```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler
from datetime import datetime


# Load the dataset (make sure the CSV file is in the working directory)
df = pd.read_csv("TCS Historical Data.csv")


# Convert the 'Date' column to datetime and set as index
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
```

```
<ipython-input-31-61bcd5824fb3>:2: UserWarning: Parsing dates in %d-%m-%Y format when dayfirst=False (the default) was specified. Pass `
  df['Date'] = pd.to_datetime(df['Date'])
```

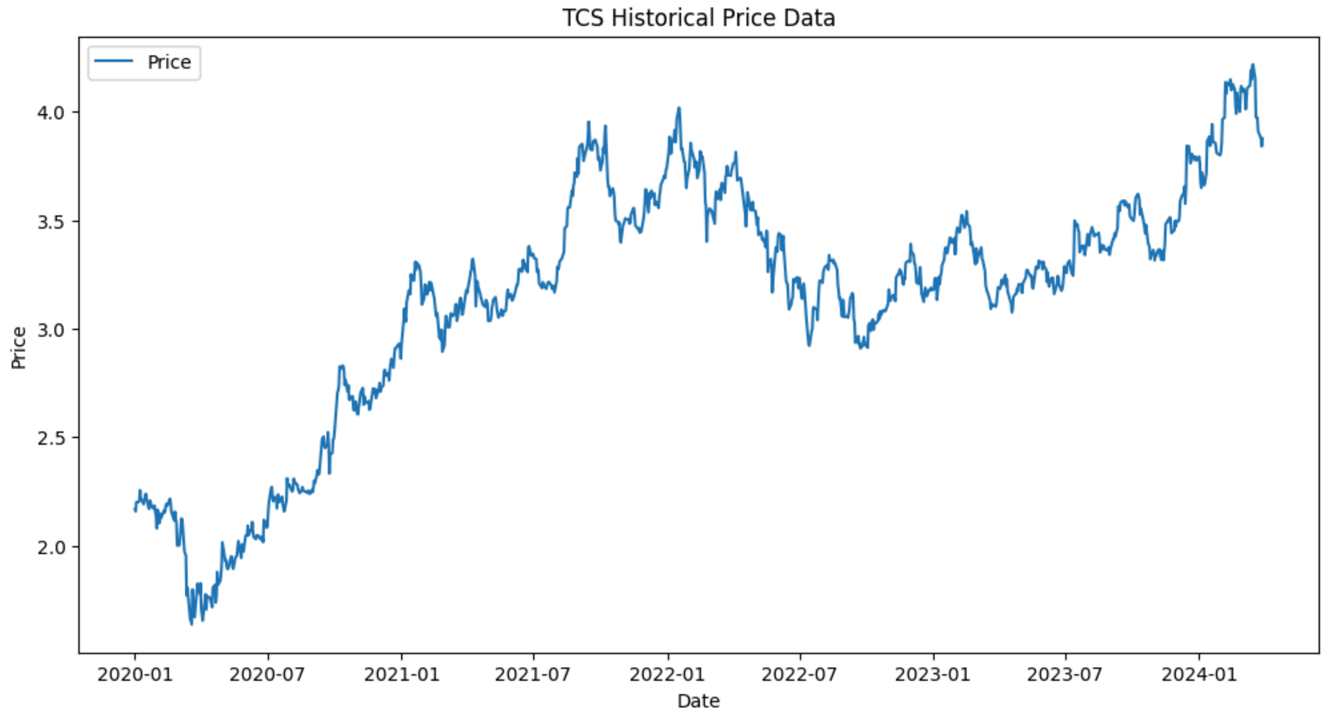
```
# df[df['Price'].astype(str).str.contains(r'\d+\.\d+\.\d+')]


df['Price'] = df['Price'].astype(str).str.replace(r'[^\d,]', '', regex=True)  # Remove non-numeric characters except commas
df['Price'] = df['Price'].str.replace(',', '.').astype(float)
```

