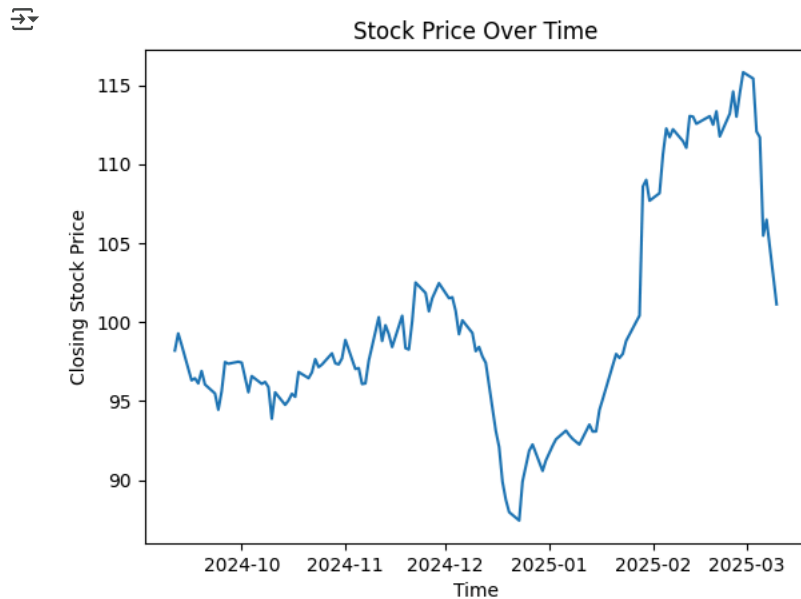


```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

Step 1: Load stock data

```
stock_data = pd.read_csv('drive/MyDrive/SBUX(1).csv', index_col='Date', parse_dates=True)
stock_data['Close/Last'] = pd.to_numeric(stock_data['Close/Last'].str.replace('\$', '', regex=True))
stock_prices = stock_data['Close/Last'].values # Use the 'Close' column for prices
```

```
plt.plot(stock_data.index, stock_prices)
plt.xlabel('Time')
plt.ylabel('Closing Stock Price')
plt.title('Stock Price Over Time')
plt.show()
```



Step 2: Normalisation

```
scaler = MinMaxScaler(feature_range=(-1, 1))
stock_prices_scaled = scaler.fit_transform(stock_prices.reshape(-1, 1)).flatten()
```

```
# Step 3: Create sequences for RNN input-output
def create_sequences(data, seq_length, forecast_length):
    X, y = [], []
    for i in range(len(data) - seq_length - forecast_length):
        X.append(data[i:i+seq_length])
        y.append(data[i+seq_length:i+seq_length+forecast_length])
    return np.array(X), np.array(y)

# Prepare sequences for RNN
seq_length = 30 # Lookback for prediction
forecast_length = 3 # Number of time steps to predict
X, y = create_sequences(stock_prices_scaled, seq_length, forecast_length)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Reshape data for RNN input [samples, time steps, features]
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

Start coding or [generate](#) with AI.

Start coding or [generate](#) with AI.

MODEL 1

```
# Step 5: Build the RNN model
model = Sequential()
model.add(SimpleRNN(128, activation='relu', return_sequences=True))
model.add(Dropout(0.2))
model.add(SimpleRNN(64, activation='relu', return_sequences=True))
model.add(Dropout(0.2))
model.add(SimpleRNN(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(forecast_length))
model.compile(optimizer='adam', loss='mse')



# Step 6: Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
```

 <keras.src.callbacks.history.History at 0x7978a80f1710>

```
y_pred = model.predict(X_test)
y_pred_rescaled = scaler.inverse_transform(y_pred)
y_test_rescaled = scaler.inverse_transform(y_test)
```

```
mse = mean_squared_error(y_test_rescaled, y_pred_rescaled)
print(f"MSE: {mse}")
```

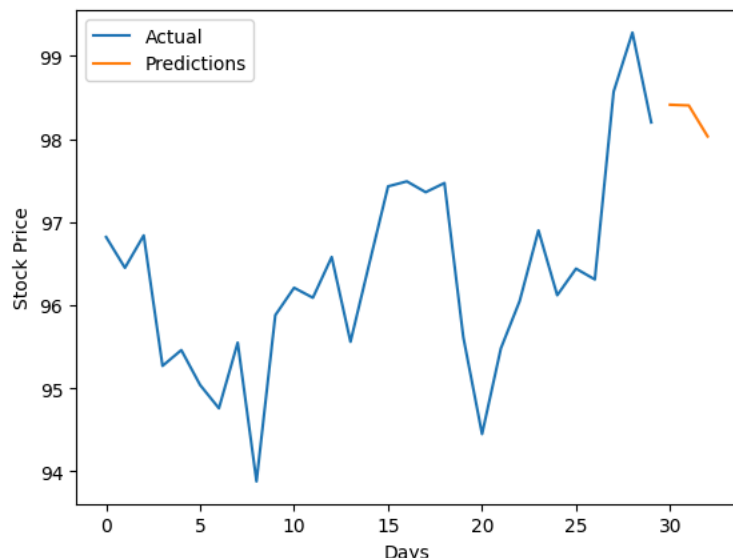
```
rmse = np.sqrt(mse)
print(f"RMSE: {rmse}")
```

