Parker Christenson Assignment 3

Instructions

1. `Data Preparation`: Select a portion of the S&P 500 stock data for analysis. Perform preprocessing steps such as feature selection, normalization, and scaling to make the data suitable for the RNN model.
2. `Model Development`: Construct an RNN model using libraries like TensorFlow, or PyTorch. Incorporate LSTM units to address the vanishing gradient problem and improve memory retention across time steps.
3. `Training`: Train the RNN model on the prepared dataset, optimizing the loss function and choosing an appropriate optimizer to enhance model performance.
4. `Prediction`: Enable the model to forecast future stock prices, starting from a given initial stock price input.

Ensure the RNN model efficiently learns from the selected stock price data and accurately forecasts future trends.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
import visualkeras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Dense, Dropout, Fla
from tensorflow.keras.utils import model_to_dot
import matplotlib.pyplot as plt
```

In [ ]:
```
df = pd.read_csv('M3-AAPL.csv')
df.head()
```

Out[ ]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2019-05-28	44.730000	45.147499	44.477501	44.557499	43.002316	111792800
1	2019-05-29	44.105000	44.837502	44.000000	44.345001	42.797237	113924800
2	2019-05-30	44.487499	44.807499	44.167500	44.575001	43.019211	84873600
3	2019-05-31	44.057499	44.497501	43.747501	43.767502	42.239895	108174400
4	2019-06-03	43.900002	44.480000	42.567501	43.325001	41.812828	161584400

In [ ]:

```
# making a new column % change from the previous close to the next days open
df['on%_change'] = (df['Open'].shift(-1) - df['Close']) / df['Close'] * 100

df.head()
```

Out[ ]:

	Date	Open	High	Low	Close	Adj Close	Volume	on%_change
0	2019-05-28	44.730000	45.147499	44.477501	44.557499	43.002316	111792800	-1.015539
1	2019-05-29	44.105000	44.837502	44.000000	44.345001	42.797237	113924800	0.321339
2	2019-05-30	44.487499	44.807499	44.167500	44.575001	43.019211	84873600	-1.160969
3	2019-05-31	44.057499	44.497501	43.747501	43.767502	42.239895	108174400	0.302736
4	2019-06-03	43.900002	44.480000	42.567501	43.325001	41.812828	161584400	1.234853

In [ ]:

```
# Now getting daily % change in the stock price (open to close)

df ['daily_change'] = (df['Close'] - df['Open']) / df['Open'] * 100
df.head()
```

Out[ ]:

	Date	Open	High	Low	Close	Adj Close	Volume	on%_change
0	2019-05-28	44.730000	45.147499	44.477501	44.557499	43.002316	111792800	-1.015539
1	2019-05-29	44.105000	44.837502	44.000000	44.345001	42.797237	113924800	0.321339
2	2019-05-30	44.487499	44.807499	44.167500	44.575001	43.019211	84873600	-1.160969
3	2019-05-31	44.057499	44.497501	43.747501	43.767502	42.239895	108174400	0.302736
4	2019-06-03	43.900002	44.480000	42.567501	43.325001	41.812828	161584400	1.234853

In [ ]:
```
# if the close is greater than the open then the stock price increased and we will

df['green_red_day'] = np.where(df['Close'].shift(-1) > df['Open'], 1, 0)
df.head()
```

Out[ ]:

	Date	Open	High	Low	Close	Adj Close	Volume	on%_change
0	2019-05-28	44.730000	45.147499	44.477501	44.557499	43.002316	111792800	-1.015539
1	2019-05-29	44.105000	44.837502	44.000000	44.345001	42.797237	113924800	0.321339
2	2019-05-30	44.487499	44.807499	44.167500	44.575001	43.019211	84873600	-1.160969
3	2019-05-31	44.057499	44.497501	43.747501	43.767502	42.239895	108174400	0.302736
4	2019-06-03	43.900002	44.480000	42.567501	43.325001	41.812828	161584400	1.234853

In [ ]:
```
# getting the longest red and green streaks

def longest_streak(arr, value):
    max_streak = 0
    current_streak = 0

    for num in arr:
        if num == value:
            current_streak += 1
        else:
            if current_streak > max_streak:
                max_streak = current_streak
            current_streak = 0
```

```
    if current_streak > max_streak:
        max_streak = current_streak

    return max_streak

# calc
longest_red_streak = longest_streak(df['green_red_day'], 0)
longest_green_streak = longest_streak(df['green_red_day'], 1)

# print
print(f"longest streak of red days (0): {longest_red_streak}")
print(f"longest streak of green days (1): {longest_green_streak}")
```

```
longest streak of red days (0): 9
longest streak of green days (1): 17
```

In [ ]:

```
# now getting the date ranges of the longest red and green streaks

def longest_streak_range(arr, value):
    max_streak = 0
    current_streak = 0
    start = 0
    end = 0
    current_start = 0

    for i, num in enumerate(arr):
        if num == value:
            current_streak += 1
            if current_streak == 1:
                current_start = i
        else:
            if current_streak > max_streak:
                max_streak = current_streak
                start = current_start
                end = i
            current_streak = 0

    if current_streak > max_streak:
        max_streak = current_streak
        start = current_start
        end = len(arr)

    return (start, end)

# calc
longest_red_streak_range = longest_streak_range(df['green_red_day'], 0)
longest_green_streak_range = longest_streak_range(df['green_red_day'], 1)

# print
print(f"longest streak of red days (0) range: {longest_red_streak_range}")
print(f"longest streak of green days (1) range: {longest_green_streak_range}")
```

```
longest streak of red days (0) range: (819, 828)
longest streak of green days (1) range: (1114, 1131)
```

Model buildling using TensorFlow and netron

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

Out[ ]:

	Date	Open	High	Low	Close	Adj Close	Volume	on%_change
0	2019-05-28	44.730000	45.147499	44.477501	44.557499	43.002316	111792800	-1.015539
1	2019-05-29	44.105000	44.837502	44.000000	44.345001	42.797237	113924800	0.321339
2	2019-05-30	44.487499	44.807499	44.167500	44.575001	43.019211	84873600	-1.160969
3	2019-05-31	44.057499	44.497501	43.747501	43.767502	42.239895	108174400	0.302736
4	2019-06-03	43.900002	44.480000	42.567501	43.325001	41.812828	161584400	1.234853

Our target column is going to be the green_red_day column, because we are going to predict if the stock price will increase or decrease the next day, which will we we are going to take positions based off the predictions.

In [ ]:
```
# Converting the green_red_day values to a green or red 0 for red and 1 for green

df['green_red_day'] = np.where(df['green_red_day'] == 0, 'red', 'green')
df.head()
```

Out[ ]:

	Date	Open	High	Low	Close	Adj Close	Volume	on%_change
0	2019-05-28	44.730000	45.147499	44.477501	44.557499	43.002316	111792800	-1.015539
1	2019-05-29	44.105000	44.837502	44.000000	44.345001	42.797237	113924800	0.321339
2	2019-05-30	44.487499	44.807499	44.167500	44.575001	43.019211	84873600	-1.160969
3	2019-05-31	44.057499	44.497501	43.747501	43.767502	42.239895	108174400	0.302736
4	2019-06-03	43.900002	44.480000	42.567501	43.325001	41.812828	161584400	1.234853

In [ ]:
```
# Now we are going to scale the data using the StandardScaler

scaler = StandardScaler()

# scaling the data
scaled_data = scaler.fit_transform(df[['Open', 'High', 'Low', 'Close', 'Volume', 'o
scaled_data

# creating a new dataframe with the scaled data
df_scaled = pd.DataFrame(data=scaled_data, columns=['Open', 'High', 'Low', 'Close',

