POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Authors: **Samuele Scherini 10674683**

Matteo Spreafico 10669138

Ludovica Tassini 10663137

Academic Year: 2023-2024

Contents

1 | Introduction

In this report, we will describe the analyses we have performed on two open datasets taken from the web, using two different technologies.

1.1. First Dataset (MongoDB)

The first study utilizes a dataset named SteamDB, which was assembled by Martin Bustos using Steam APIs. The dataset contains all the games available on Steam, and includes a wide range of variables, such as prices and genres, allowing for the formulation of detailed and multifaceted analytical queries. The selection of this dataset was deliberate, based on its richness and comprehensiveness, providing researchers with the opportunity to conduct diverse and advanced investigations in the field of digital gaming. This dataset is well-suited for a MongoDB implementation due to the large volume of data and the number of fields fitting well within the document format.

1.2. Second Dataset (Neo4j)

The second study utilizes the Instagram Influence Maximization (IM) dataset, which was curated from April to May 2020. The dataset consists of users from 24 Malaysian private universities. The data was obtained through the Instagram API and reputable third-party platforms. This dataset was chosen due to its relevance to social influence dynamics and the unique characteristics of the Instagram platform. It is well-suited for a Neo4j implementation, known for its effectiveness in social network analysis. This allows for detailed investigations into the complexities of Instagram algorithms and user influence in the Malaysian context. The dataset serves as a valuable resource for enhancing our understanding of the underlying dynamics that drive social influence on Instagram.

2 | Datasets

In this chapter, we will describe the two datasets more in detail, focusing on their content, their structure, and their non-relational implementation (providing a suited data schema).

2.1. First Dataset (MongoDB)

2.1.1. Dataset Description

This dataset, comprising 83,560 documents, unfolds the SteamDB's gaming landscape. Crafted by the author, Martin Bustos, it stems from meticulous utilization of SteamAPI and a dedicated web scraper. The scraper's codebase is openly accessible on GitHub at https://github.com/FronkonGames/Steam-Games-Scraper, underlining the dataset's transparency and reproducibility.

Dataset Credits
Author: Martin Bustos
Title: "Steam Games Dataset"
Year: 2022 - 2023
Link: https://www.kaggle.com/datasets/fronkongames/steam-games-dataset?select=games.json
DOI: 10.34740/kaggle/ds/2109585

2.1.2. Dataset Structure

The table below delineates the representation of data in the dataset, focusing specifically on the "games.json" file.

FIELD	DESCRIPTION
AppID	Game AppID
Name	Name of the videogame on steam.
Release date	Release date of the game
Estimated owners	Estimated number of people that possess the game

Peak CCU	Peak number of concurrent players
Required age	Age required to play
Price	Price in U.S. dollars
DLC count	number of DLCs
About the game	Short description of the game
Supported languages	Languages supported by the game
Full audio languages	Languages supported by the game
Reviews	Game reviews
Header image	URL of the image in the store page
Website	URL of the Website of the videogame
Support url	URL of the support for the videogame
Support email	Email of the support for the videogame
Windows	State if the game is available on Windows
Mac	State if the game is available on MacOS
Linux	State if the game is available on Linux
Metacritic score	Score of the game given by Metacritic
Metacritic url	URL of the review on the Metacritic website
User score	*The explanation is missing in the official documentation*
Positive	Number of positive reviews
Negative	Number of negative reviews
Score rank	*The explanation is missing in the official documentation*
Achievements	Number of achievements
Recommendations	Number of the recommendations of the videogame
Notes	Additional notes regarding the videogame
Average playtime forever	Average playtime since the steam release
Average playtime two weeks	Average playtime in the last two weeks
Median playtime forever	Median playtime since the steam release
Median playtime two weeks	Median playtime in the last two weeks
Developers	Name of the development company or the developer name
Publishers	Name of the publishing company
Categories	List of the categories of the videogame
Genres	List of genres of the videogame
Tags	Additional tags related to the game
Screenshots	List of screenshots from the videogame
Movies	List of movies from the videogame

Table 2.1: MongoDB Dataset Structure

2.1.3. Non-relational Implementation

Implementing the MongoDB database from the SteamDB dataset involved a systematic process. First, the dataset was divided into ten distinct chunks using a Python script provided within the assignment folder, in order to allow the MongoDB server to import them. Subsequently, the dataset's JSON fields were directly mapped to MongoDB, enabling a seamless transition of data into the document-oriented database. As a notable enhancement, a new field named `release_date_iso` (type: `ISODate`) was created based on the existing `release_date` field (type: `string`).

The following script was employed for generating the `release_date_iso` field:

```
db.steam_collection.find().forEach(function(doc) {
    var customDate = doc.release_date;
    var isoDate = new Date(customDate).toISOString();
    db.steam_collection.updateOne({_id: doc._id}, {$set: {release_date_iso:
        ISODate(isoDate)}});
})
```

This transformation proved valuable, allowing straightforward grouping by year and facilitating essential queries related to temporal patterns. These steps permit the optimization of data handling and set the stage for efficient querying within the MongoDB framework.

The following image is an example of a document that represents the result of our implementation.

Figure 2.1: Partial document in MongoDB

2.2. Second Dataset (Neo4j)

2.2.1. Dataset Description

This dataset pertains to Instagram social network data for the Influence Maximization (IM) task. Collected between April and May 2020, it originates from the Instagram users of 24 Malaysian private universities, acquired through the Instagram API and various third-party platforms. Primarily comprising Malaysian users, the dataset encompasses 70,409 nodes/users and 1,031,348 edges/connections (followees and followers).

Dataset Credits
Authors: Kristo Radion Purba; David Asirvatham; Raja Kumar Murugesan
Title: "Influence Maximization Diffusion Models Based On Engagement and Activeness on Instagram"
Journal: Journal of King Saud University - Computer and Information Sciences
Year: 2020
Paper Link: https://doi.org/10.1016/j.jksuci.2020.09.012
Dataset Link: https://www.kaggle.com/datasets/krpurba/im-instagram-70k
DOI: 10.34740/kaggle/ds/2109585

2.2.2. Dataset Structure

The following table delineates the representation of data in the dataset, focusing specifically on the "Instagram User Stats.csv" file.

pos	Number of posts
flr	Number of Followers
flg	Number of Following
eg	Engagement grade, a scale of 1 to 12 that reflects the strength of engagement rate relative to the number of followers *
er	Engagement Rate, i.e. likes+comments / followers
fg	Followers growth % in a month
op	Outsiders percentage %, i.e. non-followers that liked a post, divided by all unique likers

Table 2.2: Users Dataset Structure

* *Note that in the dataset description* ***eg*** *is said to be a value between 1 and 12, while in the dataset the values are between 0 and 1, representing the value as a percentage.*

Then, there's a second file named "Network for IC LT.txt", which contains the relationships between the nodes. More specifically, the following table delineates, for each entry, the provided attributes.

source	**target**	**weight**
Source node	Destination node	Value of the relationship (weight = 1/followees)

Table 2.3: Relationships Dataset Structure

2.2.3. Non-relational Implementation

To implement the Instagram Influence Maximization (IM) dataset in Neo4j, several steps were taken. First, data from "Instagram User Stats.csv" was used to create nodes with relevant fields. Then, relationships between users were established using the "Network for IC LT.txt" file, which facilitated IM tasks.

The following scripts were employed for importing data into Neo4j:

1. CREATE CONSTRAINT FOR (u:User) REQUIRE u.user_id IS UNIQUE;

2. LOAD CSV WITH HEADERS FROM "file:///InstagramUserStats.csv" AS row
WITH row WHERE row.id IS NOT NULL
MERGE (u:User {user_id: toInteger(row.id)})
ON CREATE SET u.num_posts = toInteger(row.pos),
u.num_followers = toInteger(row.flr),
u.num_following = toInteger(row.flg),
u.engagement_grade = toFloat(row.eg),
u.engagement_rate = toFloat(row.er),
u.followers_growth_per_month = toFloat(row.fg),
u.outsiders_percentage = toFloat(row.op)
ON MATCH SET u.count = coalesce(u.count, 0) + 1;

3. LOAD CSV WITH HEADERS FROM 'file:///Network.csv' AS row
MATCH (u1:User {user_id: toInteger(row.Source)})
MATCH (u2:User {user_id: toInteger(row.Target)})
WHERE u1.user_id <= 20000
CREATE (u1)−[:HAS_INFLUENCE_OVER {influence_value:
toFloat(row.Weight)}]−>(u2);

4. LOAD CSV WITH HEADERS FROM 'file:///Network.csv' AS row
MATCH (u1:User {user_id: toInteger(row.Source)})
MATCH (u2:User {user_id: toInteger(row.Target)})
WHERE 20000 < u1.user_id <= 40000
CREATE (u1)−[:HAS_INFLUENCE_OVER {influence_value:
toFloat(row.Weight)}]−>(u2);

5. LOAD CSV WITH HEADERS FROM 'file:///Network.csv' AS row
MATCH (u1:User {user_id: toInteger(row.Source)})
MATCH (u2:User {user_id: toInteger(row.Target)})
WHERE 40000 < u1.user_id
CREATE (u1)−[:HAS_INFLUENCE_OVER {influence_value:
toFloat(row.Weight)}]−>(u2);

The following image is an example that represents the result of our implementation.