 1/1  0s 445ms/step
MSE: 2.3002757960560576
RMSE: 1.5166660133516732

```
# Step 7: Make predictions
last_sequence = stock_prices_scaled[-seq_length:].reshape(1, seq_length, 1)
predictions_scaled = model.predict(last_sequence)
predictions = scaler.inverse_transform(predictions_scaled).flatten()
```

```
# Step 8: Plot the results
plt.plot(stock_prices[-seq_length:], label='Actual')
plt.plot(np.arange(seq_length, seq_length + forecast_length), predictions, label='Predictions')
plt.xlabel('Days')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

1/1 0s 50ms/step



```
from datetime import timedelta, datetime
last_date = stock_data.index[0]

# Generate future dates for predictions
future_dates = [last_date + timedelta(days=i) for i in range(1, forecast_length + 1)]

# Print the predictions with dates
print("Predicted Stock Prices with Dates:")
for i, (price, date) in enumerate(zip(predictions, future_dates)):
    print(f'Date: {date.strftime('%Y-%m-%d')}, Predicted Price: ${price:.2f}')
```

Predicted Stock Prices with Dates:
Date: 2025-03-11, Predicted Price: \$98.41
Date: 2025-03-12, Predicted Price: \$98.40
Date: 2025-03-13, Predicted Price: \$98.03

Start coding or [generate](#) with AI.

MODEL 2

```
# Step 5: Build the RNN model
model2 = Sequential()
model2.add(SimpleRNN(128, activation='relu', return_sequences=True))
model2.add(Dropout(0.2))
model2.add(SimpleRNN(64, activation='relu', return_sequences=True))
model2.add(Dropout(0.2))
model2.add(SimpleRNN(32, activation='relu', return_sequences=True))
model2.add(Dropout(0.2))
model2.add(SimpleRNN(16, activation='relu'))
model2.add(Dropout(0.2))
model2.add(Dense(forecast_length))
model2.compile(optimizer='adam', loss='mse')
```

```
# Step 6: Train the model
model2.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
```

<keras.src.callbacks.history.History at 0x79789b1f4990>

```

y_pred = model2.predict(X_test)
y_pred_rescaled = scaler.inverse_transform(y_pred)
y_test_rescaled = scaler.inverse_transform(y_test)

```

```

mse = mean_squared_error(y_test_rescaled, y_pred_rescaled)
print(f"MSE: {mse}")

```

```

rmse = np.sqrt(mse)
print(f"RMSE: {rmse}")

```

```

1/1 ————— 1s 568ms/step
MSE: 2.618994674740958
RMSE: 1.6183308298184762

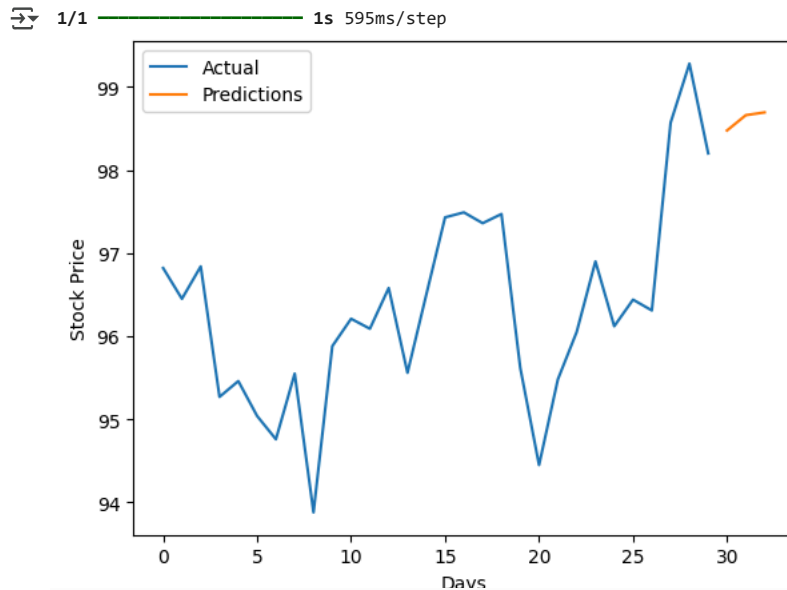
```

```

# Step 7: Make predictions
last_sequence = stock_prices_scaled[-seq_length:].reshape(1, seq_length, 1)
predictions_scaled = model2.predict(last_sequence)
predictions = scaler.inverse_transform(predictions_scaled).flatten()