# adding the green_red_day column to the new dataframe
df_scaled['green_red_day'] = df['green_red_day']

df_scaled.head()
```

Out[ ]:

	Open	High	Low	Close	Volume	on%_change	daily_change	green_
0	-2.080578	-2.090787	-2.069417	-2.087329	0.293664	-0.824802	-0.327146	
1	-2.095202	-2.097989	-2.080673	-2.092301	0.334755	0.237435	0.284106	
2	-2.086252	-2.098686	-2.076724	-2.086920	-0.225165	-0.940356	0.055681	
3	-2.096313	-2.105888	-2.086625	-2.105813	0.223925	0.222653	-0.506336	
4	-2.099998	-2.106294	-2.114442	-2.116166	1.253326	0.963281	-0.934677	

In [ ]:
```
# Now we are going to make the architecture of the neural network with LTSM units a

model = tf.keras.Sequential([
    tf.keras.layers.LSTM(128, input_shape=(7, 8), return_sequences=True),
    tf.keras.layers.LSTM(128, return_sequences=True),
    tf.keras.layers.LSTM(128),
    tf.keras.layers.Dense(64),
    tf.keras.layers.Dense(32),
    tf.keras.layers.Dense(1, activation='sigmoid')
])



model.summary()
```

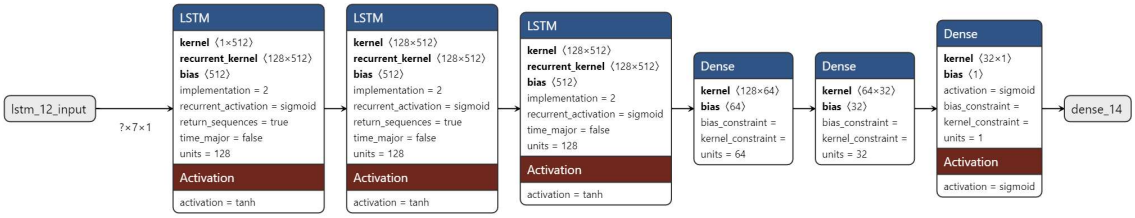
```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_18 (LSTM)              (None, 7, 128)            70144

 lstm_19 (LSTM)              (None, 7, 128)            131584

 lstm_20 (LSTM)              (None, 128)               131584

 dense_18 (Dense)            (None, 64)                8256

 dense_19 (Dense)            (None, 32)                2080

 dense_20 (Dense)            (None, 1)                 33

=================================================================
Total params: 343,681
Trainable params: 343,681
Non-trainable params: 0
_________________________________________________________________
```


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_18 (LSTM)              (None, 7, 128)            70144

 lstm_19 (LSTM)              (None, 7, 128)            131584

 lstm_20 (LSTM)              (None, 128)               131584

 dense_18 (Dense)            (None, 64)                8256

 dense_19 (Dense)            (None, 32)                2080

 dense_20 (Dense)            (None, 1)                 33

=================================================================
Total params: 343,681
Trainable params: 343,681
Non-trainable params: 0
_________________________________________________________________
```


In [ ]:
```
model.save('model.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to b
e built. `model.compile_metrics` will be empty until you train or evaluate the mode
l.
```

Here is the picture of the model



- So the shape of the input is wrong in the picture, it is actually 7 x 8 not 7 x 1

Model training

In [ ]:
```
# splitting the stock data into training and testing data
df_scaled.head()
```

Out[ ]:

	Open	High	Low	Close	Volume	on%_change	daily_change	green_
0	-2.080578	-2.090787	-2.069417	-2.087329	0.293664	-0.824802	-0.327146	
1	-2.095202	-2.097989	-2.080673	-2.092301	0.334755	0.237435	0.284106	
2	-2.086252	-2.098686	-2.076724	-2.086920	-0.225165	-0.940356	0.055681	
3	-2.096313	-2.105888	-2.086625	-2.105813	0.223925	0.222653	-0.506336	
4	-2.099998	-2.106294	-2.114442	-2.116166	1.253326	0.963281	-0.934677	

In [ ]:
```
# target col
df_scaled['green_red_day'] = np.where(df_scaled['Close'] > df_scaled['Open'], 1, 0)

# x & y
X = []
y = []

for i in range(7, df_scaled.shape[0]):
    X.append(df_scaled.iloc[i-7:i].values)
    y.append(df_scaled.iloc[i, -1])

X = np.array(X)
y = np.array(y)

# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# check the shape
print("Shape before reshaping:")
print(f"X_train: {X_train.shape}, X_test: {X_test.shape}")

print("Shape after reshaping (no reshaping needed):")
print(f"X_train: {X_train.shape}, X_test: {X_test.shape}")
```

```
# green and red days are balanced in the training and testing data
unique_train, counts_train = np.unique(y_train, return_counts=True)
unique_test, counts_test = np.unique(y_test, return_counts=True)

print("Training data balance:", dict(zip(unique_train, counts_train)))
print("Testing data balance:", dict(zip(unique_test, counts_test)))
```

```
Shape before reshaping:
X_train: (1001, 7, 8), X_test: (251, 7, 8)
Shape after reshaping (no reshaping needed):
X_train: (1001, 7, 8), X_test: (251, 7, 8)
Training data balance: {0: 488, 1: 513}
Testing data balance: {0: 131, 1: 120}
```

In [ ]:
```
# now we are going to compile the model

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [ ]:
```
# train the Model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_
```

```
Epoch 1/50
32/32 [==============================] - 7s 66ms/step - loss: 0.6946 - accuracy: 0.5
035 - val_loss: 0.7026 - val_accuracy: 0.4781
Epoch 2/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6954 - accuracy: 0.5
135 - val_loss: 0.6962 - val_accuracy: 0.4980
Epoch 3/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6925 - accuracy: 0.5
225 - val_loss: 0.6989 - val_accuracy: 0.4900
Epoch 4/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6941 - accuracy: 0.5
245 - val_loss: 0.6954 - val_accuracy: 0.4781
Epoch 5/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6929 - accuracy: 0.5
185 - val_loss: 0.6972 - val_accuracy: 0.4940
Epoch 6/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6915 - accuracy: 0.5
375 - val_loss: 0.6961 - val_accuracy: 0.4582
Epoch 7/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6929 - accuracy: 0.5
045 - val_loss: 0.6965 - val_accuracy: 0.4741
Epoch 8/50
32/32 [==============================] - 1s 16ms/step - loss: 0.6905 - accuracy: 0.5
455 - val_loss: 0.7054 - val_accuracy: 0.4582
Epoch 9/50
32/32 [==============================] - 1s 16ms/step - loss: 0.6912 - accuracy: 0.5
375 - val_loss: 0.7006 - val_accuracy: 0.4861
Epoch 10/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6898 - accuracy: 0.5
485 - val_loss: 0.7038 - val_accuracy: 0.4741
Epoch 11/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6892 - accuracy: 0.5
475 - val_loss: 0.6988 - val_accuracy: 0.4940
Epoch 12/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6891 - accuracy: 0.5
335 - val_loss: 0.7065 - val_accuracy: 0.4980
Epoch 13/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6875 - accuracy: 0.5
504 - val_loss: 0.7005 - val_accuracy: 0.4821
Epoch 14/50
32/32 [==============================] - 0s 16ms/step - loss: 0.6888 - accuracy: 0.5
405 - val_loss: 0.7122 - val_accuracy: 0.4741
Epoch 15/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6879 - accuracy: 0.5
425 - val_loss: 0.7016 - val_accuracy: 0.4701
Epoch 16/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6900 - accuracy: 0.5
205 - val_loss: 0.7040 - val_accuracy: 0.4661
Epoch 17/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6856 - accuracy: 0.5
504 - val_loss: 0.7448 - val_accuracy: 0.4622
Epoch 18/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6842 - accuracy: 0.5
485 - val_loss: 0.7100 - val_accuracy: 0.4940
Epoch 19/50
32/32 [==============================] - 1s 18ms/step - loss: 0.6888 - accuracy: 0.5
```

```
385 - val_loss: 0.7060 - val_accuracy: 0.4781
Epoch 20/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6857 - accuracy: 0.5
564 - val_loss: 0.7534 - val_accuracy: 0.4542
Epoch 21/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6851 - accuracy: 0.5
534 - val_loss: 0.7587 - val_accuracy: 0.4821
Epoch 22/50
32/32 [==============================] - 1s 16ms/step - loss: 0.6788 - accuracy: 0.5
554 - val_loss: 0.7424 - val_accuracy: 0.4701
Epoch 23/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6790 - accuracy: 0.5
584 - val_loss: 0.7233 - val_accuracy: 0.5020
Epoch 24/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6765 - accuracy: 0.5
704 - val_loss: 0.7991 - val_accuracy: 0.4701
Epoch 25/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6802 - accuracy: 0.5
744 - val_loss: 0.7299 - val_accuracy: 0.4980
Epoch 26/50
32/32 [==============================] - 1s 16ms/step - loss: 0.6770 - accuracy: 0.5
724 - val_loss: 0.7656 - val_accuracy: 0.4701
Epoch 27/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6754 - accuracy: 0.5
724 - val_loss: 0.7628 - val_accuracy: 0.4861
Epoch 28/50
32/32 [==============================] - 0s 16ms/step - loss: 0.6761 - accuracy: 0.5
614 - val_loss: 0.7539 - val_accuracy: 0.4661
Epoch 29/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6729 - accuracy: 0.5
784 - val_loss: 0.7477 - val_accuracy: 0.4980
Epoch 30/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6710 - accuracy: 0.5
784 - val_loss: 0.7521 - val_accuracy: 0.4821
Epoch 31/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6715 - accuracy: 0.5
824 - val_loss: 0.7552 - val_accuracy: 0.4582
Epoch 32/50
32/32 [==============================] - 1s 16ms/step - loss: 0.6674 - accuracy: 0.5
884 - val_loss: 0.7326 - val_accuracy: 0.4900
Epoch 33/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6692 - accuracy: 0.6
024 - val_loss: 0.7071 - val_accuracy: 0.5020
Epoch 34/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6681 - accuracy: 0.5
904 - val_loss: 0.8035 - val_accuracy: 0.4940
Epoch 35/50
32/32 [==============================] - 1s 17ms/step - loss: 0.6612 - accuracy: 0.5
834 - val_loss: 0.7526 - val_accuracy: 0.4980
Epoch 36/50
32/32 [==============================] - 1s 17ms/step - loss: 0.6512 - accuracy: 0.6
034 - val_loss: 0.7241 - val_accuracy: 0.4821
Epoch 37/50
32/32 [==============================] - 0s 15ms/step - loss: 0.6452 - accuracy: 0.6
074 - val_loss: 0.7870 - val_accuracy: 0.4900
Epoch 38/50
```

