```
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```
/kaggle/input/book-recommendation-dataset/Ratings.csv
/kaggle/input/book-recommendation-dataset/Users.csv
/kaggle/input/book-recommendation-dataset/classicRec.png
/kaggle/input/book-recommendation-dataset/Books.csv
/kaggle/input/book-recommendation-dataset/DeepRec.png
/kaggle/input/book-recommendation-dataset/recsys_taxonomy2.png
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

# Load dataset
data =
pd.read_csv('/kaggle/input/book-recommendation-dataset/Books.csv',
low_memory=False)
data = data.iloc[:5000]

print("Columns in dataset:", data.columns)


data['metadata'] = data['Book-Title'].fillna('') + ' ' + data['Book-
Author'].fillna('')
```

```
data['metadata'] = data['metadata'].str.lower()

# Model 1: TF-IDF with Cosine Similarity
print("Model 1: TF-IDF with Cosine Similarity")

def tfidf_recommendation(book_title):
    # TF-IDF Vectorizer
    tfidf = TfidfVectorizer(stop_words='english')
    tfidf_matrix = tfidf.fit_transform(data['metadata'])

    # Cosine Similarity
    cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

    # Book Index
    indices = pd.Series(data.index, index=data['Book-
Title']).drop_duplicates()

    # Recommendation
    idx = indices[book_title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:6]
    book_indices = [i[0] for i in sim_scores]

    recommended_books = data['Book-Title'].iloc[book_indices]
    recommended_metadata = data['metadata'].iloc[book_indices]
    input_metadata = data['metadata'].iloc[idx]

    return input_metadata, recommended_books, recommended_metadata

book_title = input("Enter book title: ")
input_metadata, recommended_books, recommended_metadata =
tfidf_recommendation(book_title)

print("Input Book Metadata:")
print(input_metadata)
print("\nRecommended Books:")
print(recommended_books)
print("\nMetadata of Recommended Books:")
print(recommended_metadata)



Columns in dataset: Index(['ISBN', 'Book-Title', 'Book-Author', 'Year-
Of-Publication', 'Publisher',
       'Image-URL-S', 'Image-URL-M', 'Image-URL-L'],
      dtype='object')
Model 1: TF-IDF with Cosine Similarity

Enter book title:  Timeline
```

```
Input Book Metadata:
timeline michael crichton

Recommended Books:
2430              Prey
1947    The Lost World
27            Airframe
2446          Airframe
207              Congo
Name: Book-Title, dtype: object

Metadata of Recommended Books:
2430              prey michael crichton
1947    the lost world michael crichton
27            airframe michael crichton
2446          airframe michael crichton
207              congo michael crichton
Name: metadata, dtype: object
```

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

# Load and preprocess dataset
data =
pd.read_csv('/kaggle/input/book-recommendation-dataset/Books.csv',
low_memory=False)
data = data.iloc[:1000]
data['metadata'] = data['Book-Title'].fillna('') + ' ' + data['Book-
Author'].fillna('')
data['metadata'] = data['metadata'].str.lower()

# Tokenization and padding
tokenizer = Tokenizer()
tokenizer.fit_on_texts(data['metadata'])
sequences = tokenizer.texts_to_sequences(data['metadata'])
word_index = tokenizer.word_index
data_padded = pad_sequences(sequences, maxlen=100)

# Label encoding
label_encoder = LabelEncoder()
data['encoded_labels'] = label_encoder.fit_transform(data['Book-
Title'])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(data_padded,
```

```
data['encoded_labels'], test_size=0.2, random_state=42)

# Build the RNN model
model = Sequential()
model.add(Embedding(input_dim=len(word_index) + 1, output_dim=100))
model.add(LSTM(128, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(128))
model.add(Dropout(0.2))
model.add(Dense(100, activation='relu'))
model.add(Dense(len(data['encoded_labels'].unique()),
activation='softmax'))

model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=64,
validation_data=(X_test, y_test))

# Making predictions
def rnn_recommendation(input_metadata):

    input_seq = tokenizer.texts_to_sequences([input_metadata])
    input_padded = pad_sequences(input_seq, maxlen=100)
    predictions = model.predict(input_padded)
    top_indices = predictions.argsort()[0][-10:][::-1]  # Increase
range to 10
    recommended_books = label_encoder.inverse_transform(top_indices)

    # Remove duplicates while maintaining order
    recommended_books = list(dict.fromkeys(recommended_books))
    recommended_books = recommended_books[:5]  # Keep top 5 unique
recommendations

    recommended_metadata = data.loc[data['Book-
Title'].isin(recommended_books), 'metadata']
    return recommended_books, recommended_metadata


input_metadata = input("Enter metadata of the book: ")
recommended_books, recommended_metadata =
rnn_recommendation(input_metadata)

print("Recommended Books:")
print(recommended_books)
print("\nMetadata of Recommended Books:")
print(recommended_metadata)
```

```
Epoch 1/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - accuracy: 0.0000e+00 - loss:
6.8904 - val_accuracy: 0.0000e+00 - val_loss: 6.8962
```

