UNIVERSITY SCHOOL OF AUTOMATION & ROBOTICS (USAR)

Guru Gobind Singh Indraprastha University, East Delhi Campus, Surajmal Vihar, Delhi 110092

Recommendation System Lab

(ARD453)

Submitted to:

Mr. Rishu Goel

Asst. Professor, USAR

Submitted By:

Pratham Kumar

AI-DS(B-1)

00119011921

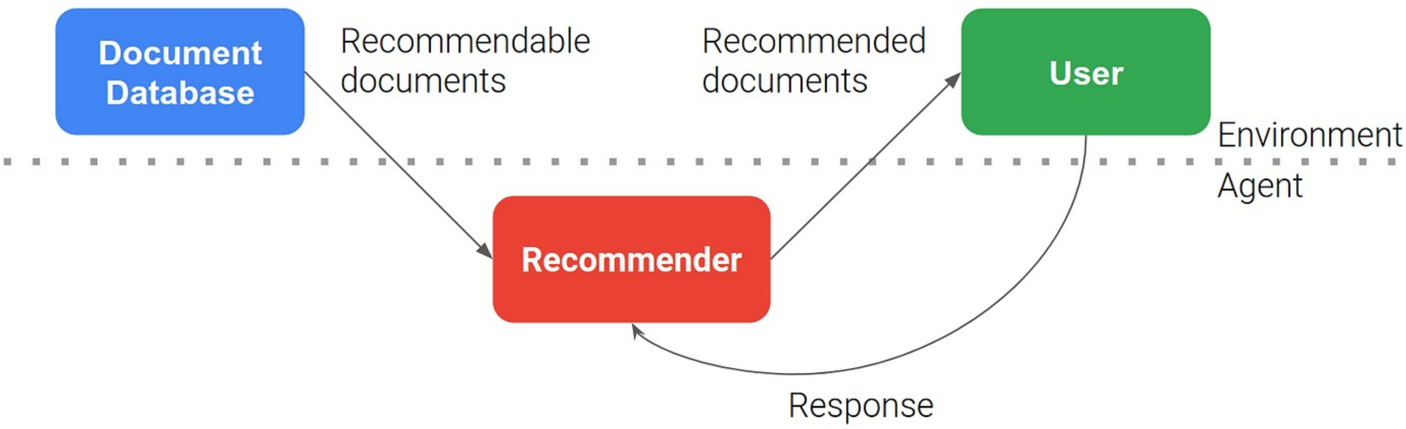
Practical No 1 : Study and analyze the tool RecSim.

RecSim: A Configurable Simulation Platform for Recommender Systems

Overview of RecSim

RecSim simulates a recommender agent's interaction with an environment consisting of a user model, a document model and a user choice model. The agent interacts with the environment by recommending sets or lists of documents (known as slates) to users, and has access to observable features of simulated individual users and documents to make recommendations. The user model samples users from a distribution over (configurable) user features (e.g., latent features, like interests or satisfaction; observable features, like user demographic; and behavioral features, such as visit frequency or time budget). The document model samples items from a prior distribution over document features, both latent (e.g., quality) and observable (e.g., length, popularity). This prior, as all other components of RecSim, can be specified by the simulation developer, possibly informed (or learned) from application data.

- **Google's RecSim is an open source simulation framework for recommender system.**
- **It is used to study reinforcement learning algorithm in recommender system.**
- **Reinforcement learning is theoretically one of the most effective machine learning methods. However, in practive it does not handle complex problems. RecSim, an Reinforcement learning framework, allows optimization of complex recommender system.**




```
@article{ie2019recsim,
title={RecSim: A Configurable Simulation Platform for Recommender Systems},
    author={Eugene Ie and Chih-wei Hsu and Martin Mladenov and Vihan Jain and Sanmit Narvekar and Jing Wang and Rui Wu and Craig Boutilier},
    year={2019},
    eprint={1909.04847},
    archivePrefix={arXiv},
    primaryClass={cs.LG}
}
```

In [ ]:

```
!pip install recsim
```

In [ ]:

```
!pip install git+https://github.com/google/dopamine.git
```

In [ ]:

```
!git clone https://github.com/google-research/recsim
%cd recsim/recsim
!python main.py --logtostderr \
  --base_dir="/tmp/recsim/interest_exploration_full_slate_q" \
  --agent_name=full_slate_q \
  --environment_name=interest_exploration \
  --episode_log_file='episode_logs.tfrecord' \
  --gin_bindings=simulator.runner_lib.Runner.max_steps_per_episode=100 \
  --gin_bindings=simulator.runner_lib.TrainRunner.num_iterations=10 \
  --gin_bindings=simulator.runner_lib.TrainRunner.max_training_steps=100 \
  --gin_bindings=simulator.runner_lib.EvalRunner.max_eval_episodes=5
```

In [ ]:

```
# Install TensorFlow 1.x
!pip install tensorflow
```

In [ ]:

```
!pip uninstall tensorflow -y
!pip install tensorflow==2.17.0
```

```
Found existing installation: tensorflow 2.17.0
Uninstalling tensorflow-2.17.0:
  Successfully uninstalled tensorflow-2.17.0
Collecting tensorflow==2.17.0
  Using cached tensorflow-2.17.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.w
hl.metadata (4.2 kB)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages
(from tensorflow==2.17.0) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packag
es (from tensorflow==2.17.0) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-pac
kages (from tensorflow==2.17.0) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/pyth
on3.10/dist-packages (from tensorflow==2.17.0) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-pack
ages (from tensorflow==2.17.0) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (f
rom tensorflow==2.17.0) (3.11.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-package
s (from tensorflow==2.17.0) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-
packages (from tensorflow==2.17.0) (0.4.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packag
es (from tensorflow==2.17.0) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from
tensorflow==2.17.0) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.2
1.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.17.
0) (3.20.3)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-pack
ages (from tensorflow==2.17.0) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (fro
m tensorflow==2.17.0) (71.0.4)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (fr
om tensorflow==2.17.0) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-package
s (from tensorflow==2.17.0) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist
-packages (from tensorflow==2.17.0) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (
from tensorflow==2.17.0) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-pack
ages (from tensorflow==2.17.0) (1.64.1)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-
packages (from tensorflow==2.17.0) (2.17.0)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (f
```

```
rom tensorflow==2.17.0) (3.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/pyt
hon3.10/dist-packages (from tensorflow==2.17.0) (0.37.1)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-pac
kages (from tensorflow==2.17.0) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packa
ges (from astunparse>=1.6.0->tensorflow==2.17.0) (0.44.0)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from kera
s>=3.2.0->tensorflow==2.17.0) (13.8.0)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from ker
as>=3.2.0->tensorflow==2.17.0) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from ke
ras>=3.2.0->tensorflow==2.17.0) (0.12.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist
-packages (from requests<3,>=2.21.0->tensorflow==2.17.0) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (f
rom requests<3,>=2.21.0->tensorflow==2.17.0) (3.8)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packa
ges (from requests<3,>=2.21.0->tensorflow==2.17.0) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packa
ges (from requests<3,>=2.21.0->tensorflow==2.17.0) (2024.8.30)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages
(from tensorboard<2.18,>=2.17->tensorflow==2.17.0) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/py
thon3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow==2.17.0) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages
(from tensorboard<2.18,>=2.17->tensorflow==2.17.0) (3.0.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packag
es (from werkzeug>=1.0.1->tensorboard<2.18,>=2.17->tensorflow==2.17.0) (2.1.5)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-pa
ckages (from rich->keras>=3.2.0->tensorflow==2.17.0) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-
packages (from rich->keras>=3.2.0->tensorflow==2.17.0) (2.16.1)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (fro
m markdown-it-py>=2.2.0->rich->keras>=3.2.0->tensorflow==2.17.0) (0.1.2)
Using cached tensorflow-2.17.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(601.3 MB)
Installing collected packages: tensorflow
Successfully installed tensorflow-2.17.0
```

In [ ]:

```python
import tensorflow as tf

# Check if estimator exists
print(tf.__version__)
print(hasattr(tf, 'estimator'))  # Should return True in TF 2.x
```

```
2.17.0
False
```

In [ ]:

```
!tensorboard --logdir=/tmp/recsim/interest_exploration_full_slate_q/ --port=2222
```

Conclusion

RecSim offers a versatile platform for simulating recommender systems. By enabling researchers to customize user and document models, it facilitates the development and evaluation of RL-based recommendation algorithms. Its open-source nature promotes collaboration and innovation, driving advancements in the field. RecSim's potential to enhance personalized recommendations holds significant promise for various real-world applications.

Practical No 2: a) Implement the User-User collaborative filtering and suggest recommendation for the users.

b) Calculate Prediction for a particular user by using method of Cosine similarity , Euclidean distance and Pearson correlation on user-user based collaborative filtering.

User-based filtering is a recommendation system technique that predicts a user's preferences based on the ratings of similar users.