```
32/32 [==============================] - 1s 16ms/step - loss: 0.6412 - accuracy: 0.6
304 - val_loss: 0.8411 - val_accuracy: 0.5020
Epoch 39/50
32/32 [==============================] - 1s 20ms/step - loss: 0.6423 - accuracy: 0.6
344 - val_loss: 0.7291 - val_accuracy: 0.5259
Epoch 40/50
32/32 [==============================] - 1s 19ms/step - loss: 0.6280 - accuracy: 0.6
284 - val_loss: 0.7703 - val_accuracy: 0.5299
Epoch 41/50
32/32 [==============================] - 1s 16ms/step - loss: 0.6180 - accuracy: 0.6
344 - val_loss: 0.7727 - val_accuracy: 0.4980
Epoch 42/50
32/32 [==============================] - 0s 16ms/step - loss: 0.6072 - accuracy: 0.6
494 - val_loss: 0.8011 - val_accuracy: 0.5219
Epoch 43/50
32/32 [==============================] - 1s 16ms/step - loss: 0.5936 - accuracy: 0.6
563 - val_loss: 0.8262 - val_accuracy: 0.5219
Epoch 44/50
32/32 [==============================] - 0s 16ms/step - loss: 0.5815 - accuracy: 0.6
663 - val_loss: 0.9218 - val_accuracy: 0.5139
Epoch 45/50
32/32 [==============================] - 0s 15ms/step - loss: 0.5696 - accuracy: 0.6
693 - val_loss: 0.8313 - val_accuracy: 0.4940
Epoch 46/50
32/32 [==============================] - 1s 17ms/step - loss: 0.5642 - accuracy: 0.6
653 - val_loss: 0.9171 - val_accuracy: 0.5179
Epoch 47/50
32/32 [==============================] - 0s 16ms/step - loss: 0.5488 - accuracy: 0.6
863 - val_loss: 0.8715 - val_accuracy: 0.5219
Epoch 48/50
32/32 [==============================] - 1s 17ms/step - loss: 0.5282 - accuracy: 0.6
983 - val_loss: 0.9048 - val_accuracy: 0.5060
Epoch 49/50
32/32 [==============================] - 1s 16ms/step - loss: 0.5161 - accuracy: 0.7
083 - val_loss: 1.0891 - val_accuracy: 0.5339
Epoch 50/50
32/32 [==============================] - 0s 15ms/step - loss: 0.5218 - accuracy: 0.7
193 - val_loss: 0.9310 - val_accuracy: 0.5060
```

In [ ]:
```python
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
```

```
8/8 [==============================] - 0s 7ms/step - loss: 0.9310 - accuracy: 0.5060
Test Loss: 0.9309968948364258
Test Accuracy: 0.5059760808944702
```

In [ ]:
```python
# doing some predictions
unique_train, counts_train = np.unique(y_train, return_counts=True)
unique_test, counts_test = np.unique(y_test, return_counts=True)

print("Training data balance:", dict(zip(unique_train, counts_train)))
print("Testing data balance:", dict(zip(unique_test, counts_test)))
```

```
Training data balance: {0: 488, 1: 513}
Testing data balance: {0: 131, 1: 120}
```

In [ ]:
```
# Re normalizing and re-testing
from sklearn.preprocessing import MinMaxScaler # using MinMaxScaler to normalize th

if 'Date' in df.columns:
    dates = df['Date']
    df = df.drop(columns=['Date'])

df['green_red_day'] = np.where(df['Close'] > df['Open'], 1, 0)

target = df['green_red_day']
df = df.drop(columns=['green_red_day'])

scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)

df_scaled['green_red_day'] = target.values


X = []
y = []

for i in range(7, df_scaled.shape[0]):
    X.append(df_scaled.iloc[i-7:i].values)
    y.append(df_scaled.iloc[i, df_scaled.columns.get_loc('green_red_day')])

X = np.array(X)
y = np.array(y)

# split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# split checks
unique_train, counts_train = np.unique(y_train, return_counts=True)
unique_test, counts_test = np.unique(y_test, return_counts=True)

print("Training data balance:", dict(zip(unique_train, counts_train)))
print("Testing data balance:", dict(zip(unique_test, counts_test)))

# build the model
model = tf.keras.Sequential([
    tf.keras.layers.LSTM(64, input_shape=(X_train.shape[1], X_train.shape[2]), retu
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.LSTM(64, return_sequences=True),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
Training data balance: {0: 454, 1: 547}
Testing data balance: {0: 122, 1: 129}
```

In [ ]:
```
# setting the optimizer and compiling the model
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
```

```
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy']

# callbacks
callbacks = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5
    tf.keras.callbacks.ModelCheckpoint('best_model.h5', monitor='val_loss', save_be
]
```

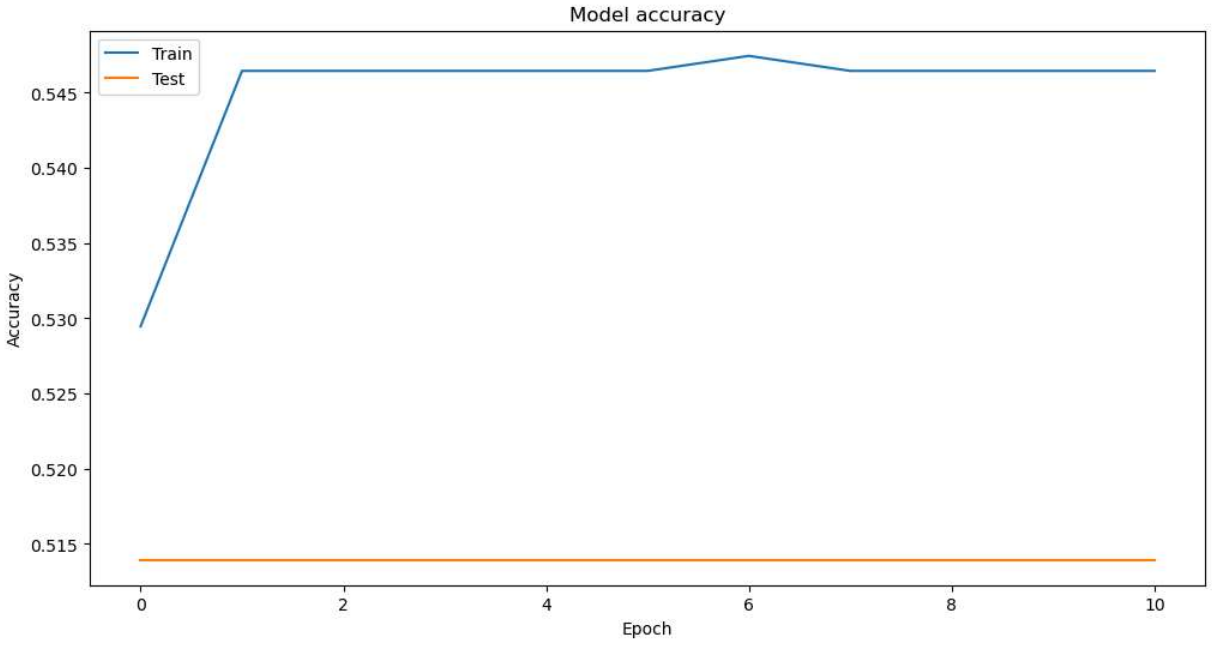
In [ ]:
```
# Now training the mdoel