```
Epoch 2/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0000e+00 - loss:
6.8863 - val_accuracy: 0.0000e+00 - val_loss: 6.9162
Epoch 3/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0000e+00 - loss:
6.8781 - val_accuracy: 0.0000e+00 - val_loss: 7.0954
Epoch 4/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0031 - loss:
6.8342 - val_accuracy: 0.0000e+00 - val_loss: 7.6144
Epoch 5/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0016 - loss:
6.7735 - val_accuracy: 0.0000e+00 - val_loss: 8.2339
Epoch 6/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0016 - loss:
6.6934 - val_accuracy: 0.0000e+00 - val_loss: 8.8826
Epoch 7/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0011 - loss:
6.5288 - val_accuracy: 0.0000e+00 - val_loss: 9.8015
Epoch 8/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0070 - loss:
6.2881 - val_accuracy: 0.0000e+00 - val_loss: 10.8946
Epoch 9/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.0058 - loss:
6.0456 - val_accuracy: 0.0000e+00 - val_loss: 12.0786
Epoch 10/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0052 - loss:
5.8704 - val_accuracy: 0.0000e+00 - val_loss: 12.9510

Enter metadata of the book:  Timeline

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 162ms/step
Recommended Books:
['The Street Lawyer', 'Hannibal', 'Angels &amp;amp; Demons', 'The
Testament', 'The Door to December']

Metadata of Recommended Books:
18              the testament john grisham
52          the street lawyer john grisham
118          angels &amp;amp; demons dan brown
239          angels &amp;amp; demons dan brown
410                 hannibal thomas harris
588         the street lawyer john grisham
850    the door to december dean r. koontz
877                 hannibal thomas harris
886    the door to december dean r. koontz
949             the testament john grisham
Name: metadata, dtype: object

def tfidf_recommendation(book_title):
    # TF-IDF Vectorizer
```

```
    tfidf = TfidfVectorizer(stop_words='english')
    tfidf_matrix = tfidf.fit_transform(data['metadata'])

    # Cosine Similarity
    cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

    # Book Index
    indices = pd.Series(data.index, index=data['Book-
Title']).drop_duplicates()

    # Ensure the book title exists in the dataset
    if book_title not in indices:
        print(f"Book title '{book_title}' not found in the dataset.")
        return None, None, None

    idx = indices[book_title]
    sim_scores = cosine_sim[idx]

    # Extract similarity scores for all books
    sim_scores = list(enumerate(sim_scores))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:6]  # Top 5 recommendations, excluding
the book itself
    book_indices = [i[0] for i in sim_scores]

    recommended_books = data['Book-Title'].iloc[book_indices]
    recommended_metadata = data['metadata'].iloc[book_indices]
    input_metadata = data['metadata'].iloc[idx]

    return input_metadata, recommended_books, recommended_metadata

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Tokenization and padding
tokenizer = Tokenizer()
tokenizer.fit_on_texts(data['metadata'])
sequences = tokenizer.texts_to_sequences(data['metadata'])
word_index = tokenizer.word_index
data_padded = pad_sequences(sequences, maxlen=100)

# Label encoding
label_encoder = LabelEncoder()
data['encoded_labels'] = label_encoder.fit_transform(data['Book-
Title'])
X_train, X_test, y_train, y_test = train_test_split(data_padded,
data['encoded_labels'], test_size=0.2, random_state=42)
from tensorflow.keras.models import Sequential
```

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

model = Sequential()
model.add(Embedding(input_dim=len(word_index) + 1, output_dim=100))
model.add(LSTM(128, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(128))
model.add(Dropout(0.2))
model.add(Dense(100, activation='relu'))
model.add(Dense(len(data['encoded_labels'].unique()),
activation='softmax'))

model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=10, batch_size=64,
validation_data=(X_test, y_test))
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()


def evaluate_rnn(input_metadata, k=5):
    recommended_books, _ = rnn_recommendation(input_metadata)

    # Replace with actual relevant books for your dataset
    ground_truth = set(["To Kill a Mockingbird", "Other Relevant Book
1", "Other Relevant Book 2"])

    recommended_books_set = set(recommended_books)
    relevant_recommendations =
ground_truth.intersection(recommended_books_set)

    precision = len(relevant_recommendations) /
len(recommended_books_set)
    recall = len(relevant_recommendations) / len(ground_truth)

    print(f"RNN Model -> Precision@{k}: {precision:.2f}, Recall@{k}:
{recall:.2f}")
    return precision, recall

evaluate_rnn("To Kill a Mockingbird")  # Input metadata or book title
```

```
import seaborn as sns

def plot_similarity(book_title):
    tfidf = TfidfVectorizer(stop_words='english')
    tfidf_matrix = tfidf.fit_transform(data['metadata'])
    cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

    indices = pd.Series(data.index, index=data['Book-
Title']).drop_duplicates()
    idx = indices[book_title]
    sim_scores = cosine_sim[idx]

    # Plot top 10 most similar books
    top_indices = sim_scores.argsort()[-5:][::-1]
    top_scores = sim_scores[top_indices]
    top_books = data['Book-Title'].iloc[top_indices]

    sns.barplot(x=top_scores, y=top_books, palette="viridis")
    plt.title("Cosine Similarity Scores")
    plt.xlabel("Similarity Score")
    plt.ylabel("Book Titles")
    plt.show()

plot_similarity("To Kill a Mockingbird")

tfidf_precision, tfidf_recall = evaluate_tfidf("To Kill a
Mockingbird")
rnn_precision, rnn_recall = evaluate_rnn("To Kill a Mockingbird")

print("\nModel Comparison:")
print(f"TF-IDF -> Precision: {tfidf_precision:.2f}, Recall:
{tfidf_recall:.2f}")
print(f"RNN -> Precision: {rnn_precision:.2f}, Recall:
{rnn_recall:.2f}")
```