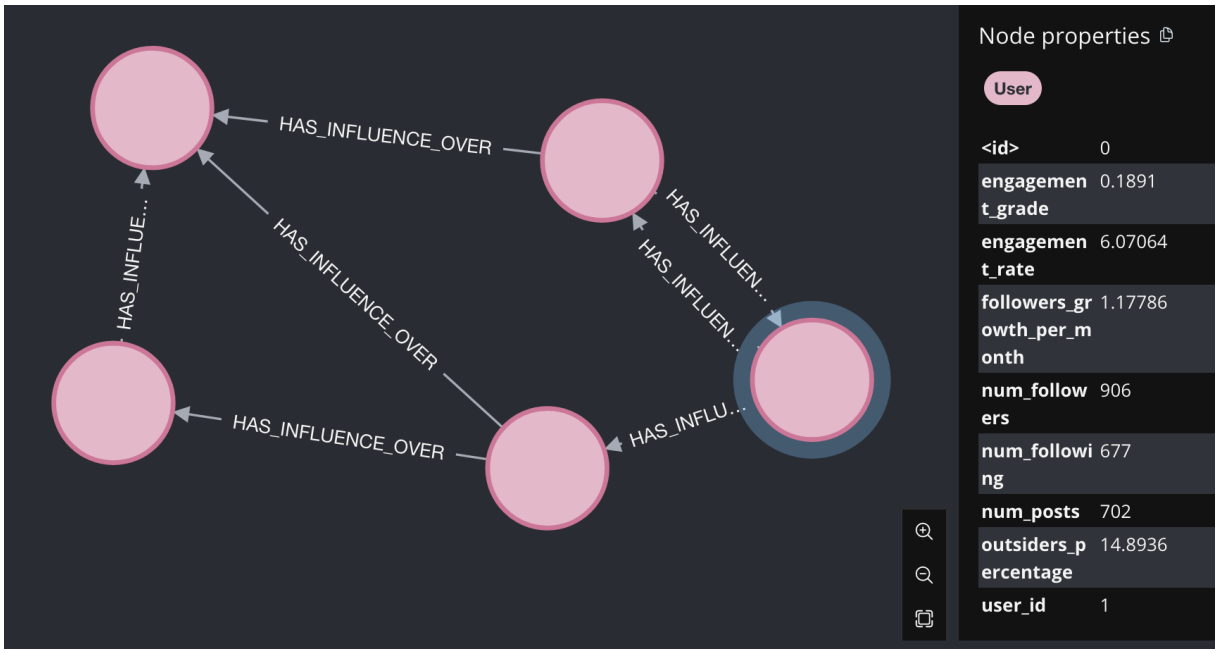

Figure 2.2: Partial graph taken from our Neo4j implementation

3 | Queries

In this chapter, we will show the queries we have performed against the two datasets. In particular, we will describe what they are supposed to do, their text in the appropriate query language, and their (potentially partial) results.

3.1. First Dataset (MongoDB)

In this section, we explore MongoDB queries on the SteamDB dataset, investigating gaming dynamics such as pricing trends, Metacritic scores, platform support, language diversity, and genre/publisher distributions. This detailed analysis offers various insights, forming a foundation for informed decision-making and strategic analysis in the gaming industry.

3.1.1. Query #1

Find the non-free game that has the lowest price among the games that have a Metacritic score of at least 90/100:

```
db.steam_collection.aggregate([
    {"$match": {"$and": [{"metacritic_score": {"$gte": 90}}, {"price": {"$gt": 0}}]}},
    {"$sort": {"price": 1}}, {"$limit": 1}
])
```

```
{
  _id: ObjectId('658721f251907a3901acf4d3'),
  name: 'Silent Hunter® III',
  price: 2.49,
  metacritic_score: 90
}
```

Figure 3.1: MongoDB Query 1 Results

3.1.2. Query #2

Find how many free games are available per genre in the "Single-player" category:

```
db.steam_collection.aggregate([
    {"$match": {"$and": [{"price": 0}, {"categories": "Single-player"}]}},
    {"$unwind": "$genres"},
    {"$group": {"_id": "$genres", "count": {"$sum": 1}}}
])
```

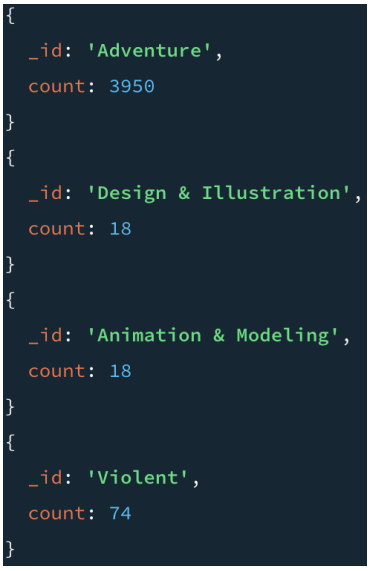

Figure 3.2: MongoDB Query 2 Results (Partial)

3.1.3. Query #3

Compute the total number of "Racing" tags for each publisher. Show only the publishers for which there's at least one "Racing" tag:

```
db.steam_collection.aggregate([
    {"$unwind": "$publishers"},
    {"$group": {"_id": "$publishers", "totalRacingTags": {"$sum": "$tags.Racing"}}},
    {"$match": {"totalRacingTags": {"$gt": 0}}}
])
```

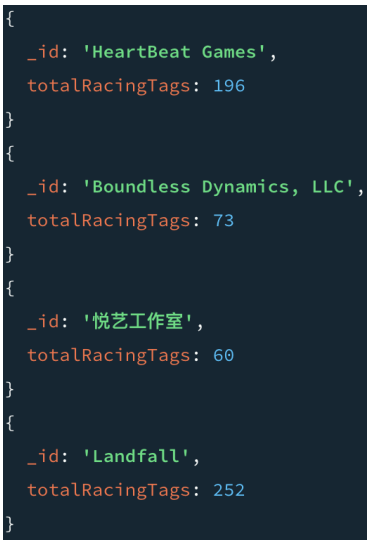

Figure 3.3: MongoDB Query 3 Results (Partial)

3.1.4. Query #4

For each genre compute the average playtime (both forever and two weeks):

```
db.steam_collection.aggregate([
    {"$unwind": "$genres"},
    {"$group": {
        "_id": "$genres",
        "averagePlaytime": {"$avg": "$average_playtime_forever"},
        "averagePlaytime2Weeks": {"$avg": "$average_playtime_2weeks"}
        }
    },
    {"$project": {
        "_id": 0,
        "genre": "$_id",
        "averagePlaytime": 1,
        "averagePlaytime2Weeks": 1
        }
    },
    {"$sort": {"averagePlaytime": -1, "averagePlaytime2Weeks": -1}}
])
```

```
{
  averagePlaytime: 1056.765,
  averagePlaytime2Weeks: 7.13,
  genre: 'Audio Production'
}
{
  averagePlaytime: 983.6,
  averagePlaytime2Weeks: 22.98181818181818,
  genre: 'Web Publishing'
}
{
  averagePlaytime: 818.6957040572793,
  averagePlaytime2Weeks: 27.471360381861576,
  genre: 'Utilities'
}
{
  averagePlaytime: 635.1483739837398,
  averagePlaytime2Weeks: 9.28048780487805,
  genre: 'Design & Illustration'
}
```


Figure 3.4: MongoDB Query 4 Results (Partial)

3.1.5. Query #5

For each year compute the average price of the non-free games released:

```
db.steam_collection.aggregate([
    {"$match": {"price": { "$ne": 0 }}},
    {"$group": {
        "_id": {"year": {"$year": "$release_date"}},
        "averagePrice": {"$avg": "$price"},
        "count": {"$sum": 1}
        }
    },
    {"$project": {
        "_id": 0,
        "year": "$_id.year",
        "averagePrice": 1,
        "gameCount": "$count"
        }
    },
    {"$sort": {"year": 1}}
])
```

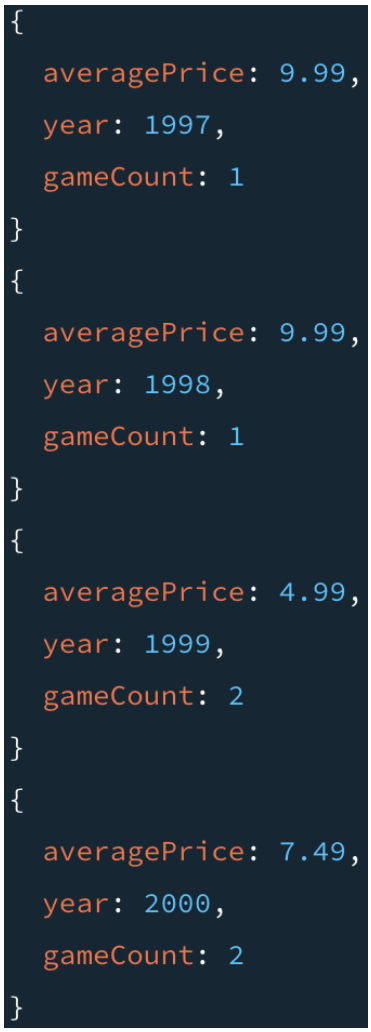

Figure 3.5: MongoDB Query 5 Results (Partial)

3.1.6. Query #6

For each publisher compute the average price of the non-free games it has released:

```
db.steam_collection.aggregate([
    {"$match": {"price": { "$ne": 0 }}},
    {"$unwind": "$publishers"},
    {"$group": {"_id": "$publishers", "gameCount": { "$sum": 1 }, "avgPrice": {
        "$avg": "$price"}}},
    {"$project": {"_id": 0, "publisher": "$_id", "gameCount": 1, "avgPrice": 1}},
    {"$sort": {"gameCount": -1}}
])
```

Figure 3.6: MongoDB Query 6 Results (Partial)