history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X
```

```
Epoch 1/100
32/32 [==============================] - 7s 51ms/step - loss: 0.6918 - accuracy: 0.5
295 - val_loss: 0.6935 - val_accuracy: 0.5139 - lr: 0.0010
Epoch 2/100
32/32 [==============================] - 0s 13ms/step - loss: 0.6891 - accuracy: 0.5
465 - val_loss: 0.6937 - val_accuracy: 0.5139 - lr: 0.0010
Epoch 3/100
32/32 [==============================] - 0s 13ms/step - loss: 0.6898 - accuracy: 0.5
465 - val_loss: 0.6940 - val_accuracy: 0.5139 - lr: 0.0010
Epoch 4/100
32/32 [==============================] - 0s 14ms/step - loss: 0.6897 - accuracy: 0.5
465 - val_loss: 0.6938 - val_accuracy: 0.5139 - lr: 0.0010
Epoch 5/100
32/32 [==============================] - 0s 14ms/step - loss: 0.6888 - accuracy: 0.5
465 - val_loss: 0.6947 - val_accuracy: 0.5139 - lr: 0.0010
Epoch 6/100
32/32 [==============================] - 0s 13ms/step - loss: 0.6892 - accuracy: 0.5
465 - val_loss: 0.6972 - val_accuracy: 0.5139 - lr: 0.0010
Epoch 7/100
32/32 [==============================] - 0s 13ms/step - loss: 0.6881 - accuracy: 0.5
475 - val_loss: 0.6960 - val_accuracy: 0.5139 - lr: 2.0000e-04
Epoch 8/100
32/32 [==============================] - 0s 13ms/step - loss: 0.6884 - accuracy: 0.5
465 - val_loss: 0.6965 - val_accuracy: 0.5139 - lr: 2.0000e-04
Epoch 9/100
32/32 [==============================] - 0s 13ms/step - loss: 0.6869 - accuracy: 0.5
465 - val_loss: 0.6967 - val_accuracy: 0.5139 - lr: 2.0000e-04
Epoch 10/100
32/32 [==============================] - 0s 13ms/step - loss: 0.6874 - accuracy: 0.5
465 - val_loss: 0.6968 - val_accuracy: 0.5139 - lr: 2.0000e-04
Epoch 11/100
32/32 [==============================] - 0s 13ms/step - loss: 0.6867 - accuracy: 0.5
465 - val_loss: 0.6977 - val_accuracy: 0.5139 - lr: 2.0000e-04
```

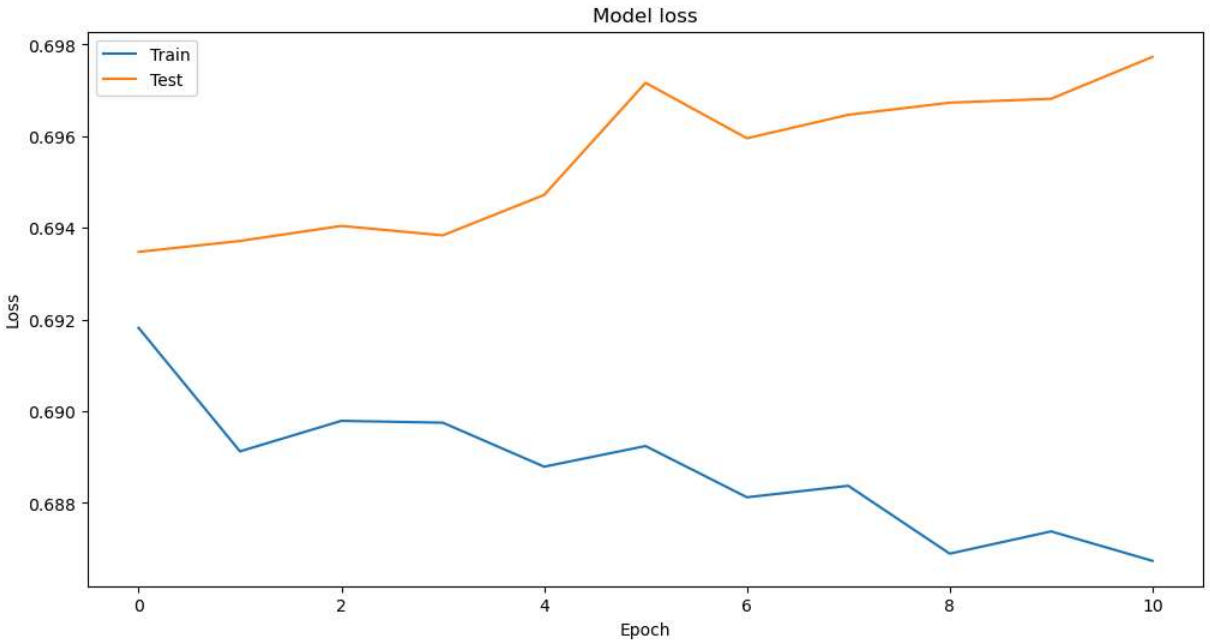
In [ ]:
```
# eval
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
```

```
8/8 [==============================] - 0s 6ms/step - loss: 0.6935 - accuracy: 0.5139
Test Loss: 0.6934756636619568
Test Accuracy: 0.5139442086219788
```

In [ ]:
```
# training and eval scores

plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



In [ ]:
```
# train and val loss
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



So I had it stop with early stopping because the model did not continue to improve, but I am going to run it for 100 epochs to see if it improves increasing the patience to 50

In [ ]:
```
# adjust
callbacks = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=20, restore_best_
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1
    tf.keras.callbacks.ModelCheckpoint('best_model.h5', monitor='val_loss', save_be
]

# re-run
history = model.fit(X_train, y_train, epochs=100, batch_size=100, validation_data=(
```

```
Epoch 1/100
11/11 [==============================] - 0s 28ms/step - loss: 0.6889 - accuracy: 0.5
465 - val_loss: 0.6932 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 2/100
11/11 [==============================] - 0s 22ms/step - loss: 0.6890 - accuracy: 0.5
465 - val_loss: 0.6931 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 3/100
11/11 [==============================] - 0s 22ms/step - loss: 0.6891 - accuracy: 0.5
465 - val_loss: 0.6931 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 4/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6893 - accuracy: 0.5
465 - val_loss: 0.6931 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 5/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6891 - accuracy: 0.5
465 - val_loss: 0.6931 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 6/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6890 - accuracy: 0.5
465 - val_loss: 0.6932 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 7/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6894 - accuracy: 0.5
465 - val_loss: 0.6932 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 8/100
11/11 [==============================] - 0s 18ms/step - loss: 0.6891 - accuracy: 0.5
465 - val_loss: 0.6932 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 9/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6891 - accuracy: 0.5
465 - val_loss: 0.6931 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 10/100
11/11 [==============================] - 0s 21ms/step - loss: 0.6891 - accuracy: 0.5
465 - val_loss: 0.6932 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 11/100
11/11 [==============================] - 0s 19ms/step - loss: 0.6886 - accuracy: 0.5
465 - val_loss: 0.6933 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 12/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6887 - accuracy: 0.5
465 - val_loss: 0.6933 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 13/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6888 - accuracy: 0.5
465 - val_loss: 0.6935 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 14/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6885 - accuracy: 0.5
465 - val_loss: 0.6936 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 15/100
11/11 [==============================] - 0s 18ms/step - loss: 0.6883 - accuracy: 0.5
465 - val_loss: 0.6935 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 16/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6889 - accuracy: 0.5
465 - val_loss: 0.6934 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 17/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6888 - accuracy: 0.5
465 - val_loss: 0.6932 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 18/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6891 - accuracy: 0.5
465 - val_loss: 0.6931 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 19/100
11/11 [==============================] - 0s 21ms/step - loss: 0.6890 - accuracy: 0.5
```

```
465 - val_loss: 0.6931 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 20/100
11/11 [==============================] - 0s 21ms/step - loss: 0.6892 - accuracy: 0.5
465 - val_loss: 0.6931 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 21/100
11/11 [==============================] - 0s 21ms/step - loss: 0.6894 - accuracy: 0.5
465 - val_loss: 0.6931 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 22/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6895 - accuracy: 0.5
465 - val_loss: 0.6931 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 23/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6893 - accuracy: 0.5
465 - val_loss: 0.6931 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 24/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6893 - accuracy: 0.5
465 - val_loss: 0.6931 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 25/100
11/11 [==============================] - 0s 18ms/step - loss: 0.6893 - accuracy: 0.5
465 - val_loss: 0.6932 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 26/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6890 - accuracy: 0.5
465 - val_loss: 0.6933 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 27/100
11/11 [==============================] - 0s 22ms/step - loss: 0.6891 - accuracy: 0.5
465 - val_loss: 0.6934 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 28/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6884 - accuracy: 0.5
465 - val_loss: 0.6936 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 29/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6878 - accuracy: 0.5
465 - val_loss: 0.6935 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 30/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6886 - accuracy: 0.5
465 - val_loss: 0.6935 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 31/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6881 - accuracy: 0.5
465 - val_loss: 0.6936 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 32/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6883 - accuracy: 0.5
465 - val_loss: 0.6937 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 33/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6883 - accuracy: 0.5
465 - val_loss: 0.6937 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 34/100
11/11 [==============================] - 0s 18ms/step - loss: 0.6880 - accuracy: 0.5
465 - val_loss: 0.6939 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 35/100
11/11 [==============================] - 0s 18ms/step - loss: 0.6881 - accuracy: 0.5
465 - val_loss: 0.6938 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 36/100
11/11 [==============================] - 0s 21ms/step - loss: 0.6880 - accuracy: 0.5
465 - val_loss: 0.6937 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 37/100
11/11 [==============================] - 0s 21ms/step - loss: 0.6882 - accuracy: 0.5
465 - val_loss: 0.6940 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 38/100
```