+ Code + Text

```
print(df['Price'].dtype)  # Should print 'float64'
print(df.head())  # Display cleaned values
```

```
float64
              Price      Open      High       Low   Vol. Change %
Date
2024-03-28  3.87630  3,850.10  3,915.00  3,840.50  4.31M    0.92%
2024-03-27  3.84090  3,888.50  3,895.00  3,829.40  1.97M   -0.94%
2024-03-26  3.87750  3,875.00  3,946.70  3,871.45  3.44M   -0.85%
2024-03-22  3.91090  3,897.00  3,938.00  3,855.00  5.85M   -1.56%
2024-03-21  3.97295  3,990.05  4,008.40  3,948.00  3.83M    0.05%
```

```
# Visualize the data (optional)
plt.figure(figsize=(12,6))
plt.plot(df.index, df['Price'], label="Price")
plt.xlabel("Date")
plt.ylabel("Price")
plt.title("TCS Historical Price Data")
plt.legend()
plt.show()
```




```
# Scale the data to the range [0,1]
scaler = MinMaxScaler(feature_range=(0, 1))
price_data = scaler.fit_transform(df[['Price']])
```

```
# Define a function to create sequences for training/testing
def create_dataset(data, look_back=60):
    X, y = [], []
    for i in range(len(data) - look_back):
        X.append(data[i:i + look_back, 0])
        y.append(data[i + look_back, 0])
    return np.array(X), np.array(y)
```

```
# Set look_back period (e.g., using 60 days of past data to predict the next day)
look_back = 60
X, y = create_dataset(price_data, look_back)
```

```
# Reshape input to be [samples, time steps, features]
X = np.reshape(X, (X.shape[0], X.shape[1], 1))
```

```
# Optionally, split the data into training and testing sets
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

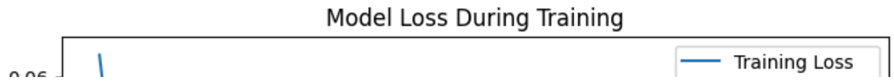
```
# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=25))
model.add(Dense(units=1))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(**kwargs)
```