3.1.7. Query #7

Find the name of all the free-to-play games for only 18+ old people that are supported by all the major operating systems (windows, macOS, linux):

```
db.steam_collection.aggregate([
    {"$match": {"genres": {"$in": ["Free to Play"]}}},
    {"$match": {"required_age": {"$gte": 18}}},
    {"$match": {"windows": true}},
    {"$match": {"linux": true}},
    {"$match": {"mac": true}},
    {"$project": {"_id": 0, "name": 1}}
])
```

Figure 3.7: MongoDB Query 7 Results

3.1.8. Query #8

Find the name of all the "Strategy" games that support the Italian language (both text and full audio support):

```
db.steam_collection.aggregate([
    {"$match": {"supported_languages": {"$in": ["Italian"]}}},
    {"$match": {"full_audio_languages": {"$in": ["Italian"]}}},
    {"$match": {"genres": {"$in": ["Strategy"]}}},
    {"$project": {"_id": 0, "name": 1}}
])
```

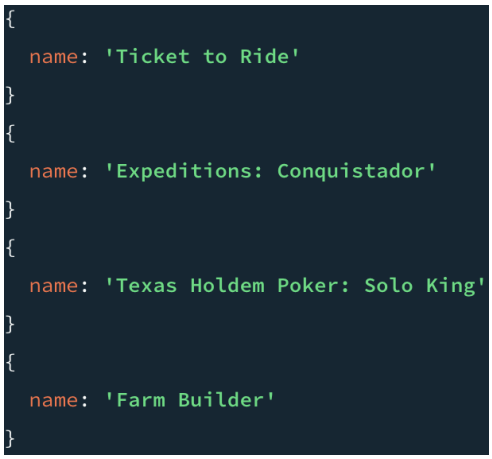

Figure 3.8: MongoDB Query 8 Results (Partial)

3.1.9. Query #9

Find the number of games developed by FromSoftware Inc. present on Steam:

```
db.steam_collection.countDocuments(
    {"developers": {"$in": ["FromSoftware Inc."]}}
)
```

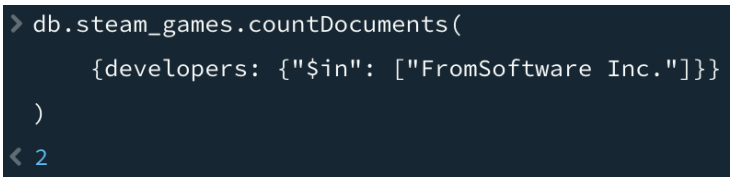

Figure 3.9: MongoDB Query 9 Results

3.1.10. Query #10

Find the name and price of the most expensive game published by Amazon Games:

```
db.steam_collection.aggregate([
    {"$match": {"publishers": {"$in": ["Amazon Games"]}}},
    {"$sort": {"price": -1}}, {"$limit": 1},
    {"$project": {"_id": 0, "name": 1, "price": 1}}
])
```

Figure 3.10: MongoDB Query 10 Results

3.2. Second Dataset (Neo4j)

This section explores Neo4j queries on the Instagram IM dataset. It aims to unravel user connections, identify influential figures, decipher engagement patterns, and profile users based on follower growth. The following analysis allows to make insights into social dynamics, highlighting user behaviors and network influence.

3.2.1. Query #1

Find the top 10 users that are influenced by the smallest possible number of users and with no influence value greater than 0.1:

```
MATCH (other:User)-[r:HAS_INFLUENCE_OVER]->(u:User)
WITH u, other, COLLECT(r) AS relationships
WHERE ALL(rel IN relationships WHERE rel.influence_value <= 0.1)
WITH u, COUNT(other) AS num_incoming_relationships
ORDER BY num_incoming_relationships ASC LIMIT 10
RETURN u
```

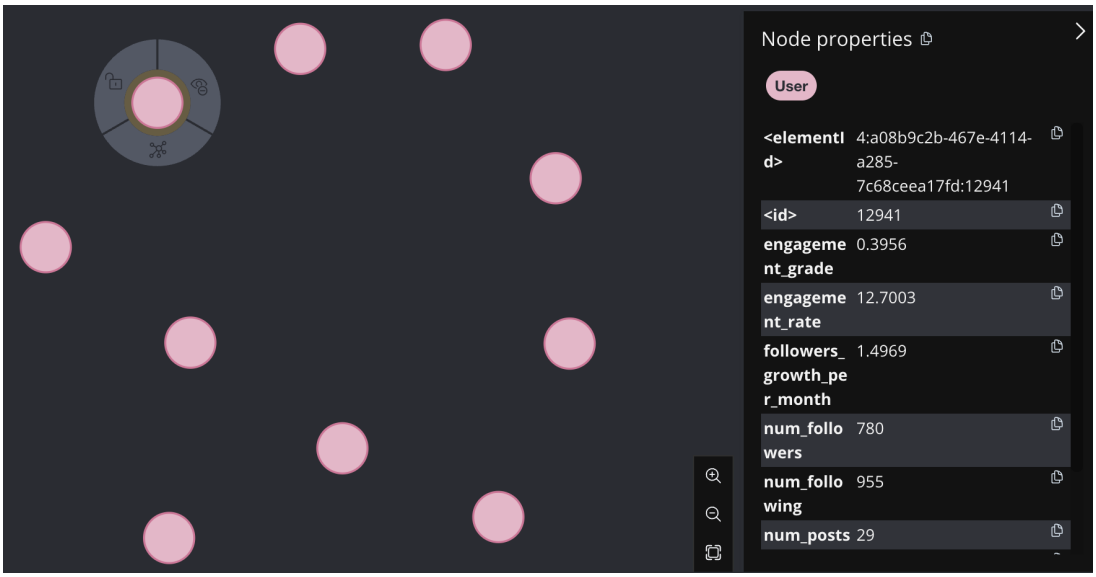

Figure 3.11: Neo4j Query 1 Results (Partial)

3.2.2. Query #2

Find the pairs of users that influence each other:

```
MATCH (u1:User)−[:HAS_INFLUENCE_OVER]−>
    (u2:User)−[:HAS_INFLUENCE_OVER]−>(u1:User)
RETURN u1, u2
```

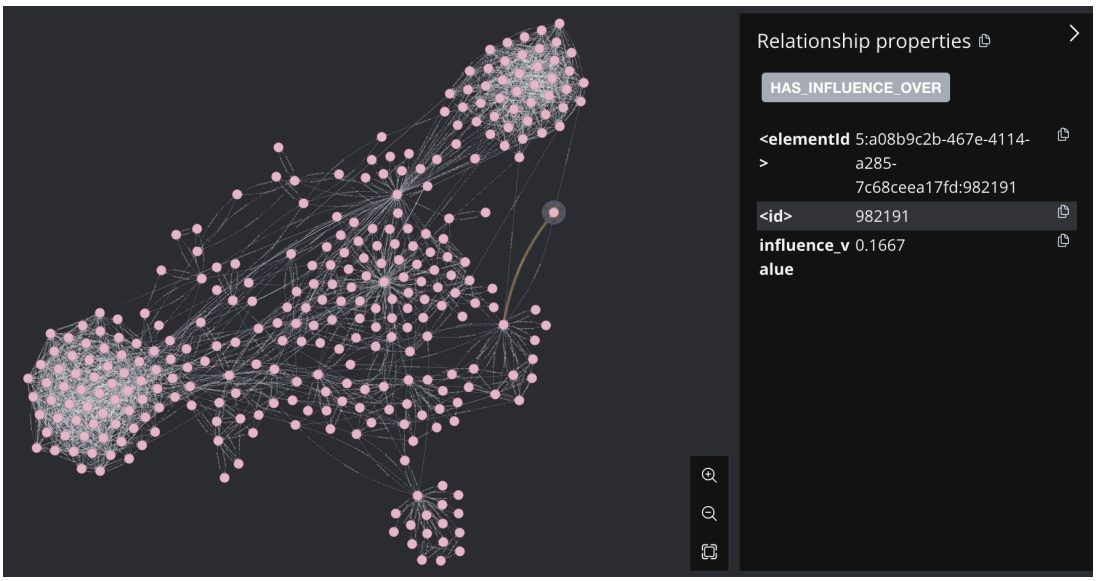

Figure 3.12: Neo4j Query 2 Results (Partial)

3.2.3. Query #3

Compute the average number of posts of the user(s) with the highest engagement rate:

```
MATCH (u1:User) WITH MAX(u1.engagement_rate) AS max_engagement_rate
MATCH (u2:User {engagement_rate: max_engagement_rate})
RETURN AVG(u2.num_posts)
```

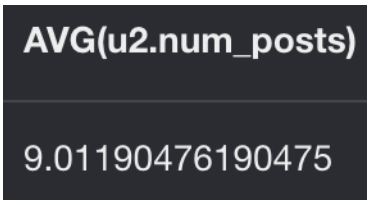

Figure 3.13: Neo4j Query 3 Results

3.2.4. Query #4

Knowing that the engagement rate = (numOfComments + numOfLikes) / numOfFollowers, find the top 5 users with the highest total number of interactions (= likes + comments):

```
MATCH (u:User)
WITH u, u.engagement_rate * u.num_followers AS total_interactions
ORDER BY total_interactions DESC
RETURN u.user_id, total_interactions
LIMIT 5
```

	u.user_id	total_interactions
1	41643	3552536.0
2	29422	3275510.877
3	65034	3178568.6
4	69959	2881863.6
5	50222	2670743.523