```
11/11 [==============================] - 0s 17ms/step - loss: 0.6883 - accuracy: 0.5
465 - val_loss: 0.6942 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 39/100
11/11 [==============================] - 0s 18ms/step - loss: 0.6881 - accuracy: 0.5
465 - val_loss: 0.6941 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 40/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6882 - accuracy: 0.5
465 - val_loss: 0.6939 - val_accuracy: 0.5139 - lr: 1.0000e-04
Epoch 41/100
11/11 [==============================] - 0s 17ms/step - loss: 0.6881 - accuracy: 0.5
465 - val_loss: 0.6937 - val_accuracy: 0.5139 - lr: 1.0000e-04
```

I ran the model about 5 times, and the best accuracy was 54ish. So we are going to rebuild the model, and change some other things about it, and just try predicting the close price based off the previous 4 days as the input.

In [ ]:
```
# reload the data

df = pd.read_csv('M3-AAPL.csv')

df['on%_change'] = (df['Open'].shift(-1) - df['Close']) / df['Close'] * 100

df.head()

df ['daily_change'] = (df['Close'] - df['Open']) / df['Open'] * 100
df.head()
```

Out[ ]:

	Date	Open	High	Low	Close	Adj Close	Volume	on%_change
0	2019-05-28	44.730000	45.147499	44.477501	44.557499	43.002316	111792800	-1.015539
1	2019-05-29	44.105000	44.837502	44.000000	44.345001	42.797237	113924800	0.321339
2	2019-05-30	44.487499	44.807499	44.167500	44.575001	43.019211	84873600	-1.160969
3	2019-05-31	44.057499	44.497501	43.747501	43.767502	42.239895	108174400	0.302736
4	2019-06-03	43.900002	44.480000	42.567501	43.325001	41.812828	161584400	1.234853

In [ ]:
```
# Re normalizing and re-testing using MinMaxScaler

if 'Date' in df.columns:
    dates = df['Date']
    df = df.drop(columns=['Date'])
```

```
df['green_red_day'] = np.where(df['Close'] > df['Open'], 1, 0)

target = df['green_red_day']
df = df.drop(columns=['green_red_day'])

scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)

df_scaled['green_red_day'] = target.values

X = []
y = []

for i in range(7, df_scaled.shape[0]):
    X.append(df_scaled.iloc[i-7:i].values)
    y.append(df_scaled.iloc[i, df_scaled.columns.get_loc('green_red_day')])

X = np.array(X)
y = np.array(y)

# split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# split checks
unique_train, counts_train = np.unique(y_train, return_counts=True)
unique_test, counts_test = np.unique(y_test, return_counts=True)

print("Training data balance:", dict(zip(unique_train, counts_train)))
print("Testing data balance:", dict(zip(unique_test, counts_test)))


# building the model again

model = tf.keras.Sequential([
    tf.keras.layers.LSTM(64, input_shape=(X_train.shape[1], X_train.shape[2]), retu
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.LSTM(64, return_sequences=True),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1)  # No activation function this time because we are goi
])


# setting the optimizer and compiling the model
o=optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mae'])  # s
```

```
Training data balance: {0: 454, 1: 547}
Testing data balance: {0: 122, 1: 129}
```

In [ ]:

```
# training not using callbacks
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X
```

```
Epoch 1/100
32/32 [==============================] - 7s 76ms/step - loss: 0.3162 - mae: 0.5104 -
val_loss: 0.2493 - val_mae: 0.4962
Epoch 2/100
32/32 [==============================] - 1s 28ms/step - loss: 0.2536 - mae: 0.4944 -
val_loss: 0.2506 - val_mae: 0.4984
Epoch 3/100
32/32 [==============================] - 1s 25ms/step - loss: 0.2527 - mae: 0.4954 -
val_loss: 0.2509 - val_mae: 0.5003
Epoch 4/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2530 - mae: 0.4983 -
val_loss: 0.2531 - val_mae: 0.4989
Epoch 5/100
32/32 [==============================] - 1s 27ms/step - loss: 0.2521 - mae: 0.4949 -
val_loss: 0.2530 - val_mae: 0.4988
Epoch 6/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2571 - mae: 0.4998 -
val_loss: 0.2525 - val_mae: 0.5012
Epoch 7/100
32/32 [==============================] - 1s 27ms/step - loss: 0.2482 - mae: 0.4937 -
val_loss: 0.2591 - val_mae: 0.4968
Epoch 8/100
32/32 [==============================] - 1s 25ms/step - loss: 0.2524 - mae: 0.4973 -
val_loss: 0.2519 - val_mae: 0.4998
Epoch 9/100
32/32 [==============================] - 1s 25ms/step - loss: 0.2527 - mae: 0.4951 -
val_loss: 0.2561 - val_mae: 0.4984
Epoch 10/100
32/32 [==============================] - 1s 25ms/step - loss: 0.2507 - mae: 0.4964 -
val_loss: 0.2658 - val_mae: 0.4965
Epoch 11/100
32/32 [==============================] - 1s 25ms/step - loss: 0.2514 - mae: 0.4942 -
val_loss: 0.2581 - val_mae: 0.4979
Epoch 12/100
32/32 [==============================] - 1s 28ms/step - loss: 0.2505 - mae: 0.4919 -
val_loss: 0.2628 - val_mae: 0.4974
Epoch 13/100
32/32 [==============================] - 1s 27ms/step - loss: 0.2521 - mae: 0.4945 -
val_loss: 0.2518 - val_mae: 0.4997
Epoch 14/100
32/32 [==============================] - 1s 27ms/step - loss: 0.2482 - mae: 0.4938 -
val_loss: 0.2504 - val_mae: 0.4998
Epoch 15/100
32/32 [==============================] - 1s 25ms/step - loss: 0.2517 - mae: 0.4983 -
val_loss: 0.2509 - val_mae: 0.5000
Epoch 16/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2495 - mae: 0.4953 -
val_loss: 0.2519 - val_mae: 0.4994
Epoch 17/100
32/32 [==============================] - 1s 27ms/step - loss: 0.2506 - mae: 0.4959 -
val_loss: 0.2569 - val_mae: 0.4982
Epoch 18/100
32/32 [==============================] - 1s 25ms/step - loss: 0.2487 - mae: 0.4943 -
val_loss: 0.2575 - val_mae: 0.4979
Epoch 19/100
32/32 [==============================] - 1s 24ms/step - loss: 0.2505 - mae: 0.4952 -
```

```
val_loss: 0.2565 - val_mae: 0.4983
Epoch 20/100
32/32 [==============================] - 1s 27ms/step - loss: 0.2501 - mae: 0.4948 -
val_loss: 0.2521 - val_mae: 0.4999
Epoch 21/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2470 - mae: 0.4912 -
val_loss: 0.2508 - val_mae: 0.5004
Epoch 22/100
32/32 [==============================] - 1s 25ms/step - loss: 0.2501 - mae: 0.4979 -
val_loss: 0.2528 - val_mae: 0.4991
Epoch 23/100
32/32 [==============================] - 1s 25ms/step - loss: 0.2483 - mae: 0.4924 -
val_loss: 0.2534 - val_mae: 0.5018
Epoch 24/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2484 - mae: 0.4919 -
val_loss: 0.2568 - val_mae: 0.4983
Epoch 25/100
32/32 [==============================] - 1s 25ms/step - loss: 0.2494 - mae: 0.4968 -
val_loss: 0.2539 - val_mae: 0.4987
Epoch 26/100
32/32 [==============================] - 1s 28ms/step - loss: 0.2478 - mae: 0.4939 -
val_loss: 0.2513 - val_mae: 0.5001
Epoch 27/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2477 - mae: 0.4951 -
val_loss: 0.2531 - val_mae: 0.4992
Epoch 28/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2487 - mae: 0.4950 -
val_loss: 0.2564 - val_mae: 0.4982
Epoch 29/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2476 - mae: 0.4898 -
val_loss: 0.2525 - val_mae: 0.5014
Epoch 30/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2486 - mae: 0.4948 -
val_loss: 0.2514 - val_mae: 0.5000
Epoch 31/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2474 - mae: 0.4935 -
val_loss: 0.2526 - val_mae: 0.4998
Epoch 32/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2477 - mae: 0.4947 -
val_loss: 0.2517 - val_mae: 0.4999
Epoch 33/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2500 - mae: 0.4965 -
val_loss: 0.2547 - val_mae: 0.4989
Epoch 34/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2481 - mae: 0.4941 -
val_loss: 0.2519 - val_mae: 0.4995
Epoch 35/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2472 - mae: 0.4931 -
val_loss: 0.2522 - val_mae: 0.4995
Epoch 36/100
32/32 [==============================] - 1s 28ms/step - loss: 0.2473 - mae: 0.4947 -
val_loss: 0.2513 - val_mae: 0.4996
Epoch 37/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2480 - mae: 0.4936 -
val_loss: 0.2525 - val_mae: 0.4998
Epoch 38/100
```