# Step 8: Plot the results
plt.plot(stock_prices[-seq_length:], label='Actual')
plt.plot(np.arange(seq_length, seq_length + forecast_length), predictions, label='Predictions')
plt.xlabel('Days')
plt.ylabel('Stock Price')
plt.legend()
plt.show()

```



```

from datetime import timedelta, datetime
last_date = stock_data.index[0]

# Generate future dates for predictions
future_dates = [last_date + timedelta(days=i) for i in range(1, forecast_length + 1)]

# Print the predictions with dates
print("Predicted Stock Prices with Dates:")
for i, (price, date) in enumerate(zip(predictions, future_dates)):
    print(f>Date: {date.strftime('%Y-%m-%d')}, Predicted Price: ${price:.2f}")

```

```

Predicted Stock Prices with Dates:
Date: 2025-03-11, Predicted Price: $98.48
Date: 2025-03-12, Predicted Price: $98.66
Date: 2025-03-13, Predicted Price: $98.69

```

MODEL 3

```

# Step 5: Build the RNN model
model3 = Sequential()
model3.add(SimpleRNN(128, activation='relu', return_sequences=True))
model3.add(Dropout(0.2))

```

```

model3.add(SimpleRNN(128, activation='relu', return_sequences=True))
model3.add(Dropout(0.2))
model3.add(SimpleRNN(64, activation='relu', return_sequences=True))
model3.add(Dropout(0.2))
model3.add(SimpleRNN(32, activation='relu', return_sequences=True))
model3.add(Dropout(0.2))
model3.add(SimpleRNN(16, activation='relu'))
model3.add(Dropout(0.2))
model3.add(Dense(forecast_length))
model3.compile(optimizer='adam', loss='mse')

```

Step 6: Train the model

```
model3.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
```

↗ <keras.src.callbacks.history.History at 0x79789f4f3c10>

```

y_pred = model3.predict(X_test)
y_pred_rescaled = scaler.inverse_transform(y_pred)
y_test_rescaled = scaler.inverse_transform(y_test)

```

```

mse = mean_squared_error(y_test_rescaled, y_pred_rescaled)
print(f"MSE: {mse}")

```

```

rmse = np.sqrt(mse)
print(f"RMSE: {rmse}")

```

↗ WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed 1/1 ————— 1s 987ms/step
MSE: 2.2044481408144225
RMSE: 1.4847384082101542

Step 7: Make predictions

```

last_sequence = stock_prices_scaled[-seq_length:].reshape(1, seq_length, 1)
predictions_scaled = model3.predict(last_sequence)
predictions = scaler.inverse_transform(predictions_scaled).flatten()

```

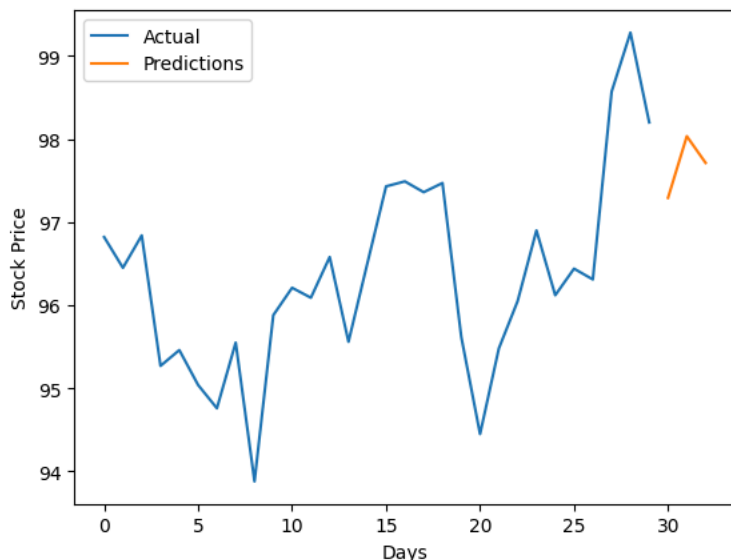
Step 8: Plot the results

```

plt.plot(stock_prices[-seq_length:], label='Actual')
plt.plot(np.arange(seq_length, seq_length + forecast_length), predictions, label='Predictions')
plt.xlabel('Days')
plt.ylabel('Stock Price')
plt.legend()
plt.show()

```

↗ WARNING:tensorflow:6 out of the last 6 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed 1/1 ————— 1s 728ms/step



```

from datetime import timedelta, datetime
last_date = stock_data.index[0]

```

```
# Generate future dates for predictions
```