Approach:

Find Similar Users: Calculate similarity between the target user and other users using metrics like Pearson correlation or cosine similarity. Identify Rated Items: Determine items rated by similar users but not by the target user. Predict Ratings: Predict the target user's rating for these items based on the ratings of similar users. Recommend Items: Recommend items with the highest predicted ratings.

Example:

If User A and User B have rated similar movies highly, and User A has rated a new movie highly, the system might recommend that movie to User B.

In [ ]:

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
```

In [ ]:

```
movies = pd.read_csv('/content/drive/MyDrive/RS dataset/movies.csv')
ratings = pd.read_csv('/content/drive/MyDrive/RS dataset/ratings.csv')
```

In [ ]:

```
movies.head()
```

Out[ ]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [ ]:

```
ratings.head()
```

Out[ ]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247

	userId	movieId	rating	timestamp
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

In [ ]:

```
user_item_matrix = ratings.pivot(index='userId', columns='movieId', values='rating')
```

In [ ]:

```
user_item_matrix.head()
```

Out[ ]:

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193
userId																		
1	4.0	NaN	4.0	NaN	NaN	4.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N
5	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N

5 rows × 9724 columns

In [ ]:

```
user_item_matrix_filled = user_item_matrix.fillna(0)
```

In [ ]:

```
user_item_matrix_filled.head()
```

Out[ ]:

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193583	193585
userId																			
1	4.0	0.0	4.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 9724 columns

In [ ]:

```
user_similarity = cosine_similarity(user_item_matrix_filled)
```

In [ ]:

```
print(user_similarity)
```

```
[[1.         0.02728287 0.05972026 ... 0.29109737 0.09357193 0.14532081]
 [0.02728287 1.         0.         ... 0.04621095 0.0275654  0.10242675]
 [0.05972026 0.         1.         ... 0.02112846 0.         0.03211875]
 ...
 [0.29109737 0.04621095 0.02112846 ... 1.         0.12199271 0.32205486]
 [0.09357193 0.0275654  0.         ... 0.12199271 1.         0.05322546]
```

```
 [0.14532081 0.10242675 0.03211875 ... 0.32205486 0.05322546 1.         ]]
```

In [ ]:

```python
user_correlation = user_item_matrix_filled.corr(method='pearson')
```

In [ ]:

```python
from scipy.spatial.distance import pdist, squareform
user_distance = pdist(user_item_matrix_filled, metric='euclidean')
```

In [ ]:

```python
user_similarity_df = pd.DataFrame(user_similarity, index=user_item_matrix.index, columns
=user_item_matrix.index)
```

In [ ]:

```python
user_correlation_df = pd.DataFrame(user_correlation, index=user_item_matrix.index, colum
ns=user_item_matrix.index)
user_correlation_df = user_correlation_df.fillna(0)
```

In [ ]:

```python
user_distance_df = pd.DataFrame(squareform(user_distance), index=user_item_matrix.index,
columns=user_item_matrix.index)
```

In [ ]:

```python
user_similarity_df.head()
```

Out[ ]:

userId	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**...**	**601**	
userId													
1	1.000000	0.027283	0.059720	0.194395	0.129080	0.128152	0.158744	0.136968	0.064263	0.016875	...	0.080554	0.16
2	0.027283	1.000000	0.000000	0.003726	0.016614	0.025333	0.027585	0.027257	0.000000	0.067445	...	0.202671	0.01
3	0.059720	0.000000	1.000000	0.002251	0.005020	0.003936	0.000000	0.004941	0.000000	0.000000	...	0.005048	0.00
4	0.194395	0.003726	0.002251	1.000000	0.128659	0.088491	0.115120	0.062969	0.011361	0.031163	...	0.085938	0.12
5	0.129080	0.016614	0.005020	0.128659	1.000000	0.300349	0.108342	0.429075	0.000000	0.030611	...	0.068048	0.41

5 rows × 610 columns

In [ ]:

```python
user_correlation_df.head()
```

Out[ ]:

userId	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**...**	**601**	**602**
userId													
1	1.000000	0.231327	0.173213	-0.028917	0.192474	0.192686	0.143743	0.085477	0.177245	0.183382	...	0.0	0.054760
2	0.231327	1.000000	0.191945	0.071269	0.200526	0.158341	0.127569	0.141540	-0.021045	0.285086	...	0.0	-0.018291
3	0.173213	0.191945	1.000000	0.067143	0.370171	0.196442	0.351513	0.296897	0.275812	0.136916	...	0.0	-0.011729
4	-0.028917	0.071269	0.067143	1.000000	0.167910	0.053755	0.258075	0.148726	-0.016025	0.056000	...	0.0	-0.004138
5	0.192474	0.200526	0.370171	0.167910	1.000000	0.215503	0.429890	0.265777	0.308085	0.110833	...	0.0	-0.011456

userId	1	2	3	4	5	6	7	8	9	10	...	601	602
userId													

5 rows × 610 columns

In [ ]:

```
user_distance_df.head()
```

Out[ ]:

userId	1	2	3	4	5	6	7	8	9	10	...	
userId												
1	0.000000	70.436141	69.336138	78.797208	68.985506	86.752522	74.161985	68.883960	70.192592	78.574805	...	77.
2	70.436141	0.000000	29.457597	59.692964	32.806249	66.777616	47.662879	32.927952	32.128648	45.241021	...	45.
3	69.336138	29.457597	0.000000	59.114296	31.882597	66.682082	47.439435	32.194720	30.975797	45.765708	...	48.
4	78.797208	59.692964	59.114296	0.000000	58.034473	80.826976	66.355105	59.732738	60.282667	68.234888	...	68.
5	68.985506	32.806249	31.882597	58.034473	0.000000	61.049161	47.370877	26.907248	34.438351	47.518417	...	49.

5 rows × 610 columns

In [ ]:

```
user_predicted_ratings_1 = pd.DataFrame(index=user_item_matrix.index, columns=user_item_
matrix.columns)
user_predicted_ratings_2 = pd.DataFrame(index=user_item_matrix.index, columns=user_item_
matrix.columns)
user_predicted_ratings_3 = pd.DataFrame(index=user_item_matrix.index, columns=user_item_
matrix.columns)
```

In [ ]:

```
for user in user_item_matrix.index:
    sim_scores = user_similarity_df[user]
    weighted_sum = sim_scores.values @ user_item_matrix_filled
    sim_sum = np.abs(sim_scores).sum()
    user_predicted_ratings_1.loc[user] = weighted_sum / sim_sum
```

In [ ]:

```
user_predicted_ratings_1.head()
```

Out[ ]:

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	1
userId													
1	1.808175	0.831877	0.423003	0.027315	0.271766	1.026076	0.317491	0.046305	0.080917	1.053528	...	0.000247	0.0
2	1.365	0.620288	0.15023	0.009878	0.154428	0.614357	0.122923	0.025785	0.025225	0.623624	...	0.012876	0.0
3	1.584153	0.796827	0.452255	0.022544	0.226689	1.143088	0.317018	0.059864	0.074613	0.941106	...	0.0	
4	1.768037	0.747967	0.349428	0.03166	0.262678	0.923377	0.351348	0.040625	0.062458	0.929745	...	0.000597	0.0
5	1.74818	0.910214	0.371129	0.059913	0.368868	0.869849	0.425598	0.068568	0.078188	1.216	...	0.0	

5 rows × 9724 columns

In [ ]:

```
for user in user_item_matrix.index:
    sim_scores = user_correlation_df[user]
    weighted_sum = sim_scores.values @ user_item_matrix_filled
    sim_sum = np.abs(sim_scores).sum()
```

```
    if sim_sum > 0:
        user_predicted_ratings_2.loc[user] = weighted_sum / sim_sum
    else:
        user_predicted_ratings_2.loc[user] = 0
```

In [ ]:

```
user_predicted_ratings_2.head()
```

Out[ ]:

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	1
userId													
1	1.32743	0.576657	0.400538	0.02836	0.193142	0.783765	0.275632	0.031117	0.071235	0.699074	...	0.004773	0.(
2	1.332922	0.555271	0.328859	0.012324	0.197977	0.662324	0.255308	0.028262	0.097404	0.693928	...	0.003521	0.(
3	1.341055	0.566074	0.300964	0.016746	0.196545	0.650674	0.247816	0.050625	0.079413	0.683748	...	-0.000864	0.(
4	1.357986	0.586288	0.251771	0.009451	0.206963	0.575975	0.244019	0.031773	0.080707	0.74	...	-0.000357	0.(
5	1.45674	0.608162	0.308476	0.026141	0.21245	0.693792	0.283012	0.044369	0.078662	0.666353	...	0.004322	0.(

5 rows × 9724 columns

In [ ]:

```
for user in user_item_matrix.index:
    sim_scores = user_distance_df[user]
    weighted_sum = sim_scores.values @ user_item_matrix_filled
    sim_sum = np.abs(sim_scores).sum()
    if sim_sum > 0:
        user_predicted_ratings_3.loc[user] = weighted_sum / sim_sum
    else:
        user_predicted_ratings_3.loc[user] = 0
```

In [ ]:

```
def recommend_items_cosine_similarity(user_id, num_recommendations=10):
    user_ratings = user_predicted_ratings_1.loc[user_id].sort_values(ascending=False)
    already_rated = user_item_matrix.loc[user_id].dropna().index
    recommendations = user_ratings.drop(already_rated)
    return recommendations.head(num_recommendations)
```

In [ ]:

```
def recommend_items_correlation(user_id, num_recommendations=10):
    user_ratings = user_predicted_ratings_2.loc[user_id].sort_values(ascending=False)
    already_rated = user_item_matrix.loc[user_id].dropna().index
    recommendations = user_ratings.drop(already_rated)
    return recommendations.head(num_recommendations)
```

In [ ]:

```
def recommend_items_distance(user_id, num_recommendations=10):
    user_ratings = user_predicted_ratings_3.loc[user_id].sort_values(ascending=False)
    already_rated = user_item_matrix.loc[user_id].dropna().index
    recommendations = user_ratings.drop(already_rated)
    return recommendations.head(num_recommendations)
```

In [ ]:

```
recommend_items_cosine_similarity(1, 5)
```

Out[ ]:

1

movieId	1
318	2.622414
589	2.06192
858	1.836914
2762	1.643315
4993	1.605043

dtype: object

In [ ]:

```
recommend_items_correlation(1, 5)
```

Out[ ]:

movieId	1
318	2.182474
589	1.411039
858	1.293493
4993	1.242883
5952	1.224389

dtype: object

In [ ]:

```
recommend_items_distance(1, 5)
```

Out[ ]:

movieId	1
318	2.386846
589	1.548258
858	1.482452
4993	1.464515
7153	1.371832

dtype: object

Conclusion

User-based collaborative filtering is a popular technique for recommending items to users based on their similarity to other users. By leveraging similarity metrics like cosine similarity, Pearson correlation, or Euclidean distance, we can effectively predict user preferences and provide personalized recommendations. However, this technique can be computationally expensive, especially for large datasets.

In [ ]:

```
```

Practical No 3: a) Implement the Item-Item based collaborative filtering and suggest recommendation for the users.

b) Calculate Prediction for a particular user by using method of Cosine similarity , Euclidean distance and Pearson correlation on Item-Item based collaborative filtering.

Item-Based Collaborative Filtering

This technique recommends items to a user based on their similarity to items the user has previously rated highly.

Approach:

Find Similar Items: Calculate similarity between items using metrics like cosine similarity or correlation. Recommend Similar Items: Recommend items similar to those the user has rated highly.

Example:

If a user has rated "The Lord of the Rings" highly, the system might recommend "The Hobbit" because these items are similar in terms of genre, author, and theme.

In [ ]:

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
```

In [ ]:

```
movies = pd.read_csv('/content/drive/MyDrive/RS dataset/movies.csv')
ratings = pd.read_csv('/content/drive/MyDrive/RS dataset/ratings.csv')
```

In [ ]:

```
movies.head()
```

Out[ ]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [ ]:

```
ratings.head()
```

Out[ ]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247

	userId	movieId	rating	timestamp
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

In [ ]:

```
item_user_matrix = ratings.pivot(index='movieId', columns='userId', values='rating')
```

In [ ]:

```
item_user_matrix.head()
```

Out[ ]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609
movieId																				
1	4.0	NaN	NaN	NaN	4.0	NaN	4.5	NaN	NaN	NaN	...	4.0	NaN	4.0	3.0	4.0	2.5	4.0	2.5	3.0
2	NaN	NaN	NaN	NaN	NaN	4.0	NaN	4.0	NaN	NaN	...	NaN	4.0	NaN	5.0	3.5	NaN	NaN	2.0	NaN
3	4.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	NaN
4	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN

5 rows × 610 columns

In [ ]:

```
item_user_matrix_filled = item_user_matrix.fillna(0)
item_user_matrix_filled.head()
```

Out[ ]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	0.0	0.0	0.0	4.0	0.0	4.5	0.0	0.0	0.0	...	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	0.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	0.0	0.0	...	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0	0.0
3	4.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 610 columns

In [ ]:

```
item_similarity = cosine_similarity(item_user_matrix_filled)
```

In [ ]:

```
item_correlation = item_user_matrix_filled.corr(method='pearson')
```

In [ ]:

```
from scipy.spatial.distance import pdist, squareform
item_distance = pdist(item_user_matrix_filled, metric='euclidean')
```

In [ ]:

```
item_similarity_df = pd.DataFrame(item_similarity, index=item_user_matrix.index, columns
=item_user_matrix.index)
```

In [ ]:

```
item_similarity_df
```

Out[ ]:

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193
movieId													
1	1.000000	0.410562	0.296917	0.035573	0.308762	0.376316	0.277491	0.131629	0.232586	0.395573	...	0.0	
2	0.410562	1.000000	0.282438	0.106415	0.287795	0.297009	0.228576	0.172498	0.044835	0.417693	...	0.0	
3	0.296917	0.282438	1.000000	0.092406	0.417802	0.284257	0.402831	0.313434	0.304840	0.242954	...	0.0	
4	0.035573	0.106415	0.092406	1.000000	0.188376	0.089685	0.275035	0.158022	0.000000	0.095598	...	0.0	
5	0.308762	0.287795	0.417802	0.188376	1.000000	0.298969	0.474002	0.283523	0.335058	0.218061	...	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
193581	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	1.0	
193583	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	1.0	
193585	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	1.0	
193587	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	1.0	
193609	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.072542	...	0.0	

9724 rows × 9724 columns

In [ ]:

```
item_correlation_df = pd.DataFrame(item_correlation, index=item_user_matrix.index, colum
ns=item_user_matrix.index)
item_correlation_df = item_correlation_df.fillna(0)
```

In [ ]:

```
item_distance_df = pd.DataFrame(squareform(item_distance), index=item_user_matrix.index,
columns=item_user_matrix.index)
```

In [ ]:

```
item_predicted_ratings_1 = pd.DataFrame(index=item_user_matrix.index, columns=item_user_
matrix.columns)
item_predicted_ratings_2 = pd.DataFrame(index=item_user_matrix.index, columns=item_user_
matrix.columns)
item_predicted_ratings_3 = pd.DataFrame(index=item_user_matrix.index, columns=item_user_
matrix.columns)
```

In [ ]:

```
for user in item_user_matrix.index:
    sim_scores = item_similarity_df[user]
    weighted_sum = sim_scores.values @ item_user_matrix_filled
    sim_sum = np.abs(sim_scores).sum()
    item_predicted_ratings_1.loc[user] = weighted_sum / sim_sum
```

In [ ]:

```
for user in item_user_matrix.index:
    sim_scores = item_correlation_df[user]
    weighted_sum = sim_scores.values @ item_user_matrix_filled
    sim_sum = np.abs(sim_scores).sum()
    item_predicted_ratings_2.loc[user] = weighted_sum / sim_sum
```

In [ ]:

```
for user in item_user_matrix.index:
```

```python
for user in item_user_matrix.index:
    sim_scores = item_distance_df[user]
    weighted_sum = sim_scores.values @ item_user_matrix_filled
    sim_sum = np.abs(sim_scores).sum()
    item_predicted_ratings_3.loc[user] = weighted_sum / sim_sum
```

In [ ]:

```python
def recommend_users_cosine_similarity(user_id, rating, num_recommendations=10):
    user_ratings = rating.loc[user_id].sort_values(ascending=False)
    already_rated = item_user_matrix.loc[user_id].dropna().index
    recommendations = user_ratings.drop(already_rated)
    return recommendations.head(num_recommendations)
```

In [ ]:

```python
recommend_users_cosine_similarity(1, item_predicted_ratings_1, 5)
```

Out[ ]:

	1
userId	
387	0.634556
305	0.600262
318	0.547063
489	0.513742
105	0.492343

dtype: object

In [ ]:

```python
recommend_users_cosine_similarity(1, item_predicted_ratings_2, 5)
```

Out[ ]:

	1
userId	
6	1.760473
84	0.960144
602	0.876858
58	0.833532
117	0.824498

dtype: object

In [ ]:

```python
recommend_users_cosine_similarity(1, item_predicted_ratings_3, 5)
```

Out[ ]:

	1
userId	
387	0.34275
318	0.33897
105	0.306704
305	0.272841

111	0.220724

userId
~~dtype: object~~

Conclusion

Item-based collaborative filtering is a popular technique for recommending items to users based on their similarity to items the user has previously rated highly. By leveraging similarity metrics like cosine similarity, Pearson correlation, or Euclidean distance, we can effectively identify similar items and provide personalized recommendations. This approach is computationally efficient compared to user-based filtering, making it suitable for large-scale recommendation systems.

In [ ]:

```

```

Practical No 4 : Implement the SVD algorithm and analyze it.

