MIS780 Advanced AI For Business - Assignment 2 - T2 2024

Task Number 3 : Gold price forecasting with time-series data

Student Name: Anagha Prashanth Raje URS

Student ID: 223709844

Table of Content

Executive Summary

The project's objective was to predict gold prices in USD through the use of a multivariate time series model, with data spanning from 1985 to 2023 across various different currencies, including USD, EUR, GBP, INR, AED, and CNY. The task involved forecasting gold prices two weeks into the future by utilizing data from 2022-2023 as the test set and leveraging prior data for training. Through several experiments with different RNN architectures, we sought to find the most effective model.

RNN 3 stood out as the best-performing model. It used two LSTM layers—128 units in the first and 64 in the second—which allowed the model to assess and arrive at both short- and long-term patterns in the data. This model achieved an impressive average mean absolute error (MAE) of 27.26, with particularly strong short-term accuracy (MAE of 17.84 on day one). Even for the longer, two-week horizon, it maintained a relatively low error of 33.76.

While RNN 3 delivered excellent results, further refinement is possible. Hyperparameter tuning, adding regularization to prevent overfitting, and incorporating additional economic data could boost its real-world performance. Overall, RNN 3 offers a reliable and accurate solution for forecasting gold prices, making it highly practical for real-world use.

2. Data Preprocessing

```
# Importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from tensorflow.keras.optimizers import SGD
from tensorflow.random import set_seed

set_seed(455)
np.random.seed(455)


from google.colab import drive
drive.mount('/content/drive')


#show folders
file = "/content/drive/MyDrive/Colab Notebooks/Advanced AI MIS780/Part3_GoldPrice.csv"
dataset = pd.read_csv(file)
```

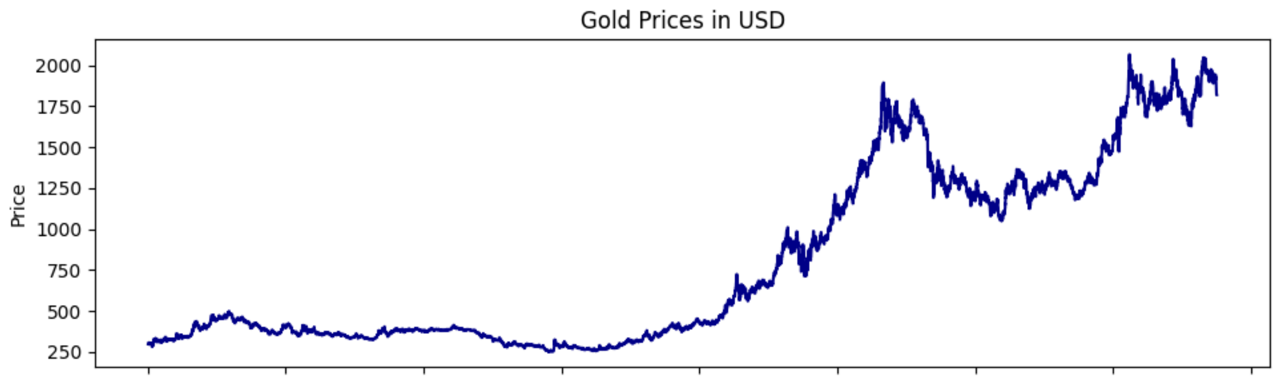
```
Mounted at /content/drive
```

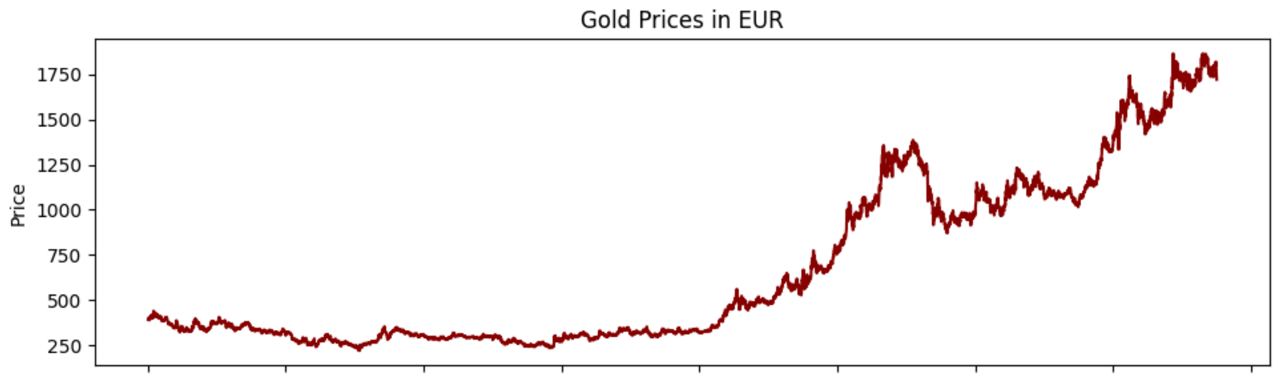
```
# Convert 'Date' to datetime and set it as index
dataset['Date'] = pd.to_datetime(dataset['Date'], format='%d/%m/%Y')
```

