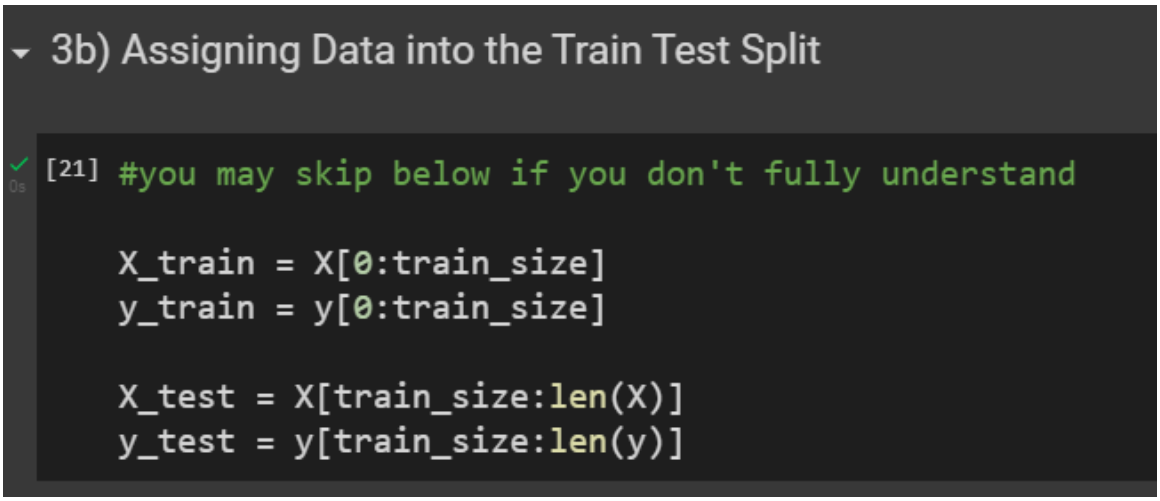
```
[19] train_size
     #166 rows of data = 166 days used for training the model
```

```
166
```

```
[20] test_size
     #82 rows of data = 82 days used for testing the model

     #total 166 + 82  = 248 days (market opening days)
```

```
82
```

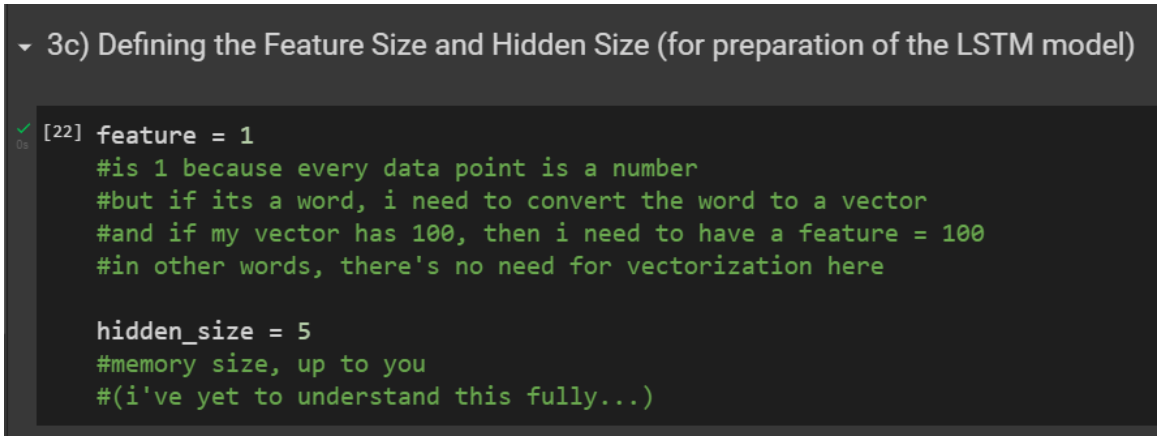
B. ASSIGNING DATA INTO THE TRAIN TEST SPLIT



3b) Assigning Data into the Train Test Split

```
[21] #you may skip below if you don't fully understand

     X_train = X[0:train_size]
     y_train = y[0:train_size]

     X_test = X[train_size:len(X)]
     y_test = y[train_size:len(y)]
```

