DEEP LEARNING ASSIGNMENT NO. 5

Name : Manjiri Netankar

GROUP MEMBERS :
SUPRIYA MASKAR(202201040049)
SAPNA DAHIKAMBLE(202201070065)
MANJIRI NETANKAR(202201040206)

Colab Link :
https://colab.research.google.com/drive/1cR8B567aUxFddGNSmxxUPowiKweMNKhi?usp=sharing

GitHub Link : https://github.com/supriyamaskar/LSTM_deepLearning

Experiment 5.1:
Objective: To forecast future values of a univariate time series using LSTM-based models.

```
# 1. INSTALL REQUIRED LIBRARIES
!pip install -q pandas numpy scikit-learn matplotlib tensorflow

# 2. LOAD THE EXTRACTED CSV FILE
import pandas as pd

# Make sure this path is correct based on where your CSV is extracted
df = pd.read_csv('day_wise.csv')

# 3. PREPROCESSING
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error

df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
df = df[['New cases']]

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df)

def create_sequences(data, seq_length=14):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)

seq_length = 14
X, y = create_sequences(scaled_data, seq_length)

split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
```

```python
y_train, y_test = y[:split], y[split:]

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))

# 4. BUILD AND TRAIN LSTM MODEL
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(seq_length, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error',
metrics=['mae'])

history = model.fit(X_train, y_train, epochs=50, batch_size=8,
validation_split=0.1, verbose=1)

# 5. PREDICT AND EVALUATE
y_pred = model.predict(X_test)
y_pred_inv = scaler.inverse_transform(y_pred)
y_test_inv = scaler.inverse_transform(y_test)

rmse = np.sqrt(mean_squared_error(y_test_inv, y_pred_inv))
mae = mean_absolute_error(y_test_inv, y_pred_inv)

print(f"RMSE: {rmse:.2f}")
print(f"MAE : {mae:.2f}")

# 6. PLOT PREDICTION VS ACTUAL
plt.figure(figsize=(12,6))
plt.plot(y_test_inv, label='Actual', marker='o')
plt.plot(y_pred_inv, label='Predicted', marker='x')
plt.title('Prediction vs Actual - New COVID-19 Cases')
plt.xlabel('Days')
plt.ylabel('New Cases')
plt.legend()
plt.grid(True)
plt.show()
```

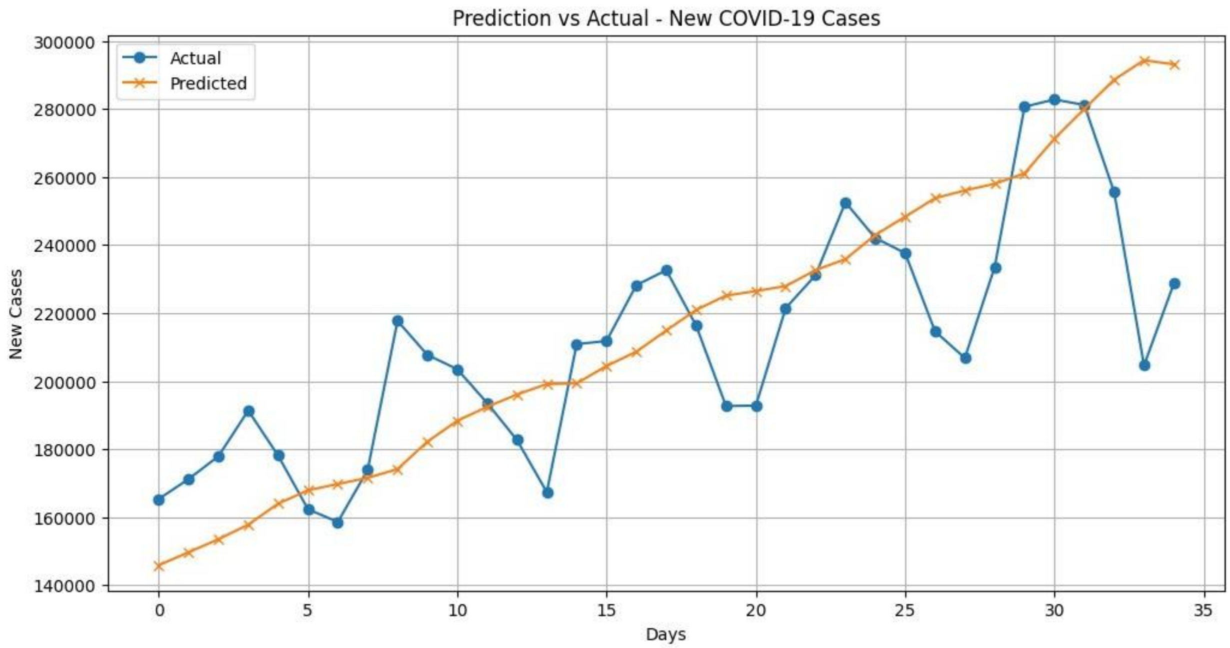
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/
rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)