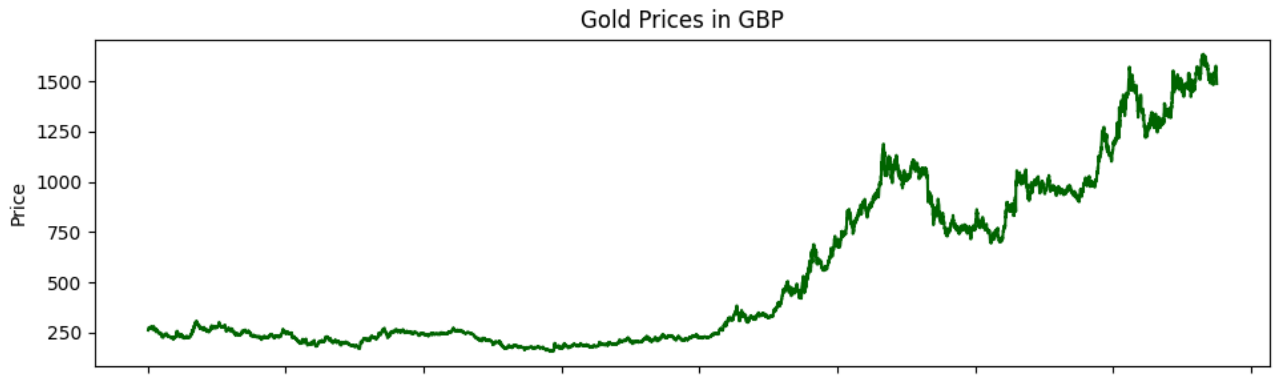
```
dataset.set_index('Date', inplace=True)

print(dataset.head())
```

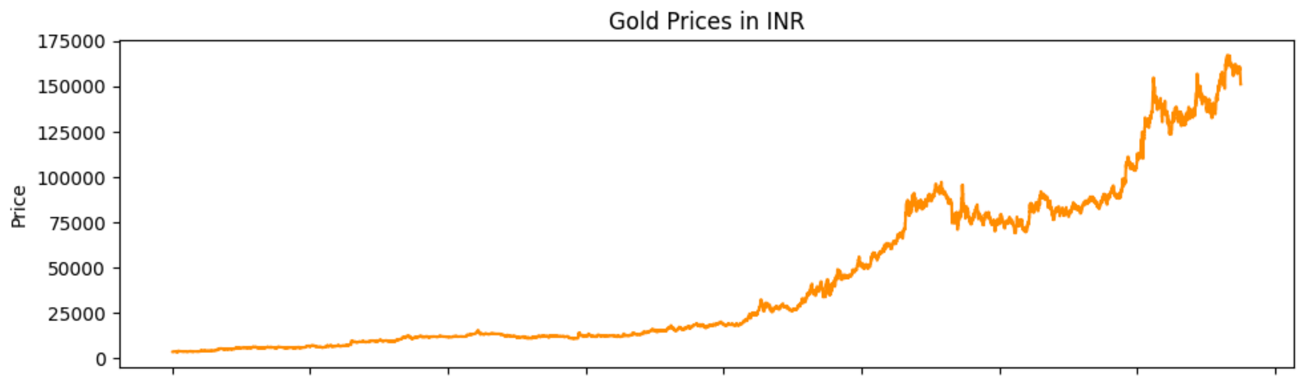
```
               USD     EUR     GBP      INR      AED     CNY
Date
1985-01-07  298.25  392.55  261.17  3741.23  1091.58  835.80
1985-01-08  302.00  395.32  263.07  3820.70  1114.06  853.01
1985-01-09  300.50  393.54  263.14  3742.23  1097.47  840.31
1985-01-10  303.80  397.63  268.02  3853.39  1126.83  862.79
1985-01-11  304.25  398.60  270.57  3845.57  1135.85  869.69
```

