```
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.metrics.pairwise import cosine_similarity
import re

def load_data(movies_path, ratings_path):
    # Load only required columns to optimize memory
    movies_df = pd.read_csv(movies_path, usecols=["movieId", "title"])
    ratings_df = pd.read_csv(ratings_path, usecols=["userId", "movieId", "rating"])

    # Remove years from movie titles using vectorized string operations
    movies_df["title"] = movies_df["title"].str.replace(r"\s*\(\d{4}\)", "", regex=True)

    # Merge datasets
    return ratings_df.merge(movies_df, on="movieId")

def create_user_movie_matrix(data):
    # Pivot table to create user-item matrix
    user_movie_matrix = data.pivot_table(index="userId", columns="title", values="rating", aggfunc="mean")

    # Convert to a sparse matrix for efficiency
    user_movie_sparse = csr_matrix(user_movie_matrix.fillna(0).values)

    return user_movie_matrix, user_movie_sparse

def compute_similarity(user_movie_sparse, user_movie_matrix):
    # Compute cosine similarity (more efficient with sparse matrices)
    movie_similarity_sparse = cosine_similarity(user_movie_sparse.T, dense_output=False)

    # Convert to a DataFrame
    return pd.DataFrame.sparse.from_spmatrix(movie_similarity_sparse, index=user_movie_matrix.columns, columns=user_movie_matrix.col

def recommend_movies(movie_similarity_df, user_movie_matrix, movie_name, n=5):
    if movie_name not in movie_similarity_df.index:
        return f"Movie '{movie_name}' not found in dataset."

    # Get top N similar movies efficiently
    similar_movies = movie_similarity_df[movie_name].nlargest(n+1).iloc[1:]

    # Retrieve average ratings
    avg_ratings = user_movie_matrix[similar_movies.index].mean()

    # Return recommendations with ratings
    return pd.DataFrame({"Movie": similar_movies.index, "Average Rating": avg_ratings.values})

# Define file paths (replace with actual paths)
movies_path = "/content/movies.csv"
ratings_path = "/content/ratings.csv.csv"

# Load and process data
data = load_data(movies_path, ratings_path)
user_movie_matrix, user_movie_sparse = create_user_movie_matrix(data)
movie_similarity_df = compute_similarity(user_movie_sparse, user_movie_matrix)

# Get user input for movie recommendation
movie_name = input("Enter a movie name to get recommendations: ")
n_recommendations = int(input("Enter the number of recommendations: "))

# Display results
print(recommend_movies(movie_similarity_df, user_movie_matrix, movie_name, n_recommendations))
```

```
Enter a movie name to get recommendations: Toy Story
Enter the number of recommendations: 4
                              Movie  Average Rating
0       Independence Day (a.k.a. ID4)        3.374783
1  Star Wars: Episode IV - A New Hope        4.139803
2                  Back to the Future        3.932658
3                         Toy Story 2        3.800509
```

8. Post-Experiments Exercise

A. Extended Theory: (Soft Copy)

1. Explain in detail the Problem statement and methodology used to perform recommendation in the experiment

ANS: Recommendation systems are widely used in various domains such as e-commerce, streaming platforms, and online education to suggest relevant items to users. The goal of this experiment is to develop a recommendation system that provides personalized recommendations using machine learning techniques such as collaborative filtering and content-based filtering.

Methodology:

1. Data Collection:
    - Gather data related to user preferences, such as ratings, purchases, or interactions with items.
    - The dataset can be obtained from sources like MovieLens, Amazon reviews, or manually created datasets.
2. Data Preprocessing:
    - Clean the dataset by handling missing values and duplicates.
    - Convert categorical data into numerical format (if necessary).
3. Feature Engineering:
    - Extract useful features such as TF-IDF scores for text-based recommendations.
    - Use similarity metrics like cosine similarity or Pearson correlation.

4. Recommendation Techniques:
    - Content-Based Filtering:
      Recommends items based on their content similarity with previously liked items.
      Uses TF-IDF and cosine similarity to find related items.

B. Questions:

1. Explain in short cosine similarity used for recommendation

ANS:

Cosine similarity is a metric used to measure how similar two items or users are by calculating the cosine of the angle between their feature vectors. It is given by:

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\| \times \|B\|}$$

Where:

- AAA and BBB are vector representations of two items or users.

- A · BA \cdot BA · B is the dot product of the vectors.
- ∥ A ∥ \|A\| ∥ A ∥ and ∥ B ∥ \|B\| ∥ B ∥ are the magnitudes of the vectors.

It is used in recommendation systems to find items similar to a given item (content-based \filtering) or users with similar preferences (collaborative filtering).

2. Explain TFIDF (Term Frequency Inverse document frequency) similarity matrix
ANS: TF-IDF is a technique used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It is used in text-based recommendation systems to find similar items based on content.

$$\text{TF-IDF} = \text{TF} \times \text{IDF}$$

Where:

- **Term Frequency (TF):** Measures how often a term appears in a document.
  TF=Number of times term appears in a documentTotal terms in the documentTF=Total terms in the documentNumber of times term appears in a document
- **Inverse Document Frequency (IDF):** Reduces the weight of common words across all documents.
  IDF=logTotal number of documentsNumber of documents containing the termIDF=logNumber of documents containing the termTotal number of documents

Use in Recommendations:

- TF-IDF helps in identifying important words in a document (e.g., movie descriptions, book summaries).

- The similarity matrix is built using cosine similarity on TF-IDF vectors to recommend items based on textual descriptions.