Epoch 1/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 45ms/step - loss: 0.0540 - mae: 0.1807 -
val_loss: 0.1548 - val_mae: 0.3901
Epoch 2/50
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0287 - mae: 0.1439 - val_loss: 0.0692 - val_mae: 0.2584
Epoch 3/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.0111 - mae: 0.0950 - val_loss: 0.0113 - val_mae: 0.0946
Epoch 4/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - loss: 0.0068 - mae: 0.0652 - val_loss: 0.0102 - val_mae: 0.0887
Epoch 5/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 53ms/step - loss: 0.0050 - mae: 0.0587 - val_loss: 0.0055 - val_mae: 0.0601
Epoch 6/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 70ms/step - loss: 0.0036 - mae: 0.0512 - val_loss: 0.0024 - val_mae: 0.0333
Epoch 7/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 56ms/step - loss: 0.0025 - mae: 0.0446 - val_loss: 0.0028 - val_mae: 0.0376
Epoch 8/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - loss: 0.0020 - mae: 0.0391 - val_loss: 0.0040 - val_mae: 0.0513
Epoch 9/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.0019 - mae: 0.0357 - val_loss: 0.0025 - val_mae: 0.0339
Epoch 10/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0022 - mae: 0.0374 - val_loss: 0.0044 - val_mae: 0.0553
Epoch 11/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0014 - mae: 0.0294 - val_loss: 0.0028 - val_mae: 0.0382
Epoch 12/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0014 - mae: 0.0298 - val_loss: 0.0025 - val_mae: 0.0341
Epoch 13/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0015 - mae: 0.0312 - val_loss: 0.0025 - val_mae: 0.0337
Epoch 14/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0012 - mae: 0.0278 - val_loss: 0.0025 - val_mae: 0.0340
Epoch 15/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0014 - mae: 0.0288 - val_loss: 0.0026 - val_mae: 0.0356
Epoch 16/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0013 - mae: 0.0271 - val_loss: 0.0024 - val_mae: 0.0329
Epoch 17/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0013 - mae: 0.0294 - val_loss: 0.0027 - val_mae: 0.0371
Epoch 18/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0010 - mae: 0.0265 -
```

```
val_loss: 0.0025 - val_mae: 0.0340
Epoch 19/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0013 - mae: 0.0300 -
val_loss: 0.0024 - val_mae: 0.0338
Epoch 20/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0011 - mae: 0.0252 -
val_loss: 0.0029 - val_mae: 0.0397
Epoch 21/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0012 - mae: 0.0269 -
val_loss: 0.0028 - val_mae: 0.0390
Epoch 22/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 8.5924e-04 - mae:
0.0226 - val_loss: 0.0033 - val_mae: 0.0436
Epoch 23/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0012 - mae: 0.0269 -
val_loss: 0.0030 - val_mae: 0.0404
Epoch 24/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0010 - mae: 0.0260 -
val_loss: 0.0036 - val_mae: 0.0463
Epoch 25/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 9.0056e-04 - mae:
0.0232 - val_loss: 0.0028 - val_mae: 0.0386
Epoch 26/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 9.4760e-04 - mae:
0.0233 - val_loss: 0.0034 - val_mae: 0.0447
Epoch 27/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0011 - mae: 0.0273 -
val_loss: 0.0032 - val_mae: 0.0433
Epoch 28/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 8.2738e-04 - mae:
0.0222 - val_loss: 0.0028 - val_mae: 0.0381
Epoch 29/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.0011 - mae: 0.0255 -
val_loss: 0.0028 - val_mae: 0.0380
Epoch 30/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 9.2817e-04 - mae:
0.0224 - val_loss: 0.0037 - val_mae: 0.0470
Epoch 31/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - loss: 0.0011 - mae: 0.0255 -
val_loss: 0.0030 - val_mae: 0.0406
Epoch 32/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 9.8017e-04 - mae:
0.0241 - val_loss: 0.0035 - val_mae: 0.0452
Epoch 33/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0012 - mae: 0.0252 -
val_loss: 0.0037 - val_mae: 0.0470
Epoch 34/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - loss: 0.0011 - mae: 0.0248 -
val_loss: 0.0035 - val_mae: 0.0460
```