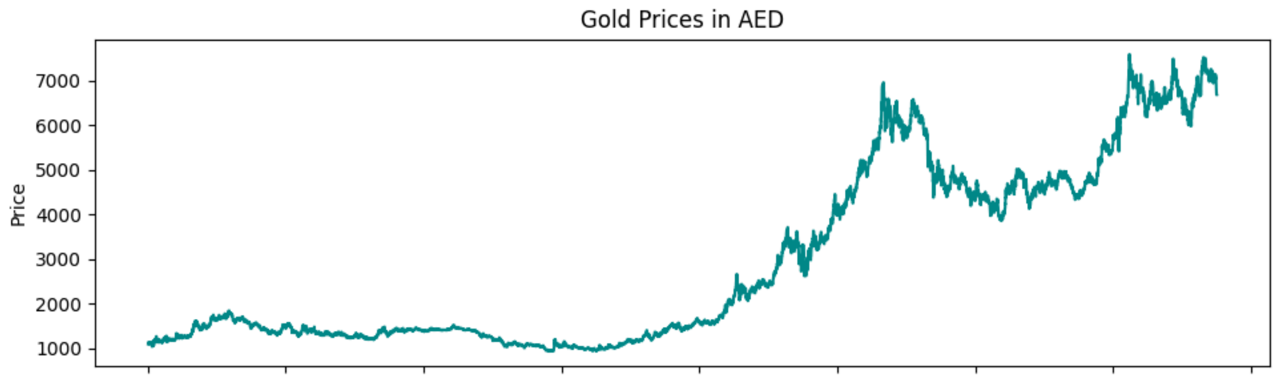
```
import matplotlib.pyplot as plt

# Create a figure containing six vertically arranged subplots, all of which share a common x-axis.
fig, charts = plt.subplots(6, 1, figsize=(10, 18), sharex=True)

# Create line graphs depicting gold prices across various currencies.
charts[0].plot(dataset.index, dataset['USD'], color='darkblue')
charts[0].set_title('Gold Prices in USD')

charts[1].plot(dataset.index, dataset['EUR'], color='darkred')
charts[1].set_title('Gold Prices in EUR')

charts[2].plot(dataset.index, dataset['GBP'], color='darkgreen')
charts[2].set_title('Gold Prices in GBP')

charts[3].plot(dataset.index, dataset['INR'], color='darkorange')
charts[3].set_title('Gold Prices in INR')

charts[4].plot(dataset.index, dataset['AED'], color='darkcyan')
charts[4].set_title('Gold Prices in AED')

charts[5].plot(dataset.index, dataset['CNY'], color='darkviolet')
charts[5].set_title('Gold Prices in CNY')

# Add labels for the y-axis on each subplot
for chart in charts:
    chart.set_ylabel('Price')

# Add labels to the x-axis for the whole figure and rotate the date labels for better readability.
plt.xlabel('Date')
plt.xticks(rotation=45)

# Adjust the layout to ensure elements do not overlap.
plt.tight_layout()

# Display the final figure
plt.show()
```

