```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.12/dist-packages (1.7.4.5)
Requirement already satisfied: bleach in /usr/local/lib/python3.12/dist-packages (from kaggle) (6.3.0)
Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.12/dist-packages (from kaggle) (2026.1.4)
Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.12/dist-packages (from kaggle) (3.4.4)
Requirement already satisfied: idna in /usr/local/lib/python3.12/dist-packages (from kaggle) (3.11)
Requirement already satisfied: protobuf in /usr/local/lib/python3.12/dist-packages (from kaggle) (5.29.6)
Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.12/dist-packages (from kaggle) (2.9.0.post0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.12/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from kaggle) (2.32.4)
Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.12/dist-packages (from kaggle) (75.2.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.12/dist-packages (from kaggle) (1.17.0)
Requirement already satisfied: text-unidecode in /usr/local/lib/python3.12/dist-packages (from kaggle) (1.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from kaggle) (4.67.3)
Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.12/dist-packages (from kaggle) (2.5.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.12/dist-packages (from kaggle) (0.5.1)
```

Importing the Dependencies

```
import os
import json

from zipfile import ZipFile
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

Close

```
**Data Collection- Kaggle API**
# (OPTIONAL IF USING DATASET DOWNLOADED DIRECTLY FROM LINK THEREFORE COMMENT
IT BEFORE USING)
```

Data Collection- Kaggle API

(OPTIONAL IF USING DATASET DOWNLOADED DIRECTLY FROM LINK THEREFORE COMMENT IT BEFORE USING)

```
kaggle_dictionary = json.load(open("/content/kaggle (1).json"))
```

```
kaggle_dictionary.keys()
```

```
dict_keys(['username', 'key'])
```

```
# setup kaggle credentials as environment variables
os.environ["KAGGLE_USERNAME"] = kaggle_dictionary["username"]
os.environ["KAGGLE_KEY"] = kaggle_dictionary["key"]
```

```
!kaggle datasets download -d lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
```

```
Dataset URL: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
License(s): other
Downloading imdb-dataset-of-50k-movie-reviews.zip to /content
  0% 0.00/25.7M [00:00<?, ?B/s]
100% 25.7M/25.7M [00:00<00:00, 741MB/s]
```

```
!ls
```

```
 imdb-dataset-of-50k-movie-reviews.zip  'kaggle (1).json'   sample_data
```

```
# unzip the dataset file
with ZipFile("imdb-dataset-of-50k-movie-reviews.zip", "r") as zip_ref:
  zip_ref.extractall()
```

```
!ls
```

```
'IMDB Dataset.csv'                     'kaggle (1).json'
 imdb-dataset-of-50k-movie-reviews.zip   sample_data
```

Loading teh Dataset

```
data = pd.read_csv("/content/IMDB Dataset.csv")
```

```
data.shape
```

```
(50000, 2)
```

```
data.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Next steps: Generate code with `data` New interactive sheet

```
data.tail()
```

	review	sentiment
49995	I thought this movie did a down right good job...	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative
49997	I am a Catholic taught in parochial elementary...	negative
49998	I'm going to have to disagree with the previou...	negative
49999	No one expects the Star Trek movies to be high...	negative

```
data["sentiment"].value_counts()
```

sentiment	count
positive	25000
negative	25000

dtype: int64

```
data.replace({"sentiment": {"positive": 1, "negative": 0}}, inplace=True)
```

```
/tmp/ipython-input-2568826810.py:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, expli
  data.replace({"sentiment": {"positive": 1, "negative": 0}}, inplace=True)
```

```
data.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	1
1	A wonderful little production. The...	1
2	I thought this was a wonderful way to spend ti...	1
3	Basically there's a family where a little boy ...	0
4	Petter Mattei's "Love in the Time of Money" is...	1

Next steps: Generate code with data | New interactive sheet

```
data["sentiment"].value_counts()
```

sentiment	count
1	25000
0	25000

dtype: int64

```
# split data into training data and test data
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)
```

```
print(train_data.shape)
print(test_data.shape)
```

```
(40000, 2)
(10000, 2)
```

Data Preprocessing

```
# Tokenize text data
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(train_data["review"])
X_train = pad_sequences(tokenizer.texts_to_sequences(train_data["review"]), maxlen=200)
X_test = pad_sequences(tokenizer.texts_to_sequences(test_data["review"]), maxlen=200)
```

