Here's the **completed and refined project documentation** with a **clear, professional, and structured** approach.

ML-Based Movie Recommendation System

Goal

Our goal is to build a **content-based movie recommendation system** that suggests movies based on similar features (genres, keywords, cast, and crew).

Project Structure

```
movies-recommender-system/
├── app.py                      # Streamlit web app
├── movie-recommender.ipynb      # Jupyter Notebook for model building
├── ProjectDocument.odt          # Project documentation
├── similarity.pkl               # Precomputed similarity matrix
├── dataset/                     # Contains dataset files
│   ├── tmdb_5000_movies.csv
│   ├── tmdb_5000_credits.csv
├── movies.pkl                   # Processed movie dataset
├── requirements.txt              # Required Python libraries
```

Project Workflow

1. Data Collection

- **Dataset Name**: TMDB 5000 Movie Dataset
- **Source**: https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata
- **Contents**: Movie titles, genres, keywords, cast, crew, overview, ratings, etc.

2. Data Preprocessing

- **Merging Datasets**: `movies` and `credits` merged on `title`.
- **Feature Selection**: Extracted relevant columns:

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

- **Handling Missing Data**: Dropped missing values in the `overview` column.
- **Data Formatting**: Converted `genres`, `keywords`, `cast`, and `crew` from dictionaries to lists.

3. Feature Engineering

- Created **movie "tags"** by combining:

  ```
  overview + genres + keywords + cast + crew
  ```

- Converted text to lowercase and removed spaces in multi-word phrases.
- Applied **Stemming** using `PorterStemmer` to normalize words (e.g., "loved" → "love").

4. Model Building

- **Vectorization**: Converted text tags into numerical vectors using **CountVectorizer (max_features=5000, stop_words='english')**.
- **Similarity Calculation**: Used **Cosine Similarity** to measure movie similarity.

5. Web App Development

- **Framework**: Streamlit
- **Features**:
    - Movie selection via dropdown
    - Fetch movie posters dynamically using TMDB API
    - Display top **5 similar movies** with posters
    - Integrated **Lottie animation** for an engaging UI

6. Deployment

- **Platform**: Heroku (or Streamlit Cloud)

Data Preprocessing Steps

Handling JSON-like Features (Genres, Keywords, Cast, Crew)

- Some features (e.g., `genres`) were stored as dictionaries within lists:

```
[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}]
```

- To extract meaningful values, we converted them into lists:

```
import ast
def fconvert(obj):
    return [i['name'] for i in ast.literal_eval(obj)]
movies['genres'] = movies['genres'].apply(fconvert)
```

Extracting Director from Crew

- The `crew` column contained multiple roles; we filtered only the **Director**:

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

Creating "Tags" for Each Movie

- Combined **overview, genres, keywords, cast, and crew**:

```
movies['tags'] = movies['overview'] + movies['genres'] + movies['keywords'] +
movies['cast'] + movies['crew']
```

- Converted tags into lowercase and applied **stemming**:

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

Model Implementation (Jupyter Notebook Code)

Convert Text Data to Vectors

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

Compute Movie Similarities

```
from sklearn.metrics.pairwise import cosine_similarity
similarity = cosine_similarity(vectors)
```

Recommendation Function

```
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]

    for i in movies_list:
        print(movies.iloc[i[0]].title)
```

Example:

```
recommend('Avatar')
```

Output:

```
Aliens vs Predator: Requiem
Aliens
Falcon Rising
Independence Day
Titan A.E.
```

Save Processed Data for Web App

```
import pickle as pkl
pkl.dump(movies, open('movies.pkl', 'wb'))
pkl.dump(similarity, open('similarity.pkl', 'wb'))
```

Web App Implementation (app.py)

Features

Interactive Movie Selection
Movie Poster Fetching via TMDB API
5 Movie Recommendations with Posters
Lottie Animation for Better UI

Streamlit App Code

```
import streamlit as st
import pickle as pkl
import pandas as pd
import requests
from streamlit_lottie import st_lottie

# Load Data
movies = pkl.load(open("movies.pkl", 'rb'))
similarity = pkl.load(open("similarity.pkl", 'rb'))

# Function to fetch movie poster
def fetch_poster(movie_id):
    url = f'https://api.themoviedb.org/3/movie/{movie_id}?
api_key=YOUR_TMDB_API_KEY&language=en-US'
    response = requests.get(url)
    data = response.json()
    return f"http://image.tmdb.org/t/p/w500/{data['poster_path']}"

# Function to recommend movies
def recommend_movie(movie):
    movie_index = movies[movies['title'] == movie].index[0]
    distances = sorted(list(enumerate(similarity[movie_index])), reverse=True,
key=lambda x: x[1])

    recommended_movies = []
    recommended_posters = []
    for i in distances[1:6]:
        movie_id = movies.iloc[i[0]].id
        recommended_movies.append(movies.iloc[i[0]].title)
        recommended_posters.append(fetch_poster(movie_id))

    return recommended_movies, recommended_posters

# UI Components
st.title("   Movie Recommendation System")
selected_movie_name = st.selectbox("Choose a movie:", movies['title'].values)
```

```
if st.button("Recommend"):
    names, posters = recommend_movie(selected_movie_name)
    col1, col2, col3, col4, col5 = st.columns(5)

    for i, col in enumerate([col1, col2, col3, col4, col5]):
        with col:
            st.text(names[i])
            st.image(posters[i])
```

Future Enhancements

- **Add Movie Duration**
- **Hybrid Recommendation (Content + Collaborative Filtering)**
- **Genre-Based Filtering**
- **Trending & Popular Movie Suggestions**

Conclusion

This **Movie Recommendation System** provides **accurate and engaging recommendations** based on content similarity. The **interactive web app** enhances user experience by displaying **movie posters & animations**, making it visually appealing.

Deployed on Streamlit Cloud!