Figure 3.14: Neo4j Query 4 Results

3.2.5. Query #5

Compute the average number of posts and the average total interactions for the users whose followers growth per month is > 0.5:

```
MATCH (u:User)
WHERE u.followers_growth_per_month > 0.5
WITH u, u.engagement_rate * u.num_followers AS total_interactions
RETURN AVG(u.num_posts) AS avg_posts, AVG(total_interactions) AS
    avg_total_intearctions
```

avg_posts	avg_total_intearctions
219.98142704783865	10108.138889314798

Figure 3.15: Neo4j Query 5 Results

3.2.6. Query #6

Compute, for the users who are considered "influencers", the number of followers who are also considered "influencers":

```
MATCH (u1:User)-[r:HAS_INFLUENCE_OVER]->(u2:User)
WHERE u1.num_followers > 50000 AND u2.num_followers > 50000
WITH u1, COUNT(u2) AS influencedUsersCount
RETURN u1.user_id, influencedUsersCount
ORDER BY influencedUsersCount DESC LIMIT 10
```

Note that for this query, a user is considered an "influencer" if it has at least 50000 followers.

u1.user_id	influencedUsersCount
29422	15
56340	14
41643	14
50222	12
15012	10
19188	10
24046	9
30650	9
32135	8
48558	8

Figure 3.16: Neo4j Query 6 Results

3.2.7. Query #7

Find the id, the number of followers, the outsider percentage and the number of posts that were made by the user with the overall highest influence value:

```
MATCH (u1:User)−[r:HAS_INFLUENCE_OVER]−>(u2:User)
WITH u1, SUM(r.influence_value) AS totalInfluence
ORDER BY totalInfluence DESC
LIMIT 1
RETURN u1.user_id, u1.num_followers, u1.outsiders_percentage, u1.num_posts,
    totalInfluence
```

u1.user_id	u1.num_followers	u1.outsiders_percentage	u1.num_posts	totalInfluence
41643	1300000	100.0	284	752.6921000000115

Figure 3.17: Neo4j Query 7 Results

3.2.8. Query #8

Find the id, the number of followers, the outsider percentage and the number of posts that were made by the user who influences the biggest number of users:

```
MATCH (u1:User)−[:HAS_INFLUENCE_OVER]−>(u2:User)
WITH u1, COUNT(u2) AS num_relationships
ORDER BY num_relationships DESC
LIMIT 1
RETURN u1.user_id, u1.num_followers, u1.outsiders_percentage, u1.num_posts,
    num_relationships
```

u1.user_id	u1.num_followers	u1.outsiders_percentage	u1.num_posts	num_relationships
41643	1300000	100.0	284	7620

Figure 3.18: Neo4j Query 8 Results

3.2.9. Query #9

Find the total influence of the users that are losing most followers per month, ordered by total influence:

```
MATCH (u1:User)
WITH MIN(u1.followers_growth_per_month) AS min_growth
MATCH (u2:User {followers_growth_per_month:
    min_growth})-[r:HAS_INFLUENCE_OVER]->(u3:User)
WITH u2, SUM(r.influence_value) AS totalInfluence ORDER BY totalInfluence
RETURN u2.user_id, totalInfluence
```

	u2.user_id	totalInfluence
1	48044	0.0005
2	15364	0.0011
3	5696	0.0012
4	56429	0.0013
5	13653	0.0017
6	34369	0.0017

Figure 3.19: Neo4j Query 9 Results (Partial)

3.2.10. Query #10

Find the number of users that have more followers than followees and vice versa:

```
MATCH (u:User)
WITH u,
CASE
WHEN u.num_followers > u.num_following THEN 'more_followers'
WHEN u.num_followers < u.num_following THEN 'more_following'
ELSE 'equal_followers_following'
END AS relationshipType
RETURN relationshipType, COUNT(u) AS numberOfUsers
```

	relationshipType	numberOfUsers
1	"more_followers"	21995
2	"more_following"	47825
3	"equal_followers_following"	589

Figure 3.20: Neo4j Query 10 Results (Partial)

4 | Extra

To perform the extra analyses, we have used two Python notebooks on Google Colab, on which we have mounted a Google Drive folder where we uploaded the datasets. The notebooks have been provided within the assignment folder.

4.1. First Dataset (MongoDB)

As extra analyses for the first dataset, we have plotted some interesting statistics about the dataset and then presented the data profiling results, highlighting some data quality issues.

4.1.1. Interesting Plots

The following plots highlight the value distribution differences between the 10 most frequent values and all the others, for some significant categoric features. Notice that, most notably, we will not show any developers' or publishers' plot, as they are many and there is no significant frequency to underline.

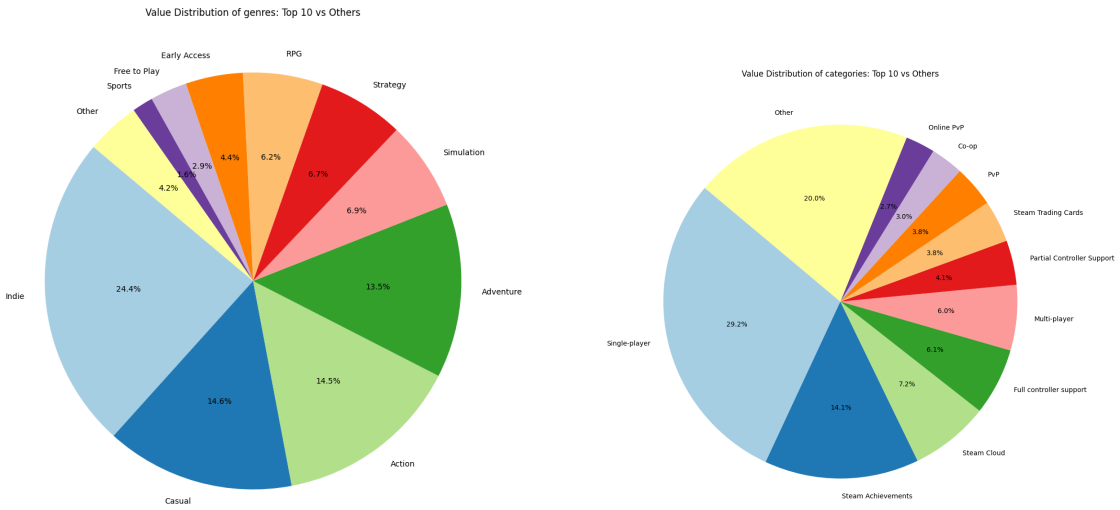

Figure 4.1: Value Distribution Plots: Genres and Categories

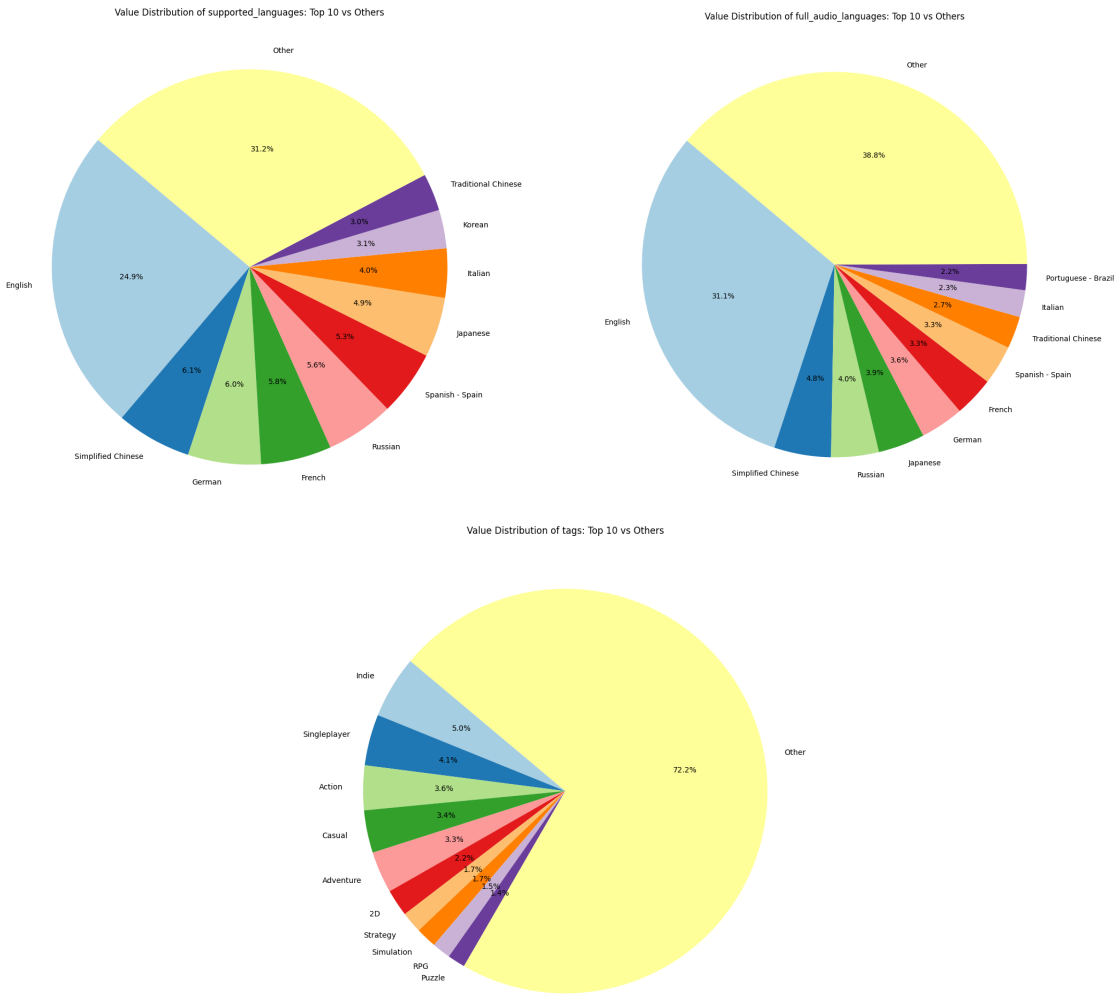

Figure 4.2: Value Distribution Plots: Supported Languages, Full Audio Languages, and Tags

4.1.2. Data Profiling Results

The data profiling tool we have used is provided by the `ydata_profiling` library.

