Steam games Analysis

using

PostgreSQL, Neo4j and MongoDB

Chenze Fan, Jingyi Zhou, Sirui Li

Dec 12, 2023

Part 1

Introduction

Background, Context, Goals

Backgroud

Extensive Game Library

Steam hosts a vast array of games, from indie titles to blockbuster releases. It has become a go-to platform for developers to release their games due to its large user base and comprehensive support for game publishing.

Community and Social Features

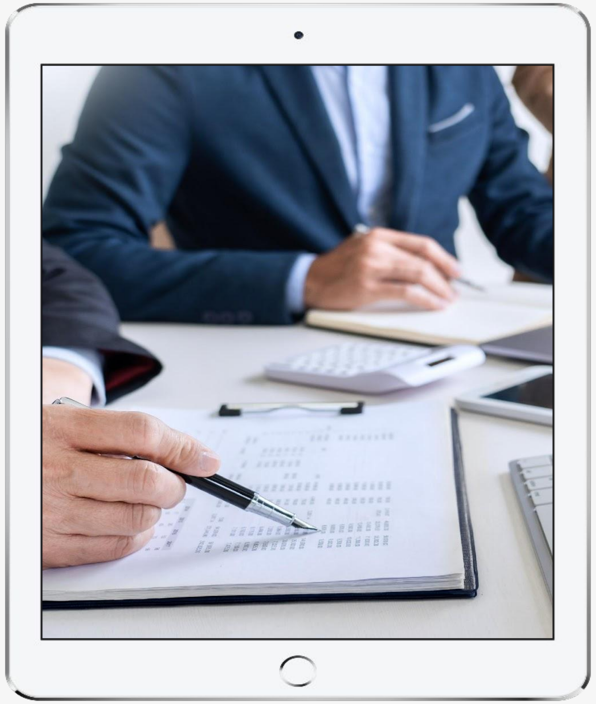
Steam provides robust community features like user reviews, forums, and groups. This ecosystem allows gamers to connect, share experiences, and discover new games through social interactions.

Steam Sales

Known for its seasonal sales, Steam offers significant discounts on a wide range of games, attracting a large number of users and boosting sales for developers.

Context

Leveraging a combination of PostgreSQL, Neo4j, and MongoDB databases with Python to extract, process, and analyze complex datasets related to games which are on the Steam platform.



Goals

Provide comprehensive insights into gaming trends on Steam aid in offering tailored game recommendations and understand industry collaboration patterns.

1. Conducting game score analysis to identify top genres and developers based on popularity and media.

2. Developing game recommendation system based on genres and descriptions.

3. Performing a collaboration analysis to understand the impact of developer-publisher collaborations on game popularity and pricing.

Part 2

Data Sources

Github repository:

https://github.com/leinstay/steamdb/blame/main/steamdb.json

Kaggle datasets:

Steam Store Games: https://www.kaggle.com/datasets/nikdavis/steam-store-games/

Steam Games Genres: https://www.kaggle.com/datasets/danieliusv/steam-games-genres

Steam Video Games: https://www.kaggle.com/datasets/tamber/steam-video-games

Data in PostgreSQL

columns we used:

appid (primary key)

name

release_date

categories

positive_rating

negative_rating

price

```
cur.execute("""
            CREATE TABLE IF NOT EXISTS steam (
                appid INT,
                name TEXT,
                release_date VARCHAR(100),
                english BOOLEAN,
                platforms VARCHAR(100),
                required_age INT,
                categories TEXT,
                steamspy_tags TEXT,
                achievements INT,
                positive_ratings INT,
                negative_ratings INT,
                average_playtime INT,
                median_playtime INT,
                owners VARCHAR(100),
                price FLOAT
            );
        """)
        conn.commit()
```

Over 25,000 rows

appid	name	release_dat	english	platforms	required_ag	categories	steamspy_t	achievemen	positive_ra	negative_ra	average_pl	median_pl	owners	price
10	Counter-Strike	2000/11/1	1	windows;mac;linux	0	Multi-playe	Action;FPS;	0	124534	3339	17612	317	10000000-200000	7.19
20	Team Fortress Classic	1999/4/1	1	windows;mac;linux	0	Multi-playe	Action;FPS;	0	3318	633	277	62	5000000-1000000	3.99
30	Day of Defeat	2003/5/1	1	windows;mac;linux	0	Multi-playe	FPS;World	0	3416	398	187	34	5000000-1000000	3.99
40	Deathmatch Classic	2001/6/1	1	windows;mac;linux	0	Multi-playe	Action;FPS;	0	1273	267	258	184	5000000-1000000	3.99
50	Half-Life: Opposing Force	1999/11/1	1	windows;mac;linux	0	Single-playe	FPS;Action;	0	5250	288	624	415	5000000-1000000	3.99
60	Ricochet	2000/11/1	1	windows;mac;linux	0	Multi-playe	Action;FPS;	0	2758	684	175	10	5000000-1000000	3.99
70	Half-Life	1998/11/8	1	windows;mac;linux	0	Single-playe	FPS;Classic	0	27755	1100	1300	83	5000000-1000000	7.19
80	Counter-Strike: Condition Z	2004/3/1	1	windows;mac;linux	0	Single-playe	Action;FPS;	0	12120	1439	427	43	10000000-200000	7.19
130	Half-Life: Blue Shift	2001/6/1	1	windows;mac;linux	0	Single-playe	FPS;Action;	0	3822	420	361	205	5000000-1000000	3.99
220	Half-Life 2	#######	1	windows;mac;linux	0	Single-playe	FPS;Action;	33	67902	2419	691	402	10000000-200000	7.19
240	Counter-Strike: Source	2004/11/1	1	windows;mac;linux	0	Multi-playe	Action;FPS;	147	76640	3497	6842	400	10000000-200000	7.19
280	Half-Life: Source	2004/6/1	1	windows;mac;linux	0	Single-playe	FPS;Action;	0	3767	1053	190	214	2000000-5000000	0
300	Day of Defeat: Source	2010/7/12	1	windows;mac;linux	0	Multi-playe	FPS;World	54	10489	1210	1356	134	5000000-1000000	7.19
320	Half-Life 2: Deathmatch	2004/11/1	1	windows;mac;linux	0	Multi-playe	Action;FPS;	0	6020	787	311	32	10000000-200000	3.99
340	Half-Life 2: Lost Coast	#######	1	windows;mac;linux	0	Single-playe	FPS;Action;	0	5783	1020	46	29	10000000-200000	0
360	Half-Life Deathmatch: Sour	2006/5/1	1	windows;mac;linux	0	Multi-playe	Action;FPS;	0	1362	473	102	81	5000000-1000000	0
380	Half-Life 2: Episode One	2006/6/1	1	windows;mac;linux	0	Single-playe	FPS;Action;	13	7908	517	281	184	5000000-1000000	5.79
400	Portal	#######	1	windows;mac;linux	0	Single-playe	Puzzle;First	15	51801	1080	288	137	10000000-200000	7.19
420	Half-Life 2: Episode Two	#######	1	windows;mac;linux	0	Single-playe	FPS;Action;	22	13902	696	354	301	5000000-1000000	5.79
440	Team Fortress 2	#######	1	windows;mac;linux	0	Multi-playe	Free to Play	520	515879	34036	8495	623	20000000-500000	0
500	Left 4 Dead	#######	1	windows;mac	0	Single-playe	Zombies;Co	73	17951	948	897	278	5000000-1000000	7.19
550	Left 4 Dead 2	#######	1	windows;mac;linux	0	Single-playe	Zombies;Co	70	251789	8418	1615	566	10000000-200000	7.19
570	Dota 2	2013/7/9	1	windows;mac;linux	0	Multi-playe	Free to Play	0	863507	142079	23944	801	100000000-20000	0
620	Portal 2	2011/4/18	1	windows;mac;linux	0	Single-playe	Puzzle;Co-c	51	138220	1891	1102	520	10000000-200000	7.19
630	Alien Swarm	2010/7/19	1	windows	0	Single-playe	Free to Play	66	17435	941	371	83	2000000-5000000	0
730	Counter-Strike: Global Offe	2012/8/21	1	windows;mac;linux	0	Multi-playe	FPS;Multip	167	2644404	402313	22494	6502	50000000-100000	0
1002	Rag Doll Kung Fu	#######	1	windows	0	Single-playe	Indie;Fighti	0	40	17	0	0	20000-50000	5.99
1200	Red Orchestra: Ostfront 41-	2006/3/14	1	windows;mac;linux	0	Multi-playe	World War	44	1562	223	232	258	500000-1000000	3.99
1250	Killing Floor	2009/5/14	1	windows;mac;linux	0	Single-playe	FPS;Zombi	285	53710	2649	1328	306	2000000-5000000	14.99
1300	SiN Episodes: Emergence	2006/5/10	1	windows	0	Single-playe	Action;FPS;	0	468	61	0	0	100000-200000	7.19
1500	Darwinia	2005/7/14	1	windows;mac;linux	0	Single-playe	Strategy;Inc	0	472	158	182	273	500000-1000000	7.19
1510	Uplink	2006/8/23	1	windows;mac;linux	0	Single-playe	Hacking;In	0	1602	152	65	77	500000-1000000	6.99
1520	DEFCON	2006/9/29	1	windows;mac;linux	0	Single-playe	Strategy;Inc	22	2057	344	80	119	500000-1000000	7.19

