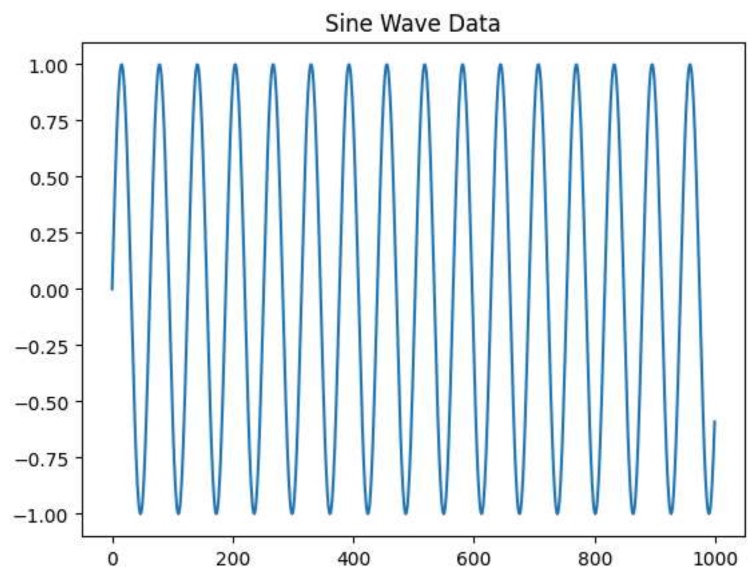
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, LSTM
```

```
# Generate a simple sine wave dataset as an example
data = np.sin(np.arange(0, 100, 0.1))
plt.plot(data)
plt.title("Sine Wave Data")
plt.show()
```



```
# Reshape data to [samples, timesteps, features]
scaler = MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data.reshape(-1, 1))

# Create sequences (X, y)
def create_sequences(data, time_steps=10):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:i + time_steps])
        y.append(data[i + time_steps])
    return np.array(X), np.array(y)

time_steps = 10
X, y = create_sequences(data, time_steps)

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

```
model = Sequential()
model.add(SimpleRNN(50, activation='tanh', input_shape=(time_steps, 1)))
model.add(Dense(1))

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

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

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 50)	2,600
dense (Dense)	(None, 1)	51

```
 Total params: 2,651 (10.36 KB)
 Trainable params: 2,651 (10.36 KB)
 Non-trainable params: 0 (0.00 B)
```

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

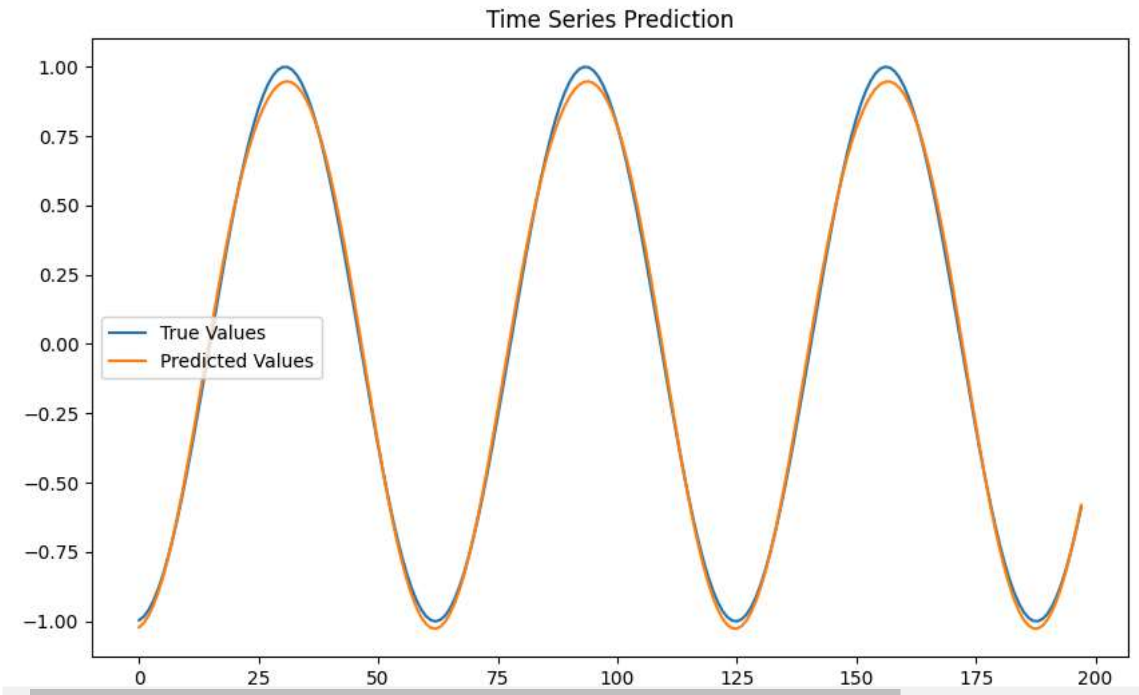
```
Epoch 1/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - loss: 0.3510 - val_loss: 0.0455
Epoch 2/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0332 - val_loss: 0.0075
Epoch 3/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0046 - val_loss: 0.0011
Epoch 4/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0011 - val_loss: 9.2052e-04
Epoch 5/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 9.2084e-04 - val_loss: 8.0366e-04
Epoch 6/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 8.6191e-04 - val_loss: 7.2477e-04
Epoch 7/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 7.3684e-04 - val_loss: 6.5739e-04
Epoch 8/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 6.4727e-04 - val_loss: 5.9032e-04
Epoch 9/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 6.0719e-04 - val_loss: 7.5308e-04
Epoch 10/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 6.2399e-04 - val_loss: 4.9252e-04
Epoch 11/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 5.2413e-04 - val_loss: 5.1167e-04
Epoch 12/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 4.9647e-04 - val_loss: 3.9166e-04
Epoch 13/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 3.8924e-04 - val_loss: 3.5334e-04
Epoch 14/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 3.7095e-04 - val_loss: 3.1989e-04
Epoch 15/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 3.2894e-04 - val_loss: 2.9211e-04
Epoch 16/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 3.0628e-04 - val_loss: 2.6229e-04
Epoch 17/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 2.7335e-04 - val_loss: 2.4134e-04
Epoch 18/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 2.5150e-04 - val_loss: 2.1345e-04
Epoch 19/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 2.1575e-04 - val_loss: 2.5415e-04
Epoch 20/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 2.3107e-04 - val_loss: 1.8369e-04
```

```
# Predict using the test set
y_pred = model.predict(X_test)

# Inverse transform predictions and true values
y_test_inv = scaler.inverse_transform(y_test)
y_pred_inv = scaler.inverse_transform(y_pred)

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(y_test_inv, label='True Values')
plt.plot(y_pred_inv, label='Predicted Values')
plt.title('Time Series Prediction')
plt.legend()
plt.show()
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
```



```
# prompt: find accuracy score of model

from sklearn.metrics import mean_squared_error, mean_absolute_error

# Calculate RMSE
rmse = np.sqrt(mean_squared_error(y_test_inv, y_pred_inv))
print(f"RMSE: {rmse}")

# Calculate MAE
mae = mean_absolute_error(y_test_inv, y_pred_inv)
print(f"MAE: {mae}")

# You can also define a custom accuracy metric based on your specific needs.
# For example, if you want to measure the accuracy within a certain tolerance:
tolerance = 0.05  # Example tolerance
correct_predictions = np.sum(np.abs(y_test_inv - y_pred_inv) < tolerance)
accuracy = correct_predictions / len(y_test_inv)
print(f"Accuracy within {tolerance} tolerance: {accuracy}")
```

```
RMSE: 0.02710590778533943
MAE: 0.02323366663132452
Accuracy within 0.05 tolerance: 0.9242424242424242
```