Singular Value Decomposition (SVD)

SVD is a matrix factorization technique that decomposes a matrix into three matrices: U, Σ, and V^T. This decomposition is useful for various applications, including dimensionality reduction and feature extraction.

Approach:

Decomposition: Decompose the matrix A into three matrices: A = UΣV^T, where U and V are orthogonal matrices, and Σ is a diagonal matrix containing singular values.

Dimensionality Reduction: Reduce the dimensionality of the matrix by selecting the top k singular values and corresponding columns of U and V.

Feature Extraction: Extract latent features from the reduced matrix.

By reducing the dimensionality of the data, SVD can improve the performance of various machine learning algorithms, including recommender systems.

In [ ]:

```
def transpose(matrix):
    return [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0]))]
```

In [ ]:

```
def multiplyMatrices(A, B):
    result = [[sum(A[i][k] * B[k][j] for k in range(len(B))) for j in range(len(B[0]))]
for i in range(len(A))]
    return result
```

In [ ]:

```
def eigSymmetric(matrix):
    n = len(matrix)
    eigenvalues = [matrix[i][i] for i in range(n)]
    eigenvectors = [[1 if i == j else 0 for i in range(n)] for j in range(n)]
    return eigenvalues, eigenvectors
```

In [ ]:

```
def svd(A):
    AT = transpose(A)
    # A^T.A and A.A^T
    ATA = multiplyMatrices(AT, A)
    AAT = multiplyMatrices(A, AT)

    eigenvalues_V, V = eigSymmetric(ATA)
    eigenvalues_U, U = eigSymmetric(AAT)

    Sigma = [[(eigenvalues_U[i] ** 0.5 if i == j else 0) for j in range(len(U))] for i i
n range(len(V))]

    return U, Sigma, V
```

In [ ]:

```
A = [[1, 1], [7, 7]]
```

In [ ]:

```
U, Sigma, V = svd(A)

print("Left Singular Vectors (U):", U)
print("Singular Values (Sigma):", Sigma)
print("Right Singular Vectors (V):", V)
```

```
Left Singular Vectors (U): [[1, 0], [0, 1]]
Singular Values (Sigma): [[1.4142135623730951, 0], [0, 9.899494936611665]]
Right Singular Vectors (V): [[1, 0], [0, 1]]
```

In [ ]:

```
import numpy as np
```

In [ ]:

```
ans = np.dot(U, Sigma)
ans = np.dot(ans, V)
print("SVD of A: ", ans)
```

```
SVD of A:  [[1.41421356 0.        ]
 [0.         9.89949494]]
```

In [ ]:

```
import pandas as pd
movies = pd.read_csv('/content/drive/MyDrive/RS dataset/movies.csv')
ratings = pd.read_csv('/content/drive/MyDrive/RS dataset/ratings.csv')
```

In [ ]:

```
movies.head()
```

Out[ ]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [ ]:

```
ratings.head()
```

Out[ ]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

In [ ]:

```
n_users = ratings.userId.unique().shape[0]
n_movies = ratings.movieId.unique().shape[0]
print('Number of users = ' + str(n_users) + ' | Number of movies = ' + str(n_movies))
```

```
Number of users = 610 | Number of movies = 9724
```

In [ ]:

```
raw_ratings_pivot = ratings.pivot(index = 'userId', columns ='movieId', values = 'rating
')
```

In [ ]:

```
ratings_pivot = raw_ratings_pivot.copy().fillna(0)
```

In [ ]:

```
user_ratings_mean = np.mean(ratings_pivot.values, axis=1)
user_ratings_demeaned = ratings_pivot.values - user_ratings_mean.reshape(-1, 1)
```

In [ ]:

```
from scipy.sparse.linalg import svds
```

In [ ]:

```
U, Sigma, VT = svds(user_ratings_demeaned, k=10)
```

In [ ]:

```
sigma = np.diag(Sigma)
```

In [ ]:

```
all_user_predicted_ratings = np.dot(np.dot(U, sigma), VT) + user_ratings_mean.reshape(-1
, 1)
```

In [ ]:

```
preds = pd.DataFrame(all_user_predicted_ratings, columns = ratings_pivot.columns)
preds.head()
```

Out[ ]:

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	1
0	2.835809	0.928402	0.967718	-0.024039	0.221835	1.724209	0.126740	-0.013477	0.154454	2.017108	...	-0.018993	0.(
1	0.190676	-0.026294	-0.026036	0.005521	0.028316	0.088451	-0.061647	0.008842	-0.006453	-0.070653	...	0.009709	0.(
2	0.033353	0.008683	0.018793	0.003493	-0.014331	0.075373	-0.015139	0.004100	0.015454	0.065764	...	0.008557	0.(
3	1.558919	0.275447	0.271616	0.043859	0.183769	0.273353	0.346929	-0.054245	-0.036465	0.068682	...	-0.017409	0.(
4	1.272888	0.991241	0.420050	0.122955	0.535151	0.753330	0.634397	0.117590	0.110667	1.151538	...	-0.003708	0.(

5 rows × 9724 columns

Analysis

SVD provides a robust approach to dimensionality reduction, leading to significant computational efficiency in recommender systems. However, it can be sensitive to noise and missing data, which may impact the accuracy of predictions.

Conclusion

Singular Value Decomposition (SVD) offers a powerful technique for dimensionality reduction in recommender systems. By decomposing the user-item rating matrix into three matrices, SVD captures the underlying latent

systems. By decomposing the user-item rating matrix into three matrices, SVD captures the underlying latent factors that influence user preferences and item characteristics. This allows for efficient representation of the data while preserving key information for accurate prediction of user ratings and personalized recommendations.

In [ ]:

```

```

Practical No 5 : Implement the SVD++ algorithm and analyze it. Compare its results with SVD algorithm.

SVD++ (Singular Value Decomposition Plus Plus)

SVD++ is an extension of the traditional SVD technique for recommender systems. It incorporates implicit feedback (user interactions) to improve the accuracy of predictions, especially for items that users haven't explicitly rated.

Approach:

1. **Start**
2. **Initialize parameters (P, Q, y, b_u, b_i, global_mean)**
3. **Preprocess user interactions**
4. **For each epoch (repeat n_epochs):**

- **For each rating (u, i, r):**

```
* Calculate implicit feedback sum
* Predict rating
* Compute error
* Update biases and latent factors (b_u, b_i, P[u], Q[i], y[j])
```

1. **Output trained model**
2. **End**

In [1]:

```
import pandas as pd
movies = pd.read_csv('/content/drive/MyDrive/RS dataset/movies.csv')
ratings = pd.read_csv('/content/drive/MyDrive/RS dataset/ratings.csv')
```

In [2]:

```
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
from math import sqrt
```

In [3]:

```
def show_predictions(model, ratings):
    predictions = []
    for _, row in ratings.iterrows():
        user = int(row['userId'])
        item = int(row['movieId'])
        actual_rating = row['rating']
        predicted_rating = model.predict_single(user, item)
        predictions.append((user, item, actual_rating, predicted_rating))

    predictions_df = pd.DataFrame(predictions, columns=['User', 'Movie', 'Actual Rating'
, 'Predicted Rating'])
    return predictions_df
```

In [4]:

```
class SVD:
    def __init__(self, n_factors=10, lr=0.005, reg=0.02, n_epochs=20):
        self.n_factors = n_factors
        self.lr = lr
        self.reg = reg
```

```
        self.n_epochs = n_epochs

    def fit(self, ratings):
        n_users = int(ratings['userId'].max()) + 1
        n_items = int(ratings['movieId'].max()) + 1

        self.P = np.random.normal(0, 0.1, (n_users, self.n_factors))
        self.Q = np.random.normal(0, 0.1, (n_items, self.n_factors))
        self.b_u = np.zeros(n_users)
        self.b_i = np.zeros(n_items)
        self.global_mean = ratings['rating'].mean()

        for epoch in range(self.n_epochs):
            for _, row in ratings.iterrows():
                u, i, r = int(row['userId']), int(row['movieId']), row['rating']
                pred = self.predict_single(u, i)
                err = r - pred

                self.b_u[u] += self.lr * (err - self.reg * self.b_u[u])
                self.b_i[i] += self.lr * (err - self.reg * self.b_i[i])
                self.P[u, :] += self.lr * (err * self.Q[i, :] - self.reg * self.P[u, :])
                self.Q[i, :] += self.lr * (err * self.P[u, :] - self.reg * self.Q[i, :])

    def predict_single(self, user, item):
        pred = self.global_mean + self.b_u[user] + self.b_i[item]
        pred += np.dot(self.P[user, :], self.Q[item, :])
        return pred
```

