```
!pip install pyvis
```

```
Requirement already satisfied: pyvis in /usr/local/lib/python3.12/dist-packages (0.3.2)
Requirement already satisfied: ipython>=5.3.0 in /usr/local/lib/python3.12/dist-packages (from pyvis) (7.34.0)
Requirement already satisfied: jinja2>=2.9.6 in /usr/local/lib/python3.12/dist-packages (from pyvis) (3.1.6)
Requirement already satisfied: jsonpickle>=1.4.1 in /usr/local/lib/python3.12/dist-packages (from pyvis) (4.1.1)
Requirement already satisfied: networkx>=1.11 in /usr/local/lib/python3.12/dist-packages (from pyvis) (3.6.1)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.12/dist-packages (from ipython>=5.3.0->pyvis) (75.2.0)
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.12/dist-packages (from ipython>=5.3.0->pyvis) (0.19.2)
Requirement already satisfied: decorator in /usr/local/lib/python3.12/dist-packages (from ipython>=5.3.0->pyvis) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.12/dist-packages (from ipython>=5.3.0->pyvis) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.12/dist-packages (from ipython>=5.3.0->pyvis) (5.7.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from ipy
Requirement already satisfied: pygments in /usr/local/lib/python3.12/dist-packages (from ipython>=5.3.0->pyvis) (2.19.2)
Requirement already satisfied: backcall in /usr/local/lib/python3.12/dist-packages (from ipython>=5.3.0->pyvis) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.12/dist-packages (from ipython>=5.3.0->pyvis) (0.2.1)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.12/dist-packages (from ipython>=5.3.0->pyvis) (4.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2>=2.9.6->pyvis) (3.0.3)
Requirement already satisfied: parso<0.9.0,>=0.8.4 in /usr/local/lib/python3.12/dist-packages (from jedi>=0.16->ipython>=5.3.0->
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.12/dist-packages (from pexpect>4.3->ipython>=5.3.0->pyv
Requirement already satisfied: wcwidth in /usr/local/lib/python3.12/dist-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2
```

```
import pandas as pd
import numpy as np
import networkx as nx
from sklearn.metrics.pairwise import cosine_similarity
import pyvis.network
from pyvis.network import Network
import matplotlib.pyplot as plt
```

importing pandas for file reading, numpy for numurical operations, networkx, cosine_similariy, network, for graph theory , plt for visualization

Loading Data

```
anime=pd.read_csv('anime.csv')
rating=pd.read_csv('rating.csv')
```

```
print(anime.info())
print(rating.info())
print(anime.isnull().sum())
print(rating.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12294 entries, 0 to 12293
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   anime_id  12294 non-null  int64  
 1   name      12294 non-null  object 
 2   genre     12232 non-null  object 
 3   type      12269 non-null  object 
 4   episodes  12294 non-null  object 
 5   rating    12064 non-null  float64
 6   members   12294 non-null  int64  
dtypes: float64(1), int64(2), object(4)
memory usage: 672.5+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7813737 entries, 0 to 7813736
Data columns (total 3 columns):
 #   Column    Dtype
---  ------    -----
 0   user_id   int64
 1   anime_id  int64
 2   rating    int64
dtypes: int64(3)
memory usage: 178.8 MB
None
anime_id      0
name          0
genre        62
```

```
type         25
episodes      0
rating      230
members       0
dtype: int64
user_id     0
anime_id    0
rating      0
dtype: int64
```

```
print(anime.describe())
print(rating.describe())
```

```
           anime_id        rating       members
count  12294.000000  12064.000000  1.229400e+04
mean   14058.221653      6.473902  1.807134e+04
std    11455.294701      1.026746  5.482068e+04
min        1.000000      1.670000  5.000000e+00
25%     3484.250000      5.880000  2.250000e+02
50%    10260.500000      6.570000  1.550000e+03
75%    24794.500000      7.180000  9.437000e+03
max    34527.000000     10.000000  1.013917e+06
            user_id      anime_id        rating
count  7.813737e+06  7.813737e+06  7.813737e+06
mean   3.672796e+04  8.909072e+03  6.144030e+00
std    2.099795e+04  8.883950e+03  3.727800e+00
min    1.000000e+00  1.000000e+00 -1.000000e+00
25%    1.897400e+04  1.240000e+03  6.000000e+00
50%    3.679100e+04  6.213000e+03  7.000000e+00
75%    5.475700e+04  1.409300e+04  9.000000e+00
max    7.351600e+04  3.451900e+04  1.000000e+01
```