```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))
```

```
Epoch 1/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 5s 56ms/step - loss: 0.1448 - val_loss: 0.0157
Epoch 2/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - loss: 0.0095 - val_loss: 0.0154
Epoch 3/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - loss: 0.0073 - val_loss: 0.0162
Epoch 4/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0062 - val_loss: 0.0118
Epoch 5/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - loss: 0.0059 - val_loss: 0.0123
Epoch 6/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.0049 - val_loss: 0.0110
Epoch 7/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.0052 - val_loss: 0.0081
Epoch 8/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - loss: 0.0048 - val_loss: 0.0079
Epoch 9/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.0038 - val_loss: 0.0066
Epoch 10/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - loss: 0.0037 - val_loss: 0.0053
Epoch 11/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - loss: 0.0044 - val_loss: 0.0050
Epoch 12/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - loss: 0.0038 - val_loss: 0.0052
Epoch 13/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - loss: 0.0036 - val_loss: 0.0044
Epoch 14/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 0.0039 - val_loss: 0.0055
Epoch 15/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.0034 - val_loss: 0.0045
Epoch 16/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 0.0038 - val_loss: 0.0046
Epoch 17/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - loss: 0.0029 - val_loss: 0.0047
Epoch 18/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - loss: 0.0029 - val_loss: 0.0036
Epoch 19/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 59ms/step - loss: 0.0027 - val_loss: 0.0048
Epoch 20/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 0.0028 - val_loss: 0.0043
```

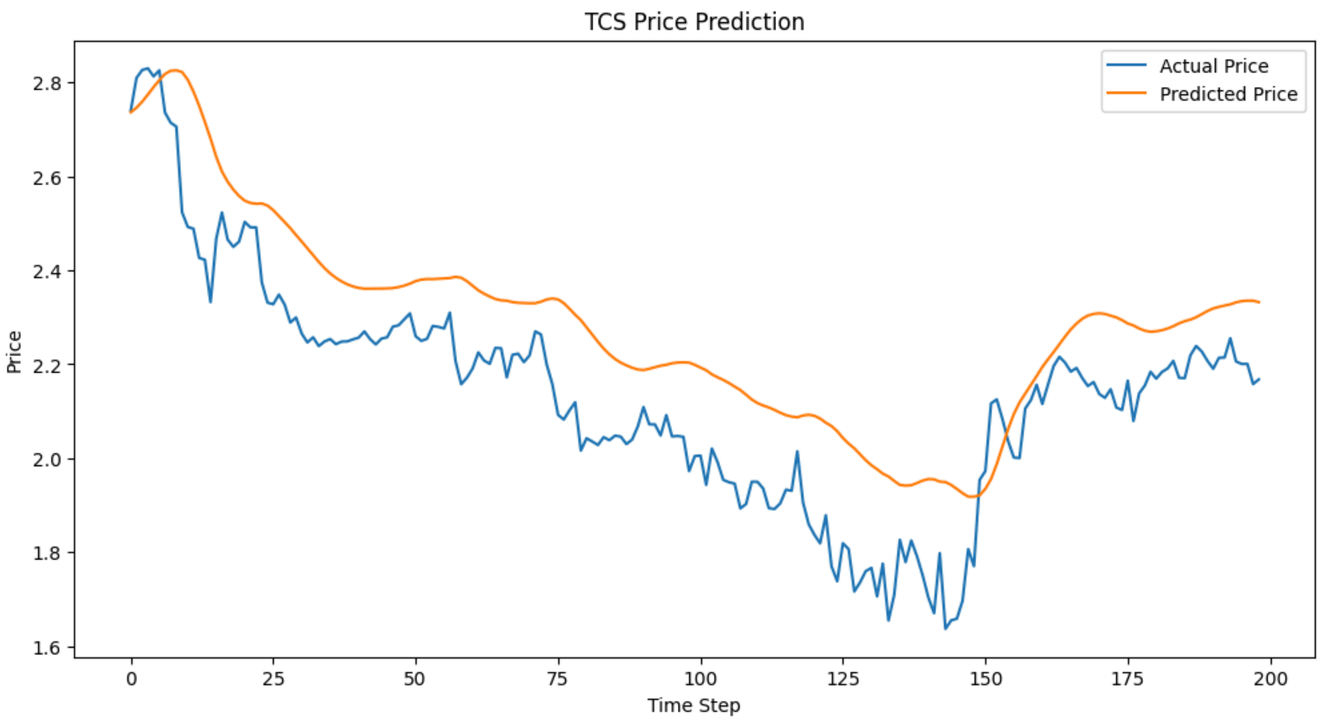
```
# Plot training and validation loss
plt.figure(figsize=(8,4))
plt.plot(history.history['loss'], label="Training Loss")
plt.plot(history.history['val_loss'], label="Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Model Loss During Training")
plt.legend()
plt.show()
```




```
# Make predictions on the test set
predictions = model.predict(X_test)
predictions = scaler.inverse_transform(predictions)
y_test_actual = scaler.inverse_transform(y_test.reshape(-1, 1))
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 134ms/step
```

```
# Plot predictions vs actual prices
plt.figure(figsize=(12,6))
plt.plot(y_test_actual, label="Actual Price")
plt.plot(predictions, label="Predicted Price")
plt.xlabel("Time Step")
plt.ylabel("Price")
plt.title("TCS Price Prediction")
plt.legend()
plt.show()
```



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM,Dense,Dropout
from sklearn.preprocessing import MinMaxScaler
```

```
df=pd.read_csv("./airplane_passenger_prediction.csv")
```

```
df
```

	Month	Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121
...	...	...
139	1960-08	606
140	1960-09	508
141	1960-10	461
142	1960-11	390
143	1960-12	432

144 rows × 2 columns

```
df["Month"]=pd.to_datetime(df["Month"])
df.set_index("Month",inplace=True)
df
```

Month	Passengers
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121
...	...
1960-08-01	606
1960-09-01	508
1960-10-01	461
1960-11-01	390
1960-12-01	432