```
32/32 [==============================] - 1s 26ms/step - loss: 0.2477 - mae: 0.4953 -
val_loss: 0.2516 - val_mae: 0.4995
Epoch 39/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2492 - mae: 0.4961 -
val_loss: 0.2519 - val_mae: 0.4999
Epoch 40/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2475 - mae: 0.4950 -
val_loss: 0.2508 - val_mae: 0.5000
Epoch 41/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2473 - mae: 0.4935 -
val_loss: 0.2514 - val_mae: 0.5004
Epoch 42/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2484 - mae: 0.4944 -
val_loss: 0.2554 - val_mae: 0.4990
Epoch 43/100
32/32 [==============================] - 1s 25ms/step - loss: 0.2488 - mae: 0.4964 -
val_loss: 0.2532 - val_mae: 0.4990
Epoch 44/100
32/32 [==============================] - 1s 26ms/step - loss: 0.2472 - mae: 0.4938 -
val_loss: 0.2511 - val_mae: 0.5002
Epoch 45/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2481 - mae: 0.4959 -
val_loss: 0.2534 - val_mae: 0.4989
Epoch 46/100
32/32 [==============================] - 1s 30ms/step - loss: 0.2510 - mae: 0.4974 -
val_loss: 0.2632 - val_mae: 0.4971
Epoch 47/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2479 - mae: 0.4920 -
val_loss: 0.2511 - val_mae: 0.5003
Epoch 48/100
32/32 [==============================] - 1s 30ms/step - loss: 0.2481 - mae: 0.4932 -
val_loss: 0.2510 - val_mae: 0.5005
Epoch 49/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2504 - mae: 0.4981 -
val_loss: 0.2513 - val_mae: 0.4998
Epoch 50/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2469 - mae: 0.4939 -
val_loss: 0.2519 - val_mae: 0.5000
Epoch 51/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2476 - mae: 0.4953 -
val_loss: 0.2546 - val_mae: 0.4990
Epoch 52/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2483 - mae: 0.4936 -
val_loss: 0.2523 - val_mae: 0.4997
Epoch 53/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2471 - mae: 0.4933 -
val_loss: 0.2522 - val_mae: 0.5003
Epoch 54/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2476 - mae: 0.4941 -
val_loss: 0.2522 - val_mae: 0.5008
Epoch 55/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2477 - mae: 0.4938 -
val_loss: 0.2524 - val_mae: 0.4997
Epoch 56/100
32/32 [==============================] - 1s 31ms/step - loss: 0.2486 - mae: 0.4954 -
val_loss: 0.2511 - val_mae: 0.5007
```

```
Epoch 57/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2471 - mae: 0.4950 -
val_loss: 0.2546 - val_mae: 0.4990
Epoch 58/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2482 - mae: 0.4916 -
val_loss: 0.2513 - val_mae: 0.5009
Epoch 59/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2477 - mae: 0.4947 -
val_loss: 0.2541 - val_mae: 0.4994
Epoch 60/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2480 - mae: 0.4933 -
val_loss: 0.2518 - val_mae: 0.5001
Epoch 61/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2477 - mae: 0.4943 -
val_loss: 0.2522 - val_mae: 0.5002
Epoch 62/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2469 - mae: 0.4945 -
val_loss: 0.2516 - val_mae: 0.5003
Epoch 63/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2479 - mae: 0.4936 -
val_loss: 0.2515 - val_mae: 0.5006
Epoch 64/100
32/32 [==============================] - 1s 31ms/step - loss: 0.2480 - mae: 0.4929 -
val_loss: 0.2523 - val_mae: 0.5000
Epoch 65/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2482 - mae: 0.4948 -
val_loss: 0.2532 - val_mae: 0.4993
Epoch 66/100
32/32 [==============================] - 1s 30ms/step - loss: 0.2476 - mae: 0.4949 -
val_loss: 0.2512 - val_mae: 0.5002
Epoch 67/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2476 - mae: 0.4942 -
val_loss: 0.2521 - val_mae: 0.4996
Epoch 68/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2487 - mae: 0.4958 -
val_loss: 0.2526 - val_mae: 0.4996
Epoch 69/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2484 - mae: 0.4949 -
val_loss: 0.2527 - val_mae: 0.4995
Epoch 70/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2484 - mae: 0.4948 -
val_loss: 0.2517 - val_mae: 0.5001
Epoch 71/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2495 - mae: 0.4965 -
val_loss: 0.2532 - val_mae: 0.4998
Epoch 72/100
32/32 [==============================] - 1s 31ms/step - loss: 0.2465 - mae: 0.4912 -
val_loss: 0.2527 - val_mae: 0.5006
Epoch 73/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2480 - mae: 0.4955 -
val_loss: 0.2535 - val_mae: 0.4998
Epoch 74/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2473 - mae: 0.4945 -
val_loss: 0.2531 - val_mae: 0.5001
Epoch 75/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2468 - mae: 0.4936 -
```

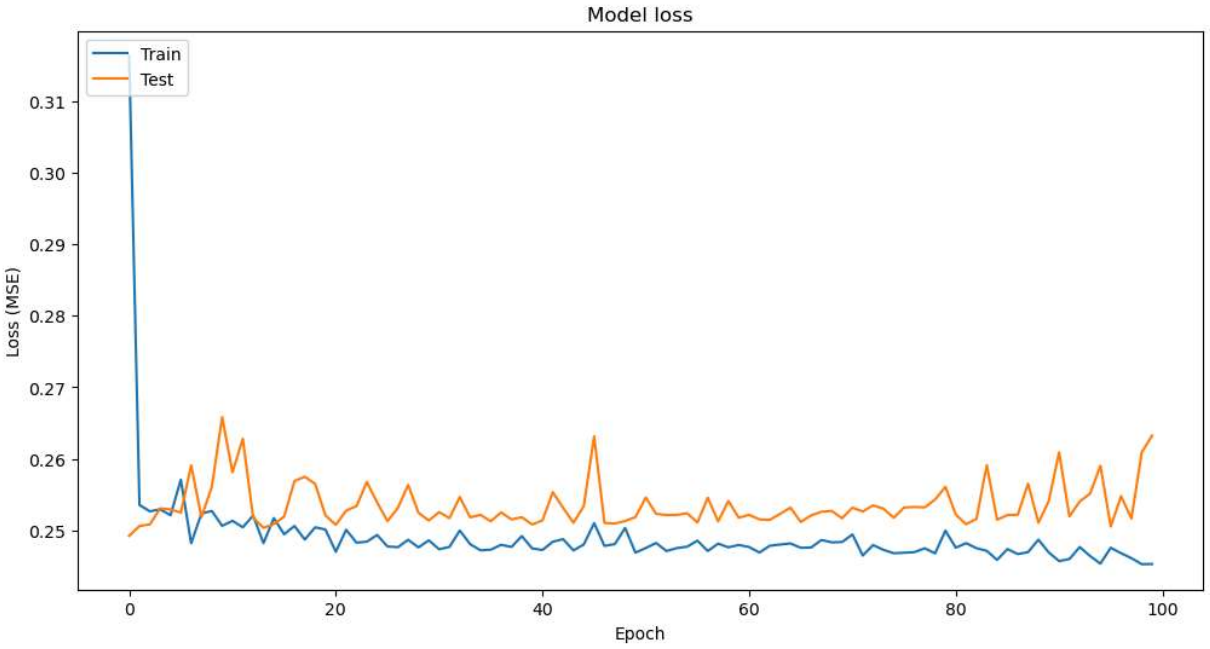
```
val_loss: 0.2518 - val_mae: 0.4999
Epoch 76/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2469 - mae: 0.4936 -
val_loss: 0.2532 - val_mae: 0.5005
Epoch 77/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2470 - mae: 0.4935 -
val_loss: 0.2533 - val_mae: 0.5003
Epoch 78/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2475 - mae: 0.4927 -
val_loss: 0.2532 - val_mae: 0.5007
Epoch 79/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2468 - mae: 0.4921 -
val_loss: 0.2543 - val_mae: 0.4999
Epoch 80/100
32/32 [==============================] - 1s 31ms/step - loss: 0.2500 - mae: 0.4951 -
val_loss: 0.2561 - val_mae: 0.4976
Epoch 81/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2476 - mae: 0.4944 -
val_loss: 0.2522 - val_mae: 0.4998
Epoch 82/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2482 - mae: 0.4954 -
val_loss: 0.2509 - val_mae: 0.5001
Epoch 83/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2475 - mae: 0.4944 -
val_loss: 0.2516 - val_mae: 0.5005
Epoch 84/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2472 - mae: 0.4942 -
val_loss: 0.2591 - val_mae: 0.5016
Epoch 85/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2459 - mae: 0.4914 -
val_loss: 0.2515 - val_mae: 0.4999
Epoch 86/100
32/32 [==============================] - 1s 28ms/step - loss: 0.2474 - mae: 0.4938 -
val_loss: 0.2522 - val_mae: 0.4987
Epoch 87/100
32/32 [==============================] - 1s 31ms/step - loss: 0.2467 - mae: 0.4922 -
val_loss: 0.2522 - val_mae: 0.5001
Epoch 88/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2470 - mae: 0.4934 -
val_loss: 0.2565 - val_mae: 0.5022
Epoch 89/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2487 - mae: 0.4926 -
val_loss: 0.2511 - val_mae: 0.4997
Epoch 90/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2469 - mae: 0.4936 -
val_loss: 0.2541 - val_mae: 0.5000
Epoch 91/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2457 - mae: 0.4897 -
val_loss: 0.2609 - val_mae: 0.5058
Epoch 92/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2460 - mae: 0.4913 -
val_loss: 0.2520 - val_mae: 0.4998
Epoch 93/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2477 - mae: 0.4923 -
val_loss: 0.2541 - val_mae: 0.5010
Epoch 94/100
```