Data in MongoDB

columns we used:

description

popularity

sid (primary key)

meta_score

meta_uscore

igdb_score

igdb_uscore

igdb_popularity

```
[83]: data[0]
```


```
[83]: {'sid': 10,
 'store_url': 'https://store.steampowered.com/app/10',
 'store_promo_url': 'https://www.youtube.com/watch?v=oKC9SAF4JAc',
 'store_uscore': 97,
 'published_store': '2000-11-01',
 'published_meta': '2000-11-08',
 'published_stsp': '2000-11-01',
 'published_hltb': '1999-06-12',
 'published_igdb': '1999-06-12',
 'image': 'https://steamcdn-a.akamaihd.net/steam/apps/10/header.jpg',
 'name': 'Counter-Strike',
 'description': "Play the world's number 1 online action game. Engage in an incredibly realistic brand of terrorist
warfare in this wildly popular team-based game. Ally with teammates to complete strategic missions. Take out enemy
sites. Rescue hostages. Your role affects your team's success. Your team's success affects your role.",
 'full_price': 999,
 'current_price': 999,
 'discount': None,
 'platforms': 'WIN,MAC,LNX',
 'developers': 'Valve',
 'publishers': 'Valve',
 'languages': 'English,French,German,Italian,Spanish - Spain,Simplified Chinese,Traditional Chinese,Korean',
 'voiceovers': 'English,French,German,Italian,Spanish - Spain,Simplified Chinese,Traditional Chinese,Korean',
 'categories': 'Multi-player,PvP,Online PvP,Shared/Split Screen PvP,Valve Anti-Cheat enabled',
 'genres': 'Action',
 'tags': "Action,FPS,Multiplayer,Shooter,Classic,Team-Based,First-Person,Competitive,Tactical,1990's,e-sports,PvP,M
ilitary,Strategy,Score Attack,Survival,Assassin,1980s,Ninja,Tower Defense",
 'achievements': None,
 'gfq_url': 'https://gamefaqs.gamespot.com/pc/429818-counter-strike?ftag=MCD-06-10aaa1f',
 'gfq_difficulty': 'Just Right-Tough',
 'gfq_difficulty_comment': '<a href="/games/rankings?platform=19&amp;genre=54&amp;list_type=diff&amp;dlc=1&amp;page
=33&amp;game_id=429818&amp;min_votes=2#1656"><b>#1656</b></a> hardest PC action game (<a href="/games/rankings?plat
form=19&amp;list_type=diff&amp;dlc=1&amp;page=111&amp;game_id=429818&amp;min_votes=2#5600"><b>#5600</b></a> on PC,
<b>#22929</b> overall)',
 'gfq_rating': 3.9,
 'gfq_rating_comment': '<a href="/games/rankings?platform=19&amp;genre=54&amp;list_type=rate&amp;dlc=1&amp;page=21&
amp;game_id=429818&amp;min_votes=2#1079"><b>#1079</b></a> highest rated PC action game (<a href="/games/rankings?pl
```

```
 'gfq_rating': 3.47,
 'gfq_rating_comment': '<a href="/games/rankings?platform=19&amp;genre=54&amp;list_type=rate&amp;view_type=1&amp;dl
c=1&amp;page=35&amp;game_id=562917&amp;min_votes=2#1799"><b>#1799</b></a> lowest rated PC action game (<a href="/ga
mes/rankings?platform=19&amp;list_type=rate&amp;view_type=1&amp;dlc=1&amp;page=148&amp;game_id=562917&amp;min_votes
=2#7435"><b>#7435</b></a> on PC, <b>#30579</b> overall)',
 'gfq_length': 50.6,
 'gfq_length_comment': '<a href="/games/rankings?platform=19&amp;genre=54&amp;list_type=time&amp;dlc=1&amp;page=2&a
mp;game_id=562917&amp;min_votes=2#127"><b>#127</b></a> longest PC action game (<a href="/games/rankings?platform=19
&amp;list_type=time&amp;dlc=1&amp;page=30&amp;game_id=562917&amp;min_votes=2#1501"><b>#1501</b></a> on PC, <b>#4994
</b> overall)',
 'stsp_owners': 3500000,
 'stsp_mdntime': 20,
 'hltb_url': 'https://howlongtobeat.com/game?id=9634',
 'hltb_single': None,
 'hltb_complete': None,
 'meta_url': 'https://www.metacritic.com/game/pc/team-fortress-classic',
 'meta_score': None,
 'meta_uscore': 71,
 'grnk_score': None,
 'igdb_url': 'https://www.igdb.com/games/team-fortress-classic',
 'igdb_single': None,
 'igdb_complete': None,
 'igdb_score': None,
 'igdb_uscore': 70,
 'igdb_popularity': 1.67}
```