First Data Profiling Report

After converting the original JSON dataset into the DataFrame format, the first step that we did was dropping many useless columns. Such columns are descriptive and thus do not hold any potential correlation between them and any other feature. Namely, they are `detailed_description`, `about_the_game`, `short_description`, `header_image`, `website`, `support_url`, `support_email`, `movies`, `screenshots`, `metacritic_url`, `notes`, and `packages`. We also removed `median_playtime_forever` and `median_playtime_2weeks`

as they are redundant. Due to the tabular nature of the DataFrame format, we also had to remove the following arrays: `supported_languages`, `full_audio_languages`, `developers`, `publishers`, `tags`, `categories`, and `genres`.

Before performing the first profiling of the dataset, we have modified the `estimated_owners` column to make it more meaningful for the following analyses. In the original dataset, it is a string that indicates a wide range of estimated owners. In the DataFrame, it has been converted into an integer that is the mean between the two ends of the aforementioned range.

The following image is an overview of the first profiling report, showing some interesting statistics about the dataset.

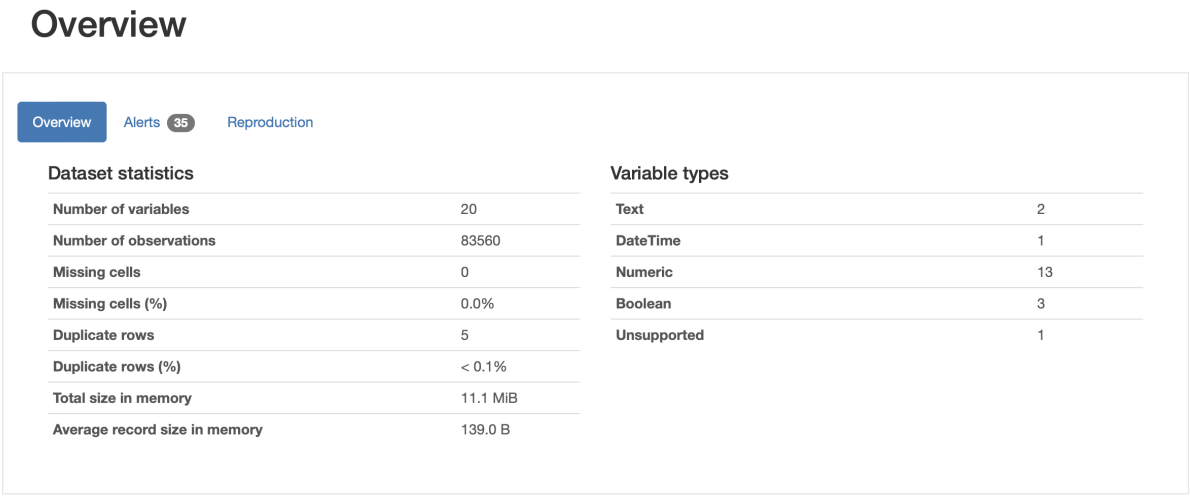

Figure 4.3: Overview of the first profiling report

The first profiling report, however, highlights many potential problems. Let's dive into them:

- The dataset has 5 duplicates.
- The `windows` feature is highly imbalanced (99.5%). However, this is not an issue of the dataset, as in reality, it is true that the majority of games are available for Windows.
- Many features have been marked as skewed. The image below provides more details about the skewness analysis.

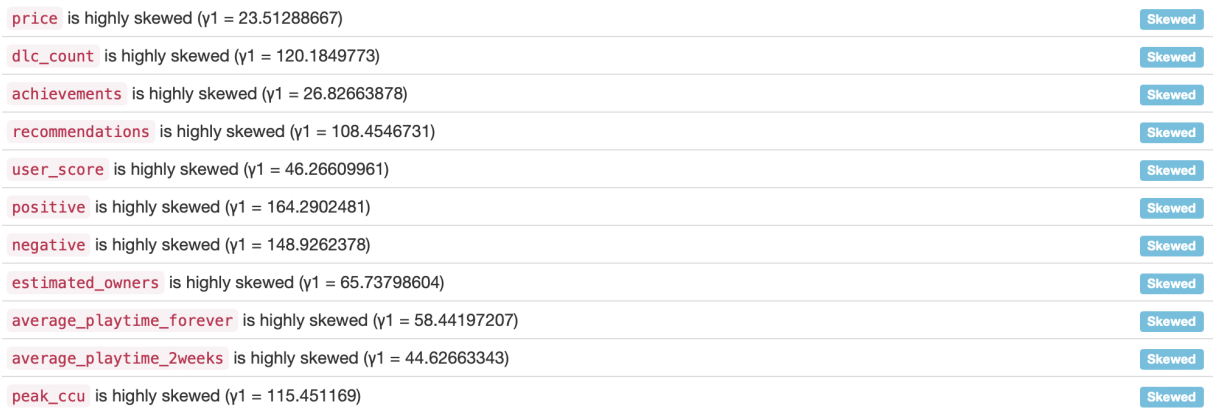

Figure 4.4: Features marked as skewed

The skewness of the distribution of these features is due to the fact that they have a high number of zeros, as we will show in a moment.

- Many features have a lot of zeros. The image below provides more details about the features with several zero values.

Figure 4.5: Features with many zeros

Notice that, even if it makes sense that some of these features have many zeros, i.e. `required_age`, `price`, `dlc_count`, `achievements`, `recommendations`, `positive`, `negative`, and `estimated_owners`, some of them, namely `average_playtime_forever`, `average_playtime_2weeks`, `user_score`, and `peak_ccu`, have too many zeros, and most probably some of those zero values mean "missing value" rather than actually zero. We will address this issue in the following analysis.

- The `score_rank` feature is marked as unsupported, thus it needs some preprocessing in order to be analyzed.

Data Preparation

Based on the profiling report we have just presented, we have prepared the dataset by doing the following:

- The exact duplicates have been dropped.
- The `user_score` column has been dropped, since it is uninformative (it has 99.9% zeros).
- The `reviews` and `score_rank` columns have been refactored in such a way that, when there's an empty string, there's `NumPy`'s `NaN` instead. This has been done to highlight that an empty review is a missing review, and that an empty score rank is a missing score rank.
- The `metacritic_score` column has been refactored in such a way that, when there's 0, there's `NumPy`'s `NaN` instead. This has been done to highlight that a zero-score review is a missing review (with some domain knowledge, we know that Metacritic reviews are made by professionals who do not assign 0 out of 100 to a game they have played).
- We assume that, if the estimated owners are more than 0, the related game has been played at least once. This is not a risky assumption, as the non-zero smallest value we have in the dataset is 10000, and we can safely assume that, out of those 10000 owners, at least one person played the game. Hence, if for a given game the `estimated_owners` feature is greater than 0 but `average_playtime_forever`, `average_playtime_2weeks`, or `peak_ccu` are equal to 0, we set such zero values to `NumPy`'s `NaN`, because we assume to be impossible that if some people own the game, no related usage data have been collected.

By running a new profile report, we have seen that the `score_rank` feature has 99.9% missing values, thus we have dropped it and we have produced the final data profiling report, which we will discuss in a moment.

Final Data Profiling Report

We will now discuss the results of the data profiling task, by presenting the most notable insights from the final report. The following image is an overview of the final profiling

report, showing some interesting statistics about the dataset.

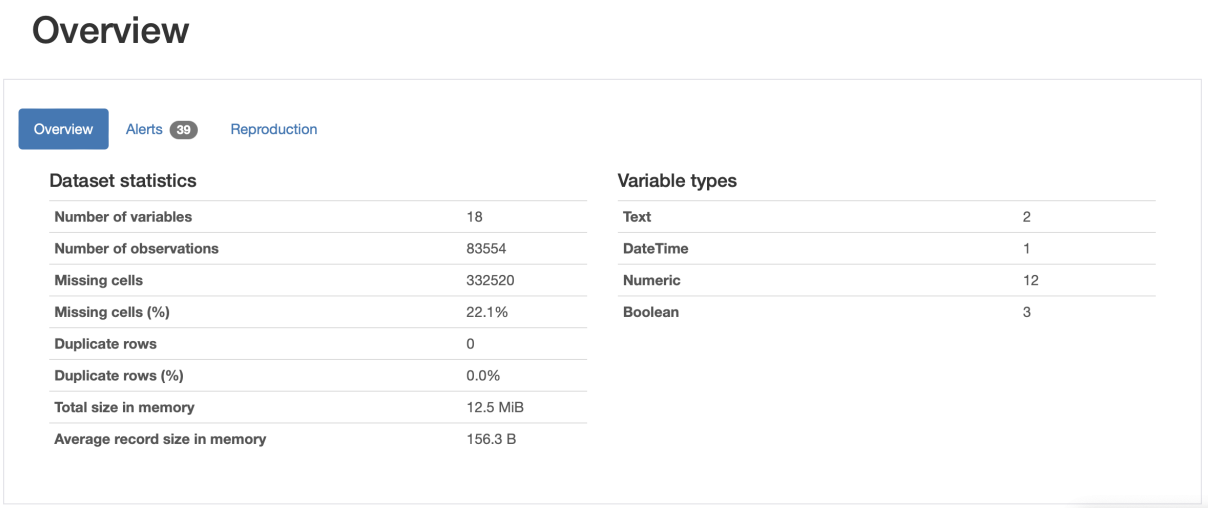

Figure 4.6: Overview of the last profiling report

The following image shows which features of the dataset have the most missing values.

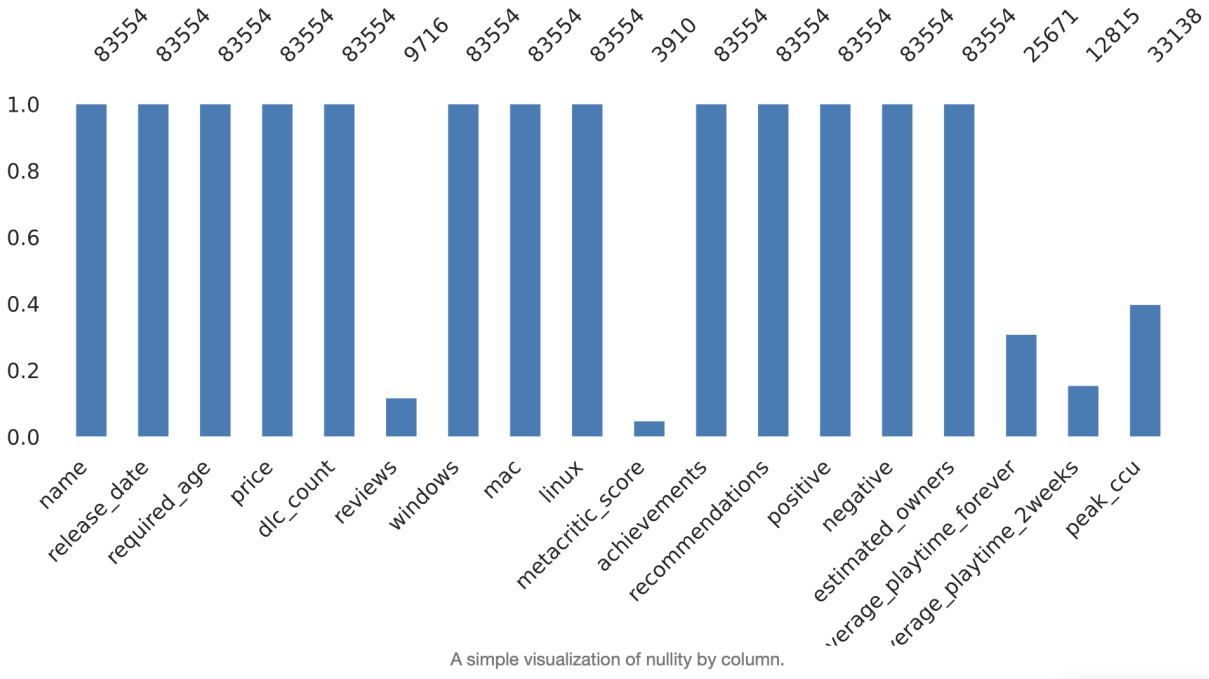

Figure 4.7: Completeness per feature

The following image highlights the correlation between the features of the dataset, by means of a heatmap.

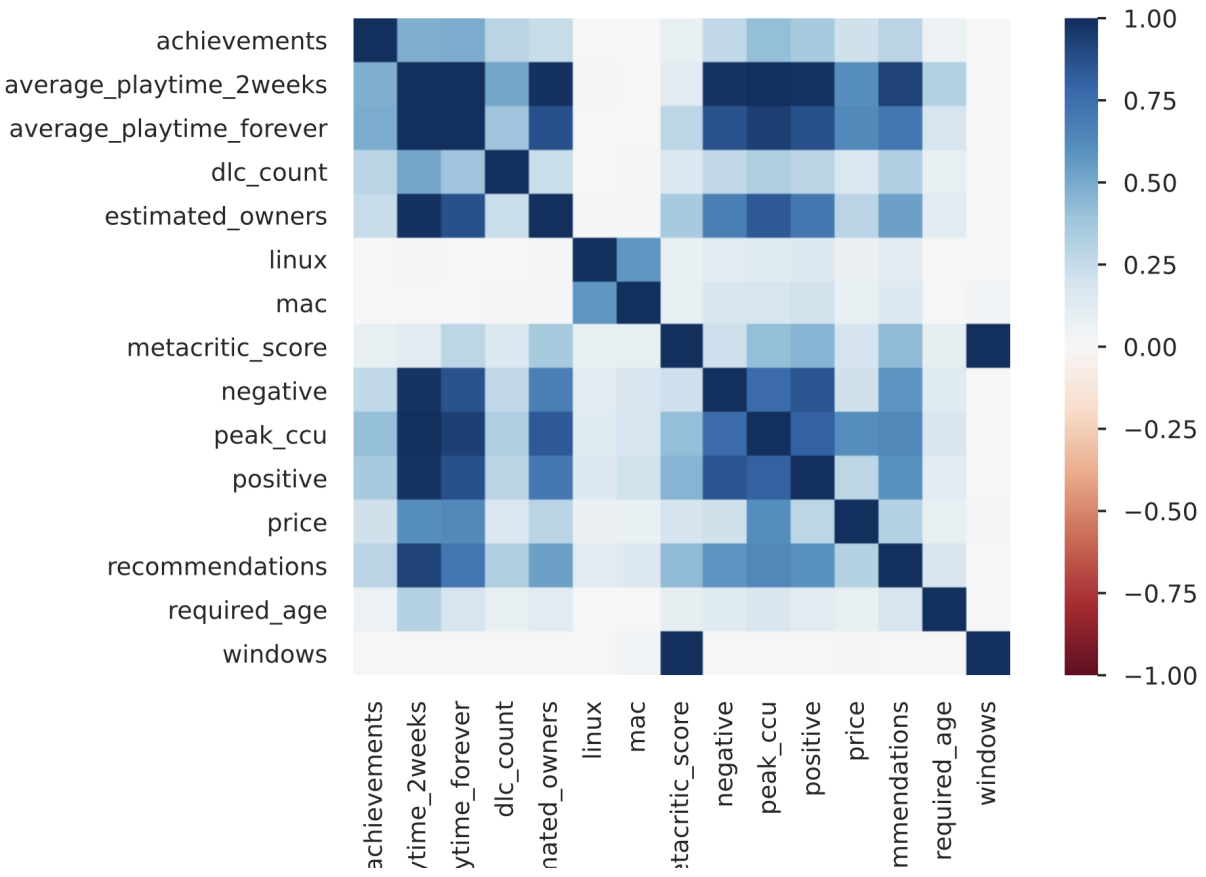

Figure 4.8: Correlation between features

From the heatmap, some conclusions can be drawn, many of which are obvious. However, there are some which are worth being pointed out:

1. Games that are bought/downloaded more tend to be the most enduring. In particular, they manage to create communities that keep on playing. This can be seen from the high, positive correlation between `estimated_owners`, `average_playtime_forever`, and `average_playtime_2weeks` (especially from the correlation between the first and the last among the mentioned features).

2. The more the required age to play a game is high, the higher the number of estimated owners tends to be. This can be seen from the positive correlation between `required_age` and `estimated_owners`, which may suggest that people tend to buy/download games that have a more adult content.

3. There's a strong, positive correlation between `peak_ccu` and both `positive` and `negative`. This could mean that games that are capable of drawing the attention (both with positive and negative reviews) are likely to create notable peaks of people

playing at the same time (perhaps because they become viral in some way). Indeed, notice that `peak_ccu` is also positively correlated with `average_playtime_2weeks`, `average_playtime_forever`, and `recommendations`.

4. The fact that `positive` and `negative` are positively, strongly correlated means that, generally, people do not leave positive reviews to games unanimously; instead, their opinions tend to be mixed.

4.2. Second Dataset (Neo4j)

As part of our supplementary analyses for the second dataset, we opted to delve into the user statistics, performing data profiling. Additionally, we undertook a clustering analysis to discern the presence of distinct clusters among users.

4.2.1. Data Profiling Results

The data profiling tool we have used is provided by the `ydata_profiling` library.

User Statistics Data Profiling Report

The following images are an overview of the profiling report, showing some interesting statistics about the dataset.

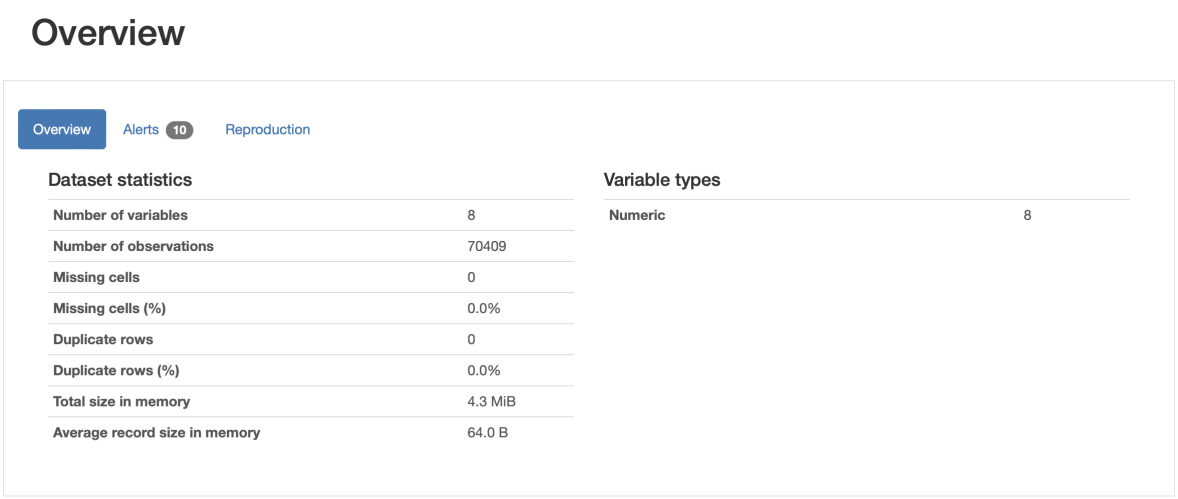

Figure 4.9: Overview of the profiling report

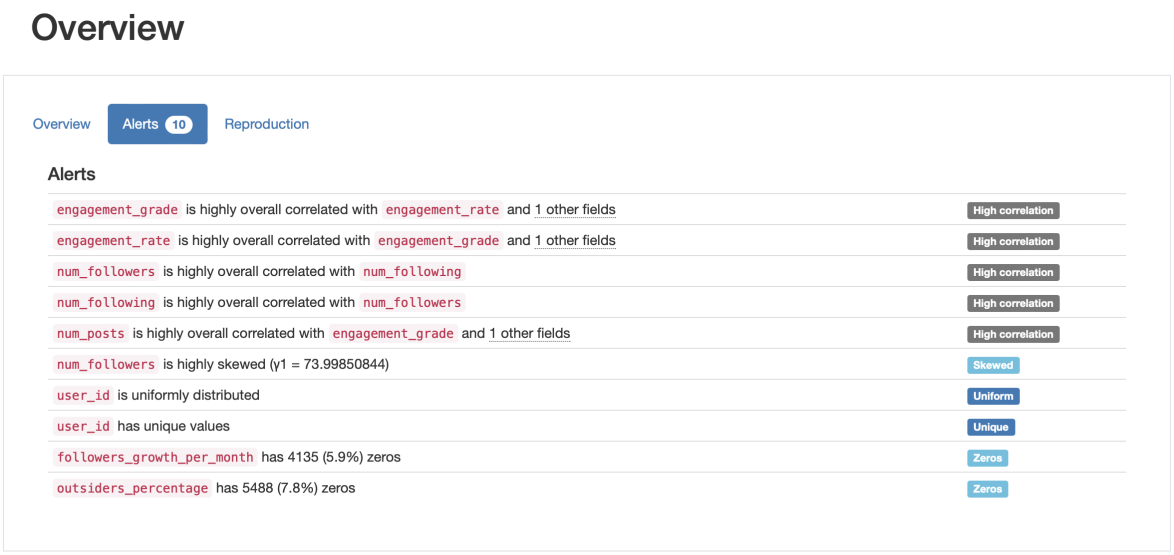

Figure 4.10: Overview of the alerts

As evident from the overview, the dataset demonstrates a high level of cleanliness, marked by the absence of duplicates and missing values. Notably, certain features exhibit strong correlations. For instance, `engagement_rate` and `engagement_grade` are inherently correlated as they encapsulate the same engagement concept, albeit measured through different metrics. Additionally, `num_followers` and `num_following` display a discernible correlation, albeit less immediate. Furthermore, `num_posts` exhibits significant correlation with `engagement_grade` and `engagement_rate`, but we will further explore this after visualizing the heatmap of the features.

The distribution of the `num_followers` feature reveals an extreme right skewness, indicating a pronounced concentration of accounts with significantly larger follower counts. This skewness aligns with common patterns observed in social network data, where a small subset of accounts, often associated with celebrities or influencers, amass a disproportionately high number of followers. The power-law nature of social network dynamics contributes to this skewed pattern, underscoring the presence of notable outliers in the dataset. `User_id` has unique values and is uniformely distributed, which makes sense since every user is unique and the ids are incremental.

Both `followers_growth_per_month` and `outsiders_percentage` exhibit zero values, indicating distinct user behaviors. A zero `followers_growth_per_month` suggests the presence of users whose accounts are not growing. Similarly, a zero `outsiders_percentage` implies that certain users lack engagement from individuals outside their immediate followers, aligning with the definition of `outsiders_percentage` as the percentage of en-

gagement from external users.

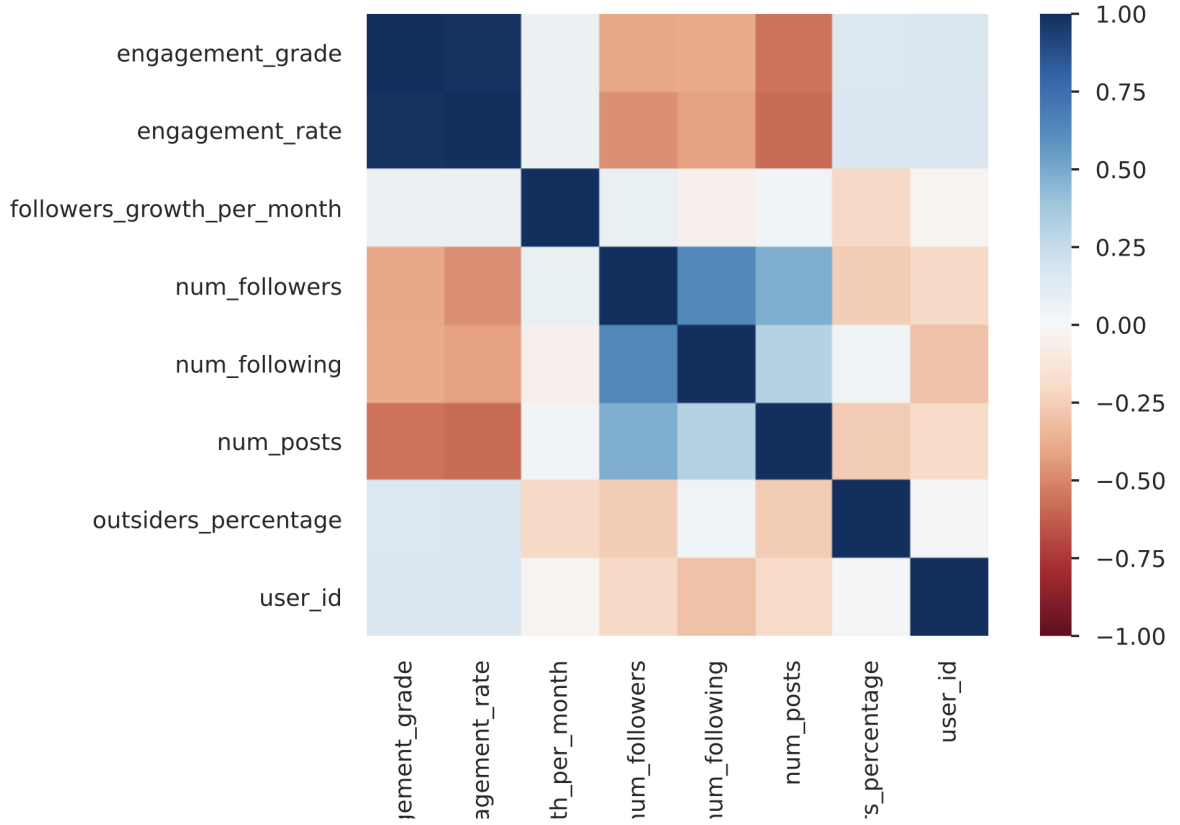

Figure 4.11: Correlation between features: Heatmap

Heatmap | Table

	engagement_grade	engagement_rate	followers_growth_per_month	num_followers	num_following
engagement_grade	1.000	0.982	0.069	-0.396	-0.388
engagement_rate	0.982	1.000	0.065	-0.471	-0.417
followers_growth_per_month	0.069	0.065	1.000	0.077	-0.049
num_followers	-0.396	-0.471	0.077	1.000	0.636
num_following	-0.388	-0.417	-0.049	0.636	1.000
num_posts	-0.558	-0.580	0.031	0.477	0.308
outsiders_percentage	0.155	0.162	-0.206	-0.251	0.039
user_id	0.156	0.165	-0.023	-0.211	-0.304

Figure 4.12: Correlation between features: Table (Part 1)

Heatmap | Table

	engagement_grade	engagement_rate	followers_growth_per_month	num_followers	num_following	num_posts	outsiders_percentage	user_id
	1.000	0.982	0.069	-0.396	-0.388	-0.558	0.155	0.156
	0.982	1.000	0.065	-0.471	-0.417	-0.580	0.162	0.165
month	0.069	0.065	1.000	0.077	-0.049	0.031	-0.206	-0.023
	-0.396	-0.471	0.077	1.000	0.636	0.477	-0.251	-0.211
	-0.388	-0.417	-0.049	0.636	1.000	0.308	0.039	-0.304
	-0.558	-0.580	0.031	0.477	0.308	1.000	-0.264	-0.199
	0.155	0.162	-0.206	-0.251	0.039	-0.264	1.000	0.016
	0.156	0.165	-0.023	-0.211	-0.304	-0.199	0.016	1.000

Figure 4.13: Correlation between features: Table (Part 2)

As we can see from the heatmap and the relative table, the correlated couples `num_posts` - `engagement_rate` and `num_posts` - `engagement_grade` are highly negative. It suggests that, on average, users who post more frequently tend to have lower engagement rates. In other words, there might be a tendency for users with a higher number of posts to receive fewer engagements (likes, comments, etc.) per post. This inverse relationship implies a potential trade-off between content quantity and quality, where higher post volume may lead to audience fatigue or reduced responsiveness, impacting overall engagement metrics adversely.

The analysis also reveals a negative correlation between `num_followers` and both `engagement_rate` and `engagement_grade`, indicating that users with larger follower counts experience lower engagement metrics. This phenomenon is attributed to audience dilution, where a broad and heterogeneous follower base diminishes the likelihood of high engagement per post. It could also be caused by content saturation, stemming from excessive post frequency (it's worth to notice that in fact `num_followers` and `num_posts` are correlated), and further contributing to reduced audience responsiveness.