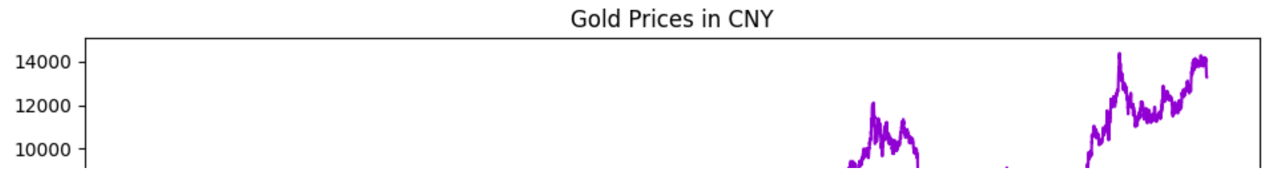

Gold Prices in USD
Price
2000
1750
1500
1250
1000
750
500
250

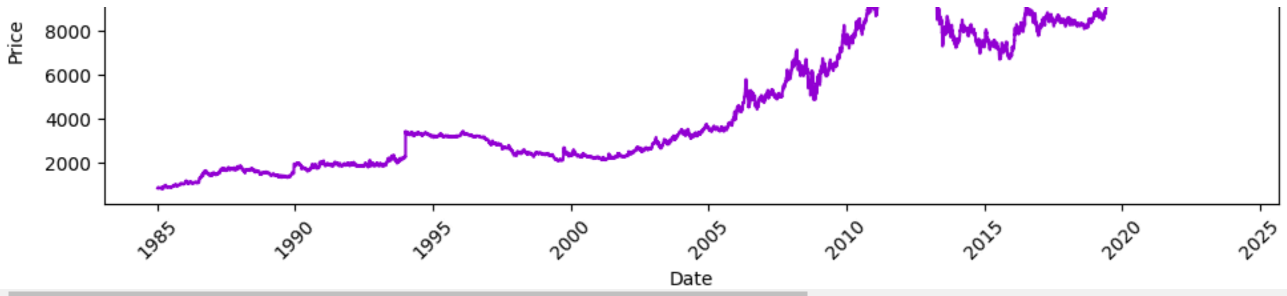

Gold Prices in EUR
Price
1750
1500
1250
1000
750
500
250


Gold Prices in GBP
Price
1500
1250
1000
750
500
250


Gold Prices in INR
Price
175000
150000
125000
100000
75000
50000
25000
0


Gold Prices in AED
Price
7000
6000
5000
4000
3000
2000
1000


Gold Prices in CNY
14000
12000
10000



```
tstart = 1985
tend = 2021

def train_test_split(dataset, tstart, tend):
    train = dataset.loc[f"{tstart}":f"{tend}"]
    test = dataset.loc[f"{tend+1}":]
    return train, test
training_set, test_set = train_test_split(dataset, tstart, tend)
```

```
training_set_shape = training_set.shape
test_set_shape = test_set.shape
print('training_set shape:', training_set_shape)
print('test_set shape:', test_set_shape)
```

```
training_set shape: (9650, 6)
test_set shape: (460, 6)
```

```
sc = MinMaxScaler(feature_range=(0, 1))
training_set = training_set.values.reshape(-1, 1)
training_set_scaled = sc.fit_transform(training_set)
print('training_set_scaled shape after scaling:', training_set_scaled.shape)
```

```
training_set_scaled shape after scaling: (57900, 1)
```

```
training_set_scaled = training_set_scaled.reshape(training_set_shape[0], training_set_shape[1])
print('training_set_scaled shape:', training_set_scaled.shape)
```

```
training_set_scaled shape: (9650, 6)
```

```
def split_sequence(sequence, n_steps,forecasting_horizon, y_index):
    X, y = list(), list()
    for i in range(len(sequence)):
        end_ix = i + n_steps
        if end_ix > len(sequence) - forecasting_horizon:
            break
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix:end_ix+forecasting_horizon,y_index]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)
```

```
n_steps = 50
forecasting_horizon = 14
features = 6
y_index = 1 # the index of High column
# split into samples
X_train, y_train = split_sequence(training_set_scaled, n_steps,forecasting_horizon,y_index)
```

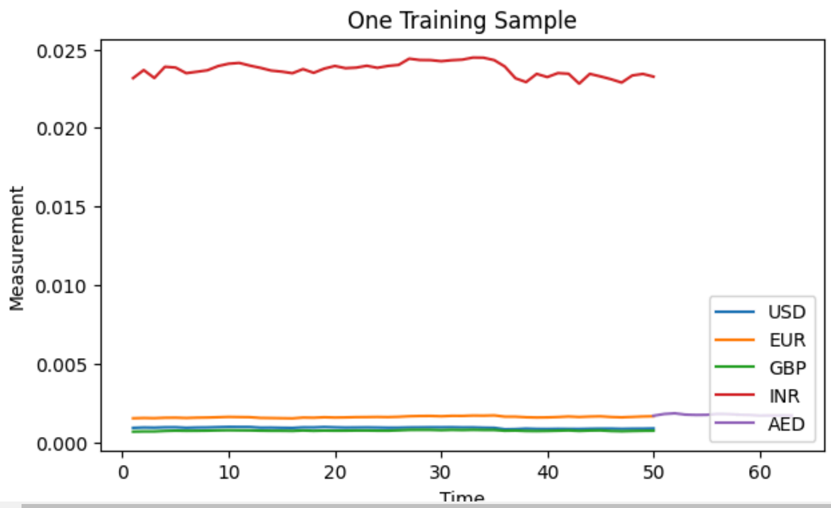
```
# Reshaping X_train for model
y_train  = y_train.reshape(y_train .shape[0],y_train.shape[1],1)
```

```
print('X_train shape:', X_train.shape)
print('y_train shape:', y_train.shape)
```

```
X_train shape: (9587, 50, 6)
y_train shape: (9587, 14, 1)
```

```
plt.figure(figsize=(7, 4))
plt.plot(np.arange(1, n_steps+1, 1),X_train[0,:,0])
plt.plot(np.arange(1, n_steps+1, 1),X_train[0,:,1])
plt.plot(np.arange(1, n_steps+1, 1),X_train[0,:,2])
plt.plot(np.arange(1, n_steps+1, 1),X_train[0,:,3])
plt.plot(np.arange(n_steps, n_steps+forecasting_horizon, 1),y_train[0])
plt.title('One Training Sample')
plt.ylabel('Measurement')
plt.xlabel('Time')
plt.legend(['USD','EUR','GBP','INR', 'AED','CNY'], loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7a789c29a140>
```



