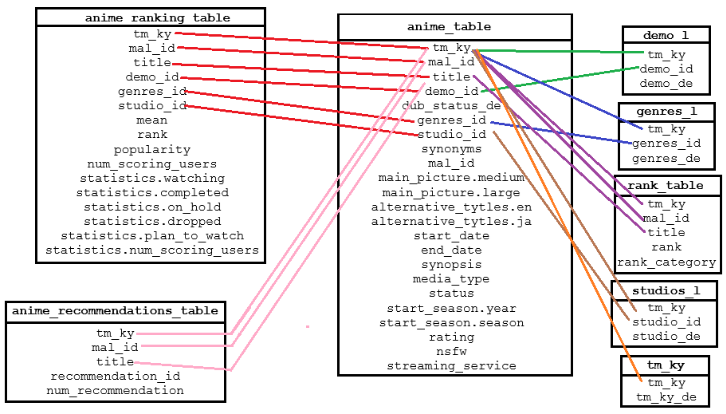
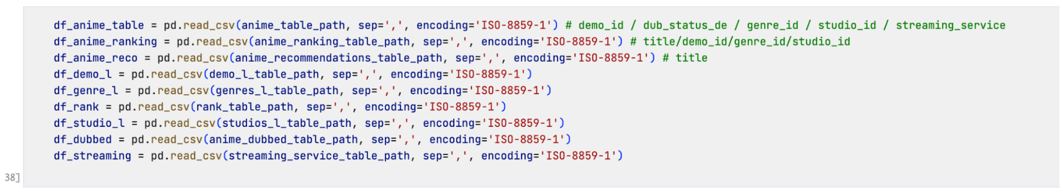
Eléonor KIOULOU - Khadija MOKHTARI - Théophile NELSON - MICHELE MOGAVERO

Report : Application Cloud

Dataset



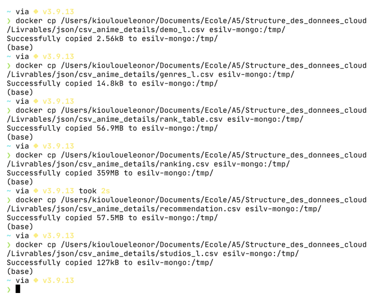


```
df_anime_table = pd.read_csv(anime_table_path, sep=',', encoding='ISO-8859-1') # demo_id / dub_status_de / genre_id / studio_id / streaming_service
df_anime_ranking = pd.read_csv(anime_ranking_table_path, sep=',', encoding='ISO-8859-1') # title/demo_id/genre_id/studio_id
df_anime_reco = pd.read_csv(anime_recommendations_table_path, sep=',', encoding='ISO-8859-1') # title
df_demo_l = pd.read_csv(demo_l_table_path, sep=',', encoding='ISO-8859-1')
df_genre_l = pd.read_csv(genres_l_table_path, sep=',', encoding='ISO-8859-1')
df_rank = pd.read_csv(rank_table_path, sep=',', encoding='ISO-8859-1')
df_studio_l = pd.read_csv(studios_l_table_path, sep=',', encoding='ISO-8859-1')
df_dubbed = pd.read_csv(anime_dubbed_table_path, sep=',', encoding='ISO-8859-1')
df_streaming = pd.read_csv(streaming_service_table_path, sep=',', encoding='ISO-8859-1')
```
38]