C. DEFINING THE FEATURE SIZE AND HIDDEN SIZE (FOR PREPARATION OF THE LSTM MODEL)



3c) Defining the Feature Size and Hidden Size (for preparation of the LSTM model)

```
[22] feature = 1
     #is 1 because every data point is a number
     #but if its a word, i need to convert the word to a vector
     #and if my vector has 100, then i need to have a feature = 100
     #in other words, there's no need for vectorization here

     hidden_size = 5
     #memory size, up to you
     #(i've yet to understand this fully...)
```

D. RESHAPING X_TRAIN AND X_TEST TO STORE INFORMATION ABOUT THE BATCH, TIMTESPTES, FEATURE

3d) Reshaping X_train and X_test to store information about the Batch, Timesteps, Feature

LSTM input definition

inputs = np.random.randn(batch, timesteps, feature))

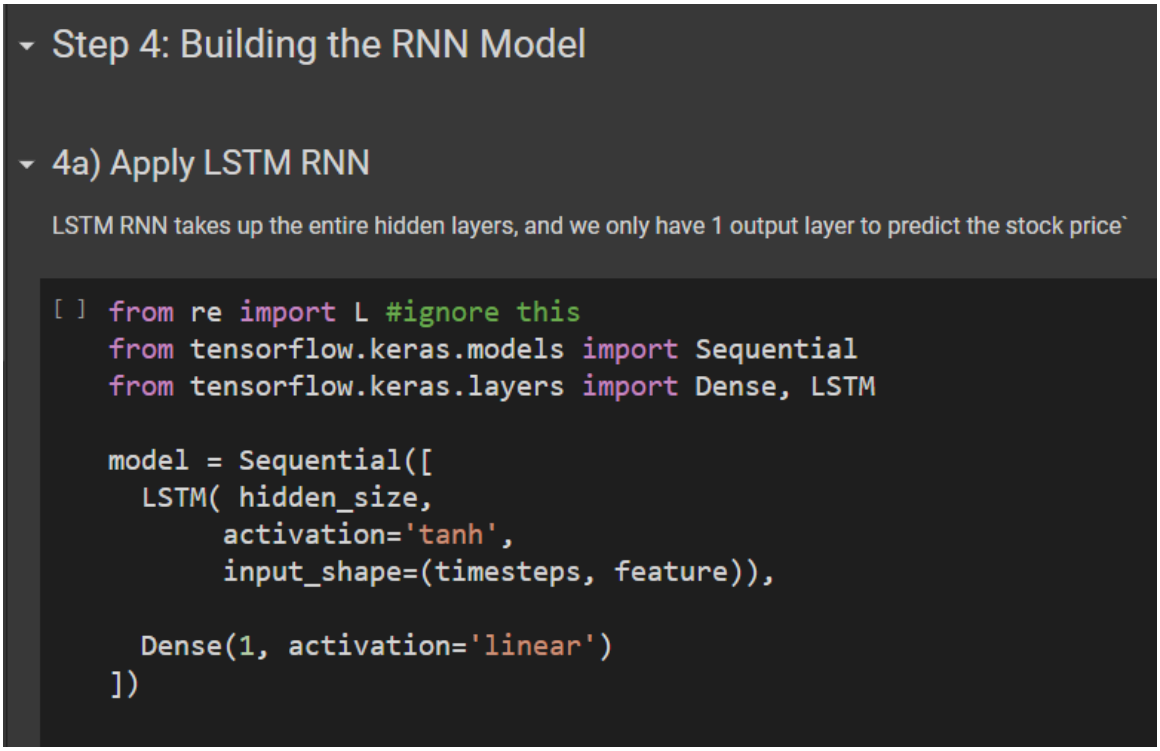
```
[ ] # inputs: A 3D tensor with shape [batch, timesteps, feature].
    X_train = np.reshape(X_train, (X_train.shape[0], timesteps, feature))

    # batch size = X_train.shape[0]
    # timesteps = 4 because its the size of sliding window
    # feature = 1 as explained earlier
```

```
[ ] X_test = np.reshape(X_test, (X_test.shape[0], timesteps, 1, feature))
```

```
[ ] X_test

array([[[[0.66447372]],

        [[0.64736856]],

        [[0.6500002 ]],

        [[0.68552633]]],


       [[[0.64736856]],

        [[0.6500002 ]],

        [[0.68552633]],

        [[0.68815797]]],


       [[[0.6500002 ]],

        [[0.68552633]],

        [[0.68815797]],

        [[0.64210528]]],


       [[[0.68552633]],

        [[0.68815797]],

        [[0.64210528]],

        [[0.66578954]]],
```

