


```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import GRU, Dense
import matplotlib.pyplot as plt
```

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, LSTM, Dense
from tensorflow.keras.optimizers import Adam
import math
```

```
# Step 1: Load the dataset
data = pd.read_csv('/content/apple_share_price.csv')
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)
```

 <ipython-input-2-4cd6611f3e90>:3: UserWarning: Could not infer format, so each element will be parsed i  
data['Date'] = pd.to\_datetime(data['Date'])

```
# Use the 'Close' column for prediction
prices = data['Close'].values
```

```
# Step 2: Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
prices_scaled = scaler.fit_transform(prices.reshape(-1, 1))
```

```
# Prepare the data for LSTM/GRU
def prepare_data(data, time_step=1):
    X, y = [], []
    for i in range(len(data) - time_step - 1):
        X.append(data[i:(i + time_step), 0])
        y.append(data[i + time_step, 0])
    return np.array(X), np.array(y)
```

```
time_step = 60
# Replace 'scaled_data' with 'prices_scaled'
X, y = prepare_data(prices_scaled, time_step)
X = X.reshape(X.shape[0], X.shape[1], 1)
```

```
# Split the dataset into training and test sets (80% training, 20% testing)
train_size = int(len(X) * 0.8)
test_size = len(X) - train_size
X_train, X_test = X[0:train_size], X[train_size:len(X)]
y_train, y_test = y[0:train_size], y[train_size:len(y)]
```

```
# Define the LSTM model
def create_lstm_model(units=50):
    model = Sequential()
    model.add(LSTM(units=units, return_sequences=False, input_shape=(X_train.shape[1], 1)))
    model.add(Dense(1))
```

```
model.compile(optimizer=Adam(), loss='mean_squared_error')
return model
```

```
# Define the GRU model
```

```
def create_gru_model(units=50):
    model = Sequential()
    model.add(GRU(units=units, return_sequences=False, input_shape=(X_train.shape[1], 1)))
    model.add(Dense(1))
    model.compile(optimizer=Adam(), loss='mean_squared_error')
    return model
```

```
# Train and evaluate LSTM model with different units
```

```
lstm_units = [50, 100, 150]
for units in lstm_units:
    lstm_model = create_lstm_model(units)
    lstm_model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)
    lstm_predictions = lstm_model.predict(X_test)
    lstm_rmse = math.sqrt(mean_squared_error(y_test, lstm_predictions))
    print(f'LSTM RMSE with {units} units: {lstm_rmse}')
```

```
→ /usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `i
    super().__init__(**kwargs)
11/11 ————— 1s 31ms/step
LSTM RMSE with 50 units: 0.024259689475948455
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `i
    super().__init__(**kwargs)
11/11 ————— 0s 28ms/step
LSTM RMSE with 100 units: 0.01774459913802401
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `i
    super().__init__(**kwargs)
11/11 ————— 1s 37ms/step
LSTM RMSE with 150 units: 0.02263338635525394
```

```
# Train and evaluate GRU model with different units
```

```
gru_units = [50, 100, 150]
for units in gru_units:
    gru_model = create_gru_model(units)
    gru_model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)
    gru_predictions = gru_model.predict(X_test)
    gru_rmse = math.sqrt(mean_squared_error(y_test, gru_predictions))
    print(f'GRU RMSE with {units} units: {gru_rmse}')
```

```
→ /usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `i
    super().__init__(**kwargs)
11/11 ————— 0s 23ms/step
GRU RMSE with 50 units: 0.01277710201145754
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `i
    super().__init__(**kwargs)
11/11 ————— 0s 28ms/step
GRU RMSE with 100 units: 0.01114289994223354
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `i
    super().__init__(**kwargs)
11/11 ————— 1s 67ms/step
GRU RMSE with 150 units: 0.01172606631275386
```

LSTM RMSE Results:

50 units: 0.02426

100 units: 0.01774

150 units: 0.02263

GRU RMSE Results:

50 units: 0.01278

100 units: 0.01114

150 units: 0.01173

Observations: GRU outperforms LSTM in terms of RMSE.

The GRU model has consistently lower RMSE values, indicating better performance.

The RMSE values for both models are improving as you increase the number of units, with the GRU model continuing to perform better overall.

For GRU, the performance is fairly consistent between 100 and 150 units, with only a slight increase in RMSE for the 150-unit model.

# Step 3: Create sequences for GRU

