```
import streamlit as st
import rottentomatoes as rt
import numpy as np
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans
import matplotlib.pyplot as plt
from pyspark.sql.functions import col, when
from pyspark.ml.feature import Imputer
from pyspark.sql.types import StructType, StructField, StringType, IntegerType
import sqlite3
import pandas as pd
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import OneHotEncoder

def get_df(movie_name,genre):

  # query ="select * from movies where genre1 IN "+ str(tuple(genre)) +" OR genre2
IN " +str(tuple(genre)) +"AND title!='"+movie_name +"'"
  formatted_genre = ','.join(["'" + item + "'" for item in genre])
  query = "SELECT * FROM movies WHERE (genre1 IN (" + formatted_genre + ") OR
genre2 IN (" + formatted_genre + ")) AND title NOT LIKE '%" + movie_name + "%'"


  print(query)
  conn = sqlite3.connect('movies.db')
  cursor = conn.cursor()
  cursor.execute(query)
  result = cursor.fetchall()
  print(result)
  df = pd.DataFrame(result)
  # print(cursor.description)
  df.columns = [col[0] for col in cursor.description]
  return df
from pyspark.sql.functions import col, when

def clustering(pandas_df):


  spark = SparkSession.builder.appName("MovieRecommender").getOrCreate()
  df = spark.createDataFrame(pandas_df)

  imputer = Imputer(strategy="mean", inputCols=["tomatometer", "audience_score"],
outputCols=["tomatometer", "audience_score"])

  # Convert string columns to numerical types
  df = df.withColumn("tomatometer", col("tomatometer").cast("float"))
  df = df.withColumn("audience_score", col("audience_score").cast("float"))
  df = df.withColumn("rating", when(col("rating") != "",
col("rating")).otherwise("NA"))
  df = df.withColumn("genre2", when(col("genre2") != "",
col("genre2")).otherwise("NA"))


  df = imputer.fit(df).transform(df)


str_obj=StringIndexer(inputCols=["genre1","genre2","actor1","actor2","rating"],outp
```

```
utCols=["new_genre1","new_genre2","new_actor1","new_actor2","new_rating"])

onehot_obj=OneHotEncoder(inputCols=["new_genre1","new_genre2","new_actor1","new_act
or2","new_rating"],outputCols=["One_Hot_genre1","One_Hot_genre2","One_Hot_actor1","
One_Hot_actor2","One_Hot_rating"])


  df = str_obj.fit(df).transform(df)
  df = onehot_obj.fit(df).transform(df)



  # Step 2: Feature Engineering
  feature_cols = ["tomatometer",
"audience_score","One_Hot_genre1","One_Hot_genre2","One_Hot_actor1","One_Hot_actor2
","One_Hot_rating"]
  assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
  newdata = assembler.transform(df)
  # Step 3: Model Training
  no_of_clusters = 3
  kmeans = KMeans(featuresCol="features").setK(no_of_clusters)
  predictions = kmeans.fit(newdata).transform(newdata)
  # predictions.show(10, truncate=False)

  #getting titles from each cluster
  cluster_titles={}
  for i in range(no_of_clusters):
      titles = predictions.filter(predictions.prediction ==
i).select("title").rdd.flatMap(lambda x: x).collect()[:5]
      #print(f"Set {i} length: {len(titles)}")
      # print(f"Cluster {i}: {titles}")
      cluster_titles['Set '+str(i+1)]=titles

  # print(cluster_titles)
  # titles.sear


  return cluster_titles

# MAIN WEB APP CODE

def fetch_rt_score(movie_title):
    try:
        my_dict={}
        my_dict["score"] = rt.tomatometer(movie_title)
        my_dict["actors"] = rt.actors(movie_title, max_actors=5)
        my_dict["audience_score"] = rt.audience_score(movie_title)
        my_dict["genres"] = rt.genres(movie_title)
        return my_dict
    except Exception as e:
        return f"Error fetching score for {movie_title}: {str(e)}"

def main():
    st.title("CinFind - Movie Recommeder")

    # Input field for movie name
    movie_name = st.text_input("Enter a movie name:", "Top Gun")

    # Fetch the score when the user clicks the button
```

```
    if st.button("Fetch Info"):
        info = fetch_rt_score(movie_name)
        score = info["score"]
        print(score)
        actors = info["actors"]
        genres = info['genres']
        df = get_df(str(movie_name).lower(),genres)
        titles = clustering(df)


        print(info)


        st.write(f"Rotten Tomatoes Score for '{movie_name}': {score}")
        st.write(f"Audience Rotten Tomatoes Score for '{movie_name}':
{info['audience_score']}")
        st.write(f"Actors for {movie_name} : {actors}")
        st.write(f"Genre for {movie_name} : {genres}")
        st.write(f"Recommended Movies are : \n" )

        st.table(titles)


if __name__ == "__main__":
    main()
```