```python
import streamlit as st
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

#  Title & Description
st.set_page_config(page_title="Movie Recommendation System",
layout="wide")
st.title("🎬 Movie Recommendation System")
st.markdown("Find movies similar to your favorites using TF-IDF and cosine
similarity.")

#  Load Data and Model (Cached)
@st.cache_data(show_spinner=True)
def load_data_and_model():
    df = pd.read_csv("movies.csv")

    # Automatically detect the text column
    possible_cols = ["overview", "description", "tags", "genres"]
    text_col = next((col for col in possible_cols if col in df.columns),
None)

    if not text_col:
        st.error("No suitable text column found. Please ensure your CSV
has 'overview', 'description', 'tags', or 'genres'.")
        st.stop()

    df = df.dropna(subset=["title", text_col])
    tfidf = TfidfVectorizer(stop_words="english", max_features=5000)
    tfidf_matrix = tfidf.fit_transform(df[text_col].astype(str))

    indices = pd.Series(df.index,
index=df["title"].str.lower()).drop_duplicates()
    return df, tfidf_matrix, indices


#  Load Cached Objects
df, tfidf_matrix, indices = load_data_and_model()

#  Recommendation Function
```

```
def get_recommendations(title, df, tfidf_matrix, indices, top_n=10):
    title = title.lower().strip()
    if title not in indices:
        st.error("Movie not found. Please check spelling or try another
title.")
        return pd.DataFrame()

    idx = indices[title]
    sim_scores = linear_kernel(tfidf_matrix[idx], tfidf_matrix).flatten()
    sim_indices = sim_scores.argsort()[-top_n-1:][::-1]
    sim_indices = [i for i in sim_indices if i != idx]

    # Identify which column was used for TF-IDF
    possible_cols = ["overview", "description", "tags", "genres"]
    text_col = next((col for col in possible_cols if col in df.columns),
None)

    # Safely return recommendations
    if text_col:
        return df.iloc[sim_indices][["title", text_col]]
    else:
        return df.iloc[sim_indices][["title"]]


#  Streamlit UI
st.subheader("Search for a Movie")
user_input = st.text_input("Enter a movie title:", "")

if st.button("Recommend"):
    if user_input.strip():
        recommendations = get_recommendations(user_input, df,
tfidf_matrix, indices)
        if len(recommendations) > 0:
            st.success(f"Top {len(recommendations)} movies similar to
'{user_input}':")
            text_col = [col for col in ["overview", "description", "tags",
"genres"] if col in recommendations.columns]
            text_col = text_col[0] if text_col else None

            for _, row in recommendations.iterrows():
```

```
                st.markdown(f"### {row['title']}")
                if text_col:
                    st.write(row[text_col])
                st.markdown("---")

    else:
        st.warning("Please enter a movie title first.")

#  Footer
st.markdown("---")
st.caption("Built with Streamlit & scikit-learn")
```

Day 1:

Basic libraries

import pandas as pd

import numpy as np

Visualization (optional today)

import matplotlib.pyplot as plt

import seaborn as sns

Show all columns

pd.set_option('display.max_columns', None)

Load the datasets

```
movies = pd.read_csv("movies.csv")

ratings = pd.read_csv("ratings.csv")


# Look at first few rows

print(movies.head())

print(ratings.head())


# Merge datasets

df = ratings.merge(movies, on='movieId')

df.head()


df.isnull().sum()

df.duplicated().sum()


df.drop_duplicates(inplace=True)


df.info()

df.describe()


# Number of unique users and movies

print("Total Users :", df['userId'].nunique())

print("Total Movies:", df['movieId'].nunique())


plt.figure(figsize=(6,4))
```

```
sns.countplot(x='rating', data=df)

plt.title("Distribution of Ratings")

plt.show()


top_movies = df['title'].value_counts().head(10)

print(top_movies)



top_movies.plot(kind='barh', figsize=(7,4), title="Top 10 Most Rated Movies")

plt.show()


df['genres'] = df['genres'].str.split('|')

df_exploded = df.explode('genres')

plt.figure(figsize=(8,5))

sns.countplot(y='genres', data=df_exploded,
order=df_exploded['genres'].value_counts().index[:10])

plt.title("Top 10 Movie Genres")

plt.show()
```

Day 2:

```
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine_similarity

from sklearn.metrics.pairwise import linear_kernel



# Combine movie title and genres into one feature for simplicity

df['genres'] = df['genres'].fillna('').astype(str)

df['combined_features'] = df['title'] + " " + df['genres']

df['combined_features'].head()



movies = df.copy()  # assuming 'df' is your dataframe containing 'title' and 'genres'



# --- Step 1: Preprocessing ---

# Remove duplicate movie titles

movies = movies.drop_duplicates(subset='title', keep='first').reset_index(drop=True)

# Fill NaN in 'genres' (important for TF-IDF)

movies['genres'] = movies['genres'].fillna('')



# --- Step 2: TF-IDF Vectorization ---
```

```
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(movies['genres'])

# --- Step 3: Build index mapping ---
indices = pd.Series(movies.index, index=movies['title']).drop_duplicates()

# --- Step 4: Define recommendation function ---
def get_recommendations(title):
        if title not in indices:
      return "Movie not found in dataset!"

        idx = indices[title]
   sim_scores = linear_kernel(tfidf_matrix[idx], tfidf_matrix).flatten()
   sim_scores_indices = sim_scores.argsort()[::-1]
   sim_scores_indices = sim_scores_indices[sim_scores_indices != idx]
   top_indices = sim_scores_indices[:10]

   return movies['title'].iloc[top_indices]

# --- Step 5: Test the function ---
print(get_recommendations('Toy Story (1995)'))
```