Data in Neo4j

information we used:

id (primary key)

game

developer

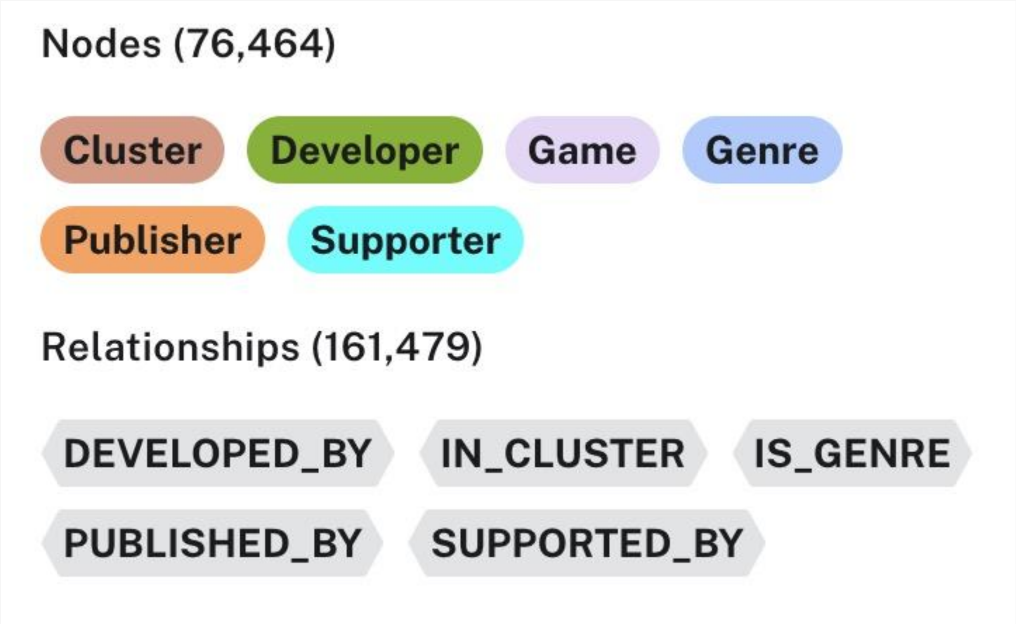
publisher

genre

developed_by

is_genre

published_by



Part 3

Methodology

Libraries, Game Score Analysis, Game Recommedation System, Collaboration Analysis

Libraries for Connection

- psycopg2: connect to postgres (local)
- pymongo: connect to mongodb (cloud)
- neo4j: connect to neo4j (cloud)

```python
def connect_postgres():
    #connect to PostgreSQL
    try:
        conn = psycopg2.connect(
            dbname="***",
            user="***",
            password="***",
            host="***",
        )
        print("Connected to the postgres successfully")
    except Exception as e:
        print("An error occurred:", e)
    cur=conn.cursor()
    return conn,cur
```

```python
def connect_mongodb():
    #connect to MongoDB
    uri = "***"
    client=MongoClient(uri,server_api=ServerApi('1'))
    db=client['steam']
    collection=db['steamdb']
    return collection,client
```

```python
def connect_neo4j():
    #connect to Neo4j
    URI="*"
    USERNAME="neo4j"
    PASSWORD="*"
    driver = GraphDatabase.driver(URI, auth=(USERNAME, PASSWORD))
    driver.verify_connectivity()
    return driver
```

Other Libraries

```
from collections import defaultdict
import pandas as pd
import numpy as np
from prettytable import PrettyTable
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.cluster import MiniBatchKMeans
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
import re
import nltk
from nltk.corpus import stopwords
```

Game Score Analysis

Queries:

- Top 5 highest media score and user score genres of all time
- The popularity shift of top 5 highest score genres over time
- Top 5 highest media score and user score developer of all time

Query 1: Top 5 highest media score and user score genres of all time

Pipeline: Use mongodb and postgres

select media score from mongodb

Insert result into Postgres

select steam table from postgres

Merge tables based on game ID
Apply Query on Postgres

Detailed query

From mongodb

```python
#get game info from MongoDB
json_games=collection.find({},{"_id":0,"sid":1,"gfq_rating":1,"meta_score":1,\
    "meta_uscore":1,"igdb_score":1,"igdb_uscore":1,"igdb_popularity":1})
```

Postgres query to merge and select top 5 genres with highest score

```sql
with t as (select s.appid,s.name,s.release_date,s.genres,j.gfq_rating,j.meta_score,
   j.meta_uscore,j.igdb_score,j.igdb_uscore,j.igdb_popularity
   from steam as s inner join json_scores as j on j.sid=s.appid)
   SELECT unnested_genres,round(AVG(gfq_rating),2) AS
avg_gfq_rating,round(AVG(meta_score),2) AS avg_meta_score,round(AVG(meta_uscore),2) AS
avg_meta_uscore,round(AVG(igdb_score),2) AS avg_igdb_score,round(AVG(igdb_uscore),2) AS
avg_igdb_uscore,round(AVG(igdb_popularity),2) AS avg_igdb_popularity
   FROM (SELECT unnest(string_to_array(genres, ';')) AS unnested_genres, gfq_rating,
meta_score,meta_uscore,igdb_score,igdb_uscore,igdb_popularity
    FROM t) GROUP BY unnested_genres
```

Query1 Result

Top 5 genres with highest media score and user score

	unnested_genres	avg_gfq_rating	avg_igdb_popularity	avg_score	avg_uscore
0	RPG	3.39	3.72	71.72	67.40
1	Early Access	3.15	2.71	71.68	67.06
2	Strategy	3.41	3.26	71.65	66.20
3	Sports	3.51	1.56	71.51	64.20
4	Nudity	3.22	2.60	70.62	65.16

- gain insights into user preferences.
- sets a standard for new games.

Query2: The popularity shift of top 5 highest score genres over time

```
with t as (select s.appid,s.name,s.release_date,s.genres,j.gfq_rating,j.meta_score,
j.meta_uscore,j.igdb_score,j.igdb_uscore,j.igdb_popularity
from steam as s inner join json_scores as j on j.sid=s.appid)
SELECT
unnested_genres,
release_year,
count(*) as num_games,
round(AVG(gfq_rating),2) AS avg_gfq_rating,
round(AVG(meta_score),2) AS avg_meta_score,
round(AVG(meta_uscore),2) AS avg_meta_uscore,
round(AVG(igdb_score),2) AS avg_igdb_score,
round(AVG(igdb_uscore),2) AS avg_igdb_uscore,
round(AVG(igdb_popularity),2) AS avg_igdb_popularity
FROM
(SELECT
        unnest(string_to_array(genres, ';')) AS unnested_genres,
        extract(year from to_date(release_date,'YYYY-MM-DD')) as release_year,
        gfq_rating,
        meta_score,
        meta_uscore,
        igdb_score,
        igdb_uscore,
        igdb_popularity
    FROM
        t)
GROUP BY
release_year,unnested_genres
order by unnested_genres,release_year
```

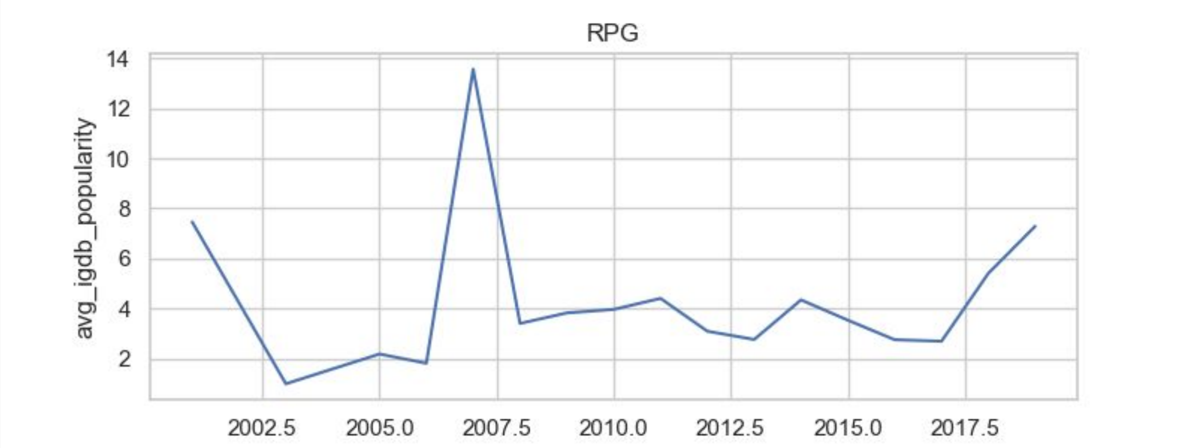
Basically similar to query 1.
Use igdb_popularity instead of average media score.
Extract year from game release date, add year to group by expression.