```
Epoch 35/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 9.0387e-04 - mae: 0.0234 - val_loss: 0.0037 - val_mae: 0.0471
Epoch 36/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 8.0381e-04 - mae: 0.0217 - val_loss: 0.0034 - val_mae: 0.0447
Epoch 37/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 6.6957e-04 - mae: 0.0201 - val_loss: 0.0038 - val_mae: 0.0483
Epoch 38/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0011 - mae: 0.0249 - val_loss: 0.0032 - val_mae: 0.0433
Epoch 39/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0014 - mae: 0.0280 - val_loss: 0.0033 - val_mae: 0.0442
Epoch 40/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 9.4109e-04 - mae: 0.0234 - val_loss: 0.0031 - val_mae: 0.0425
Epoch 41/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 7.6878e-04 - mae: 0.0206 - val_loss: 0.0035 - val_mae: 0.0463
Epoch 42/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 0.0011 - mae: 0.0255 - val_loss: 0.0039 - val_mae: 0.0493
Epoch 43/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 7.4514e-04 - mae: 0.0209 - val_loss: 0.0027 - val_mae: 0.0368
Epoch 44/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 8.4085e-04 - mae: 0.0224 - val_loss: 0.0032 - val_mae: 0.0437
Epoch 45/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 9.5099e-04 - mae: 0.0234 - val_loss: 0.0044 - val_mae: 0.0526
Epoch 46/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 7.9463e-04 - mae: 0.0211 - val_loss: 0.0030 - val_mae: 0.0417
Epoch 47/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 7.6148e-04 - mae: 0.0201 - val_loss: 0.0025 - val_mae: 0.0337
Epoch 48/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 9.3954e-04 - mae: 0.0245 - val_loss: 0.0029 - val_mae: 0.0401
Epoch 49/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 8.4489e-04 - mae: 0.0228 - val_loss: 0.0030 - val_mae: 0.0412
Epoch 50/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 9.0218e-04 - mae: 0.0232 - val_loss: 0.0030 - val_mae: 0.0420
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 263ms/step
```

```
RMSE:  28670.25
MAE : 21653.14
```




```
# 1. INSTALL REQUIRED LIBRARIES
!pip install -q pandas numpy scikit-learn matplotlib tensorflow

# 2. LOAD 'day_wise.csv'
import pandas as pd

df = pd.read_csv('/content/day_wise.csv')       # Adjust path if needed

# 3. PREPROCESSING
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error

df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
df = df[['New cases']]

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df)

def create_sequences(data, seq_length=14):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
```

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

seq_length = 14
X, y = create_sequences(scaled_data, seq_length)

split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

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))

# 4. BUILD AND TRAIN LSTM MODEL
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(seq_length, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error',
metrics=['mae'])

history = model.fit(X_train, y_train, epochs=50, batch_size=8,
validation_split=0.1, verbose=1)

# 5. PREDICT AND EVALUATE
y_pred = model.predict(X_test)
y_pred_inv = scaler.inverse_transform(y_pred)
y_test_inv = scaler.inverse_transform(y_test)

rmse = np.sqrt(mean_squared_error(y_test_inv, y_pred_inv))
mae = mean_absolute_error(y_test_inv, y_pred_inv)

print(f"RMSE: {rmse:.2f}")
print(f"MAE : {mae:.2f}")

# 6. PLOT PREDICTION VS ACTUAL
plt.figure(figsize=(12,6))
plt.plot(y_test_inv, label='Actual', marker='o')
plt.plot(y_pred_inv, label='Predicted', marker='x')
plt.title('Prediction vs Actual - New COVID-19 Cases')
plt.xlabel('Days')
plt.ylabel('New Cases')
plt.legend()
plt.grid(True)
plt.show()
```