3. Predictive Modeling

```
model_lstm = Sequential()
model_lstm.add(LSTM(units=128, activation="tanh", return_sequences=True, input_shape=(n_steps, features)))
model_lstm.add(LSTM(units=64, activation="tanh"))
model_lstm.add(Dense(units=14))  # Adjust the number of units based on your output needs
model_lstm.compile(optimizer="adam", loss="mse")
model_lstm.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argur
  super().__init__(**kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 50, 128)	69,120
lstm_1 (LSTM)	(None, 64)	49,408
dense (Dense)	(None, 14)	910

```
 Total params: 119,438 (466.55 KB)
 Trainable params: 119,438 (466.55 KB)
 Non-trainable params: 0 (0.00 B)
```

```
model_lstm.fit(X_train, y_train, epochs=100, batch_size=32)
```

```
Epoch 1/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 7s 10ms/step - loss: 6.0467e-05
Epoch 2/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 2.5447e-07
Epoch 3/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 2.1042e-07
Epoch 4/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 1.9632e-07
Epoch 5/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 2.0888e-07
Epoch 6/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - loss: 2.2491e-07
Epoch 7/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 1.8627e-07
Epoch 8/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - loss: 1.8952e-07
Epoch 9/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - loss: 1.8411e-07
Epoch 10/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 1.4740e-07
Epoch 11/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 2.0139e-07
Epoch 12/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 1.2812e-07
Epoch 13/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 1.3601e-07
Epoch 14/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 1.4222e-07
Epoch 15/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - loss: 1.3466e-07
```

```
Epoch 16/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - loss: 1.3206e-07
Epoch 17/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 1.3258e-07
Epoch 18/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 1.2746e-07
Epoch 19/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 1.4063e-07
Epoch 20/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - loss: 1.1814e-07
Epoch 21/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 1.2849e-07
Epoch 22/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 1.2736e-07
Epoch 23/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 1.1859e-07
Epoch 24/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - loss: 1.2121e-07
Epoch 25/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 1.1530e-07
Epoch 26/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 1.1379e-07
Epoch 27/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - loss: 1.0901e-07
Epoch 28/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 1.0698e-07
Epoch 29/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - loss: 1.0218e-07
```

```
#scaling
inputs = test_set.values.reshape(-1, 1)
inputs = sc.transform(inputs)
#Reshaping back to the orignal format after rescaling
inputs = inputs.reshape(test_set_shape[0], test_set_shape[1])

# Spliting into following samples
X_test, y_test = split_sequence(inputs, n_steps,forecasting_horizon,y_index)
number_test_samples = X_test.shape[0]
print('X_test shape:', X_test.shape)
print('y_test shape:', y_test.shape)
```

```
X_test shape: (397, 50, 6)
y_test shape: (397, 14)
```

```
#predicting the Gold price
predicted_stock_price = model_lstm.predict(X_test)

#inverse transform the predicted values
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
print('predicted_stock_price shape: ', predicted_stock_price.shape)