4.2.2. Clustering Results

The clustering algorithm we decided to use is KMeans. The implementation we have used is provided by the `sklearn.cluster` library.

Before starting our analysis, we converted the "Network for IC LT.txt" file into a csv and then created a DataFrame to better manage the data. After scaling the dataset we had to choose the right number of clusters (K). To do so, we used the Elbow Method to find an optimal value for K.

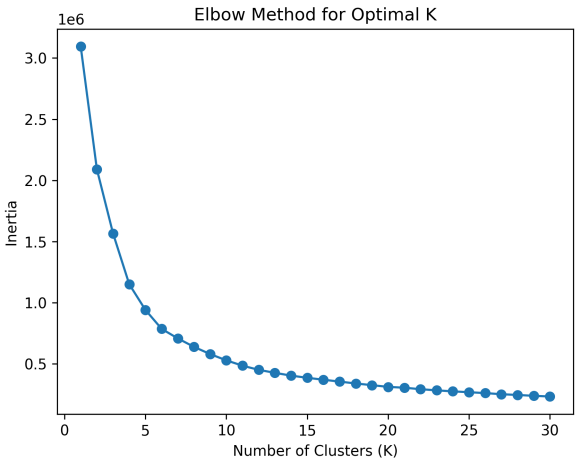

Figure 4.14: Elbow Method Plot

As evident from the plot, the elbow point occurs between K values of 5 and 10. To determine the optimal K, we conducted multiple iterations with varying K values and compared the results. For simplicity, we present and utilize the best-performing K, which is 9. Next, we employed KMeans clustering with the chosen K and incorporated the predicted clusters into our DataFrame. Visualization of the clusters involved creating scatter plots for different feature pairs, with data points color-coded based on their assigned cluster. Additionally, we marked cluster centroids with red 'X' markers to provide a clear representation of central tendencies within each cluster.

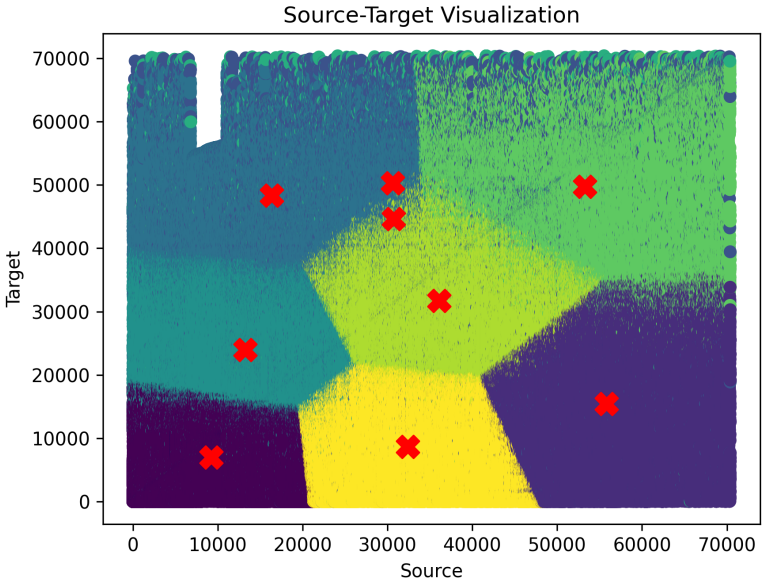

Figure 4.15: Cluster Visualization: Source-Target Plot

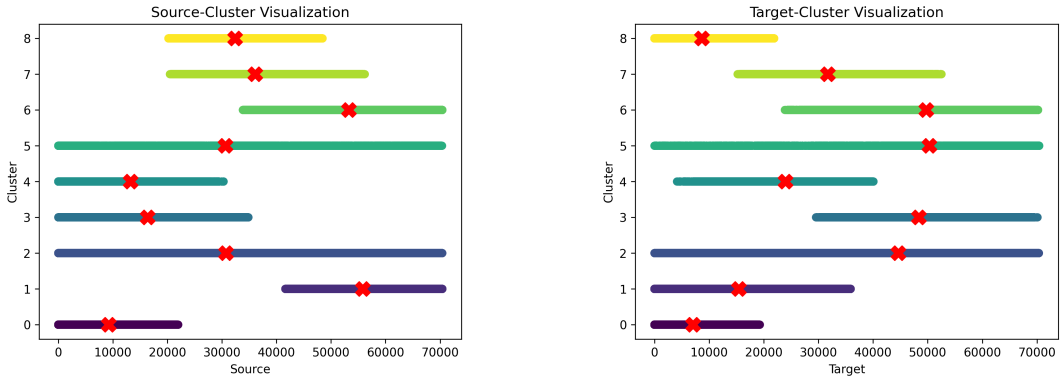

Figure 4.16: Cluster Visualization: Source-Cluster and Target-Cluster Plot

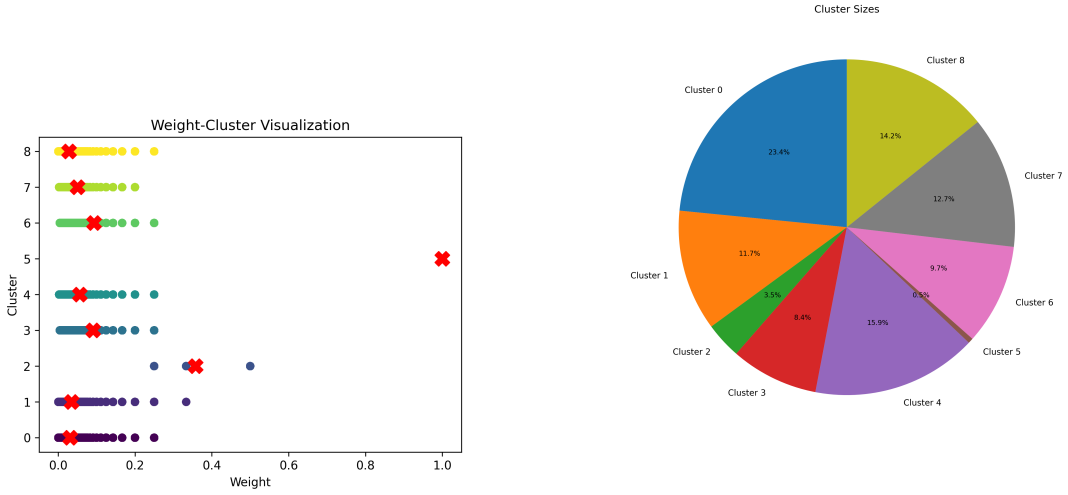

Figure 4.17: Cluster Sizes Plot

Figure 4.18: Cluster Visualization: Weight-Cluster and Cluster Sizes Plot

The visualization of the `Target` and `Source` features reveals distinct boundaries, indicating the presence of well-defined clusters. Several plausible interpretations arise from this observation:

- Community Structure: the discernible boundaries suggest the existence of distinct communities or user groups within the social network. Users sharing the same community tend to exhibit stronger connections compared to those in different communities.
- Homophily: homophily, the inclination of individuals to connect with others who share similar attributes or behaviors, may contribute to the observed boundaries.

Users with common characteristics form tightly connected groups, creating visible distinctions in the plot.

- Influencer Networks: the boundaries hint at the presence of influence networks, where certain users, potentially high-degree nodes or influencers, establish robust connections within their group but display fewer connections outside it.
- Interest-Based Segmentation: clustering of users with similar interests or engagement patterns becomes evident, resulting in well-defined segments within the network. The observed boundaries represent the demarcation between these interest-based segments.
- Activity Patterns: variances in user activity patterns, such as posting frequency and content types, likely contribute to the establishment of clear boundaries. Users with analogous activity patterns tend to cluster together.
- Behavioral Segmentation: users exhibiting diverse behaviors, such as high engagement versus low engagement, tend to form separate clusters. The delineated boundaries represent the transition between these distinct behavioral segments.

With the data available, we cannot explore each interpretation, and thus provide an exhaustive analysis essential for a better understanding of the Instagram dynamics. Hence, the present study will not undertake further investigation into the aforementioned interpretations.

Finally, we computed the silhouette score, to ensure an unbiased assessment of the overall clustering performance. To perform this task we had to sample our dataset since the complexity was of $O(n^2)$ and n was over 1 million. We chose a sample factor of 0.2.

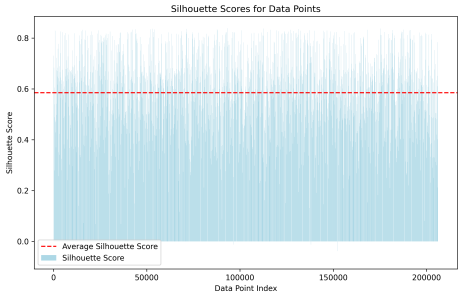

Figure 4.19: Silhouette Score Plot

The achieved average silhouette score of 0.58 underscores the robustness of the clustering solution, especially considering that all sampled data points exhibit positive silhouette

scores. This result indicates a high level of cohesion within clusters, where data points are more similar to members of their own cluster than to those in other clusters. The uniformly positive silhouette scores across the sampled data suggest a well-defined and internally consistent clustering structure. This outcome confirms the appropriateness of the chosen clustering approach and configuration, highlighting the homogeneity within clusters and the distinct separation between them. The comprehensive positive silhouette scores lend confidence to the reliability and validity of the clustering outcomes, even within the context of the sampled 20% of the dataset.

List of Figures

List of Tables