CLEANING DATA

```
rating.loc[rating['rating'] < 0, 'rating'] = 0
```

```
anime["genre"] = anime["genre"].fillna("Unknown")
```

```
anime["type"] = anime["type"].fillna("Unknown")
```

```
anime["rating"]=anime["rating"].fillna(0)
```

```
anime.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12294 entries, 0 to 12293
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   anime_id  12294 non-null  int64  
 1   name      12294 non-null  object 
 2   genre     12294 non-null  object 
 3   type      12294 non-null  object 
 4   episodes  12294 non-null  object 
 5   rating    12294 non-null  float64
 6   members   12294 non-null  int64  
dtypes: float64(1), int64(2), object(4)
memory usage: 672.5+ KB
```

```
rating
```

	user_id	anime_id	rating
0	1	20	0
1	1	24	0
2	1	79	0
3	1	226	0
4	1	241	0
...	...	...	...
7813732	73515	16512	7
7813733	73515	17187	9
7813734	73515	22145	10
7813735	73516	790	9
7813736	73516	8074	9

7813737 rows × 3 columns

Start coding or generate with AI.

Creating Graph

Its Based graph theory in mathematics which used many fields like maping social media platforms. G is a graph with vertices or nodes and edge. G=(v,e) its a graph

```
G=nx.Graph()#G is an empty graph blank map where i will  place users anime genres relationships
```

```
for user_id in rating['user_id'].unique():
  G.add_node(f"user_{user_id}", node_type="user")#Loops through all users
```

```
for _, row in anime.iterrows():
  G.add_node( f"anime_{row['anime_id']}",
        node_type="anime",
        name=row["name"],
        type=row["type"]
    )#adding anime nodes,
```

```
genre_set = set()

for genres in anime["genre"]:
    for g in genres.split(","):
        genre_set.add(g.strip())#Anime can have multiple genres i extract unique genres from the dataset and add them as nodes
```

```
for genre in genre_set:
    G.add_node(f"genre_{genre}", node_type="genre")#adding genere node
```

User to anime edges

```
for _, row in rating.iterrows():
    user = f"user_{row['user_id']}"
    anime_node = f"anime_{row['anime_id']}"

    if G.has_node(anime_node):
        weight = (row["rating"] / 10) * 3   # STRONG weight
        G.add_edge(user, anime_node, weight=weight, edge_type="rated")
```

```
G.add_edge(user, anime_node, weight=rating_weight, edge_type="rated")
```

anime to genre edges

```
anime_genre_map = anime.set_index('anime_id')['genre'].to_dict()
for anime_id, genres in anime_genre_map.items():
    for genre in genres.split(","):
        G.add_edge(
            f"anime_{anime_id}",
            f"genre_{genre.strip()}",
            weight=1,
            edge_type="belongs"
        )
```

BUILDING ANIME SIMILARITY

```
ratings_small = rating.sample(20000, random_state=42)
user_anime_matrix = ratings_small.pivot_table(
    index="user_id",
    columns="anime_id",
    values="rating",
    fill_value=0
)
```

```
'''user_anime_matrix = rating.pivot_table(
    index="user_id",
    columns="anime_id",
    values="rating",
    fill_value=0
)#cUser-Anime Matrix'''
```