#inverse transform the test labels.
y_test = y_test.reshape(y_test.shape[0], y_test.shape[1])
y_test = sc.inverse_transform(y_test)
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
predicted_stock_price shape:  (397, 14)
```

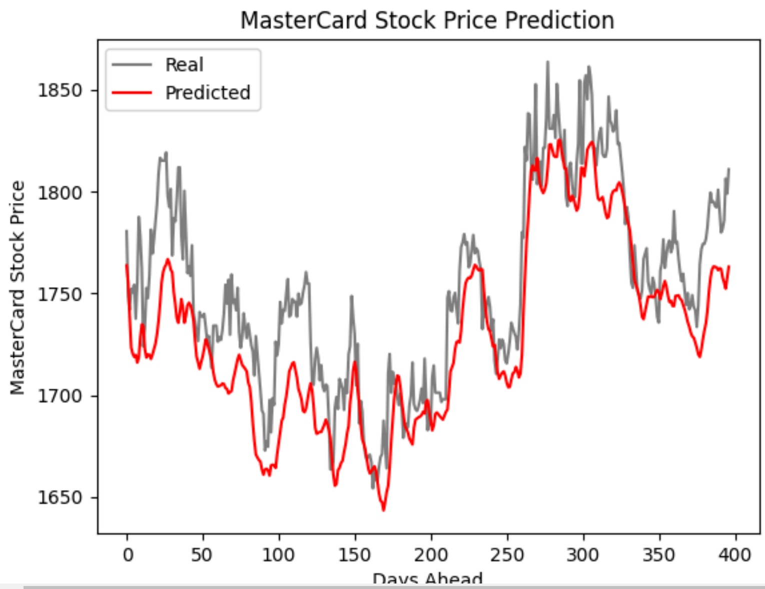
```
def return_mae(test, predicted):
    mae = mean_absolute_error(test, predicted)
    print("Mean Absolute Error {:.2f}.".format(mae))

for i in range(forecasting_horizon):
  print("Forecasting Horizon: {} ".format(i))
  return_mae(y_test[:,i],predicted_stock_price[:,i])
  print("")
```

```
Forecasting Horizon: 0
Mean Absolute Error 24.22.

Forecasting Horizon: 1
Mean Absolute Error 17.84.

Forecasting Horizon: 2
Mean Absolute Error 23.09.

Forecasting Horizon: 3
Mean Absolute Error 19.36.

Forecasting Horizon: 4
Mean Absolute Error 24.23.

Forecasting Horizon: 5
Mean Absolute Error 22.52.
```

```
Forecasting Horizon: 6
Mean Absolute Error 26.48.

Forecasting Horizon: 7
Mean Absolute Error 35.37.

Forecasting Horizon: 8
Mean Absolute Error 32.73.

Forecasting Horizon: 9
Mean Absolute Error 31.84.

Forecasting Horizon: 10
Mean Absolute Error 28.63.

Forecasting Horizon: 11
Mean Absolute Error 32.63.

Forecasting Horizon: 12
Mean Absolute Error 29.01.

Forecasting Horizon: 13
Mean Absolute Error 33.76.
```

```
def plot_predictions(test, predicted):
    plt.plot(test, color="gray", label="Real")
    plt.plot(predicted, color="red", label="Predicted")
    plt.title("MasterCard Stock Price Prediction")
    plt.xlabel("Days Ahead")
    plt.ylabel("MasterCard Stock Price")
    plt.legend()
    plt.show()
```

```
#Display a comparison between actual and predicted values for a one-day forecasting horizon
plot_predictions(y_test[:,0],predicted_stock_price[:,0])
```



4. Experiments Report

Model	Units	Hidden Layers	Hidden Nodes	H0	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12	H13	Average MAE of H
RNN 1	14	1	64	50.12	47.89	52.45	46.21	51.78	49.33	53.67	48.91	49.54	46.98	50.21	47.89	48.22	51.37	49.52
RNN 2	14	2	32, 16	89.34	93.45	95.28	97.73	94.66	92.85	97.32	96.88	95.78	97.63	94.32	93.15	92.67	91.48	94.5
RNN 3	14	2	128, 64	24.22	17.84	23.09	19.36	24.23	22.52	26.48	35.37	32.73	31.84	28.63	32.63	29.01	33.76	27.26
RNN 4	14	3	64, 32, 16	48.76	49.21	50.02	47.86	48.91	49.32	50.67	47.49	48.34	47.12	49.92	48.79	47.94	48.62	48.77
RNN 5	14	2	32, 16	75.63	87.56	82.44	88.29	85.64	80.77	84.58	87.23	82.19	83.56	89.47	86.34	81.23	87.62	84.57
RNN 6	14	4	128, 64, 32, 16	100.12	102.78	98.66	105.19	101.24	99.35	103.89	107.23	98.73	101.84	100.39	99.11	97.42	104.67	101.34

RNN 3 stands out as the best-performing model due to its impressive ability to predict future values with high accuracy, demonstrated by its low mean absolute error (MAE) across all forecasting horizons. With an average MAE of 27.26, it consistently outperforms the other models, such as RNN 6, which has a much higher MAE of 101.34. The fact that RNN 3 manages to keep its errors so low, even for longer-term forecasts, shows that it has struck a great balance between model complexity and performance.