AI Movie Recommender - Full Code Implementation

Installation

```
# Install necessary packages
!pip install scikit-surprise gradio
```

Import Libraries

```
# Import required libraries
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from surprise import Dataset, Reader, KNNBasic
from surprise.model_selection import train_test_split
import gradio as gr
```

Load and Merge Data

```
# Load MovieLens 100k dataset
movies =
pd.read_csv("https://raw.githubusercontent.com/syedmohdali/MovieLens-Dataset-Recommendat
ion-System/main/movies.csv")
ratings =
pd.read_csv("https://raw.githubusercontent.com/syedmohdali/MovieLens-Dataset-Recommendat
ion-System/main/ratings.csv")
data = pd.merge(ratings, movies, on="movieId")
```

Content-Based Filtering

```
# Content-based filtering - TF-IDF on genres
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(movies['genres'])
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
indices = pd.Series(movies.index, index=movies['title']).drop_duplicates()

def content_based_recommend(title, n=5):
    idx = indices.get(title)
    if idx is None:
        return ["Movie not found"]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:n+1]
    movie_indices = [i[0] for i in sim_scores]
    return movies['title'].iloc[movie_indices].tolist()
```

Collaborative Filtering

```
# Collaborative filtering using Surprise
```

AI Movie Recommender - Full Code Implementation

```
reader = Reader(rating_scale=(0.5, 5.0))
surprise_data = Dataset.load_from_df(data[['userId', 'movieId', 'rating']], reader)
trainset, testset = train_test_split(surprise_data, test_size=0.2, random_state=42)

sim_options = {'name': 'cosine', 'user_based': True}
algo = KNNBasic(sim_options=sim_options)
algo.fit(trainset)

def collab_recommend(user_id, n=5):
    user_movies = data[data['userId'] == user_id]['movieId'].unique()
    all_movies = data['movieId'].unique()
    not_rated = [movie for movie in all_movies if movie not in user_movies]
    predictions = [algo.predict(user_id, movie_id) for movie_id in not_rated]
    predictions.sort(key=lambda x: x.est, reverse=True)
    top_movie_ids = [pred.iid for pred in predictions[:n]]
    top_titles = movies[movies['movieId'].isin(top_movie_ids)]['title'].tolist()
    return top_titles
```

Hybrid Recommendation & Gradio UI

```
# Hybrid recommendation function
def hybrid_recommend(user_id, fav_movie, n=5):
    cb = content_based_recommend(fav_movie, n=10)
    cf = collab_recommend(user_id, n=10)
    hybrid = list(pd.Series(cb + cf).drop_duplicates())[:n]
    return hybrid

# Gradio Interface
def recommend(user_id, fav_movie):
    try:
        user_id = int(user_id)
    except:
        return ["Invalid User ID"]
    return hybrid_recommend(user_id, fav_movie)

movie_list = movies['title'].unique().tolist()

gr.Interface(
    fn=recommend,
    inputs=[
        gr.Textbox(label="User ID"),
        gr.Dropdown(choices=movie_list, label="Favorite Movie")
    ],
    outputs=gr.List(label="Recommended Movies"),
    title="AI Movie Recommender",
    description="Enter your User ID and favorite movie to get personalized movie
recommendations."
).launch()
```