# Movie Recommender System - Documentation

**Github Link:** [Kaviya-Np/Movie-Recommender-System: Movie-Recommender-System]

**Project Link:** [Streamlit]

## Introduction

This project is a Movie Recommendation System built using **Python, Pandas, Scikit-Learn, and Streamlit**. The system suggests similar movies based on a given input using **content-based filtering** with **cosine similarity**. The deployment was done using **Streamlit Community Cloud**.

---

## Dataset Preprocessing Steps

The dataset was processed and cleaned using the following steps:

### 1. Importing Necessary Libraries

```
import pandas as pd  # For data manipulation
import numpy as np  # For numerical operations
import ast  # For converting string representations of lists into actual lists
from sklearn.feature_extraction.text import CountVectorizer  # For text vectorization
from sklearn.metrics.pairwise import cosine_similarity  # For computing similarity between vectors
import pickle  # For saving and loading models
```

### 2. Loading the Dataset

```
movies = pd.read_csv("movies.csv")  # Reading the movies dataset
credits = pd.read_csv("credits.csv")  # Reading the credits dataset
```

### 3. Merging Datasets

```
movies = movies.merge(credits, on='id')  # Combining the datasets using 'id'
```

### 4. Selecting Relevant Columns

```
movies = movies[['id', 'title', 'overview', 'genres', 'keywords', 'cast', 'crew']]
```

### 5. Handling Missing Data

```
movies.dropna(inplace=True)  # Removing rows with missing values
```

### 6. Converting Text Data into Useful Format

```
def convert(obj):
    try:
        return [i['name'] for i in ast.literal_eval(obj)]  # Extracting only the 'name' from each
dictionary
    except:
        return []
movies['genres'] = movies['genres'].apply(convert)
movies['keywords'] = movies['keywords'].apply(convert)
```

## 7. Extracting Important Features from Crew and Cast

```
def extract_director(obj):
    try:
        for i in ast.literal_eval(obj):
            if i['job'] == 'Director':
                return i['name']
        return ""
    except:
        return ""
movies['director'] = movies['crew'].apply(extract_director)

def extract_top_actors(obj):
    try:
        actors = [i['name'] for i in ast.literal_eval(obj)[:3]]  # Taking top 3 actors
        return actors
    except:
        return []
movies['cast'] = movies['cast'].apply(extract_top_actors)
```

## 8. Creating a New 'Tags' Column

```
movies['tags'] = movies['overview'] + movies['genres'] + movies['keywords'] + movies['cast'] +
movies['director'].apply(lambda x: [x])
```

## 9. Converting Lists to Strings

```
movies['tags'] = movies['tags'].apply(lambda x: " ".join(x))
movies['tags'] = movies['tags'].str.lower()  # Converting all text to lowercase
```

## 10. Text Vectorization (Converting Text to Numerical Data)

```
cv = CountVectorizer(max_features=5000, stop_words='english')
vectors = cv.fit_transform(movies['tags']).toarray()
```

## 11. Stemming (Reducing Words to Their Root Form)

```
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
def stem(text):
    return " ".join([stemmer.stem(word) for word in text.split()])
movies['tags'] = movies['tags'].apply(stem)
```

## 12. Computing Cosine Similarity

```
similarity = cosine_similarity(vectors)
```

## 13. Saving the Processed Data

```
pickle.dump(movies, open("movies_dict.pkl", "wb"))
pickle.dump(similarity, open("similarity.pkl", "wb"))
```

---

# Deploying the Model with Streamlit

## App.py Implementation

- Loads the preprocessed `movies_dict.pkl` and `similarity.pkl` files.
- Uses a dropdown to select a movie.
- Fetches similar movies based on cosine similarity.
- Fetches posters from the TMDB API.
- Create a Procfile with [web: sh setup.sh && streamlit run app.py] content
- Create setup.sh with below content

```
mkdir -p ~/.streamlit/

echo "\
[server]\n\
port = $PORT\n\
enableCORS = false\n\
headless = true\n\
\n\
" > ~/.streamlit/config.toml
```

- Create .gitignore with [venv] in it.

### Function to Fetch Movie Posters

```python
def fetch_poster(movie_id):
    response = requests.get(f'https://api.themoviedb.org/3/movie/{movie_id}?api_key=YOUR_API_KEY')
    data = response.json()
    return "https://image.tmdb.org/t/p/w500/" + data['poster_path']
```

### Recommendation Function

```python
def recommend(movie):
    movie_index = movies[movies['title'] == movie].index[0]
    distances = similarity[movie_index]
    movies_list = sorted(list(enumerate(distances)), reverse=True, key=lambda x: x[1])[1:6]
    recommended_movies = []
    recommended_movies_posters = []
    for i in movies_list:
        movie_id = movies.iloc[i[0]].movie_id
        recommended_movies.append(movies.iloc[i[0]].title)
        recommended_movies_posters.append(fetch_poster(movie_id))
    return recommended_movies, recommended_movies_posters
```

---

# Deployment Steps for the Movie Recommender System on Streamlit Community Cloud

## 1. Install Required Libraries

```
pip install -r requirements.txt
```

If you don't have a `requirements.txt` file, generate it using:

```
pip freeze > requirements.txt
```

## 2. Upload the Project to GitHub

```
git init  # Initialize a new Git repository
git add .  # Add all files to staging
git commit -m "Initial commit"
git branch -M main  # Rename the default branch to 'main'
git remote add origin <repository_url>  # Connect your local repo to GitHub
git push -u origin main  # Push the code to GitHub
```

### 3. Deploy on Streamlit Community Cloud

1. Go to Streamlit Community Cloud.
2. Log in with GitHub.
3. Click on "New App".
4. Select the GitHub repository.
5. Choose the branch (`main`).
6. Specify the entry point file as `app.py`.
7. Click **Deploy**.

My Streamlit app is now live at: 📌 **Streamlit**

### 4. Updating the App

```
git add .
git commit -m "Updated app"
git push origin main
```

Then, go to **Streamlit**, navigate to the app, and click **“Rerun”** or **"Reboot"** to apply the changes.

---

## Conclusion

This project successfully recommends movies based on content similarity. The deployment was done using **Streamlit Community Cloud**, making it accessible to users online. Future improvements can include **collaborative filtering** or **deep learning-based recommendation techniques**.