IV. STEP 4: BUILDING THE RNN MODEL

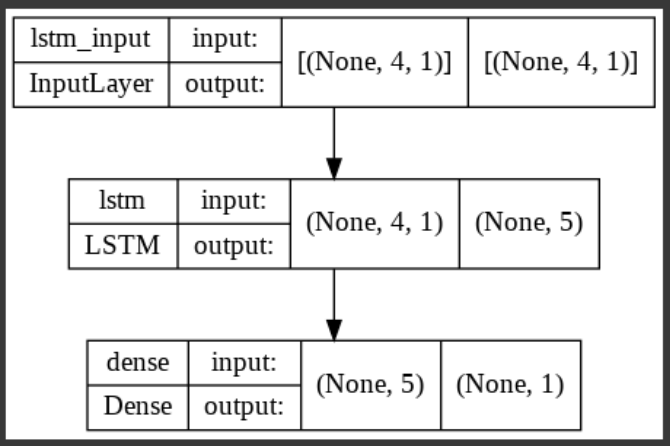
A. APPLY LSTM RNN



B. VISUALIZE THE MODEL

4b) Visualize the Model

```
[ ] import pydot
    tf.keras.utils.plot_model(model, 'model.png', show_shapes = True)
```



C. COMPILE THE MODEL

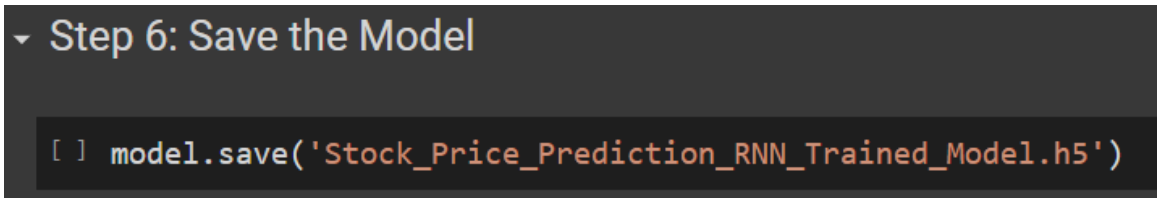
4c) Compile the Model

```
[ ] ADAM = tf.keras.optimizers.Adam(learning_rate=0.002)

    model.compile(
        loss='mse',
        optimizer=ADAM)
```

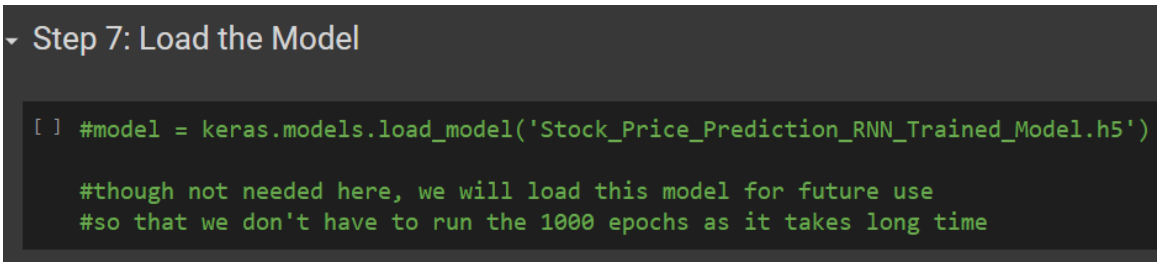
Step 5: Training the Model

```
[ ] history = model.fit(X_train, y_train, epochs=1000)
```

```
Epoch 1/1000
6/6 [==============================] - 2s 4ms/step - loss: 0.1589
Epoch 2/1000
6/6 [==============================] - 0s 4ms/step - loss: 0.1255
Epoch 3/1000
6/6 [==============================] - 0s 4ms/step - loss: 0.0959
Epoch 4/1000
6/6 [==============================] - 0s 4ms/step - loss: 0.0692
Epoch 5/1000
6/6 [==============================] - 0s 4ms/step - loss: 0.0470
Epoch 6/1000
6/6 [==============================] - 0s 4ms/step - loss: 0.0297
Epoch 7/1000
6/6 [==============================] - 0s 4ms/step - loss: 0.0180
Epoch 8/1000
6/6 [==============================] - 0s 4ms/step - loss: 0.0118
Epoch 9/1000
6/6 [==============================] - 0s 5ms/step - loss: 0.0095
Epoch 10/1000
6/6 [==============================] - 0s 4ms/step - loss: 0.0091
Epoch 11/1000
6/6 [==============================] - 0s 4ms/step - loss: 0.0091
```