```
32/32 [==============================] - 1s 29ms/step - loss: 0.2464 - mae: 0.4931 -
val_loss: 0.2552 - val_mae: 0.5009
Epoch 95/100
32/32 [==============================] - 1s 32ms/step - loss: 0.2454 - mae: 0.4903 -
val_loss: 0.2591 - val_mae: 0.5042
Epoch 96/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2476 - mae: 0.4902 -
val_loss: 0.2506 - val_mae: 0.4997
Epoch 97/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2468 - mae: 0.4928 -
val_loss: 0.2548 - val_mae: 0.5013
Epoch 98/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2461 - mae: 0.4912 -
val_loss: 0.2517 - val_mae: 0.4997
Epoch 99/100
32/32 [==============================] - 1s 31ms/step - loss: 0.2453 - mae: 0.4925 -
val_loss: 0.2609 - val_mae: 0.5031
Epoch 100/100
32/32 [==============================] - 1s 29ms/step - loss: 0.2453 - mae: 0.4884 -
val_loss: 0.2633 - val_mae: 0.5029
```

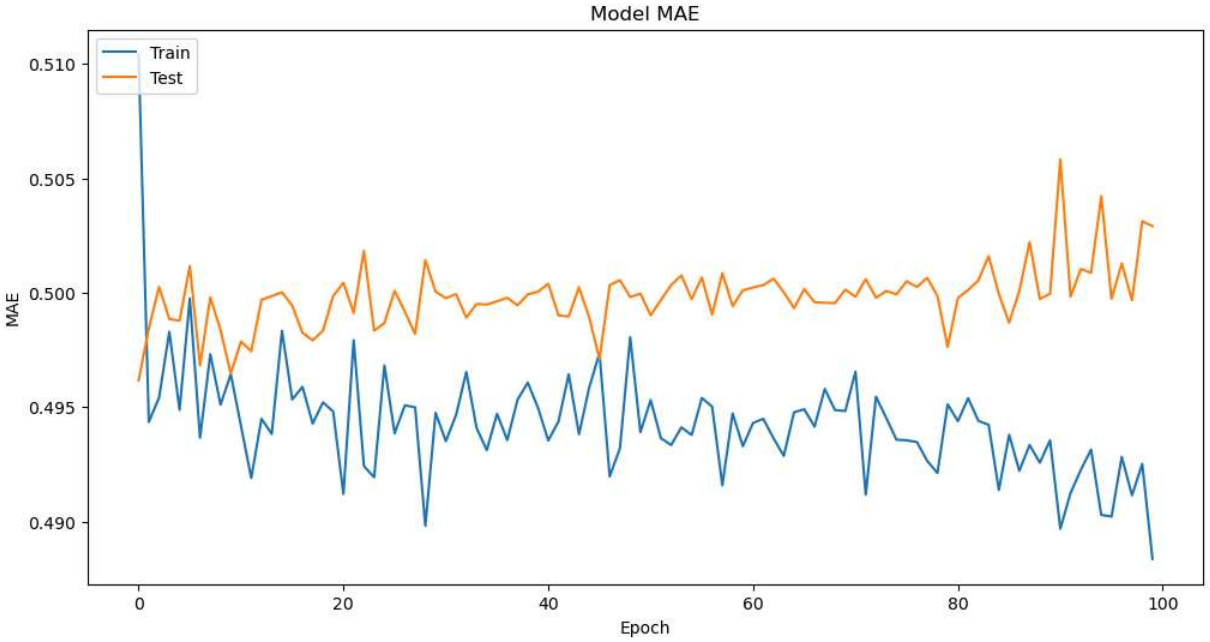
In [ ]:
```python
# Evaluate the model
loss, mae = model.evaluate(X_test, y_test)
print(f"Test Loss (MSE): {loss}")
print(f"Test MAE: {mae}")
```

```
8/8 [==============================] - 0s 9ms/step - loss: 0.2633 - mae: 0.5029
Test Loss (MSE): 0.2632591426372528
Test MAE: 0.5029081702232361
```

In [ ]:
```python
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss (MSE)')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



In [ ]:
```
plt.figure(figsize=(12, 6))
plt.plot(history.history['mae'])
plt.plot(history.history['val_mae'])
plt.title('Model MAE')
plt.ylabel('MAE')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



Attempting to build the model one more time

In [ ]:
```
# reload the data

df = pd.read_csv('M3-AAPL.csv')
```

```
df['on%_change'] = (df['Open'].shift(-1) - df['Close']) / df['Close'] * 100

df.head()
```

Out[ ]:

	Date	Open	High	Low	Close	Adj Close	Volume	on%_change
0	2019-05-28	44.730000	45.147499	44.477501	44.557499	43.002316	111792800	-1.015539
1	2019-05-29	44.105000	44.837502	44.000000	44.345001	42.797237	113924800	0.321339
2	2019-05-30	44.487499	44.807499	44.167500	44.575001	43.019211	84873600	-1.160969
3	2019-05-31	44.057499	44.497501	43.747501	43.767502	42.239895	108174400	0.302736
4	2019-06-03	43.900002	44.480000	42.567501	43.325001	41.812828	161584400	1.234853

In [ ]:
```
# col selection
data = df[['Open', 'High', 'Low', 'Close', 'Volume', 'on%_change']].values

# normalize data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

# sequences
def create_sequences(data, seq_length):
    sequences = []
    for i in range(len(data) - seq_length):
        sequences.append(data[i:i + seq_length])
    return np.array(sequences)

seq_length = 7
sequences = create_sequences(scaled_data, seq_length)