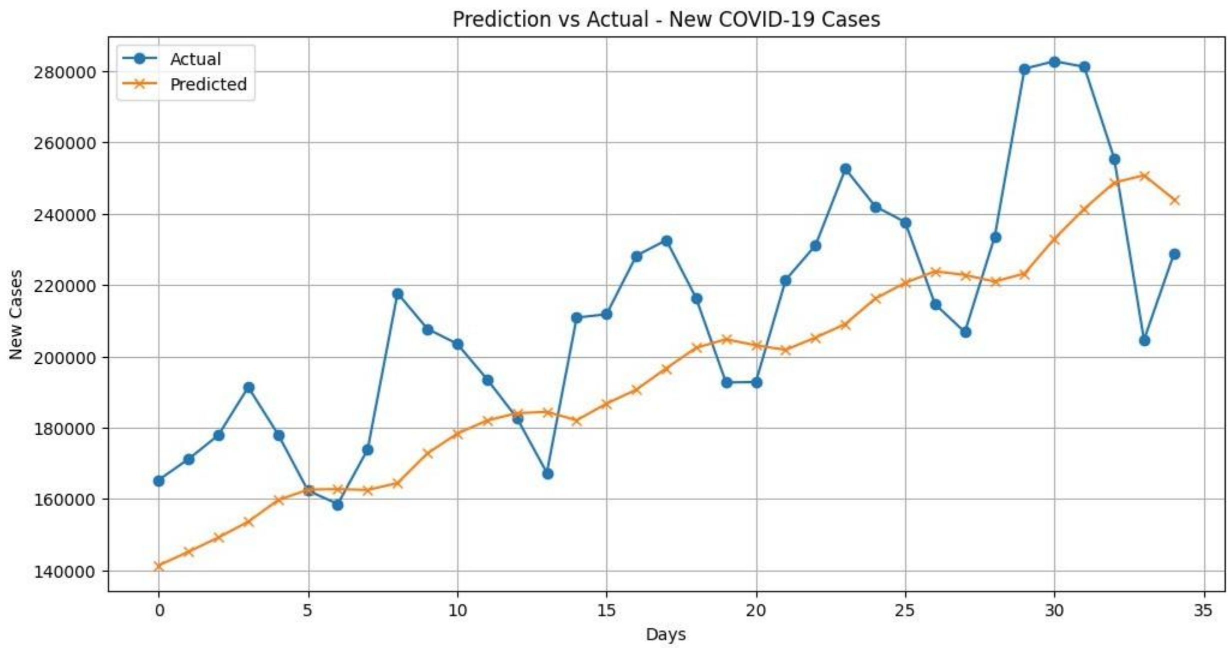
```
Epoch 1/50

/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/
rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim`
```

```
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 53ms/step - loss: 0.0415 - mae: 0.1620 -
val_loss: 0.0438 - val_mae: 0.2035
Epoch 2/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 0.0077 - mae: 0.0749 -
val_loss: 0.0079 - val_mae: 0.0745
Epoch 3/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0042 - mae: 0.0518 -
val_loss: 0.0057 - val_mae: 0.0611
Epoch 4/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0025 - mae: 0.0414 -
val_loss: 0.0028 - val_mae: 0.0371
Epoch 5/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0018 - mae: 0.0344 -
val_loss: 0.0030 - val_mae: 0.0402
Epoch 6/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0015 - mae: 0.0314 -
val_loss: 0.0026 - val_mae: 0.0345
Epoch 7/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0017 - mae: 0.0345 -
val_loss: 0.0024 - val_mae: 0.0336
Epoch 8/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0015 - mae: 0.0302 -
val_loss: 0.0026 - val_mae: 0.0360
Epoch 9/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0017 - mae: 0.0328 -
val_loss: 0.0036 - val_mae: 0.0475
Epoch 10/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0015 - mae: 0.0307 -
val_loss: 0.0025 - val_mae: 0.0335
Epoch 11/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0012 - mae: 0.0283 -
val_loss: 0.0024 - val_mae: 0.0334
Epoch 12/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0015 - mae: 0.0317 -
val_loss: 0.0029 - val_mae: 0.0393
Epoch 13/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0013 - mae: 0.0293 -
val_loss: 0.0041 - val_mae: 0.0507
Epoch 14/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0015 - mae: 0.0308 -
val_loss: 0.0025 - val_mae: 0.0337
Epoch 15/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0013 - mae: 0.0285 -
val_loss: 0.0027 - val_mae: 0.0379
Epoch 16/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0010 - mae: 0.0260 -
```

```
val_loss: 0.0033 - val_mae: 0.0442
Epoch 17/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0012 - mae: 0.0280 -
val_loss: 0.0025 - val_mae: 0.0342
Epoch 18/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0011 - mae: 0.0260 -
val_loss: 0.0026 - val_mae: 0.0358
Epoch 19/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0011 - mae: 0.0269 -
val_loss: 0.0031 - val_mae: 0.0423
Epoch 20/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0010 - mae: 0.0255 -
val_loss: 0.0025 - val_mae: 0.0340
Epoch 21/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0012 - mae: 0.0277 -
val_loss: 0.0035 - val_mae: 0.0459
Epoch 22/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0012 - mae: 0.0305 -
val_loss: 0.0029 - val_mae: 0.0398
Epoch 23/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0011 - mae: 0.0257 -
val_loss: 0.0037 - val_mae: 0.0477
Epoch 24/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 9.0265e-04 - mae:
0.0253 - val_loss: 0.0030 - val_mae: 0.0411
Epoch 25/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0010 - mae: 0.0246 -
val_loss: 0.0043 - val_mae: 0.0513