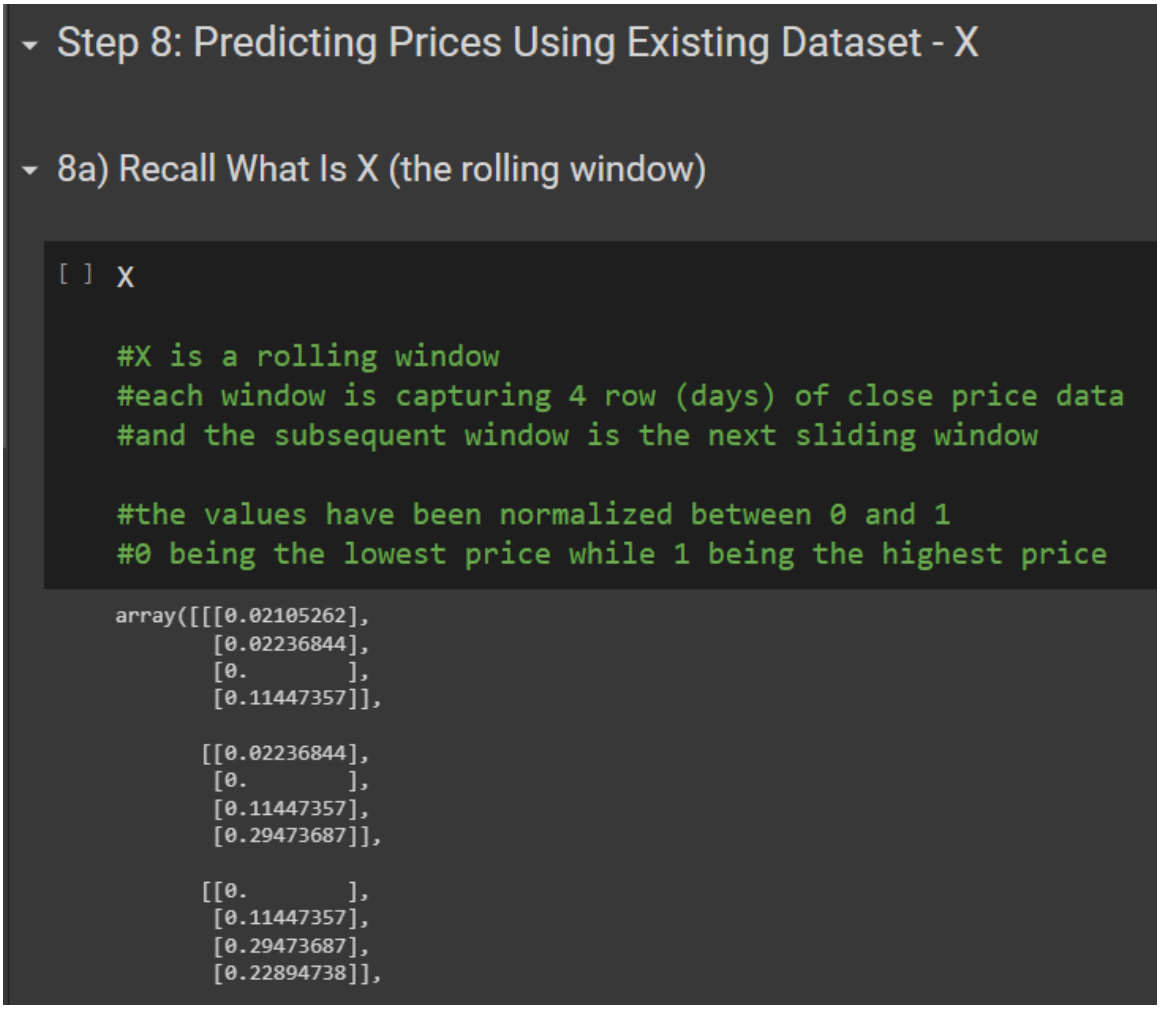


VII. STEP 7: LOAD THE MODEL



VIII. STEP 8: PREDICTING PRICES USING EXISTING DATASET – X

A. RECALL WHAT IS X (THE ROLLING WINDOW)



B. RECALL WHAT IS Y(THE NORMALIZED ACTUAL PRICES USED FOR TRAINING)

8b) Recall What is y (the normalized actual prices used for Training)

```
[ ] y[0]

    #this is the 5th day actual price (taken from X)
```

```
array([0.29473687])
```

```
[ ] y

    #y captures and stores all pricing from day 5 onwards, taking them as
    #the actual pricing for training purpose
```

```
array([[0.29473687],
       [0.22894738],
       [0.20394731],
       [0.20789477],
       [0.23815787],
       [0.2197369 ],
       [0.16184208],
       [0.18552633],
       [0.15789462],
       [0.16710536],
       [0.13026315],
       [0.09868423],
       [0.06842113],
       [0.08289464],
       [0.03421056],
       [0.        ],
```

C. OBTAIN Y_: UN-NORMALIZED ACTUAL PRICES

8c) Obtain y_: Un-Normalized Actual Prices

```
[ ] y_ = sc.inverse_transform(y)

    #we un-normalize y to get back the actual price
```

```
[ ] y_
```

```
array([[27.42000008],
       [26.92000008],
       [26.72999954],
       [26.76000023],
       [26.98999977],
       [26.85000038],
       [26.40999985],
       [26.59000015],
       [26.37999916],
       [26.45000076],
       [26.17000008],
       [25.93000031],
       [25.70000076],
       [25.80999947],
       [25.44000053],
       [25.18000031],
       [25.23999977],
       [25.27000046],
       [25.20000076],
       [25.23999977],
       [25.28000069],
```

D. USING THE TRAINED RNN MODEL TO PREDICT X DATASET (YHAT)

8d) Using the Trained RNN Model to Predict X dataset (yhat)

```
[ ] yhat = model.predict(X)

    #fitting the model to X dataset
```

```
[ ] yhat = sc.inverse_transform(yhat)

    #un-normalizing yhat to get the predicted price
```

```
[ ] yhat

    #previewing the predicted prices
```

```
array([[25.99750485],
       [27.18125749],
       [27.02904555],
       [26.89250475],
       [26.90001926],
       [27.07005102],
       [26.9858451 ],
       [26.6239624 ],
       [26.70861101],
       [26.54401577],
       [26.57010285],
       [26.35231555],
       [26.12481054],
       [25.90330234],
       [25.94048544],
       [25.65870158],
       [25.41253983],
```