144 rows × 1 columns

```
scaler=MinMaxScaler(feature_range=(0,1))
```

```
data_scaled=scaler.fit_transform(df)
```

```
def create_sequence(data,seq_length):
  x,y=[],[]
  for i in range(len(data)-seq_length):
    x.append(data[i:(i+seq_length)])
    y.append(data[i+seq_length])
```

```
    y.append(data[i+seq_length])
  return np.array(x),np.array(y)


seq_length=5
X,y=create_sequence(data_scaled,seq_length)


#split data into train-test split
size=int(0.8*len(X))
X_train,X_test=X[:size],X[size:]
y_train,y_test=y[:size],y[size:]


#Define the LSTM model
model=Sequential([LSTM(50,return_sequences=True,input_shape=(seq_length,1)),LSTM(50),Dense(1)])


model.compile(optimizer="adam",loss="mse")
model.fit(X_train,y_train,epochs=50,batch_size=16,verbose=1)
```

```
Epoch 1/50
7/7 [==============================] - 16s 12ms/step - loss: 0.0717
Epoch 2/50
7/7 [==============================] - 0s 16ms/step - loss: 0.0240
Epoch 3/50
7/7 [==============================] - 0s 19ms/step - loss: 0.0179
Epoch 4/50
7/7 [==============================] - 0s 11ms/step - loss: 0.0144
Epoch 5/50
7/7 [==============================] - 0s 15ms/step - loss: 0.0118
Epoch 6/50
7/7 [==============================] - 0s 12ms/step - loss: 0.0114
Epoch 7/50
7/7 [==============================] - 0s 13ms/step - loss: 0.0100
Epoch 8/50
7/7 [==============================] - 0s 18ms/step - loss: 0.0092
Epoch 9/50
7/7 [==============================] - 0s 11ms/step - loss: 0.0089
Epoch 10/50
7/7 [==============================] - 0s 11ms/step - loss: 0.0087
Epoch 11/50
7/7 [==============================] - 0s 12ms/step - loss: 0.0087
Epoch 12/50
7/7 [==============================] - 0s 14ms/step - loss: 0.0087
Epoch 13/50
7/7 [==============================] - 0s 12ms/step - loss: 0.0086
Epoch 14/50
7/7 [==============================] - 0s 12ms/step - loss: 0.0086
Epoch 15/50
7/7 [==============================] - 0s 12ms/step - loss: 0.0085
Epoch 16/50
7/7 [==============================] - 0s 12ms/step - loss: 0.0084
Epoch 17/50
7/7 [==============================] - 0s 11ms/step - loss: 0.0084
Epoch 18/50
7/7 [==============================] - 0s 11ms/step - loss: 0.0083
Epoch 19/50
7/7 [==============================] - 0s 12ms/step - loss: 0.0083
Epoch 20/50
7/7 [==============================] - 0s 11ms/step - loss: 0.0083
Epoch 21/50
7/7 [==============================] - 0s 23ms/step - loss: 0.0084
Epoch 22/50
7/7 [==============================] - 0s 26ms/step - loss: 0.0084
Epoch 23/50
7/7 [==============================] - 0s 13ms/step - loss: 0.0081
Epoch 24/50
7/7 [==============================] - 0s 10ms/step - loss: 0.0086
Epoch 25/50
7/7 [==============================] - 0s 10ms/step - loss: 0.0082
Epoch 26/50
7/7 [==============================] - 0s 10ms/step - loss: 0.0081
Epoch 27/50
7/7 [==============================] - 0s 10ms/step - loss: 0.0080
Epoch 28/50
7/7 [==============================] - 0s 10ms/step - loss: 0.0079
Epoch 29/50
7/7 [==============================] - 0s 10ms/step - loss: 0.0082
```

```
predictions=model.predict(X_test)
```

```
1/1 [==============================] - 3s 3s/step
```