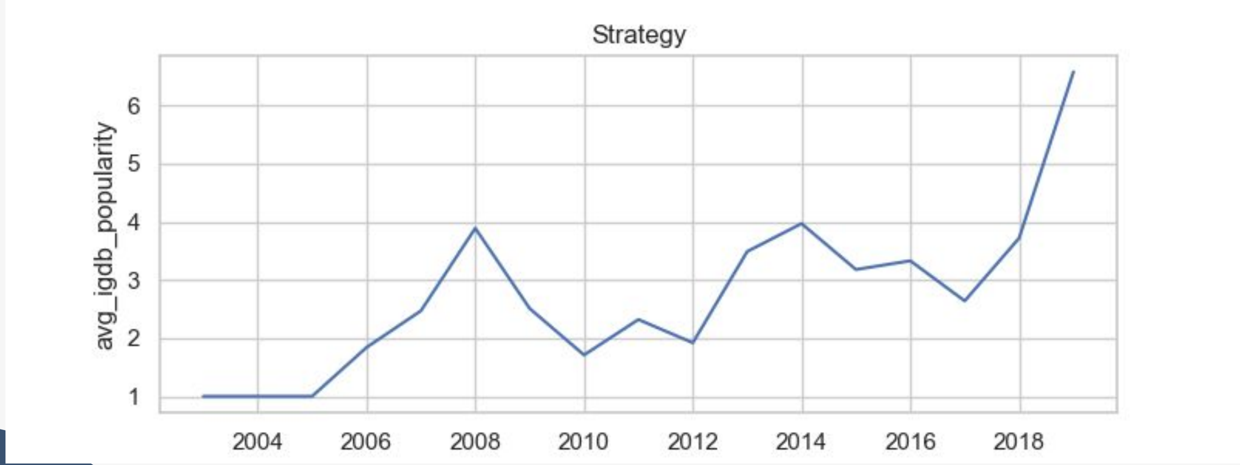
Query2 Merged Data sample

	unnested_genres	release_year	avg_igdb_popularity
0	Action	1997	1.00
1	Action	1998	14.82
2	Action	1999	2.56
3	Action	2000	14.15
4	Action	2001	4.04

Query2 Result

The popularity shift of top 5 highest score genres over time
(Take RPG and Strategy as examples)

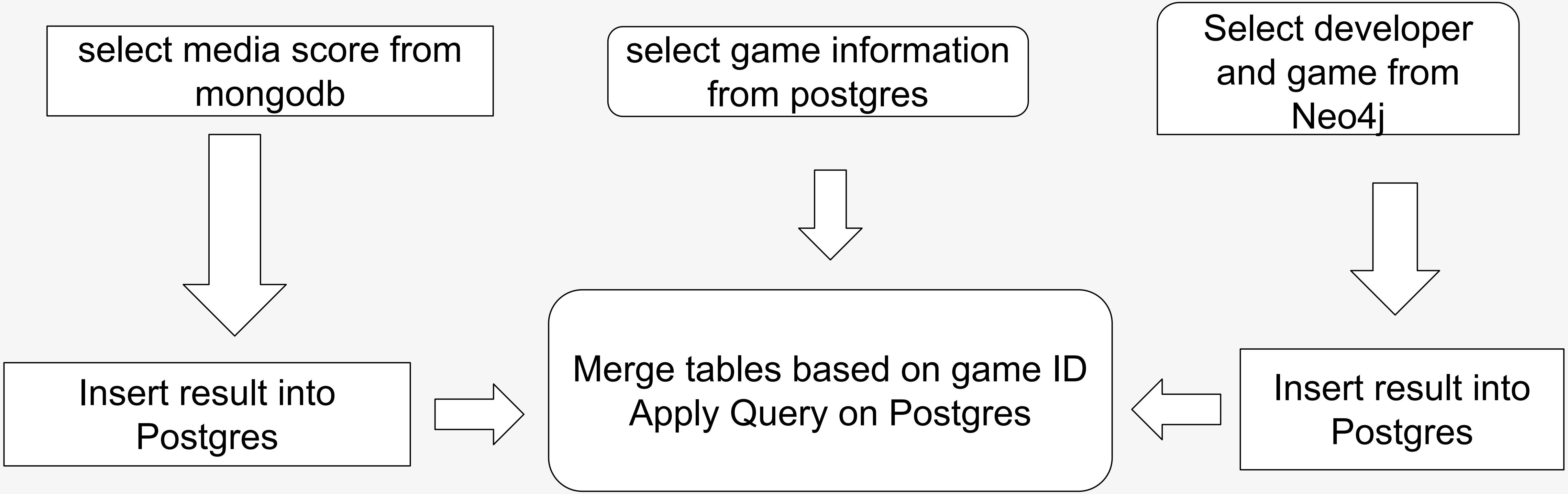




- popularity trends analysis
- help with development decisions

Query 3: Top 5 highest media score and user score developer of all time

Pipeline: Use mongodb, neo4j, and postgres



Detailed query

Neo4j query

```
query = """
    MATCH (g:Game)-[:DEVELOPED_BY]->(d:Developer)
    RETURN g.name as Name,toInteger(g.id) as appid,d.name as Developer
    """
```

Postgres query

```
with t as (
    select t1.*,s.release_date
    from (select g.appid,g."Developer",j.gfq_rating,j.meta_score,
j.meta_uscore,j.igdb_score,j.igdb_uscore,j.igdb_popularity
from game_developer as g inner join json_scores as j on j.sid=g.appid ) as t1 inner join steam as s
on s.appid=t1.appid
)
select
unnested_developer,
release_year,
count(*) as num_games,
round(AVG(gfq_rating),2) AS avg_gfq_rating,
round(AVG(meta_score),2) AS avg_meta_score,
round(AVG(meta_uscore),2) AS avg_meta_uscore,
round(AVG(igdb_score),2) AS avg_igdb_score,
round(AVG(igdb_uscore),2) AS avg_igdb_uscore,
round(AVG(igdb_popularity),2) AS avg_igdb_popularity
    FROM
(SELECT
        unnest(string_to_array(t."Developer", ';')) AS unnested_developer,
        extract(year from to_date(release_date,'YYYY-MM-DD')) as release_year,
        gfq_rating,
        meta_score,
        meta_uscore,
        igdb_score,
        igdb_uscore,
        igdb_popularity
    FROM
        t)
GROUP BY
release_year,unnested_developer
order by unnested_developer,release_year
```