Double-click (or enter) to edit

```
#anime_similarity = cosine_similarity(user_anime_matrix.T)
#anime_ids = user_anime_matrix.columns
#Compute Cosine Similarity
```

```
from sklearn.metrics.pairwise import cosine_similarity

sim = cosine_similarity(user_anime_matrix.T)
anime_ids = user_anime_matrix.columns
```

```
'''SIMILARITY_THRESHOLD = 0.6

for i in range(len(anime_ids)):
    for j in range(i + 1, len(anime_ids)):
        if anime_similarity[i, j] > SIMILARITY_THRESHOLD:
            G.add_edge(
                f"anime_{anime_ids[i]}",
                f"anime_{anime_ids[j]}",
                weight=anime_similarity[i, j],
                edge_type="similar"
            )'''
```

```
TOP_K = 5

for i, anime_id in enumerate(anime_ids):
    sims = list(enumerate(sim[i]))
    sims = sorted(sims, key=lambda x: x[1], reverse=True)[1:TOP_K+1]

    for j, score in sims:
        if score > 0.85:
            G.add_edge(
                f"anime_{anime_id}",
                f"anime_{anime_ids[j]}",
                weight=score,
                edge_type="similar"
            )
```

```python
TOP_K = 10

for i, anime_id in enumerate(anime_ids):
    sims = list(enumerate(sim[i]))
    sims = sorted(sims, key=lambda x: x[1], reverse=True)[1:TOP_K+1]

    for j, score in sims:
        if score > 0.85:
            G.add_edge(
                f"anime_{anime_id}",
                f"anime_{anime_ids[j]}",
                weight=score,
                edge_type="similar"
            )
```

```python
def build_personalization(user_id):
    p = {node: 0 for node in G.nodes()}
    user_node = f"user_{user_id}"

    if user_node not in G:
        return p

    p[user_node] = 0.6

    liked = rating[rating["user_id"] == user_id]
    for _, row in liked.iterrows():
        anime_node = f"anime_{row['anime_id']}"
        if anime_node in p:
            p[anime_node] += 0.4 * (row["rating"] / 10)

    return p
```

```python
def recommend_anime(user_id, top_n=10):
    p = build_personalization(user_id)

    pr = nx.pagerank(
        G,
        alpha=0.75,              # STRONG restart
        personalization=p,
        weight="weight"
    )

    watched = set(
        rating[rating["user_id"] == user_id]["anime_id"]
    )

    results = []
    for node, score in pr.items():
        if node.startswith("anime_"):
            anime_id = int(node.split("_")[1])
            if anime_id not in watched:
                results.append((anime_id, score))

    results.sort(key=lambda x: x[1], reverse=True)
    return results[:top_n]
```

```python
def show_user_profile(user_id):
    watched = rating[rating["user_id"] == user_id]
    watched = watched.merge(anime, on="anime_id")
    return watched.sort_values("rating_x", ascending=False).head(6)[["name", "genre", "rating_x"]]

print(show_user_profile(90))
```

```
                            name  \
66        Shingeki no Kyojin OVA
78              Noragami Aragoto
76                 Tokyo Ghoul √A
62            Shingeki no Kyojin
77  Tokyo Ghoul: &quot;Jack&quot;
73                   Tokyo Ghoul

                                               genre  rating_x
```

```
66       Action, Drama, Fantasy, Shounen, Super Power        10
78           Action, Adventure, Shounen, Supernatural        10
76  Action, Drama, Horror, Mystery, Psychological,...        10
62       Action, Drama, Fantasy, Shounen, Super Power        10
77  Action, Drama, Horror, School, Seinen, Superna...        10
73  Action, Drama, Horror, Mystery, Psychological,...        10
```

```
'''def recommend_anime(user_id, top_n=10):
    user_node = f"user_{user_id}"

    if user_node not in G:
        return []

    personalization = {node: 0 for node in G.nodes()}
    personalization[user_node] = 1

    pr_scores = nx.pagerank(G, personalization=personalization, weight="weight")

    watched = set(
        rating[rating["user_id"] == user_id]["anime_id"]
    )

    recommendations = []
    for node, score in pr_scores.items():
        if node.startswith("anime_"):
            anime_id = int(node.split("_")[1])
            if anime_id not in watched:
                recommendations.append((anime_id, score))

    recommendations.sort(key=lambda x: x[1], reverse=True)
    return recommendations[:top_n]'''
```

```
'''user_id = 2000
recs = recommend_anime(user_id)

for anime_id, score in recs:
    name = anime[anime["anime_id"] == anime_id]["name"].values[0]
    print(name, round(score, 4))'''
```

```
Great Mazinger 0.0
Uchuu Koukyoushi Maetel: Ginga Tetsudou 999 Gaiden 0.0
Gokinjo Monogatari 0.0
Lady Georgie 0.0
Dorei Kaigo 0.0
Kerokko Demetan 0.0
Nils no Fushigi na Tabi 0.0
Marie &amp; Gali 0.0
Piece 0.0
Ebiten: Kouritsu Ebisugawa Koukou Tenmonbu Specials 0.0
```

Start coding or generate with AI.