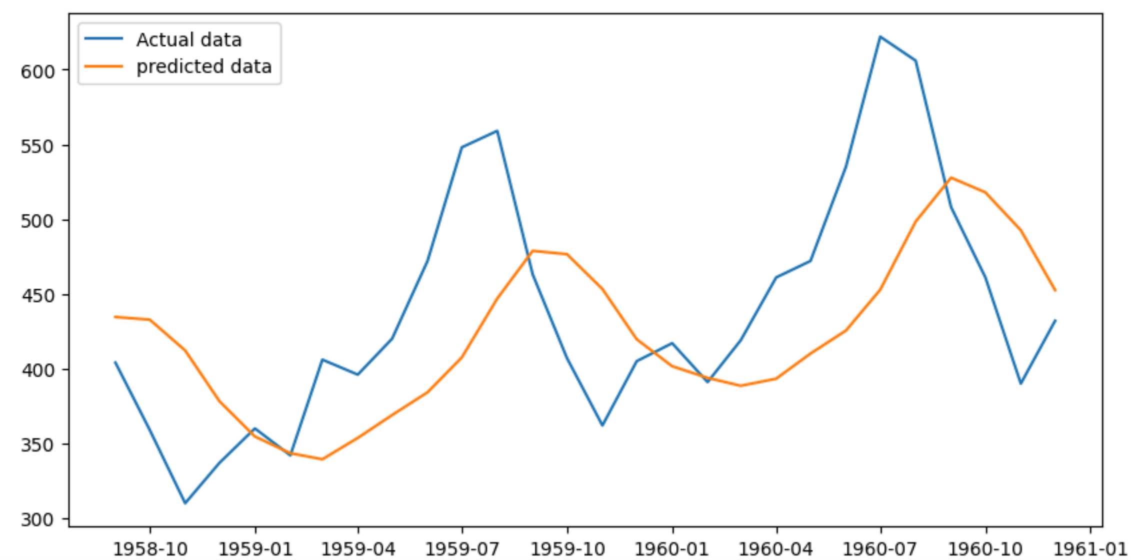
```
predictions=scaler.inverse_transform(predictions)
```

```
predictions
```

```
array([[434.537  ],
       [432.76926],
       [412.17795],
       [378.1969 ],
       [354.6024 ],
       [343.441  ],
       [339.3838 ],
       [353.66962],
       [368.94315],
       [384.16858],
       [407.48694],
       [446.86337],
       [478.76862],
       [476.536  ],
       [453.2577 ],
       [419.64987],
       [401.6198 ],
       [393.82507],
       [388.5291 ],
       [393.27304],
       [410.07288],
       [425.4471 ],
       [452.74762],
       [498.32596],
       [527.7141 ],
       [517.89636],
       [492.57117],
       [452.51154]], dtype=float32)
```

```
y_test=scaler.inverse_transform(y_test)
```

```
plt.figure(figsize=(10,5))
plt.plot(df.index[-len(y_test):],y_test,label="Actual data")
plt.plot(df.index[-len(y_test):],predictions,label="predicted data")
plt.legend()
plt.show()
```



Start coding or generate with AI.

Aim: To remember the important tasks and forget the less important tasks using LSTM

IDE: Google Colab

Theory:

Countless learning tasks require dealing with sequential data. Image captioning, speech synthesis, and music generation all require that models produce outputs consisting of sequences. In other domains, such as time series prediction, video analysis, and musical information retrieval, a model must learn from inputs that are sequences. These demands often arise simultaneously: tasks such as translating passages of text from one natural language to another, engaging in dialogue, or controlling a robot, demand that models both ingest and output sequentially-structured data.