Query3 Result

Top 5 highest media score and user score developer of all time

	unnested_developer	num_games	avg_gfq_rating	avg_igdb_popularity	avg_score	avg_uscore
0	InvertMouse	8	2.85	1.31	97.5	70.4
1	Gritfish	2	3.64	1.42	96.0	74.0
2	Flying Helmet Games	1	3.69	1.45	95.0	84.0
3	tobyfox	1	4.25	33.10	94.0	85.5
4	MAD Virtual Reality Studio	1	3.31	1.00	94.0	57.0

- help investors make decision
- help user make decision on whether to buy a game

Game Recommendation System

Queries:

- Recommend similar games based on genres

- Recommend high rating games based on key word clusters

Query 1: Recommend similar games based on genres

Pipeline: Use neo4j and postgres

1. User input a game name
2. select and output games with similar names from postgres using ilike expression
3. User input the index of correct game name, get corresponding game id
4. get genres of game from postgres based on game id, one game can have multiple genres
5. select and output games have a relation to all the genres selected in step 4 from neo4j

Detailed query

Interact with Postgres

```
#process input name for ilike
def format_name_for_ilike(name):
    # Split the name into words and convert each to lowercase
    words = name.split()
    lowercase_words = [word.lower() for word in words]

    # Join the words with '%'
    formatted_name = '%'.join(lowercase_words)

    # Add '%' at the beginning and end for the ILIKE expression
    ilike_expression = f'%{formatted_name}%'
    return ilike_expression
```

Interact with Neo4j

```
#return list of genres from id
def get_genres_from_id(id,session):
    n4j=session.run("match p=(g:Game)-[:IS_GENRE]->(genre:Genre) where g.id=$id RETURN genre",parameters={'id':id}).data()
    genres=[item['genre']['genre'] for item in n4j]
    return genres

#return dataFrame of games from genres(list)
def get_games_from_genres(genres,session):
    query = """
        MATCH (g:Game)
        WHERE ALL(genre IN $genres WHERE (g)-[:IS_GENRE]->(:Genre {genre: genre}))
        RETURN g.name as Name,g.id as Appid
        """
    n4j=session.run(query,genres=genres).data()
    return pd.DataFrame(n4j)
```

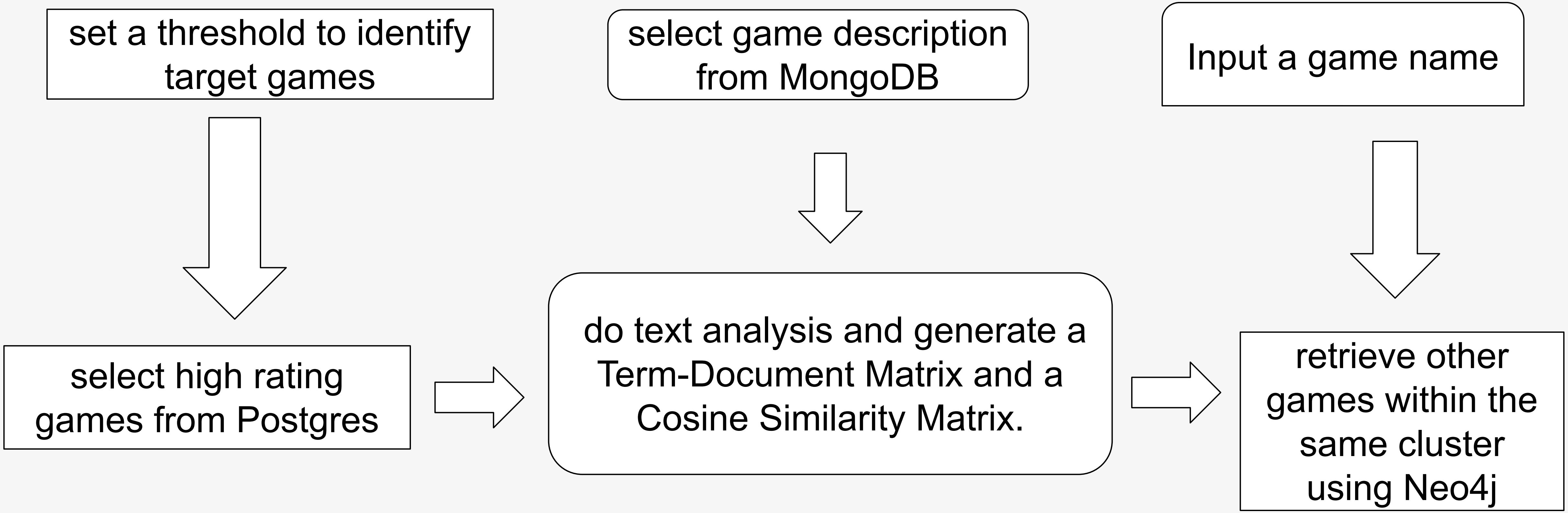
Query1 Result Example

Similar games for Sekiro™: Shadows Die Twice with genres ['Adventure', 'Action']

	Name	Appid
0	Portal 2	620
1	Thrillville®: Off the Rails™	6080
2	The Longest Journey	6310
3	Lost Planet™: Extreme Condition	6510
4	Tomb Raider: Legend	7000
5	X-Blades	7510
6	Tomb Raider: Anniversary	8000
7	Tomb Raider: Underworld	8140
8	Just Cause 2	8190
9	Death to Spies	9800
10	Blade Kitten	9940
11	Prototype™	10150
12	Bully: Scholarship Edition	12200
13	Grand Theft Auto IV	12210
14	Hunting Unlimited™ 2008	12530
15	Damnation	12790
16	Prince of Persia: Warrior Within™	13500

Query 2: Recommend high rating games based on key word clusters

Pipeline: Use postgres, mongodb and neo4j



Find high rating games

Using PostgreSQL

Calculate the positive ratings rate using the formula:
positive_ratings / (positive_ratings + negative_ratings)

```python
def get_games_above_threshold(threshold, cur):
    query = """
    SELECT name
    FROM steam
    WHERE (positive_ratings::FLOAT / NULLIF(positive_ratings + negative_ratings, 0)) > %s
    """
    cur.execute(query, (threshold,))
    return cur.fetchall()
```

Find high rating games

Using PostgreSQL

```
Pinged your deployment. You successfully connected to MongoDB!
Successfully connected to Neo4j.
Connected to the database successfully
Enter the threshold value (as a percentage, e.g., 0.8 for 80%):
0.98
```

- Input a threshold
- It will return all games with a positive rating rate that meets this threshold.

```
|                                Hexa Path                                |
|                       Pixel Puzzles 2: Paintings                        |
|                         Assault of the Robots                           |
|                           Virtual Skydiving                             |
|                              TOK HARDCORE                               |
|                               Foxyland 2                                |
|                       Bunker - Nightmare Begins                         |
|                 Pixel Art Monster - Color by Number                     |
|                          Easy puzzle: Animals                           |
|              Magic Farm 2: Fairy Lands (Premium Edition)                |
|                           The Cat and the Box                           |
|                    Awakening: The Dreamless Castle                      |
|                    Argonauts Agency: Pandora's Box                      |
|                              City Zombies                               |
|                        VTB Basketball League VR                         |
|                                Moon Pool                                |
|                        Funny Bunny: Adventures                          |
|                            Franchise Wars                               |
|                  New Yankee 6: In Pharaoh's Court                       |
|                   Azurael's Circle: Chapter 4                           |
|                              Magic Clouds                               |
|                       Die, zombie sausage, die!                         |
|                            Nyasha Valkyrie                              |
|                             Peas Adventure                              |
|                             Blacksmith Run                              |
|                            MonteCube Dodge                              |
|                   The Mystery of Bikini Island                          |
|                             CaptainMarlene                              |
|                              Old Edge II                                |
|                            Room of Pandora                              |
|                      New Yankee 7: Deer Hunters                         |
|                               Rune Lord                                 |
+-------------------------------------------------------------------------+
```

