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

# Load the Google stock prices dataset
df = pd.read_csv('Google_Stock_Price_Train.csv')
print(df.head())

# Extract Open prices
prices = df['Open'].values
prices

# Reshape price values from 1D array to 2 D array
prices = prices.reshape(-1, 1)
prices

# Normalize the prices using Min-Max scaling
scaler = MinMaxScaler(feature_range=(0, 1))
prices_scaled = scaler.fit_transform(prices)
prices_scaled

# Split the data into train and test sets (80% train, 20% test)
train_size = int(len(prices_scaled) * 0.8)
test_size = len(prices_scaled) - train_size
train_data = prices_scaled[0:train_size]
test_data = prices_scaled[train_size:len(prices_scaled)]

# Function to create X and y datasets from time series data
def create_dataset(dataset, time_step):
    X, y = [], []
    for i in range(len(dataset)-time_step-1):
        X.append(dataset[i:(i+time_step), 0])
        y.append(dataset[i + time_step, 0])
    return np.array(X), np.array(y)

# Define the time step for sequence prediction
time_step = 60

# Create the X and y datasets
x_train, y_train = create_dataset(train_data, time_step)
x_test, y_test = create_dataset(test_data, time_step)


# Reshape the input data to be 3D (batch_size, time_step, features)
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], 1)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1], 1)

# Build the RNN model
model = Sequential([
    LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1)),
    LSTM(units=50, return_sequences=True),
    LSTM(units=50),
    Dense(units=1)
])

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

```
)

# Train the model
model.fit(x_train, y_train, epochs=100, batch_size=64)

# Predictions
y_pred = model.predict(x_test)

# Inverse scaling to get actual prices
y_pred = scaler.inverse_transform(y_pred)
y_test = scaler.inverse_transform([y_test])

# Plot the test predictions vs. actual prices
plt.figure(figsize=(10,6))
plt.plot(y_test.flatten(), label='Actual Prices', color='red')
plt.plot(y_pred.flatten(), label='Predicted Prices', color='green')
plt.title('Google Stock Prices - Test Set')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()
```