In [5]:

```
class SVDPlusPlus(SVD):
    def __init__(self, n_factors=10, lr=0.005, reg=0.02, n_epochs=20):
        super().__init__(n_factors, lr, reg, n_epochs)

    def fit(self, ratings):
        n_users = int(ratings['userId'].max()) + 1
        n_items = int(ratings['movieId'].max()) + 1

        self.P = np.random.normal(0, 0.1, (n_users, self.n_factors))
        self.Q = np.random.normal(0, 0.1, (n_items, self.n_factors))
        self.y = np.random.normal(0, 0.1, (n_items, self.n_factors))  # Implicit feedbac
k
        self.b_u = np.zeros(n_users)
        self.b_i = np.zeros(n_items)
        self.global_mean = ratings['rating'].mean()

        user_interactions = ratings.groupby('userId')['movieId'].apply(list).to_dict()

        for epoch in range(self.n_epochs):
            for _, row in ratings.iterrows():
                u, i, r = int(row['userId']), int(row['movieId']), row['rating']
                implicit_sum = np.sum(self.y[user_interactions[u]], axis=0) if u in user
_interactions else np.zeros(self.n_factors)

                pred = self.global_mean + self.b_u[u] + self.b_i[i] + np.dot(self.P[u] +
implicit_sum, self.Q[i])
                err = r - pred

                self.b_u[u] += self.lr * (err - self.reg * self.b_u[u])
                self.b_i[i] += self.lr * (err - self.reg * self.b_i[i])
                self.P[u] += self.lr * (err * self.Q[i] - self.reg * self.P[u])
                self.Q[i] += self.lr * (err * (self.P[u] + implicit_sum) - self.reg * se
lf.Q[i])

                if u in user_interactions:
                    for j in user_interactions[u]:
                        self.y[j] += self.lr * (err * self.Q[i] / len(user_interactions[
u]) - self.reg * self.y[j])
```

In [9]:

```
svd = SVD(n_factors=10, n_epochs=10)
```

```
svd = SVD(n_factors=10, n_epochs=10)
svd.fit(ratings)
print("SVD Predictions:")
print(show_predictions(svd, ratings))
```

```
SVD Predictions:
        User   Movie  Actual Rating  Predicted Rating
0          1       1            4.0          4.713382
1          1       3            4.0          4.100579
2          1       6            4.0          4.731447
3          1      47            5.0          4.845244
4          1      50            5.0          5.023008
...      ...     ...            ...               ...
100831   610  166534            4.0          3.634540
100832   610  168248            5.0          3.839842
100833   610  168250            5.0          3.717777
100834   610  168252            5.0          4.115216
100835   610  170875            3.0          3.476174

[100836 rows x 4 columns]
```

In [7]:

```
svdpp = SVDPlusPlus(n_factors=10, n_epochs=10)
svdpp.fit(ratings)
print("\nSVD++ Predictions:")
print(show_predictions(svdpp, ratings))
```

```
SVD++ Predictions:
        User   Movie  Actual Rating  Predicted Rating
0          1       1            4.0          4.684986
1          1       3            4.0          4.146554
2          1       6            4.0          4.640298
3          1      47            5.0          4.676758
4          1      50            5.0          4.910485
...      ...     ...            ...               ...
100831   610  166534            4.0          3.488087
100832   610  168248            5.0          3.837149
100833   610  168250            5.0          3.780384
100834   610  168252            5.0          4.204285
100835   610  170875            3.0          3.336781

[100836 rows x 4 columns]
```

In [10]:

```
from sklearn.metrics import mean_squared_error

svd_predictions = show_predictions(svd, ratings)
svd_mse = mean_squared_error(svd_predictions['Actual Rating'], svd_predictions['Predicted
Rating'])

svdpp_predictions = show_predictions(svdpp, ratings)
svdpp_mse = mean_squared_error(svdpp_predictions['Actual Rating'], svdpp_predictions['Pre
dicted Rating'])

print("SVD MSE:", svd_mse)
print("SVD++ MSE:", svdpp_mse)
```

```
SVD MSE: 0.7156198322098477
SVD++ MSE: 0.6726591503061652
```

Analysis

SVD MSE: 0.7156 The SVD model achieves an MSE of 0.7156, indicating its error level in predictions.

SVD++ MSE: 0.6727 The SVD++ model achieves an MSE of 0.6727, which is lower than SVD's MSE.

This suggests that SVD++ performs better than SVD in this context.

Why is SVD++ better?

SVD++ extends SVD by incorporating implicit feedback (e.g., user behavior such as clicks, views, etc.), which often improves prediction accuracy, especially in recommendation systems.

Conclusion

SVD++ is a powerful extension of SVD that leverages both explicit and implicit feedback to enhance recommendation accuracy. By incorporating user interaction history, SVD++ can provide more personalized and accurate recommendations, especially for items that users have not explicitly rated.

In [ ]:

```

```

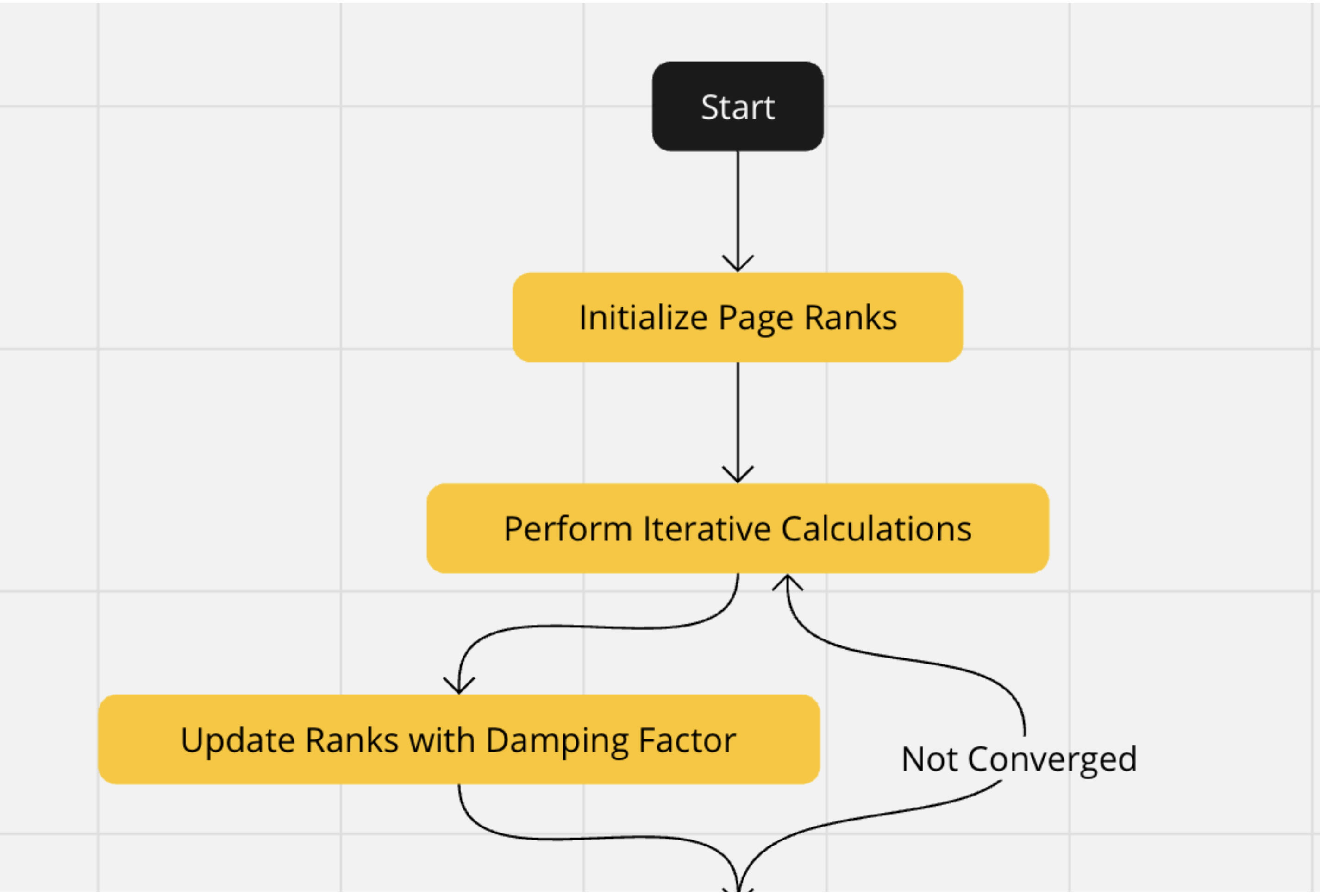
Practical No 6 : Implement the PAGE Rank algorithm and analyze it.