Text Analysis

Using MongoDB

```python
def get_game_descriptions(game_names):
    collection, _ = connect_mongodb()
    games = collection.find({"name": {"$in": game_names}}, {"_id": 0, "description": 1})
    return [game['description'] for game in games if 'description' in game and game['description']]

def custom_preprocessor(text):
    text = re.sub(r'\b\d+\b', '', text)
    return text

def analyze_texts(texts):
    if not texts:
        print("No descriptions available for analysis.")
        return None
    additional_stop_words = {'quot', 'games', 'And'}
    custom_stop_words = list(set(stopwords.words('english')).union(additional_stop_words))

    vectorizer = TfidfVectorizer(
    preprocessor=custom_preprocessor,
    token_pattern=r'\b[a-zA-Z]{2,}\b',
    max_df=0.10,
    min_df=0.01,
    stop_words=custom_stop_words)

    tfidf_matrix = vectorizer.fit_transform(texts)
    feature_names = vectorizer.get_feature_names_out()

    # print TF-IDF matrix
    print("TF-IDF Matrix:")
    dense_tfidf_matrix = tfidf_matrix.todense()
    print(pd.DataFrame(dense_tfidf_matrix, columns=vectorizer.get_feature_names_out()))

    cosine_similarities = cosine_similarity(tfidf_matrix)
    return dense_tfidf_matrix, pd.DataFrame(cosine_similarities), feature_names
```

Term-Document Matrix：

```
TF-IDF Matrix:
       AI  AND  About  Access  Achievements  Action  Adventure  After  ...  written  wrong  year  years  yet     young  zomb
ies  zone
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  0.000000
0.0   0.0
1     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  0.000000
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  0.148522
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  0.000000
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  0.000000
0.0   0.0
...   ...  ...    ...     ...           ...     ...        ...    ...  ...      ...    ...   ...    ...  ...       ...
...   ...
2936  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  0.000000
0.0   0.0
2937  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  0.000000
0.0   0.0
2938  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  0.000000
0.0   0.0
2939  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  0.000000
0.0   0.0
2940  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  0.000000
0.0   0.0

[2941 rows x 1646 columns]
```

Cosine Similarity Matrix：

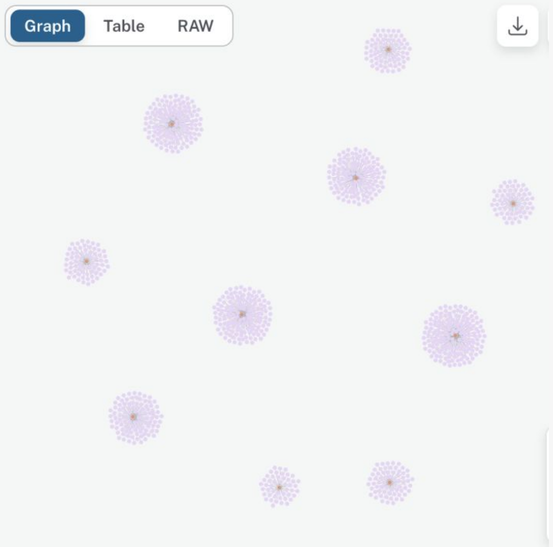
```
Cosine Similarity Matrix:
          0         1         2         3         4         5     ...      2935      2936      2937      2938      2939
 2940
0     1.000000  0.088747  0.023010  0.017112  0.008858  0.029584  ...  0.013565  0.073283  0.000000  0.021658  0.000000  0.0
20437
1     0.088747  1.000000  0.049821  0.042742  0.041497  0.013353  ...  0.040190  0.026062  0.000000  0.000000  0.013866  0.0
57071
2     0.023010  0.049821  1.000000  0.112165  0.028510  0.049703  ...  0.032929  0.000000  0.012487  0.000000  0.018178  0.0
00000
3     0.017112  0.042742  0.112165  1.000000  0.027845  0.027978  ...  0.000000  0.000000  0.000000  0.000000  0.020479  0.1
25627
4     0.008858  0.041497  0.028510  0.027845  1.000000  0.015001  ...  0.028879  0.000000  0.000000  0.000000  0.000000  0.0
00000
...        ...       ...       ...       ...       ...       ...  ...       ...       ...       ...       ...       ...
  ...
2936  0.073283  0.026062  0.000000  0.000000  0.000000  0.000000  ...  0.000000  1.000000  0.042854  0.082735  0.025372  0.1
21346
2937  0.000000  0.000000  0.012487  0.000000  0.000000  0.019364  ...  0.010546  0.042854  1.000000  0.000000  0.028204  0.0
15192
2938  0.021658  0.000000  0.000000  0.000000  0.000000  0.000000  ...  0.000000  0.082735  0.000000  1.000000  0.061516  0.0
25265
2939  0.000000  0.013866  0.018178  0.020479  0.000000  0.000000  ...  0.000000  0.025372  0.028204  0.061516  1.000000  0.0
32881
2940  0.020437  0.057071  0.000000  0.125627  0.000000  0.031768  ...  0.064598  0.121346  0.015192  0.025265  0.032881  1.0
00000

[2941 rows x 2941 columns]
```

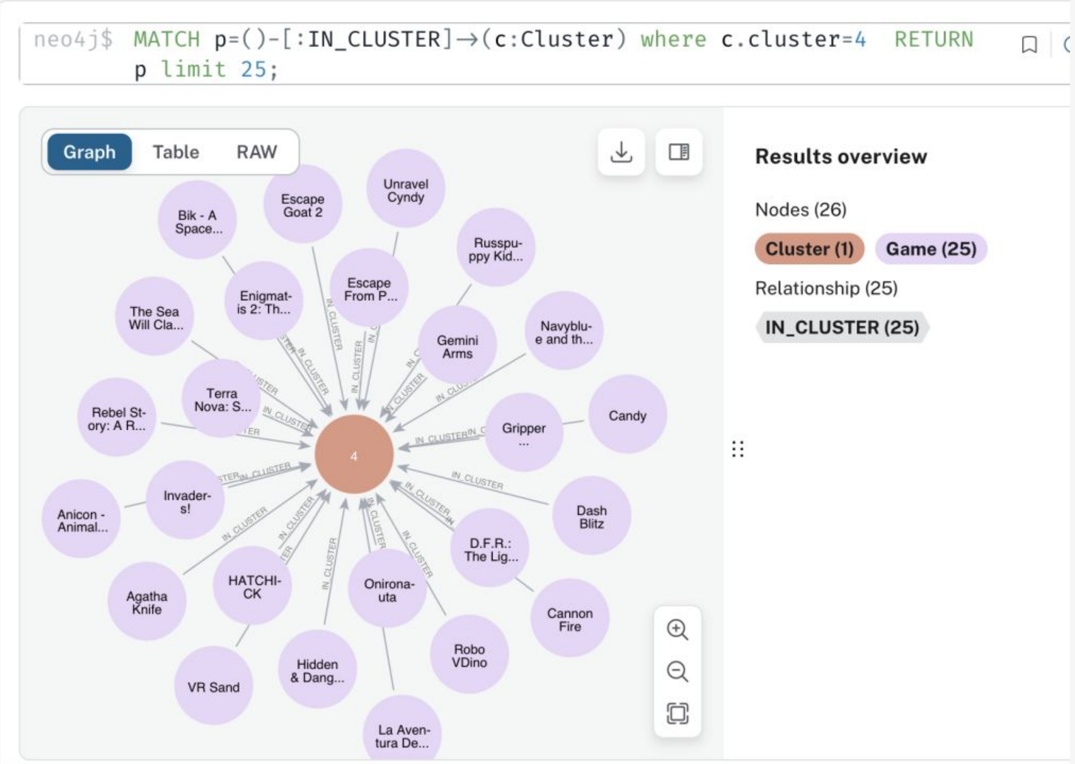
View clustering results and find similar games