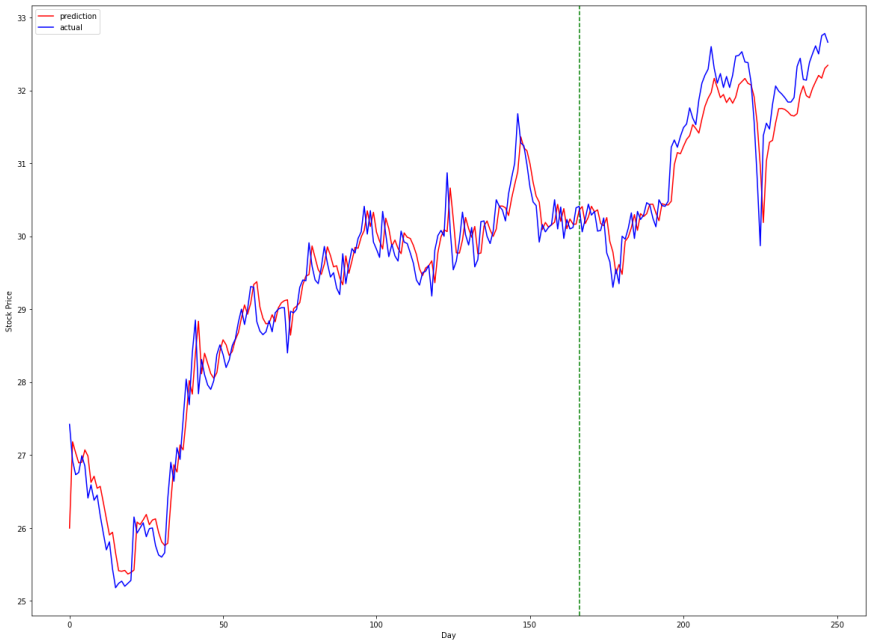
```
yhat.shape

#248 days of predicted prices
```

```
(248, 1)
```

E. VISUALIZING THE PREDICTION

8e) Visualizing the Prediction

```
[ ] plt.figure(figsize=(20,15))

    plt.axvline(x=train_size, c='g', linestyle='--')
    #draw a dotted line to show the split between Train vs Test datasets

    plt.plot(yhat,'r',label='prediction')
    plt.plot(y_,'b',label='actual')
    #yhat is the Predicted price using the RNN model
    #y_ is the Actual Price

    plt.xlabel('Day')
    plt.ylabel('Stock Price')
    plt.legend()
    plt.show()
```



F. UNDERSTANDING THE DIFFERENCE BETWEEN ORIGINAL_X // YHAT // AND Y_

1. ORIGINAL_X

8f) Understanding the Difference between original_x // yhat // and y_

8f)(i) Original_x

```
[ ] sd_1 = stock_dataset.filter(['Date', 'Close'])
    sd_1

    #we extract out only the 'Date' and 'Close' columns from the original dataset
```

Date	Close
2021-01-04	25.340000
2021-01-05	25.350000
2021-01-06	25.180000
2021-01-07	26.049999
2021-01-08	27.420000
...	...
2021-12-27	32.500000
2021-12-28	32.750000
2021-12-29	32.779999
2021-12-30	32.660000
2021-12-31	32.660000

253 rows × 1 columns

```
sd_1.insert(loc=0, column='Day', value=np.arange(len(sd_1)))
sd_1

#we insert a Day column....
```

Date	Day	Close
2021-01-04	0	25.340000
2021-01-05	1	25.350000
2021-01-06	2	25.180000
2021-01-07	3	26.049999
2021-01-08	4	27.420000
...	...	...
2021-12-27	248	32.500000
2021-12-28	249	32.750000
2021-12-29	250	32.779999
2021-12-30	251	32.660000
2021-12-31	252	32.660000

253 rows × 2 columns

```
original_x = stock_dataset['Close'].values
#we store the 'Close' price into original_x
```

```
original_x[4]
#this is Actual Close price for day 4
```

```
27.420000076293945
```

2. Y_[0]

```
8f)(ii) y_[0]

[ ] y_[0]
    #we see that this is the actual price for day 4 stored inside y_[1]
    #similar to original_x[4]

    array([27.42000008])
```

3. YHAT[0]

```
8f)(iii) yhat[0]

[ ] yhat[0]
    #this is the PREDICTED price for day 4 stored inside yhat[1]

    array([25.99750485])
```

G. TRYING TO PREDICT DAY 250

```
8g) Trying to Predict Day 250

yhat[248]
#we try to PREDICT the price for Day 252
#but an error pops up: Max size is 248
#which means that we can PREDICT only up to Day 251

#what if we want to predict after Day 251? Please see the next section...

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-151-795ec28d0580> in <module>()
----> 1 yhat[248]
      2 #we try to PREDICT the price for day 252
      3 #but an error pops up: Max size is 248
      4 #which means that

IndexError: index 248 is out of bounds for axis 0 with size 248

SEARCH STACK OVERFLOW
```