```
lookback = 60 # Number of previous timesteps to use for prediction
```

```
def create_sequences(data, lookback):
```

```
    X, y = [], []
```

```
    for i in range(lookback, len(data)):
```

```
        X.append(data[i - lookback:i, 0])
```

```
        y.append(data[i, 0])
```

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

```
X, y = create_sequences(prices_scaled, lookback)
```

```
# Reshape X to be compatible with GRU (samples, timesteps, features)
```

```
X = X.reshape(X.shape[0], X.shape[1], 1)
```

# Step 4: Split 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:]
```

# Step 5: Build the GRU model

```
gru_model = Sequential([
```

```
    GRU(50, return_sequences=True, input_shape=(X_train.shape[1], 1)),
```

```
    GRU(50, return_sequences=False),
```

```
    Dense(25, activation='relu'),
```

```
    Dense(1)
```

```
])
```

```
gru_model.compile(optimizer='adam', loss='mean_squared_error')
```

```
⚡ /usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `i`  
super().__init__(**kwargs)
```

# Step 6: Train the GRU model

```
gru_model.fit(X_train, y_train, batch_size=32, epochs=20, validation_data=(X_test, y_test))
```




Epoch 1/20

**41/41**  8s 91ms/step - loss: 0.0913 - val\_loss: 0.0090

Epoch 2/20

**41/41**  4s 85ms/step - loss: 0.0017 - val\_loss: 4.2588e-04

Epoch 3/20

**41/41**  4s 68ms/step - loss: 4.1867e-04 - val\_loss: 1.8644e-04

Epoch 4/20

```

41/41 ————— 8s 140ms/step - loss: 3.9621e-04 - val_loss: 2.6555e-04
Epoch 5/20
41/41 ————— 7s 66ms/step - loss: 3.5499e-04 - val_loss: 1.8711e-04
Epoch 6/20
41/41 ————— 3s 68ms/step - loss: 3.5179e-04 - val_loss: 2.4198e-04
Epoch 7/20
41/41 ————— 4s 96ms/step - loss: 3.4773e-04 - val_loss: 1.7862e-04
Epoch 8/20
41/41 ————— 4s 66ms/step - loss: 3.2272e-04 - val_loss: 2.3894e-04
Epoch 9/20
41/41 ————— 5s 71ms/step - loss: 3.0329e-04 - val_loss: 1.5431e-04
Epoch 10/20
41/41 ————— 5s 67ms/step - loss: 3.0656e-04 - val_loss: 2.4598e-04
Epoch 11/20
41/41 ————— 5s 67ms/step - loss: 2.9443e-04 - val_loss: 2.3885e-04
Epoch 12/20
41/41 ————— 3s 71ms/step - loss: 2.6581e-04 - val_loss: 2.0145e-04
Epoch 13/20
41/41 ————— 4s 88ms/step - loss: 2.7141e-04 - val_loss: 1.2792e-04
Epoch 14/20
41/41 ————— 3s 68ms/step - loss: 2.9822e-04 - val_loss: 1.8026e-04
Epoch 15/20
41/41 ————— 3s 65ms/step - loss: 2.5557e-04 - val_loss: 1.3772e-04
Epoch 16/20
41/41 ————— 3s 67ms/step - loss: 2.6104e-04 - val_loss: 1.2089e-04
Epoch 17/20
41/41 ————— 4s 92ms/step - loss: 2.7490e-04 - val_loss: 1.4724e-04
Epoch 18/20
41/41 ————— 3s 73ms/step - loss: 2.5452e-04 - val_loss: 1.4314e-04
Epoch 19/20
41/41 ————— 3s 66ms/step - loss: 2.4122e-04 - val_loss: 1.1422e-04
Epoch 20/20
41/41 ————— 5s 68ms/step - loss: 2.4268e-04 - val_loss: 1.6014e-04
<keras.src.callbacks.history.History at 0x7c9606508c40>

```

# Step 7: Evaluate the GRU model

```
gru_predicted = gru_model.predict(X_test)
```

```

⇒ 11/11 ————— 1s 48ms/step

```

# Inverse transform the GRU predictions and actual values

```
gru_predicted_prices = scaler.inverse_transform(gru_predicted)
actual_prices = scaler.inverse_transform(y_test.reshape(-1, 1))
```