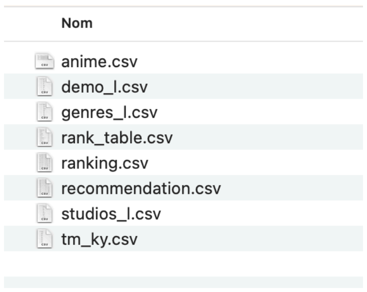
I. Performance Measures of Queries

Data Import and Container Setup

- Data is imported into MongoDB using Docker. CSV files, including 'anime.csv', 'ranking.csv', and 'recommendation.csv', others, are copied to the MongoDB container (esilv-mongo).



```
~ via ✤ v3.9.13
❯ docker cp /Users/kioulouelelonor/Documents/Ecole/A5/Structure_des_donnees_cloud
/Livrables/json/csv_anime_details/demo_l.csv esilv-mongo:/tmp/
Successfully copied 2.56kB to esilv-mongo:/tmp/
(base)
~ via ✤ v3.9.13
❯ docker cp /Users/kioulouelelonor/Documents/Ecole/A5/Structure_des_donnees_cloud
/Livrables/json/csv_anime_details/genres_l.csv esilv-mongo:/tmp/
Successfully copied 14.8kB to esilv-mongo:/tmp/
(base)
~ via ✤ v3.9.13
❯ docker cp /Users/kioulouelelonor/Documents/Ecole/A5/Structure_des_donnees_cloud
/Livrables/json/csv_anime_details/rank_table.csv esilv-mongo:/tmp/
Successfully copied 56.9MB to esilv-mongo:/tmp/
(base)
~ via ✤ v3.9.13
❯ docker cp /Users/kioulouelelonor/Documents/Ecole/A5/Structure_des_donnees_cloud
/Livrables/json/csv_anime_details/ranking.csv esilv-mongo:/tmp/
Successfully copied 359MB to esilv-mongo:/tmp/
(base)
~ via ✤ v3.9.13 took 2s
❯ docker cp /Users/kioulouelelonor/Documents/Ecole/A5/Structure_des_donnees_cloud
/Livrables/json/csv_anime_details/recommendation.csv esilv-mongo:/tmp/
Successfully copied 57.5MB to esilv-mongo:/tmp/
(base)
~ via ✤ v3.9.13
❯ docker cp /Users/kioulouelelonor/Documents/Ecole/A5/Structure_des_donnees_cloud
/Livrables/json/csv_anime_details/studios_l.csv esilv-mongo:/tmp/
Successfully copied 127kB to esilv-mongo:/tmp/
(base)
~ via ✤ v3.9.13
❯ █
```



```
Nom
anime.csv
demo_l.csv
genres_l.csv
rank_table.csv
ranking.csv
recommendation.csv
studios_l.csv
tm_ky.csv
```

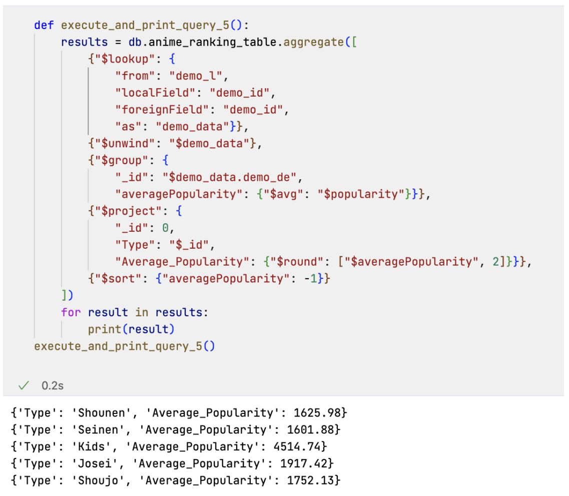
Database and Collections Structure

- The MongoDB database, named '**animeDB**', is used.
- Each CSV file is imported as a separate collection in '**animeDB**', facilitating organized data management and efficient query execution.

- Collections created include 'anime_recommendation_table' and others based on different CSV files.