# split
X = sequences[:, :-1, :]
y = sequences[:, -1, :]

# split to training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [ ]:
```
def build_model(input_shape):
    model = Sequential()

    # conv layer
    model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=inpu
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.2))

    # LSTM
    model.add(LSTM(50, return_sequences=True))
```

```
    model.add(Dropout(0.2))
    model.add(LSTM(50))
    model.add(Dropout(0.2))

    # connected layers
    model.add(Dense(25, activation='relu'))
    model.add(Dense(y_train.shape[1]))  # Output layer

    # complie
    model.compile(optimizer='adam', loss='mean_squared_error')

    return model

input_shape = (seq_length - 1, X_train.shape[2])
model = build_model(input_shape)

# Model summary
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 5, 64)             832

 max_pooling1d (MaxPooling1D  (None, 2, 64)            0
 )

 dropout_12 (Dropout)        (None, 2, 64)             0

 lstm_33 (LSTM)              (None, 2, 50)             23000

 dropout_13 (Dropout)        (None, 2, 50)             0

 lstm_34 (LSTM)              (None, 50)                20200

 dropout_14 (Dropout)        (None, 50)                0

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 5, 64)             832

 max_pooling1d (MaxPooling1D  (None, 2, 64)            0
 )

 dropout_12 (Dropout)        (None, 2, 64)             0

 lstm_33 (LSTM)              (None, 2, 50)             23000

 dropout_13 (Dropout)        (None, 2, 50)             0

 lstm_34 (LSTM)              (None, 50)                20200

 dropout_14 (Dropout)        (None, 50)                0

 dense_33 (Dense)            (None, 25)                1275

 dense_34 (Dense)            (None, 6)                 156

=================================================================
Total params: 45,463
Trainable params: 45,463
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
# train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_
```

```
Epoch 1/50
32/32 [==============================] - 6s 64ms/step - loss: 0.1792 - val_loss: 0.0
329
Epoch 2/50
32/32 [==============================] - 1s 22ms/step - loss: 0.0281 - val_loss: 0.0
138
Epoch 3/50
32/32 [==============================] - 1s 25ms/step - loss: 0.0169 - val_loss: 0.0
045
Epoch 4/50
32/32 [==============================] - 1s 26ms/step - loss: 0.0122 - val_loss: 0.0
043
Epoch 5/50
32/32 [==============================] - 1s 26ms/step - loss: 0.0113 - val_loss: 0.0
034
Epoch 6/50
32/32 [==============================] - 1s 27ms/step - loss: 0.0100 - val_loss: 0.0
036
Epoch 7/50
32/32 [==============================] - 1s 31ms/step - loss: 0.0092 - val_loss: 0.0
034
Epoch 8/50
32/32 [==============================] - 1s 25ms/step - loss: 0.0090 - val_loss: 0.0
034
Epoch 9/50
32/32 [==============================] - 1s 24ms/step - loss: 0.0084 - val_loss: 0.0
030
Epoch 10/50
32/32 [==============================] - 1s 26ms/step - loss: 0.0078 - val_loss: 0.0
031
Epoch 11/50
32/32 [==============================] - 1s 22ms/step - loss: 0.0079 - val_loss: 0.0
032
Epoch 12/50
32/32 [==============================] - 1s 30ms/step - loss: 0.0075 - val_loss: 0.0
031
Epoch 13/50
32/32 [==============================] - 1s 27ms/step - loss: 0.0075 - val_loss: 0.0
030
Epoch 14/50
32/32 [==============================] - 1s 24ms/step - loss: 0.0070 - val_loss: 0.0
029
Epoch 15/50
32/32 [==============================] - 1s 24ms/step - loss: 0.0070 - val_loss: 0.0
028
Epoch 16/50
32/32 [==============================] - 1s 32ms/step - loss: 0.0067 - val_loss: 0.0
027
Epoch 17/50
32/32 [==============================] - 1s 23ms/step - loss: 0.0067 - val_loss: 0.0
028
Epoch 18/50
32/32 [==============================] - 1s 36ms/step - loss: 0.0064 - val_loss: 0.0
029
Epoch 19/50
32/32 [==============================] - 1s 28ms/step - loss: 0.0061 - val_loss: 0.0
```

```
027
Epoch 20/50
32/32 [==============================] - 1s 27ms/step - loss: 0.0060 - val_loss: 0.0
026
Epoch 21/50
32/32 [==============================] - 1s 25ms/step - loss: 0.0058 - val_loss: 0.0
027
Epoch 22/50
32/32 [==============================] - 1s 26ms/step - loss: 0.0054 - val_loss: 0.0
029
Epoch 23/50
32/32 [==============================] - 1s 26ms/step - loss: 0.0055 - val_loss: 0.0
026
Epoch 24/50
32/32 [==============================] - 1s 30ms/step - loss: 0.0054 - val_loss: 0.0
032
Epoch 25/50
32/32 [==============================] - 1s 31ms/step - loss: 0.0053 - val_loss: 0.0
027
Epoch 26/50
32/32 [==============================] - 1s 24ms/step - loss: 0.0052 - val_loss: 0.0
024
Epoch 27/50
32/32 [==============================] - 1s 24ms/step - loss: 0.0049 - val_loss: 0.0
024
Epoch 28/50
32/32 [==============================] - 1s 22ms/step - loss: 0.0049 - val_loss: 0.0
024
Epoch 29/50
32/32 [==============================] - 1s 28ms/step - loss: 0.0050 - val_loss: 0.0
026
Epoch 30/50
32/32 [==============================] - 1s 27ms/step - loss: 0.0048 - val_loss: 0.0
025
Epoch 31/50
32/32 [==============================] - 1s 24ms/step - loss: 0.0046 - val_loss: 0.0
029
Epoch 32/50
32/32 [==============================] - 1s 26ms/step - loss: 0.0048 - val_loss: 0.0
028
Epoch 33/50
32/32 [==============================] - 1s 30ms/step - loss: 0.0045 - val_loss: 0.0
023
Epoch 34/50
32/32 [==============================] - 1s 28ms/step - loss: 0.0044 - val_loss: 0.0
023
Epoch 35/50
32/32 [==============================] - 1s 38ms/step - loss: 0.0044 - val_loss: 0.0
023
Epoch 36/50
32/32 [==============================] - 1s 30ms/step - loss: 0.0045 - val_loss: 0.0
023
Epoch 37/50
32/32 [==============================] - 1s 31ms/step - loss: 0.0043 - val_loss: 0.0
024
Epoch 38/50
```

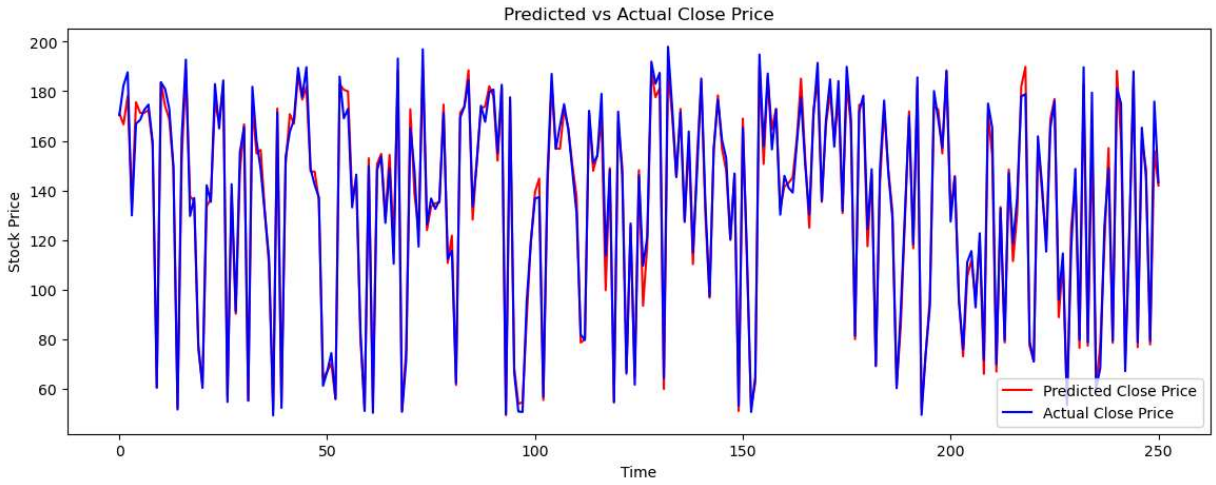
```
32/32 [==============================] - 1s 29ms/step - loss: 0.0041 - val_loss: 0.0
023
Epoch 39/50
32/32 [==============================] - 1s 29ms/step - loss: 0.0041 - val_loss: 0.0
023
Epoch 40/50
32/32 [==============================] - 1s 28ms/step - loss: 0.0041 - val_loss: 0.0
025
Epoch 41/50
32/32 [==============================] - 1s 36ms/step - loss: 0.0043 - val_loss: 0.0
026
Epoch 42/50
32/32 [==============================] - 1s 26ms/step - loss: 0.0042 - val_loss: 0.0
022
Epoch 43/50
32/32 [==============================] - 1s 20ms/step - loss: 0.0038 - val_loss: 0.0
022
Epoch 44/50
32/32 [==============================] - 1s 30ms/step - loss: 0.0038 - val_loss: 0.0
023
Epoch 45/50
32/32 [==============================] - 1s 23ms/step - loss: 0.0039 - val_loss: 0.0
022
Epoch 46/50
32/32 [==============================] - 1s 27ms/step - loss: 0.0039 - val_loss: 0.0
027
Epoch 47/50
32/32 [==============================] - 1s 32ms/step - loss: 0.0038 - val_loss: 0.0
022
Epoch 48/50
32/32 [==============================] - 1s 17ms/step - loss: 0.0037 - val_loss: 0.0
023
Epoch 49/50
32/32 [==============================] - 1s 26ms/step - loss: 0.0038 - val_loss: 0.0
023
Epoch 50/50
32/32 [==============================] - 1s 33ms/step - loss: 0.0039 - val_loss: 0.0
022
```

In [ ]:
```python
# predictions
predictions = model.predict(X_test)

# inverse
predictions = scaler.inverse_transform(predictions)
y_test = scaler.inverse_transform(y_test)

# eval
mse = np.mean((predictions - y_test) ** 2)
print(f'Mean Squared Error: {mse}')
```

```
8/8 [==============================] - 1s 2ms/step
Mean Squared Error: 178388167035632.8
```

Model eval with predictions

In [ ]:
```
# Plot the predictions vs actual values
plt.figure(figsize=(14, 5))
plt.plot(predictions[:, 3], color='red', label='Predicted Close Price')
plt.plot(y_test[:, 3], color='blue', label='Actual Close Price')
plt.title('Predicted vs Actual Close Price')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```



Conclusion

The model was not able to predict the stock price with very high accuracy, this last attempt was a little bit better, but training on stock prices is very difficult because it is very random, in the intra day fluctions of the price, and not having context behing the news that is driving the price up or down. For instance during covid the stock market sold off, but then the fed started buying up bonds, and the stock market went up. So it is very difficult to predict the stock market with just the stock prices.