IX. STEP 9: PREDICTING NEXT DAY PRICE USING NEW DATASET

A. PRESUME WE NOW HAVE A RANDOM PAST 8 DAYS OF PRICE DATA

Step 9: Predicting Next Day Price Using New Dataset

9a) Presume we now have a random past 8 days of Price data

```
past8days = pd.DataFrame([34, 36, 20, 6, 50, 10, 98, 7])

past8days
```

	0
0	34
1	36
2	20
3	6
4	50
5	10
6	98
7	7

B. WE NORMALIZE THE PAST 8 DAYS PRICES

```
9b) We Normalize the Past 8 Days Prices

[ ] past8days_norm = sc.fit_transform(past4days)

[ ] past8days_norm

    array([[0.30434783],
           [0.32608696],
           [0.15217391],
           [0.        ],
           [0.47826087],
           [0.04347826],
           [1.        ],
           [0.01086957]])
```

C. WE FIT THE PAST 8 DAYS PRICES INTO THE RNN MODEL

```
9c) We Fit the Past 8 Days Prices into the RNN Model

[ ] new_pred_norm = model.predict(past4days_norm)

[ ] new_pred = sc.inverse_transform(new_pred_norm)
```

D. PREDICTED CLOSE PRICE FOR THE 9TH DAY

9d) Predicted Close Price for the 9th Day

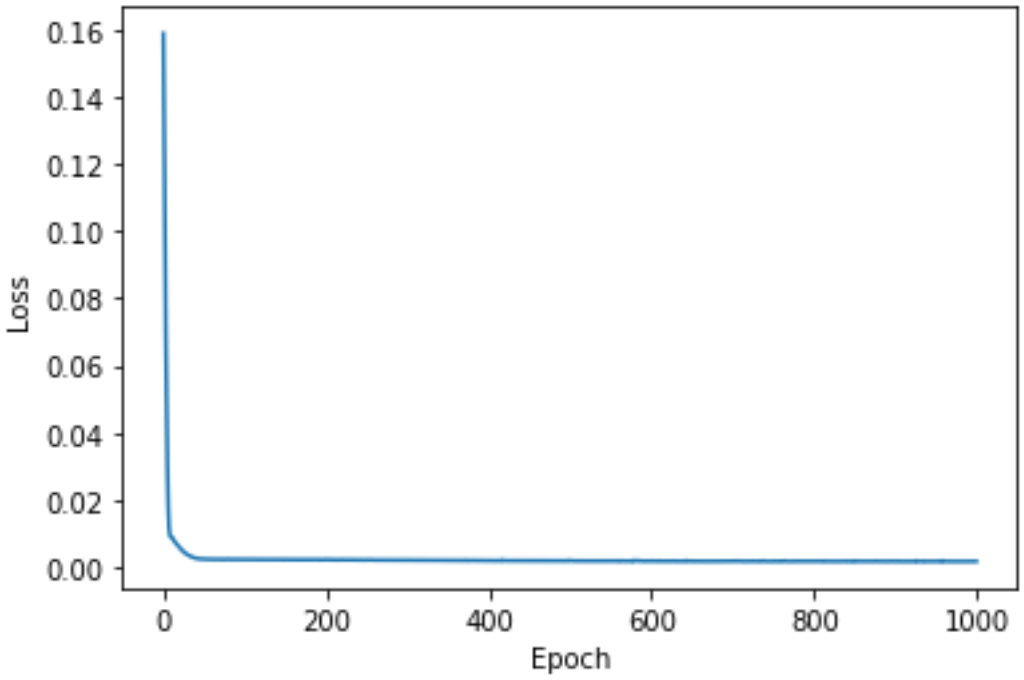
```
[ ] new_pred[0]
    #the predicted close price for the 9th day is 25.55
```

```
array([25.557405], dtype=float32)
```

Step 10: Visualizing the Loss

```
[ ] loss = history.history['loss']
    epoch = range(len(loss))
```

```
import matplotlib.pyplot as plt

plt.plot(epoch, loss)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```



ABOUT DR. ALVIN ANG

Dr. Alvin Ang earned his Ph.D., Masters and Bachelor degrees from NTU, Singapore. He is a scientist, entrepreneur, as well as a personal/business advisor. More about him at www.AlvinAng.sg.