Epoch 26/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0012 - mae: 0.0254 -
val_loss: 0.0040 - val_mae: 0.0493
Epoch 27/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 9.0281e-04 - mae:
0.0238 - val_loss: 0.0031 - val_mae: 0.0414
Epoch 28/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0012 - mae: 0.0263 -
val_loss: 0.0044 - val_mae: 0.0525
Epoch 29/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0010 - mae: 0.0251 -
val_loss: 0.0033 - val_mae: 0.0439
Epoch 30/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0013 - mae: 0.0276 -
val_loss: 0.0033 - val_mae: 0.0434
Epoch 31/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 9.5889e-04 - mae:
0.0241 - val_loss: 0.0051 - val_mae: 0.0574
Epoch 32/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0011 - mae: 0.0258 -
val_loss: 0.0033 - val_mae: 0.0434
```

```
Epoch 33/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0010 - mae: 0.0248 -
val_loss: 0.0039 - val_mae: 0.0483
Epoch 34/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 9.0899e-04 - mae:
0.0230 - val_loss: 0.0041 - val_mae: 0.0502
Epoch 35/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0011 - mae: 0.0248 -
val_loss: 0.0035 - val_mae: 0.0454
Epoch 36/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 9.1811e-04 - mae:
0.0230 - val_loss: 0.0050 - val_mae: 0.0567
Epoch 37/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0011 - mae: 0.0267 -
val_loss: 0.0028 - val_mae: 0.0388
Epoch 38/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0011 - mae: 0.0258 -
val_loss: 0.0041 - val_mae: 0.0495
Epoch 39/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0011 - mae: 0.0268 -
val_loss: 0.0040 - val_mae: 0.0492
Epoch 40/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 8.4081e-04 - mae:
0.0223 - val_loss: 0.0049 - val_mae: 0.0559
Epoch 41/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 9.1679e-04 - mae:
0.0240 - val_loss: 0.0037 - val_mae: 0.0476
Epoch 42/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - loss: 9.9494e-04 - mae:
0.0243 - val_loss: 0.0042 - val_mae: 0.0511
Epoch 43/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0010 - mae: 0.0240 -
val_loss: 0.0035 - val_mae: 0.0457
Epoch 44/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 8.6277e-04 - mae:
0.0216 - val_loss: 0.0038 - val_mae: 0.0477
Epoch 45/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 0.0011 - mae: 0.0260 -
val_loss: 0.0053 - val_mae: 0.0592
Epoch 46/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 9.3478e-04 - mae:
0.0231 - val_loss: 0.0033 - val_mae: 0.0438
Epoch 47/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - loss: 0.0011 - mae: 0.0247 -
val_loss: 0.0037 - val_mae: 0.0469
Epoch 48/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - loss: 8.4657e-04 - mae:
0.0224 - val_loss: 0.0035 - val_mae: 0.0459
Epoch 49/50
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 9.7624e-04 - mae: 0.0237 - val_loss: 0.0093 - val_mae: 0.0827
Epoch 50/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0011 - mae: 0.0252 - val_loss: 0.0034 - val_mae: 0.0446
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 213ms/step
RMSE: 28223.84
MAE : 24078.97
```