Using Neo4j

- Perform clustering using the results of the term-document matrix.
- Prior to this, due to slow code execution speed, we conducted PCA.

```
Enter the number of clusters (integer):
10
```






```python
def update_game_cluster(game_name, cluster_label, session):
    try:
        query = """
        MATCH (g:Game {name: $game_name})
        SET g.cluster = $cluster_label
        RETURN g.name, g.cluster
        """
        result = session.run(query, game_name=game_name, cluster_label=cluster_label)
        return result.data()
    except Exception as e:
        print(f"Error updating game cluster in Neo4j: {e}")
        return None
```

View clustering results and find similar games

Using Neo4j

0: Mode, Challenge, Story
1: space, ship, planet
2: Vive, HTC (maybe related to virtual reality)
3: Collector, Edition, extras
4: monsters, battle, combat
5: blocks, block, master
6: endings, mysterious, hidden
7: tiles, board, pieces
8: score, multiplayer, arcade
9: platformer, solve, Steam

We can infer the game style through keywords !

```
Cluster 0 - Keywords: Mode, Challenge, Story
                          Game Name                 Keywords
0        Farm Frenzy 3: American Pie  Mode, Challenge, Story
1                          Icebound  Mode, Challenge, Story
2                          Goscurry  Mode, Challenge, Story
3              The First Time I Died  Mode, Challenge, Story
4                  Ruler by Default  Mode, Challenge, Story
..                               ...                     ...
79  Lost Lands: Mistakes of the Past  Mode, Challenge, Story
80                          Choconoa  Mode, Challenge, Story
81                    SmartyTale 2D  Mode, Challenge, Story
82                    Time to Fight  Mode, Challenge, Story
83                          Apolune  Mode, Challenge, Story

[84 rows x 2 columns]
```

Recommend similar games:

```
Which game do you want to query for similar games?
Finding Paradise
Games in the same cluster as 'Finding Paradise':
                                   Similar Games
0                                  Sniper Fodder
1                                Marty Thinks 4D
2                                      Queendoom
3                     Creeper World 3: Arc Eternal
4                               Dungeons of Hell
..                                           ...
```

Collaboration Analysis

Goal:

Analyze how the collaborations of developers and publishers influent the igdb_popularity and price of games.

Retrieve the relationship between games, their publishers, and developers from Neo4j

Fetch igdb_popularity and appid from MongoDB

Fetch appid and price from PostgreSQL

Merge Dataframes based on appid in Python

Collaboration Analysis

Querying Neo4j:

Retrieve the relationship between games, their publishers, and developers from Neo4j, including the count of games (num_games) for each publisher-developer pair.

```
# Neo4j Query
neo4j_query = """
MATCH (publisher:Publisher)<-[:PUBLISHED_BY]-(game:Game)-[:DEVELOPED_BY]->(developer:Developer)
WITH publisher, developer, COUNT(game) AS num_games, COLLECT(game) AS games
UNWIND games AS game
RETURN toInteger(game.id) as appid, developer.name AS developer,
       publisher.name AS publisher, num_games
"""
```

Collaboration Analysis

Querying MongoDB and PostgreSQL:

1. Fetch igdb_popularity and appid from MongoDB
2. Fetch appid and price from PostgreSQL.

Note: Rename sid to appid for consistency.

```
# MongoDB Query
cursor_mongo = collection.find({}, {"sid": 1, 'igdb_popularity': 1})
df_mongo = pd.DataFrame(list(cursor_mongo))
df_mongo = df_mongo.rename(columns={'sid': 'appid'})

# PostgreSQL Query
cur.execute("SELECT appid, price FROM steam;")
columns = [desc[0] for desc in cur.description]
data = cur.fetchall()
df_postgres = pd.DataFrame(data, columns=columns)
```

Collaboration Analysis

Merging DataFrames:

Merge those data from Neo4j, MongoDB, and PostgreSQL into a single DataFrame based on the appid which means the id of game.

```
# Merge DataFrames
df_merged = pd.merge(df_neo4j, df_mongo, on='appid', how='inner')
df_merged = pd.merge(df_merged, df_postgres, on='appid', how='inner')
df_merged.dropna(subset=['igdb_popularity'], inplace=True)
```

Collaboration Analysis

Performing Analysis:

1. Group the dataframe by the developer-publisher pair
2. Calculate the mean of igdb_popularity, price, and num_games for each group.

```
# Analysis
analysis_result = df_merged.groupby(['developer', 'publisher']).agg({
    'igdb_popularity': 'mean',
    'price': 'mean',
    'num_games': 'mean'
}).reset_index()
```