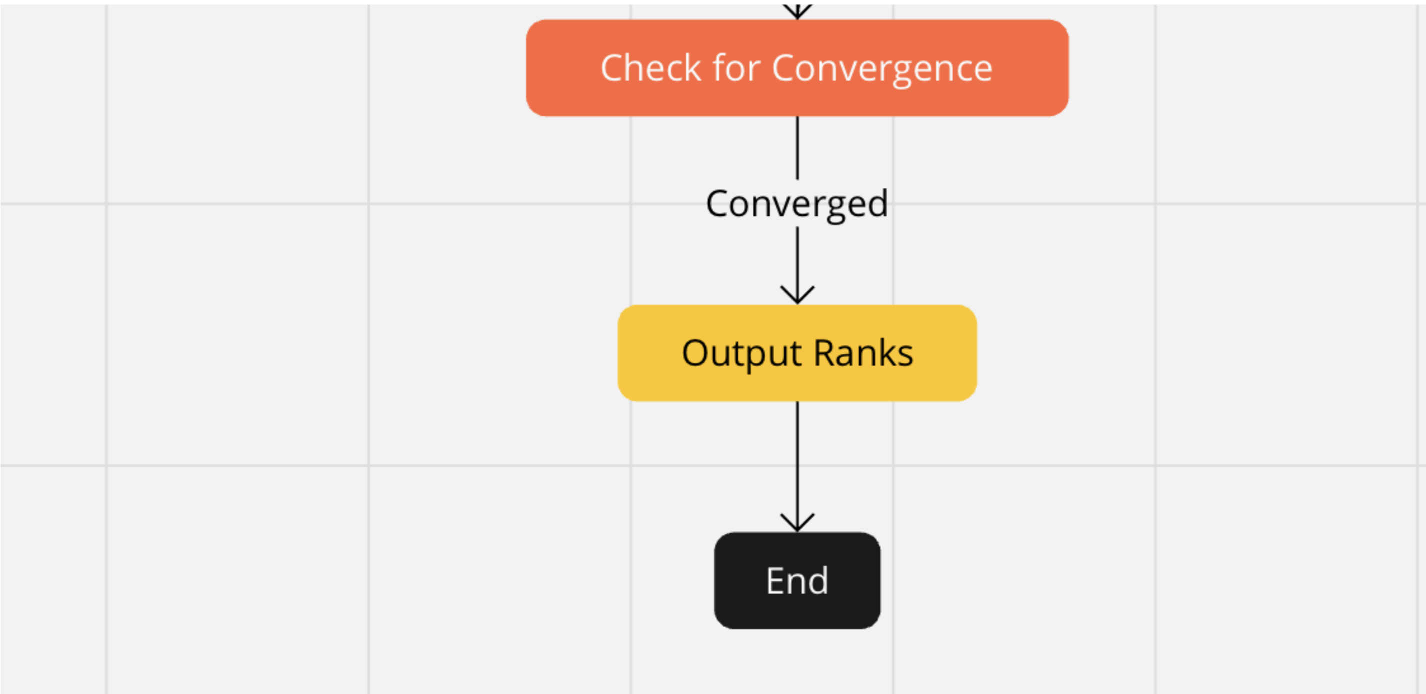
PageRank is an algorithm used by Google Search to rank web pages in their search engine results. It measures the importance of a webpage based on the number and quality of links pointing to it. The underlying assumption is that more important pages are likely to receive more links from other important pages.

$$Rank(v) = \frac{d}{N} + (1-d)\left(\sum_{u_i} \frac{Rank(u_i)}{outlink(u_i)} + \sum_{u_j} \frac{Rank(u_j)}{N}\right)$$

- d: Jump factor.
- N: Number of webpages
- u_i: Pages with links to v.
- u_j: Pages without outlinks.
- $Rank(u_i)$: The rank of the page u_i in the previous iteration.
- $outlinks(u_i)$: The number of pages u_i is pointing to.

Block Diagram





In [26]:

```
import numpy as np
```

In [27]:

```
def page_rank(links, damping=0.85, max_iterations=100, tol=1.0e-6):
    n = links.shape[0]
    column_sums = np.sum(links, axis=0)
    column_sums[column_sums == 0] = 1
    normalized_links = links / column_sums

    ranks = np.ones(n) / n
    teleport = (1 - damping) / n

    for iteration in range(max_iterations):
        new_ranks = teleport + damping * np.dot(normalized_links, ranks)

        if np.linalg.norm(new_ranks - ranks, 1) < tol:
            print(f"Converged after {iteration + 1} iterations.")
            break

        ranks = new_ranks

    return ranks
```

In [28]:

```
links = np.array([
    [0, 0, 1, 0],
    [1, 0, 0, 1],
    [0, 1, 0, 1],
    [1, 0, 0, 0],
], dtype=float)
```

In [29]:

```
scores = page_rank(links)
print("PageRank Scores:", scores)
```

```
Converged after 45 iterations.
PageRank Scores: [0.29721007 0.23343507 0.30554071 0.16381415]
```

In [30]:

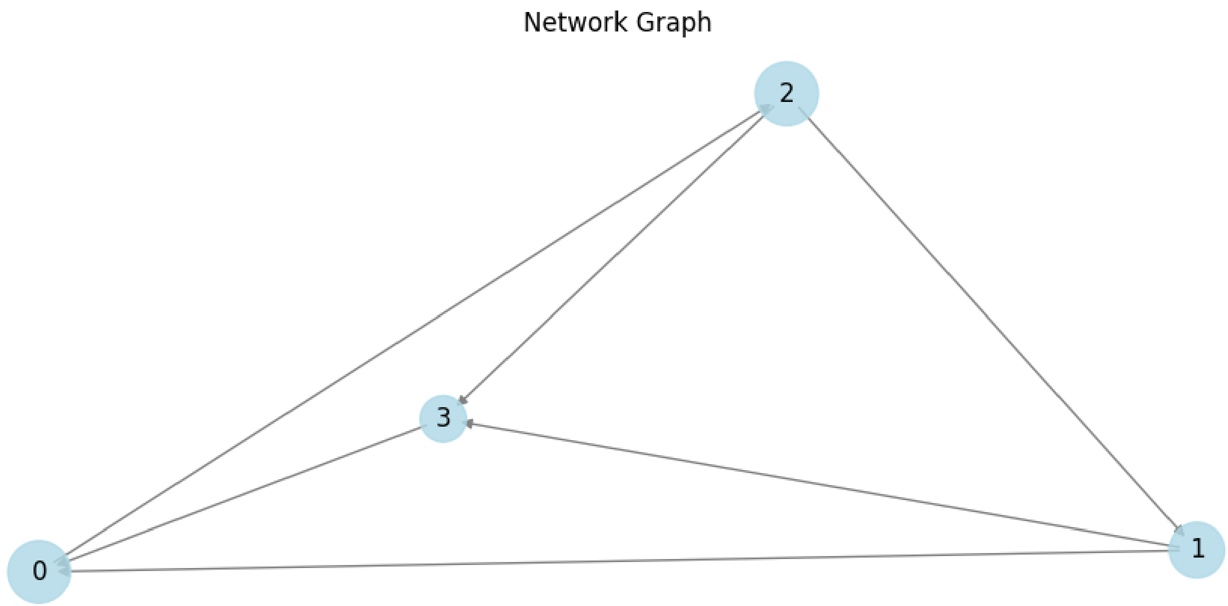
```
import networkx as nx
import matplotlib.pyplot as plt
```

In [31]:

```
G = nx.from_numpy_array(links, create_using=nx.DiGraph)
plt.figure(figsize=(12, 5))
pos = nx.spring_layout(G, seed=42)
node_sizes = scores * 3000
nx.draw_networkx_nodes(G, pos, node_color='lightblue', node_size=node_sizes, alpha=0.8)
nx.draw_networkx_edges(G, pos, edge_color='gray', arrows=True)
nx.draw_networkx_labels(G, pos)
plt.title('Network Graph')
plt.axis('off')
```

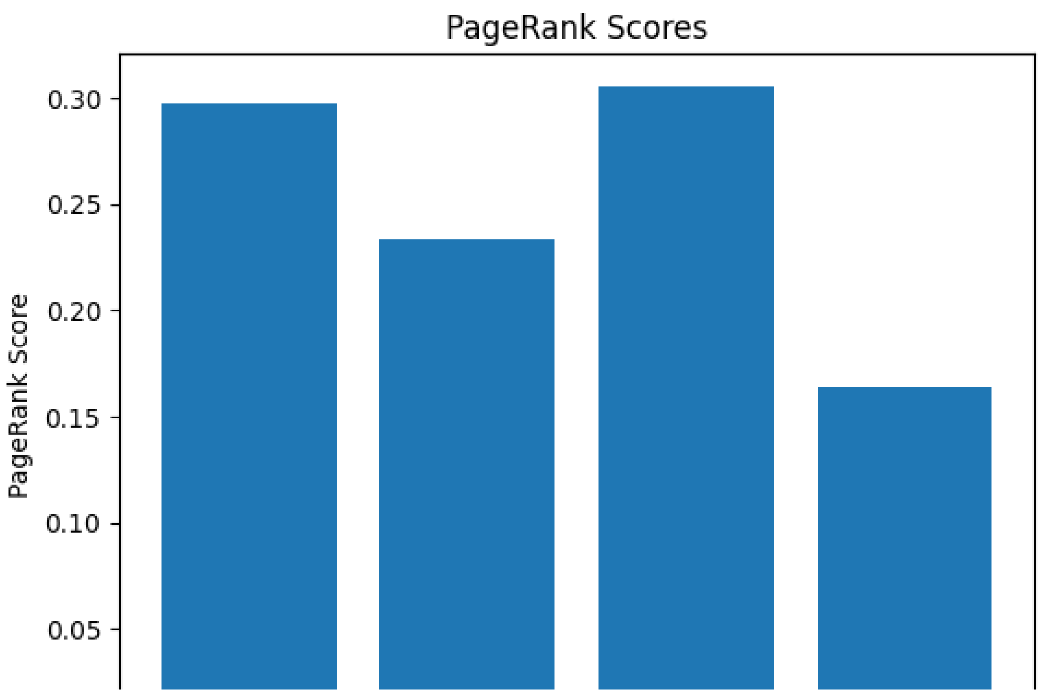
Out[31]:

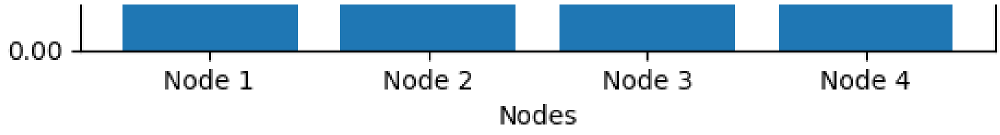
```
(-1.030902963509557,
 1.0352899746004964,
 -0.677541548473842,
 1.1594044005337136)
```



In [32]:

```
plt.bar(range(len(scores)), scores)
plt.title('PageRank Scores')
plt.xlabel('Nodes')
plt.ylabel('PageRank Score')
plt.xticks(range(len(scores)), [f'Node {i+1}' for i in range(len(scores))])
plt.show()
```





In [33]:

```
for i, score in enumerate(scores):
  print(f"Node {i+1}: {score:.4f}")
```

```
Node 1: 0.2972
Node 2: 0.2334
Node 3: 0.3055
Node 4: 0.1638
```

Analysis

Node 3 (0.3055) has the highest rank, meaning it is the most significant node in the network based on the connectivity structure.

Node 1 (0.2972) follows closely, indicating substantial importance.

Node 2 (0.2334) has a moderate rank.

Node 4 (0.1638) is the least significant, likely due to fewer or less influential incoming links.

Conclusion

PageRank is a link analysis algorithm that effectively ranks web pages based on their importance within a network. It considers the quality and quantity of incoming links, with pages receiving links from authoritative sources gaining higher PageRank scores. This approach identifies valuable and informative content, ultimately influencing search engine results and user navigation on the web.

In [ ]:

```
```

Practical No 7: Implement the linear threshold and independent cascade model for influence analysis.

The Linear Threshold (LT) and Independent Cascade (IC) models are used for influence propagation analysis in networks, particularly in the context of social networks, marketing strategies, information diffusion, and viral spread.

Linear Threshold (LT) Model

In this model:

1. **Each node has a threshold value (e.g., randomly assigned between 0 and 1).**
2. **Each edge has an influence weight.**
3. **A node is activated if the sum of weights from its active neighbors exceeds its threshold.**

Independent Cascade (IC) Model

In this model:

1. **When a node is activated, it has a single chance to activate its neighbors with a certain probability.**
2. **This process continues in discrete steps until no further activations occur.**

In [1]:

```
import networkx as nx
import random
```

In [2]:

```
def linear_threshold_model(graph, seeds, max_steps=10):
    thresholds = {node: random.uniform(0, 1) for node in graph.nodes}
    active = set(seeds)
    new_active = set(seeds)

    for _ in range(max_steps):
        current_active = set()
        for node in graph.nodes:
            if node not in active:
                influence = sum(graph[u][node]['weight'] for u in graph.predecessors(node) if u in active)
                if influence >= thresholds[node]:
                    current_active.add(node)
        if not current_active:
            break
        active.update(current_active)
        new_active = current_active

    return active
```

In [4]:

```
G = nx.DiGraph()
edges = [(1, 2, 0.2), (1, 3, 0.1), (2, 3, 0.5), (2, 4, 0.4), (3, 4, 0.3)]
G.add_weighted_edges_from(edges)
```

In [5]:

```
import matplotlib.pyplot as plt
```

In [6]:

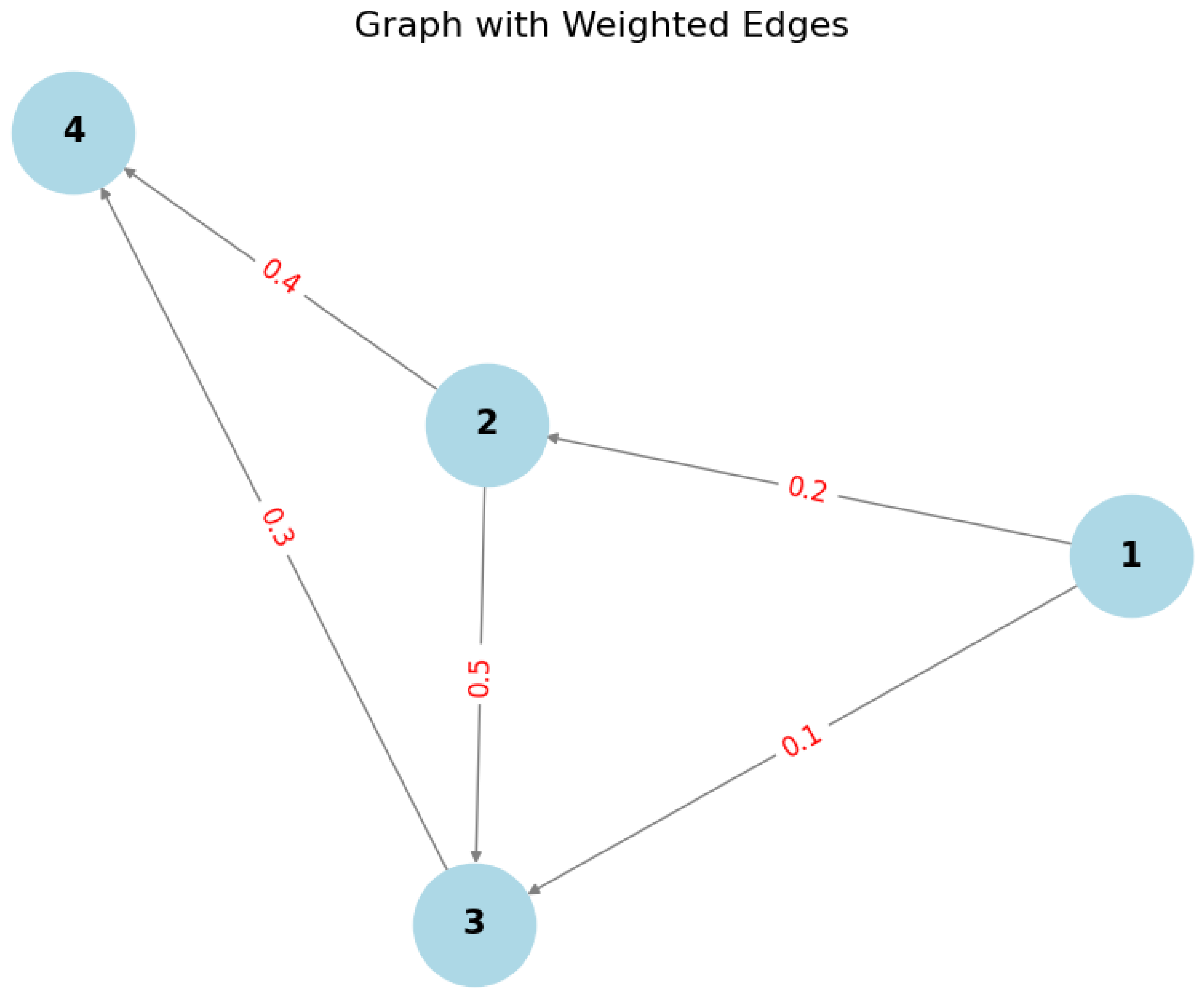
In [6]:

```
pos = nx.spring_layout(G)

plt.figure(figsize=(8, 6))
nx.draw(G, pos, with_labels=True, node_color='lightblue', edge_color='gray', node_size=3
000, font_size=15, font_weight='bold')

edge_labels = {(u, v): f"{d['weight']:.1f}" for u, v, d in G.edges(data=True)}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='red', font_siz
e=12)

plt.title("Graph with Weighted Edges", fontsize=16)
plt.show()
```



In [7]:

```
seeds = [1]
```

In [8]:

```
lt_result = linear_threshold_model(G, seeds)
print("Activated nodes (LT Model):", lt_result)
```

```
Activated nodes (LT Model): {1}
```

In [9]:

```
def independent_cascade_model(graph, seeds, propagation_prob=0.01, max_steps=10):
    active = set(seeds)
    new_active = set(seeds)

    for _ in range(max_steps):
        current_active = set()
```

```
        for node in new_active:
            for neighbor in graph.successors(node):
                if neighbor not in active:
                    if random.random() <= propagation_prob:
                        current_active.add(neighbor)
        if not current_active:
            break
        active.update(current_active)
        new_active = current_active

    return active
```

In [10]:

```
ic_result = independent_cascade_model(G, seeds)
print("Activated nodes (IC Model):", ic_result)
```

```
Activated nodes (IC Model): {1, 2}
```

Analysis

In the LT model, a node becomes activated only when the sum of the influence weights from its active neighbors exceeds its threshold.