5.2. Sequence Text Prediction using LSTM

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences, to_categorical
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense
import re

# Load Dataset
df = pd.read_csv('/content/sample_data/Jokes_subreddit_threads.csv')
# Adjust path as needed
df.dropna(inplace=True)

# Preprocess and Tokenize
text_data_list = df['selftext'].astype(str).tolist()        # Using
'selftext' for jokes
```

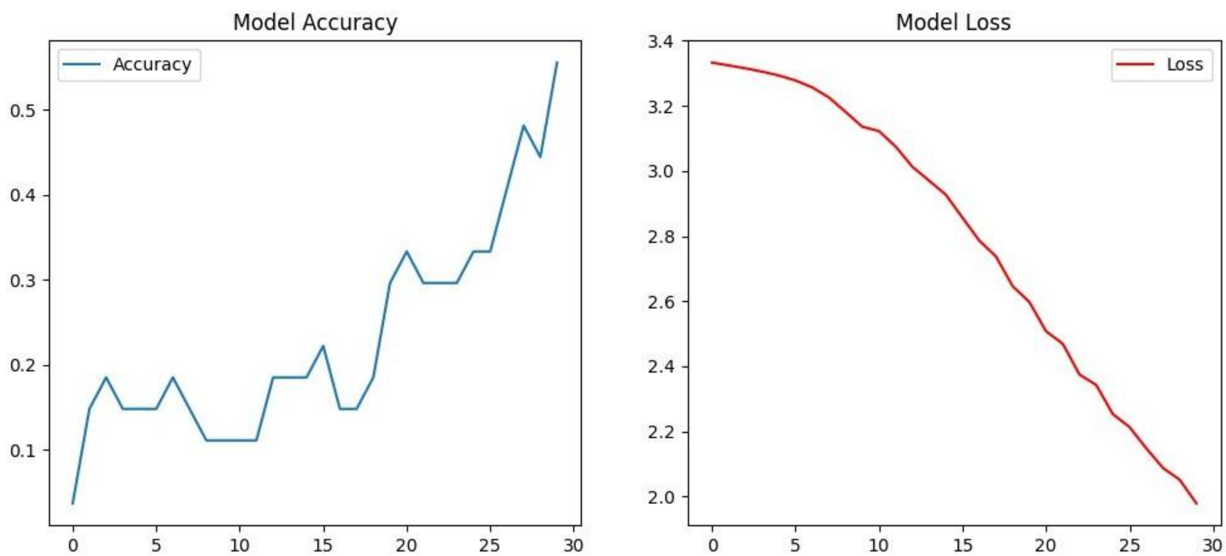
```
processed_text_data = [re.sub(r'[^a-zA-Z\s.,]', '', text) for text in
text_data_list]
text_data = ' '.join(processed_text_data)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(processed_text_data)       # Tokenize individual
jokes
total_words = len(tokenizer.word_index) + 1
print("Total words in vocab:", total_words)

# Create Sequences
input_sequences = []
tokens = tokenizer.texts_to_sequences([text_data])[0]

if len(tokens) > 1:
    for i in range(1, len(tokens)):
        n_gram_sequence = tokens[:i + 1]
        input_sequences.append(n_gram_sequence)

    max_seq_len = max([len(seq) for seq in input_sequences])
    input_sequences = pad_sequences(input_sequences,
maxlen=max_seq_len, padding='pre')

    X = input_sequences[:, :-1]
    y = input_sequences[:, -1]
    y = to_categorical(y, num_classes=total_words)

else:
    print("Not enough tokens to create sequences. Check your input
data or tokenization process.")
    X = np.array([])
    y = np.array([])
    max_seq_len = 0

# Build and Train LSTM Model
model = Sequential()
if max_seq_len > 0:
    model.add(Embedding(total_words, 100, input_length=max_seq_len -
1))
    model.add(LSTM(150))
    model.add(Dense(total_words, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.summary()

    history = model.fit(X, y, epochs=30, verbose=1)

    # Plot Accuracy and Loss
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
```

```
    plt.plot(history.history['accuracy'], label='Accuracy')
    plt.legend()
    plt.title("Model Accuracy")

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Loss', color='red')
    plt.legend()
    plt.title("Model Loss")
    plt.show()

    # Text Generation Function
    def generate_text(seed_text, next_words, model, max_seq_len):
        for _ in range(next_words):
            token_list = tokenizer.texts_to_sequences([seed_text])[0]
            token_list = pad_sequences([token_list],
maxlen=max_seq_len - 1, padding='pre')
            predicted = model.predict(token_list, verbose=0)
            predicted_word_index = np.argmax(predicted, axis=-1)[0]

            for word, index in tokenizer.word_index.items():
                if index == predicted_word_index:
                    seed_text += " " + word
                    break
        return seed_text

    # Generate Sample Text
    print("\nGenerated  Text:\n")
    print(generate_text("why did the chicken", 20, model,
max_seq_len))

else:
    print("Model cannot be built and trained due to insufficient
data.")

Total words in vocab: 28

<ipython-input-6-66999a8beb7e>:12: DtypeWarning: Columns (2) have
mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv('/content/sample_data/Jokes_subreddit_threads.csv')
# Adjust path as needed
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embeddin
g.py:90: UserWarning: Argument `input_length` is deprecated. Just
remove it.
  warnings.warn(

Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
```

```
├──────────────────────────────────┼───────────────────────┼─────────────
────────┤
│ embedding (Embedding)           │ ?                      │     0
(unbuilt) │
├──────────────────────────────────┼───────────────────────┼─────────────
────────┤
│ lstm (LSTM)                     │ ?                      │     0
(unbuilt) │
├──────────────────────────────────┼───────────────────────┼─────────────
────────┤
│ dense (Dense)                   │ ?                      │     0
(unbuilt) │
└──────────────────────────────────┴───────────────────────┴─────────────
────────┘
 Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)
Epoch 1/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 0.0370 - loss: 3.3331
Epoch 2/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.1481 - loss:
3.3245
Epoch 3/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step - accuracy: 0.1852 - loss:
3.3155
Epoch 4/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step - accuracy: 0.1481 - loss:
3.3055
Epoch 5/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - accuracy: 0.1481 - loss:
3.2935
Epoch 6/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.1481 - loss:
3.2781
Epoch 7/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step - accuracy: 0.1852 - loss:
3.2570
Epoch 8/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - accuracy: 0.1481 - loss:
3.2260
Epoch 9/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - accuracy: 0.1111 - loss:
3.1814
Epoch 10/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.1111 - loss:
3.1360
```

```
Epoch 11/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.1111 - loss:
3.1226
Epoch 12/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step - accuracy: 0.1111 - loss:
3.0747
Epoch 13/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step - accuracy: 0.1852 - loss:
3.0129
Epoch 14/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.1852 - loss:
2.9704
Epoch 15/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 160ms/step - accuracy: 0.1852 - loss:
2.9271
Epoch 16/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.2222 - loss:
2.8563
Epoch 17/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step - accuracy: 0.1481 - loss:
2.7863
Epoch 18/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.1481 - loss:
2.7374
Epoch 19/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step - accuracy: 0.1852 - loss:
2.6460
Epoch 20/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step - accuracy: 0.2963 - loss:
2.5975
Epoch 21/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step - accuracy: 0.3333 - loss:
2.5075
Epoch 22/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step - accuracy: 0.2963 - loss:
2.4693
Epoch 23/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step - accuracy: 0.2963 - loss:
2.3746
Epoch 24/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step - accuracy: 0.2963 - loss:
2.3427
Epoch 25/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step - accuracy: 0.3333 - loss:
2.2530
Epoch 26/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 125ms/step - accuracy: 0.3333 - loss:
2.2130
Epoch 27/30
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.4074 - loss:
2.1478
Epoch 28/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.4815 - loss:
2.0875
Epoch 29/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step - accuracy: 0.4444 - loss:
2.0514
Epoch 30/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 139ms/step - accuracy: 0.5556 - loss:
1.9788
```




```
Generated Text:

why did the chicken has a a input input input punchline punchline your
joke joke joke posting and if its been been been posted
```

5.3: Sequence Text Classification using LSTM

```
#   Step 1: Install & Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

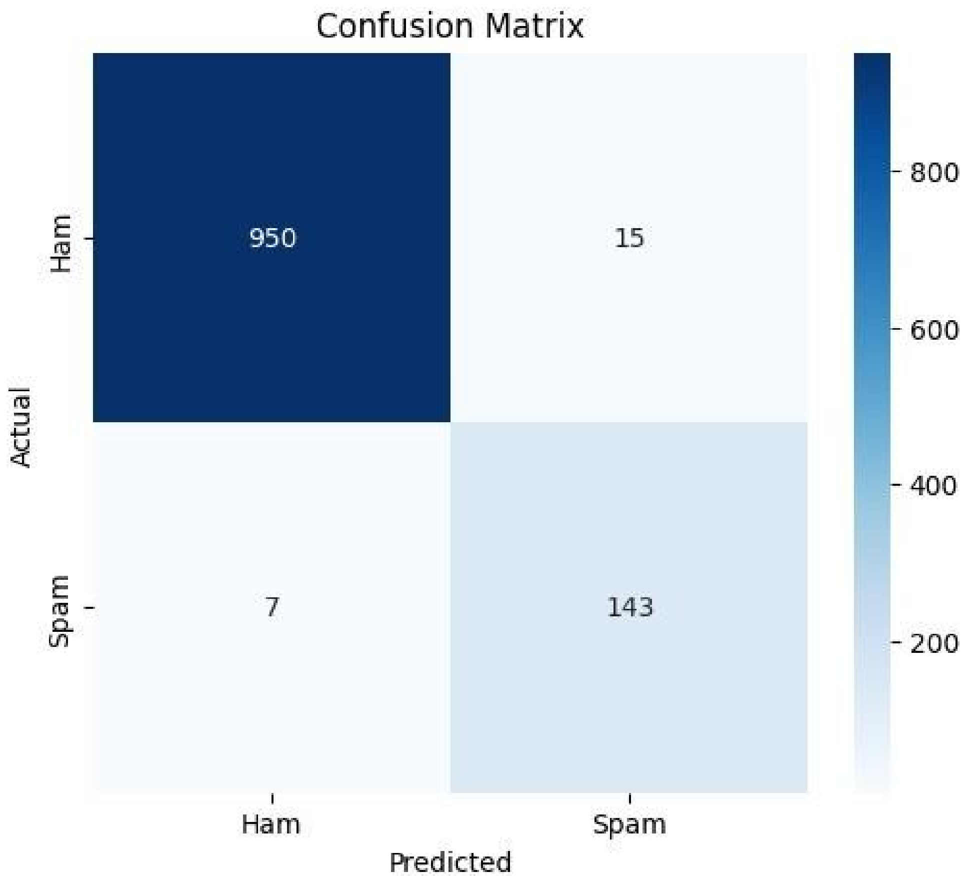
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