```
print(X_train)
```

```
[[1935    1 1200 ...  205  351 3856]
 [   3 1651  595 ...   89  103    9]
 [   0    0    0 ...    2  710   62]
 ...
 [   0    0    0 ... 1641    2  603]
 [   0    0    0 ...  245  103  125]
 [   0    0    0 ...   70   73 2062]]
```

```
print(X_test)
```

```
[[   0    0    0 ...  995  719  155]
 [  12  162   59 ...  380    7    7]
 [   0    0    0 ...   50 1088   96]
 ...
 [   0    0    0 ...  125  200 3241]
 [   0    0    0 ... 1066    1 2305]
 [   0    0    0 ...    1  332   27]]
```

```
Y_train = train_data["sentiment"]
Y_test = test_data["sentiment"]
```

```
print(Y_train)
```

```
39087    0
30893    0
45278    1
16398    0
13653    0
        ..
11284    1
44732    1
38158    0
860      1
15795    1
Name: sentiment, Length: 40000, dtype: int64
```

LSTM - Long Short-Term Memory

```
# build the model

model = Sequential()
model.add(Embedding(input_dim=5000, output_dim=128, input_length=200))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation="sigmoid"))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

```
# Now this will show the correct model summary and parameter count
model.summary()
```

Model: "sequential"

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)

```
# compile the model
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
```

Training the Model

```
model.fit(X_train, Y_train, epochs=5, batch_size=64, validation_split=0.2)
```

```
Epoch 1/5
500/500 ━━━━━━━━━━━━━━━━━━━━ 212s 408ms/step - accuracy: 0.7064 - loss: 0.5446 - val_accuracy: 0.8342 - val_loss: 0.3782
Epoch 2/5
500/500 ━━━━━━━━━━━━━━━━━━━━ 206s 413ms/step - accuracy: 0.8606 - loss: 0.3385 - val_accuracy: 0.8064 - val_loss: 0.4405
Epoch 3/5
500/500 ━━━━━━━━━━━━━━━━━━━━ 210s 421ms/step - accuracy: 0.8610 - loss: 0.3343 - val_accuracy: 0.8524 - val_loss: 0.3510
Epoch 4/5
500/500 ━━━━━━━━━━━━━━━━━━━━ 255s 408ms/step - accuracy: 0.8929 - loss: 0.2704 - val_accuracy: 0.8683 - val_loss: 0.3352
Epoch 5/5
500/500 ━━━━━━━━━━━━━━━━━━━━ 206s 412ms/step - accuracy: 0.9084 - loss: 0.2368 - val_accuracy: 0.8745 - val_loss: 0.3285
<keras.src.callbacks.history.History at 0x7fa498a83fb0>
```

Model Evaluation

```
loss, accuracy = model.evaluate(X_test, Y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 37s 118ms/step - accuracy: 0.8786 - loss: 0.3194
Test Loss: 0.3181324601173401
Test Accuracy: 0.8823999762535095
```

Building a Predictive System

```
def predict_sentiment(review):
  # tokenize and pad the review
  sequence = tokenizer.texts_to_sequences([review])
  padded_sequence = pad_sequences(sequence, maxlen=200)
  prediction = model.predict(padded_sequence)
  sentiment = "positive" if prediction[0][0] > 0.5 else "negative"
  return sentiment
```

```
# Case 1:
new_review = "This movie was fantastic. I loved it."
sentiment = predict_sentiment(new_review)
print(f"The sentiment of the review is: {sentiment}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 424ms/step
The sentiment of the review is: positive
```

```
# Case 2:
new_review = "This movie was not that good"
sentiment = predict_sentiment(new_review)
print(f"The sentiment of the review is: {sentiment}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step
The sentiment of the review is: negative
```

```
#Case 3:
new_review = "This movie was ok but not that good."
sentiment = predict_sentiment(new_review)
print(f"The sentiment of the review is: {sentiment}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step
The sentiment of the review is: negative
```

```
review = input("Enter your movie review: ")
sentiment = predict_sentiment(review)
print(f"The sentiment of the review is: {sentiment}")
```

```
Enter your movie review:
```

```
# from google.colab import files
# files.download('sentiment_model.h5')
# files.download('tokenizer.pkl')
```