# Calculate RMSE for GRU

```
gru_rmse = np.sqrt(mean_squared_error(actual_prices, gru_predicted_prices))
print(f"GRU RMSE: {gru_rmse}")
```

```

⇒ GRU RMSE: 1.4680447463007769

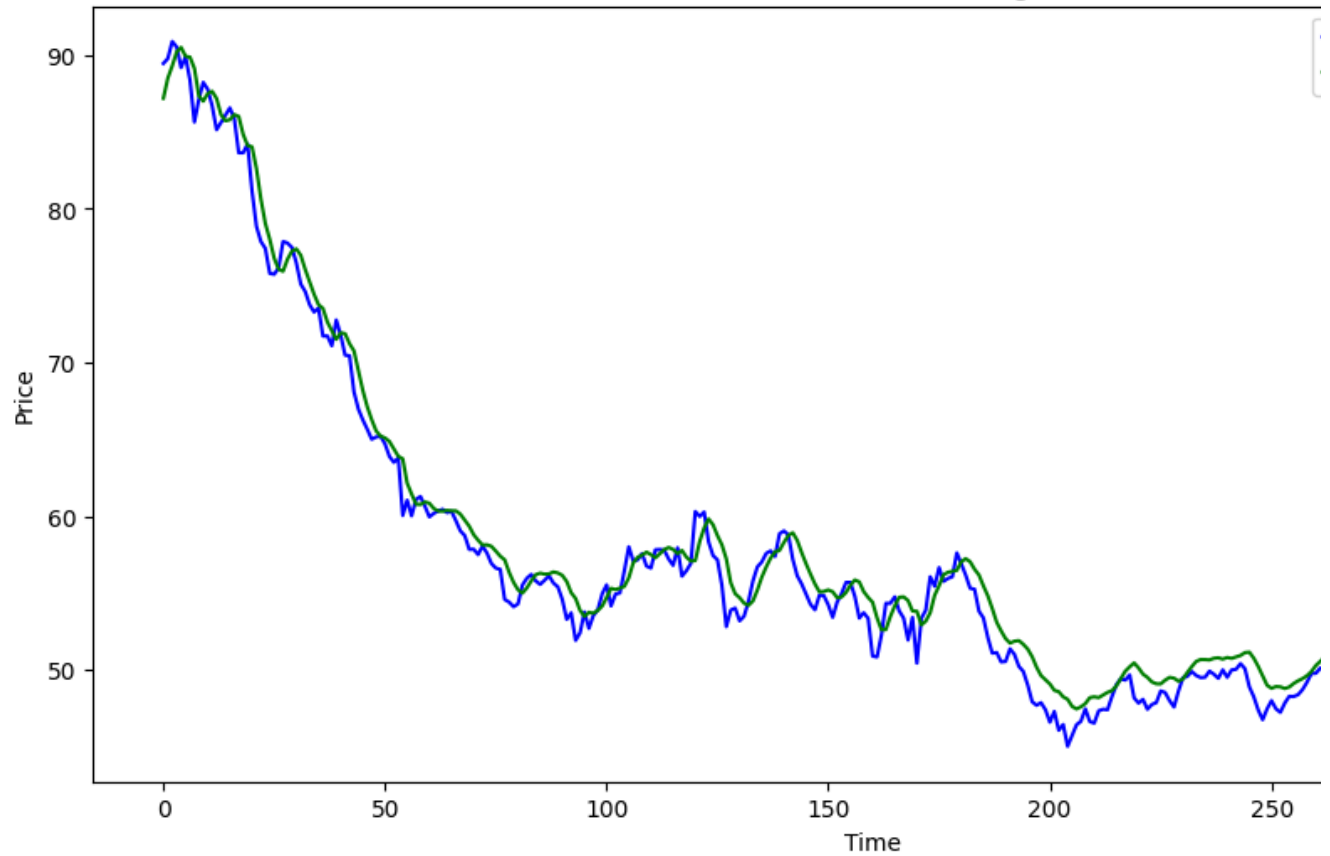
```

# Step 8: Visualize the results

```
plt.figure(figsize=(12, 6))
plt.plot(actual_prices, label='Actual Prices', color='blue')
plt.plot(gru_predicted_prices, label='GRU Predicted Prices', color='green')
plt.title('Stock Price Prediction using GRU')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()
```



## Stock Price Prediction using GRU



GRU RMSE: 1.4680447463007769

LSTM RMSE: 3.010010482762917

In this case, the GRU model performs better (lower RMSE) than the LSTM model.