Here, only the seed node (1) is activated because:

- **Nodes 2 and 3 receive influence from node 1 (weights: 0.2 and 0.1 respectively), but the total influence does not exceed their thresholds (randomly set values between 0 and 1).**
- **As a result, no further activations occur, leaving only the seed node active.**

In the IC model, when a node is activated, it tries to activate its neighbors with a probability of activation (propagation_prob, default 0.01 in the example).

Here:

- **Node 1 successfully activated node 2 with the propagation probability.**
- **Node 2 attempted to activate its neighbors (3 and 4), but the activation did not succeed (likely due to the low propagation probability).**

Thus, the set of activated nodes includes the seed (1) and node 2.

Conclusion

In this practical, we explored the Linear Threshold and Independent Cascade models for influence propagation in networks. These models provide valuable insights into how information, behaviors, or trends can spread across nodes, making them useful tools for applications in marketing, social network analysis, and epidemic modeling.

In [ ]:

Practical No 8: Implement the Jaccard and Adamic-Adar measures for link prediction.

Jaccard Similarity

Jaccard similarity is a statistical measure used to compare the similarity between two sets. In the context of networks, it measures the similarity between two nodes based on their shared neighbors.

Formula:

`Jaccard(A, B) = |A ∩ B| / |A U B|`

Where:

- **A and B are two sets (e.g., sets of neighbors for two nodes).**
- **|A ∩ B| is the cardinality of the intersection of A and B (number of common elements).**
- **|A ∪ B| is the cardinality of the union of A and B (total number of unique elements).**

A higher Jaccard similarity indicates a stronger relationship between the two nodes.

Adamic-Adar Index

The Adamic-Adar index is another similarity measure used in network analysis, particularly for link prediction. It assigns higher weights to shared neighbors that are less common, suggesting that they are more influential in connecting the two nodes.

Formula:

`Adamic-Adar(A, B) = Σ(1 / log(degree(N)))`

Where:

- **N is a common neighbor of nodes A and B.**
- **degree(N) is the degree of node N (number of neighbors).**

A higher Adamic-Adar index indicates a stronger potential connection between the two nodes.

In [1]:

```
import networkx as nx
import math
```

In [2]:

```
G = nx.Graph()
G.add_edges_from([(1, 2), (1, 3), (2, 3), (3, 4), (4, 5)])
```

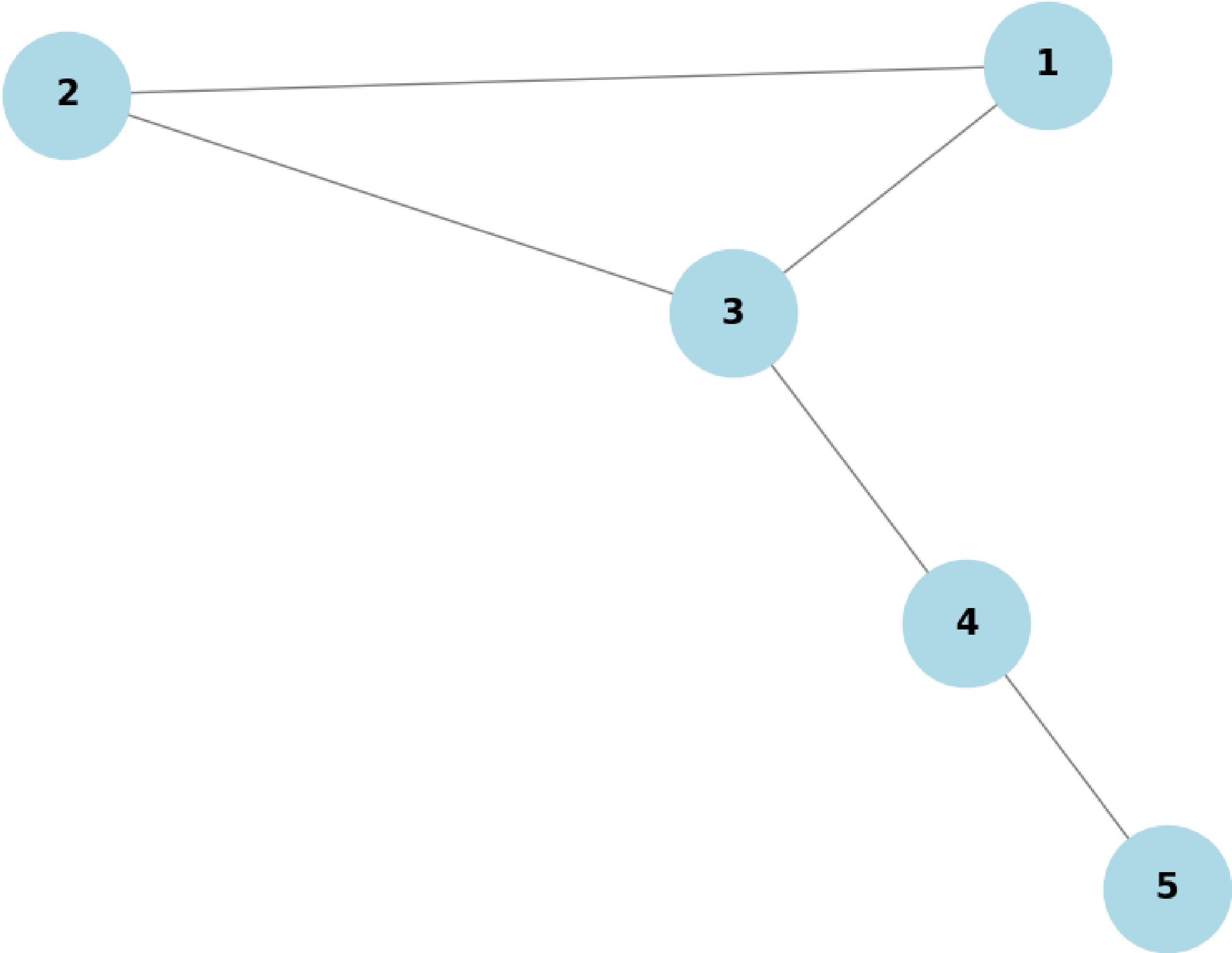
In [3]:

```
import matplotlib.pyplot as plt
```

In [6]:

```
plt.figure(figsize=(8, 6))
nx.draw(G, with_labels=True, node_color='lightblue', node_size=3000, font_size=15, font_
weight='bold', edge_color='gray')
plt.title("Graph Visualization")
plt.show()
```

Graph Visualization



In [7]:

```
def jaccard_index(G, node1, node2):
    neighbors1 = set(G.neighbors(node1))
    neighbors2 = set(G.neighbors(node2))
    intersection = len(neighbors1 & neighbors2)
    union = len(neighbors1 | neighbors2)
    return intersection / union if union != 0 else 0
```

In [13]:

```
print("Jaccard Index between nodes 1 and 4:", jaccard_index(G, 1, 4))
```

```
Jaccard Index between nodes 1 and 4: 0.3333333333333333
```

In [14]:

```
def adamic_adar_index(G, node1, node2):
    neighbors1 = set(G.neighbors(node1))
    neighbors2 = set(G.neighbors(node2))
    common_neighbors = neighbors1 & neighbors2
    score = 0
    for neighbor in common_neighbors:
        degree = len(list(G.neighbors(neighbor)))
        score += 1 / math.log(degree) if degree > 1 else 0
    return score
```

In [15]:

```
print("Adamic-Adar Index between nodes 1 and 4:", adamic_adar_index(G, 1, 4))
```

```
Adamic-Adar Index between nodes 1 and 4: 0.9102392266268373
```

Analysis

- **Jaccard Index (0.3333): This measures the similarity between nodes 1 and 4 by comparing their shared neighbors to all possible neighbors. The result (33.33%) indicates a moderate similarity based on shared connections.**
- **Adamic-Adar Index (0.9102): This index gives more weight to rare common neighbors. Since node 3 (the common neighbor of 1 and 4) has a relatively low degree, it results in a higher score (91.02%), suggesting a stronger potential for a link between nodes 1 and 4.**

Conclusion

In this practical, the Jaccard Index and Adamic-Adar Index are used to measure the similarity between nodes based on shared neighbors, with the Adamic-Adar Index giving more weight to rare common neighbors. The results indicate a moderate to high potential for a link between nodes 1 and 4, with Adamic-Adar suggesting a stronger likelihood.

In [ ]: