Construct a basic Recurrent Neural Network (RNN) model to forecast stock prices based on historical stock data

```
import yfinance as yf

# Define the ticker symbol of the company
ticker_symbol = 'AAPL'  # Example: Apple Inc.

# Download historical data
data = yf.download(ticker_symbol, start='2010-01-01', end='2022-01-01')

# Save data to a CSV file
data.to_csv(f'{ticker_symbol}_historical_data.csv')
```

```
[*********************100%%**********************]  1 of 1 completed
```

```
import numpy as np
import pandas as pd
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

# Download historical data
def download_historical_data(ticker_symbol, start_date, end_date):
    data = yf.download(ticker_symbol, start=start_date, end=end_date)
    return data

# Preprocess data
def preprocess_data(data):
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(data['Close'].values.reshape(-1, 1))
    return scaled_data, scaler

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

# Define model architecture
def build_model(sequence_length):
    model = Sequential([
        SimpleRNN(50, input_shape=(sequence_length, 1)),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

# Define parameters
ticker_symbol = 'AAPL'
start_date = '2010-01-01'
end_date = '2022-01-01'
sequence_length = 10  # Number of historical data points to look back

# Download historical data
historical_data = download_historical_data(ticker_symbol, start_date, end_date)

# Preprocess data
scaled_data, scaler = preprocess_data(historical_data)

# Create sequences for training
X, y = create_sequences(scaled_data, sequence_length)

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

# Reshape data for LSTM
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))
```

```
# Build and train the model
model = build_model(sequence_length)
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)

# Evaluate the model
loss = model.evaluate(X_test, y_test)
print('Test Loss:', loss)

# Make predictions
predictions = model.predict(X_test)
predictions = scaler.inverse_transform(predictions)

# Compare predictions with actual prices
actual_prices = scaler.inverse_transform(y_test.reshape(-1, 1))
comparison_df = pd.DataFrame({'Actual': actual_prices.flatten(), 'Predicted': predictions.flatten()})
print(comparison_df)
```

```
[*********************100%%**********************]  1 of 1 completed
Epoch 1/50
61/61 [==============================] - 2s 9ms/step - loss: 0.0012 - val_loss: 3.8546e-04
Epoch 2/50
61/61 [==============================] - 0s 6ms/step - loss: 4.5488e-05 - val_loss: 2.9210e-04
Epoch 3/50
61/61 [==============================] - 0s 6ms/step - loss: 3.8507e-05 - val_loss: 2.3714e-04
Epoch 4/50
61/61 [==============================] - 0s 7ms/step - loss: 3.4193e-05 - val_loss: 2.8078e-04
Epoch 5/50
61/61 [==============================] - 0s 7ms/step - loss: 2.9125e-05 - val_loss: 1.7012e-04
Epoch 6/50
61/61 [==============================] - 0s 7ms/step - loss: 2.3823e-05 - val_loss: 1.5562e-04
Epoch 7/50
61/61 [==============================] - 0s 7ms/step - loss: 2.0021e-05 - val_loss: 1.4795e-04
Epoch 8/50
61/61 [==============================] - 0s 6ms/step - loss: 1.8066e-05 - val_loss: 1.1065e-04
Epoch 9/50
61/61 [==============================] - 0s 6ms/step - loss: 1.5016e-05 - val_loss: 1.0072e-04
Epoch 10/50
61/61 [==============================] - 0s 6ms/step - loss: 1.3168e-05 - val_loss: 7.4579e-05
Epoch 11/50
61/61 [==============================] - 0s 7ms/step - loss: 1.1927e-05 - val_loss: 5.5885e-05
Epoch 12/50
61/61 [==============================] - 0s 8ms/step - loss: 1.3090e-05 - val_loss: 5.1345e-05
Epoch 13/50
61/61 [==============================] - 0s 7ms/step - loss: 1.1240e-05 - val_loss: 4.6895e-05
Epoch 14/50
61/61 [==============================] - 0s 7ms/step - loss: 1.0190e-05 - val_loss: 9.4728e-05
Epoch 15/50
61/61 [==============================] - 0s 7ms/step - loss: 1.0028e-05 - val_loss: 1.2240e-04
Epoch 16/50
61/61 [==============================] - 0s 7ms/step - loss: 9.6813e-06 - val_loss: 7.8087e-05
Epoch 17/50
61/61 [==============================] - 1s 9ms/step - loss: 8.0338e-06 - val_loss: 9.0063e-05
Epoch 18/50
61/61 [==============================] - 1s 9ms/step - loss: 9.0001e-06 - val_loss: 5.5214e-05
Epoch 19/50
61/61 [==============================] - 1s 9ms/step - loss: 8.3345e-06 - val_loss: 7.2945e-05
Epoch 20/50
61/61 [==============================] - 1s 9ms/step - loss: 7.5561e-06 - val_loss: 3.6442e-05
Epoch 21/50
61/61 [==============================] - 1s 9ms/step - loss: 7.9761e-06 - val_loss: 3.2679e-05
Epoch 22/50
61/61 [==============================] - 0s 6ms/step - loss: 6.5877e-06 - val_loss: 3.7000e-05
Epoch 23/50
61/61 [==============================] - 0s 6ms/step - loss: 9.1415e-06 - val_loss: 5.1531e-05
Epoch 24/50
61/61 [==============================] - 0s 6ms/step - loss: 7.1209e-06 - val_loss: 3.0517e-05
Epoch 25/50
61/61 [==============================] - 0s 5ms/step - loss: 7.3215e-06 - val_loss: 3.0134e-05
Epoch 26/50
61/61 [==============================] - 0s 6ms/step - loss: 6.4744e-06 - val_loss: 2.9631e-05
Epoch 27/50
61/61 [==============================] - 0s 6ms/step - loss: 6.3735e-06 - val_loss: 6.6620e-05
Epoch 28/50
61/61 [==============================] - 0s 6ms/step - loss: 6.0909e-06 - val_loss: 3.4219e-05
```