Recurrent neural networks (RNNs) are deep learning models that capture the dynamics of sequences via *recurrent* connections, which can be thought of as cycles in the network of nodes. This might seem counterintuitive at first. After all, it is the feedforward nature of neural networks that makes the order of computation unambiguous. However, recurrent edges are defined in a precise way that ensures that no such ambiguity can arise. Recurrent neural networks are *unrolled* across time steps (or sequence steps), with the *same* underlying parameters applied at each step. While the standard connections are applied *synchronously* to propagate each layer's activations to the subsequent layer *at the same time step*, the recurrent connections are *dynamic*, passing information across adjacent time steps. As the unfolded view reveals, RNNs can be thought of as feedforward neural networks where each layer's parameters (both conventional and recurrent) are shared across time steps. Like neural networks more broadly, RNNs have a long discipline-spanning history, originating as models of the brain popularized by cognitive scientists and subsequently adopted as practical modeling tools employed by the machine learning community.

The name of LSTM refers to the analogy that a standard RNN has both "long-term memory" and "short-term memory". The connection weights and biases in the network change once per episode of training, analogous to how physiological changes in synaptic strengths store long-term memories; the activation patterns in the network change once per time-step, analogous to how the moment-to-moment change in electric firing patterns in the brain store short-term memories. The LSTM architecture aims to provide a short-term memory for RNN that can last thousands of timesteps, thus "long short-term memory".

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications.

Methodology:

1. Load the basic libraries and packages
2. Load the dataset
3. Analyse the dataset
4. Apply LSTM Model
5. Apply the training over the dataset to minimize the loss
6. Observe the cost function vs iterations learning curve

Program (Code):

To be attached with

Results:

To be attached with

a. Model Summary
b. Training and Validation accuracy v/s epochs
c. Training and Validation loss v/s epochs

Observation and Result Analysis:

a. Nature of the dataset

____

____

____

____

b. During Training Process

____

____

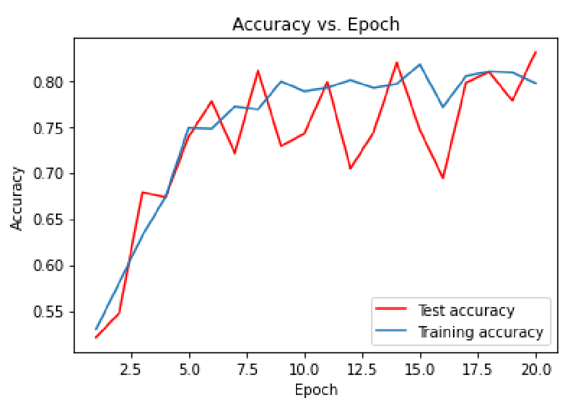
c. After the training Process

d. Observation over the Learning Curve

Post Lab Exercise:

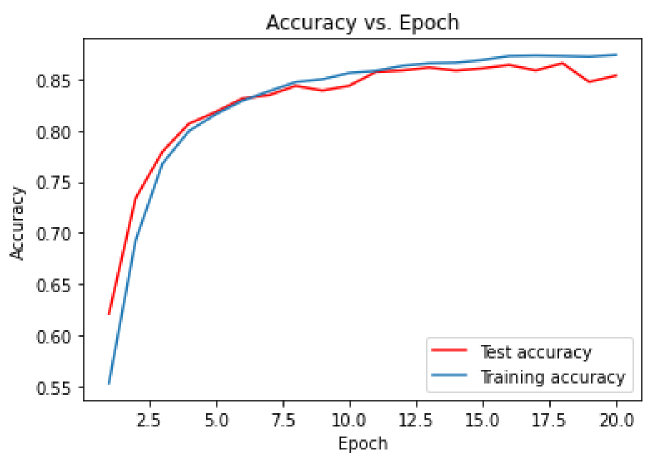
a. What is the requirement of LSTM over RNN?

b. Interpretation of graph



What is the meaning of the above graph? How can the graph be smoothened?

c.



What is the meaning of the above graph? How can the graph be smoothened?

Post Lab Activity

Obtain the prediction value of the TCS stock for the historical data obtained from the link mentioned as: https://in.investing.com/equities/tata-consultancy-services-historical-data

Select the Date Range from (01-01-2020 to 31-03-2024). Select the price range to be as “DAILY” stock values. Train your LSTM model to obtain the predicted value as close as the actual value. Paste the screenshot of the output.