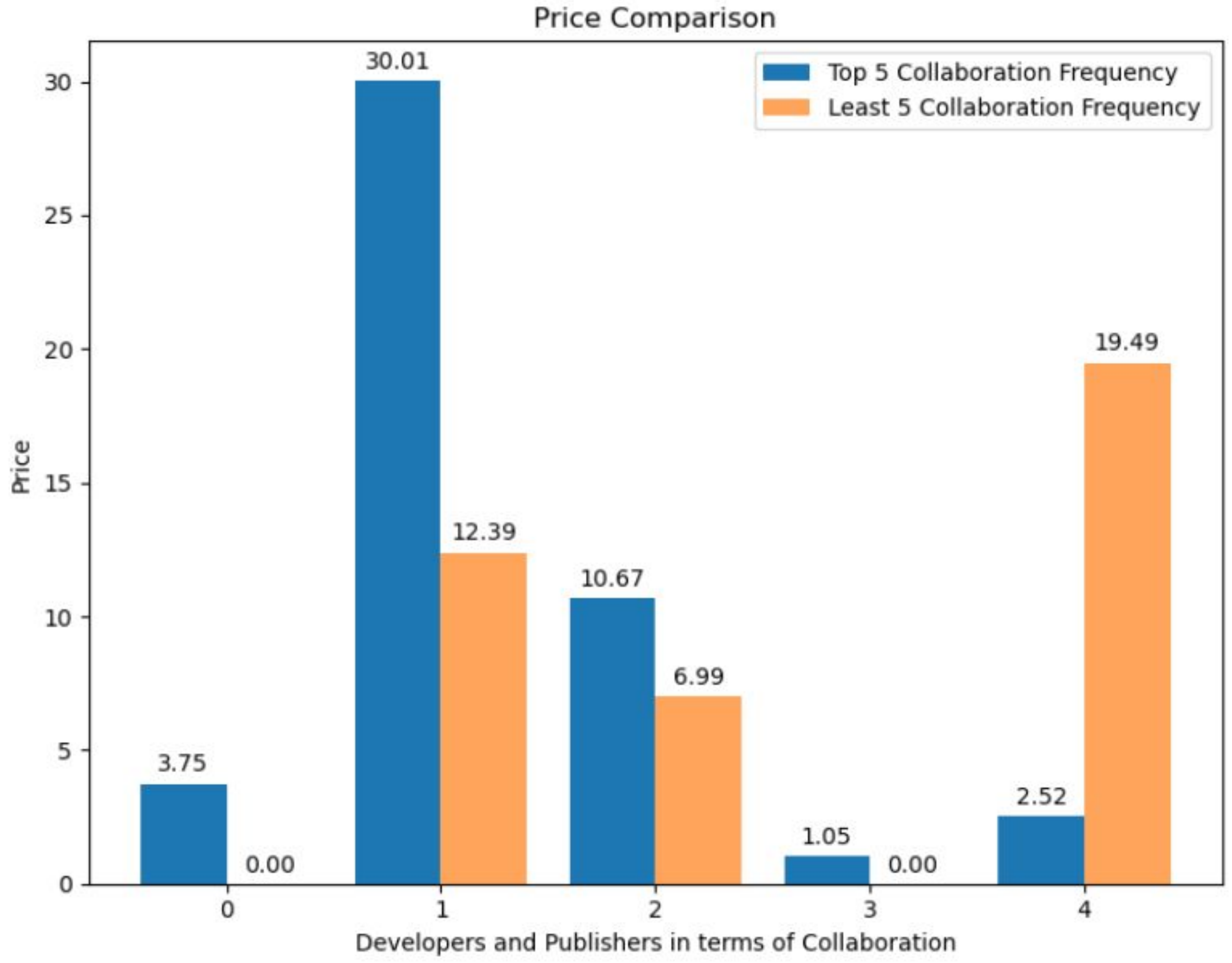
Collaboration Analysis

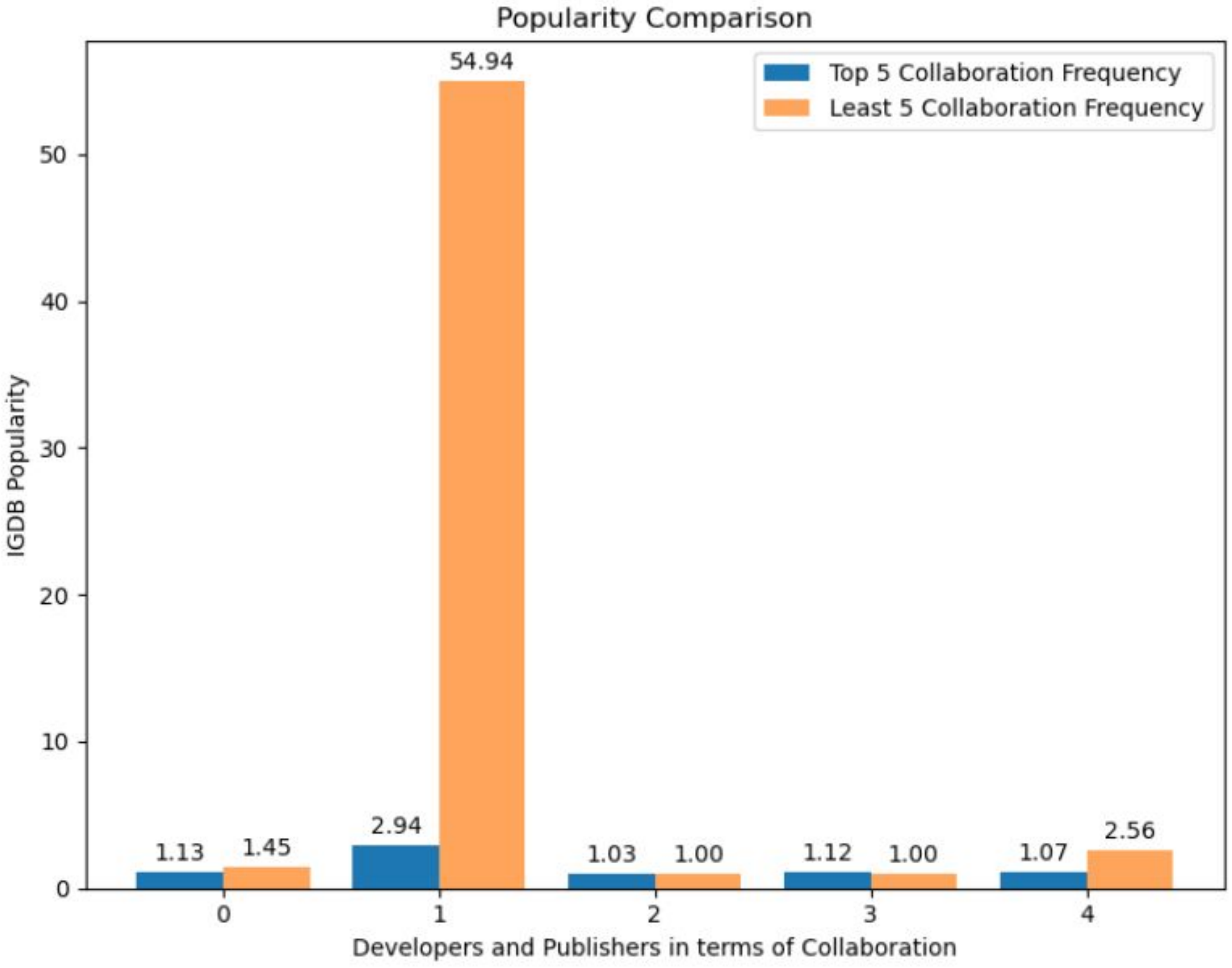
Results:

```
Top 5 num_games:
                      developer                        publisher  igdb_popularity      price  num_games
1982            Choice of Games                  Choice of Games         1.134762   3.745556       94.0
5654  KOEI TECMO GAMES CO., LTD.  KOEI TECMO GAMES CO., LTD.         2.936939  30.012857       69.0
8917            Ripknot Systems                  Ripknot Systems         1.029545  10.671818       62.0
7390         Nikita "Ghost_RUS"                  Ghost_RUS Games         1.117273   1.053636       50.0
6023    Laush Dmitriy Sergeevich                    Laush Studio         1.067000   2.520000       47.0

Least 5 num_games:
                        developer           publisher  igdb_popularity  price  num_games
4793            Hex Entertainment   Hex Entertainment             1.45   0.00        1.0
4794                HexGameStudio       HexGameStudio            54.94  12.39        1.0
4796             HexWar Games Ltd    HexWar Games Ltd             1.00   6.99        1.0
4798   Hexagon Games;NAMI TENTOU       Hexagon Games             1.00   0.00        1.0
12689                  黄昏フロンティア  SUNFISH Co., Ltd.             2.56  19.49        1.0
```

Collaboration Analysis

Results:





Part 4

Demonstration

THANK YOU!

Chenze Fan, Jingyi Zhou, Sirui Li