Queries and Data Analysis

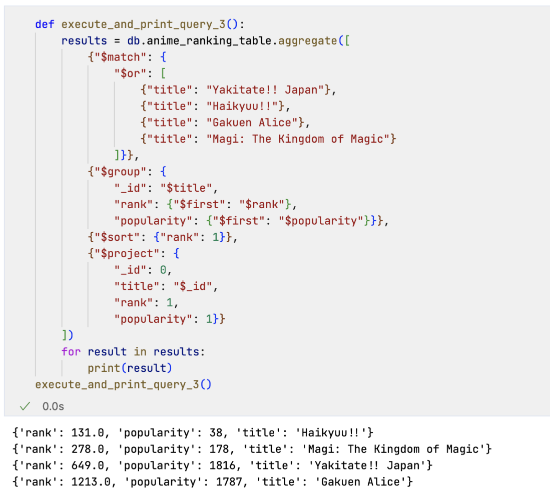
- The database is structured to support various queries related to anime genres, rankings, and studio information.
- Queries are designed to leverage MongoDB's aggregation framework, allowing for complex data manipulation and analysis.
- Example of queries with their results :



```python
def execute_and_print_query_5():
    results = db.anime_ranking_table.aggregate([
        {"$lookup": {
            "from": "demo_l",
            "localField": "demo_id",
            "foreignField": "demo_id",
            "as": "demo_data"}},
        {"$unwind": "$demo_data"},
        {"$group": {
            "_id": "$demo_data.demo_de",
            "averagePopularity": {"$avg": "$popularity"}}},
        {"$project": {
            "_id": 0,
            "Type": "$_id",
            "Average_Popularity": {"$round": ["$averagePopularity", 2]}}},
        {"$sort": {"averagePopularity": -1}}
    ])
    for result in results:
        print(result)
execute_and_print_query_5()
```

✓ 0.2s

```
{'Type': 'Shounen', 'Average_Popularity': 1625.98}
{'Type': 'Seinen', 'Average_Popularity': 1601.88}
{'Type': 'Kids', 'Average_Popularity': 4514.74}
{'Type': 'Josei', 'Average_Popularity': 1917.42}
{'Type': 'Shoujo', 'Average_Popularity': 1752.13}
```



```python
def execute_and_print_query_3():
    results = db.anime_ranking_table.aggregate([
        {"$match": {
            "$or": [
                {"title": "Yakitate!! Japan"},
                {"title": "Haikyuu!!"},
                {"title": "Gakuen Alice"},
                {"title": "Magi: The Kingdom of Magic"}
            ]}},
        {"$group": {
            "_id": "$title",
            "rank": {"$first": "$rank"},
            "popularity": {"$first": "$popularity"}}},
        {"$sort": {"rank": 1}},
        {"$project": {
            "_id": 0,
            "title": "$_id",
            "rank": 1,
            "popularity": 1}}
    ])
    for result in results:
        print(result)
execute_and_print_query_3()
```

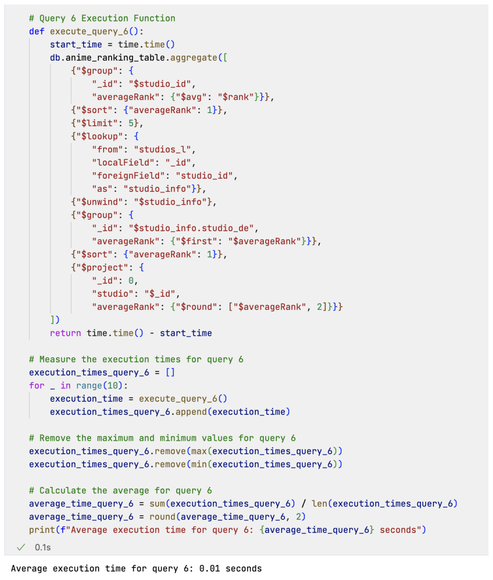
✓ 0.0s

```
{'rank': 131.0, 'popularity': 38, 'title': 'Haikyuu!!'}
{'rank': 278.0, 'popularity': 178, 'title': 'Magi: The Kingdom of Magic'}
{'rank': 649.0, 'popularity': 1816, 'title': 'Yakitate!! Japan'}
{'rank': 1213.0, 'popularity': 1787, 'title': 'Gakuen Alice'}
```