#   Step 2: Load and Prepare Dataset
df = pd.read_csv("/content/sample_data/spam.csv", encoding='latin-1')
df = df[['v1', 'v2']]
df.columns = ['label', 'text']
df.dropna(inplace=True)

#   Encode Labels: 'ham' = 0, 'spam' = 1
label_encoder = LabelEncoder()
df['label_num'] = label_encoder.fit_transform(df['label'])

#   Check class distribution
print(df['label'].value_counts())

#   Text Preprocessing (basic lowercase only)
df['text'] = df['text'].str.lower()

#   Step 3: Tokenization and Padding
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['text'])
vocab_size = len(tokenizer.word_index) + 1

sequences = tokenizer.texts_to_sequences(df['text'])
max_len = max([len(seq) for seq in sequences])  # Maximum sequence
length
X = pad_sequences(sequences, maxlen=max_len)
y = df['label_num'].values

#   Step 4: Split into Train/Test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

#   Step 5: Build and Compile LSTM Model
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=128))  # Removed
deprecated input_length
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Explicitly build the model to fix "unbuilt" summary issue
model.build(input_shape=(None, max_len))

#   View model architecture
model.summary()
```

```python
#  Step 6: Train the Model
history = model.fit(X_train, y_train, epochs=5, batch_size=64,
validation_data=(X_test, y_test), verbose=1)

#   Step 7: Evaluate Model
y_pred  =  (model.predict(X_test)  >  0.5).astype("int32")

#   Classification Report
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred, target_names=['ham',
'spam']))

#   Accuracy Score
acc = accuracy_score(y_test, y_pred)
print("Accuracy:",  acc)

#   Step 8: Confusion Matrix
cm  =  confusion_matrix(y_test,  y_pred)

#   Plot Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Ham',
'Spam'], yticklabels=['Ham', 'Spam'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
label
ham      4825
spam      747
Name: count, dtype: int64
```

```
Model: "sequential_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                          ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━━━┩
│ embedding_2 (Embedding)               │ (None, 189, 128)                      │
1,141,888 │
├─────────────────────────────────────────┼──────────────────────────────────────────────────┼──────────
───────┤
│ lstm_2 (LSTM)                         │ (None, 128)                           │
131,584 │
├─────────────────────────────────────────┼──────────────────────────────────────────────────┼──────────
───────┤
│ dense_2 (Dense)                       │ (None, 1)                             │
```

```
129  |
└──────────────────────────────────────────┴──────────────────────────────┴─────────────
────────┘

 Total params: 1,273,601 (4.86 MB)

 Trainable params: 1,273,601 (4.86 MB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/5
70/70 ━━━━━━━━━━━━━━━━━━━━ 47s 613ms/step - accuracy: 0.8886 - loss:
0.3466 - val_accuracy: 0.9794 - val_loss: 0.0804
Epoch 2/5
70/70 ━━━━━━━━━━━━━━━━━━━━ 80s 582ms/step - accuracy: 0.9875 - loss:
0.0439 - val_accuracy: 0.9848 - val_loss: 0.0619
Epoch 3/5
70/70 ━━━━━━━━━━━━━━━━━━━━ 41s 585ms/step - accuracy: 0.9946 - loss:
0.0195 - val_accuracy: 0.9830 - val_loss: 0.0750
Epoch 4/5
70/70 ━━━━━━━━━━━━━━━━━━━━ 40s 570ms/step - accuracy: 0.9991 - loss:
0.0065 - val_accuracy: 0.9830 - val_loss: 0.0660
Epoch 5/5
70/70 ━━━━━━━━━━━━━━━━━━━━ 41s 568ms/step - accuracy: 0.9998 - loss:
0.0017 - val_accuracy: 0.9803 - val_loss: 0.0718
35/35 ━━━━━━━━━━━━━━━━━━━━ 3s 76ms/step

Classification Report:
              precision    recall  f1-score   support

         ham       0.99      0.98      0.99       965
        spam       0.91      0.95      0.93       150

    accuracy                           0.98      1115
   macro avg       0.95      0.97      0.96      1115
weighted avg       0.98      0.98      0.98      1115

Accuracy:  0.9802690582959641
```



Declaration
I, Manjiri Netankar, confirm that the work submitted in this assignment is my own and has been completed following academic integrity guidelines.
Github link:
Dataset Link : https://github.com/supriyamaskar/LSTM_deepLearning
Signature: Manjiri Netankar