- performances.ipynb file

Performance Measurement

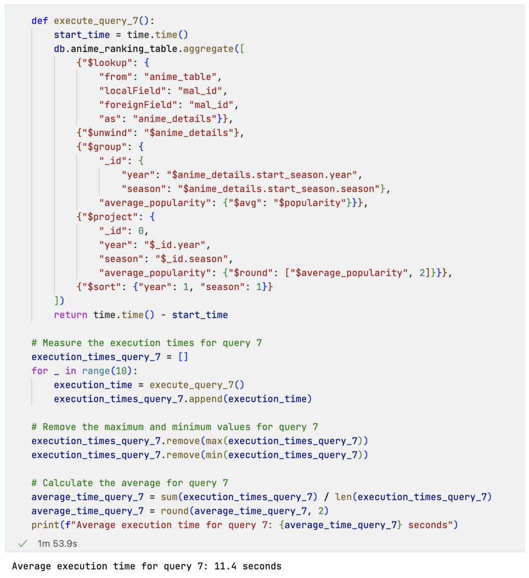
- Each query function is designed to measure the time it takes to execute a MongoDB aggregate query.
- The **execute_query_1()** function, for example, starts by recording the current time, runs the query, and then calculates the difference from the current time to get the execution time.
- This process is repeated 10 times for the query, each time appending the execution time to a list **execution_times_query_1**.
- After all repetitions are done, the maximum and minimum values are removed from this list to exclude outliers.
- The average execution time is then calculated by summing the remaining execution times and dividing by the number of these times, which should be 8 after removing the max and min values.
- The average time is then rounded to two decimal places for readability and printed out.



```python
# Query 6 Execution Function
def execute_query_6():
    start_time = time.time()
    db.anime_ranking_table.aggregate([
        {"$group": {
            "_id": "$studio_id",
            "averageRank": {"$avg": "$rank"}}},
        {"$sort": {"averageRank": 1}},
        {"$limit": 5},
        {"$lookup": {
            "from": "studios_l",
            "localField": "_id",
            "foreignField": "studio_id",
            "as": "studio_info"}},
        {"$unwind": "$studio_info"},
        {"$group": {
            "_id": "$studio_info.studio_de",
            "averageRank": {"$first": "$averageRank"}}},
        {"$sort": {"averageRank": 1}},
        {"$project": {
            "_id": 0,
            "studio": "$_id",
            "averageRank": {"$round": ["$averageRank", 2]}}}
    ])
    return time.time() - start_time

# Measure the execution times for query 6
execution_times_query_6 = []
for _ in range(10):
    execution_time = execute_query_6()
    execution_times_query_6.append(execution_time)

# Remove the maximum and minimum values for query 6
execution_times_query_6.remove(max(execution_times_query_6))
execution_times_query_6.remove(min(execution_times_query_6))

# Calculate the average for query 6
average_time_query_6 = sum(execution_times_query_6) / len(execution_times_query_6)
average_time_query_6 = round(average_time_query_6, 2)
print(f"Average execution time for query 6: {average_time_query_6} seconds")
```

✓ 0.1s

```
Average execution time for query 6: 0.01 seconds
```



```python
def execute_query_7():
    start_time = time.time()
    db.anime_ranking_table.aggregate([
        {"$lookup": {
            "from": "anime_table",
            "localField": "mal_id",
            "foreignField": "mal_id",
            "as": "anime_details"}},
        {"$unwind": "$anime_details"},
        {"$group": {
            "_id": {
                "year": "$anime_details.start_season.year",
                "season": "$anime_details.start_season.season"},
            "average_popularity": {"$avg": "$popularity"}}},
        {"$project": {
            "_id": 0,
            "year": "$_id.year",
            "season": "$_id.season",
            "average_popularity": {"$round": ["$average_popularity", 2]}}},
        {"$sort": {"year": 1, "season": 1}}
    ])
    return time.time() - start_time

# Measure the execution times for query 7
execution_times_query_7 = []
for _ in range(10):
    execution_time = execute_query_7()
    execution_times_query_7.append(execution_time)

# Remove the maximum and minimum values for query 7
execution_times_query_7.remove(max(execution_times_query_7))
execution_times_query_7.remove(min(execution_times_query_7))

# Calculate the average for query 7
average_time_query_7 = sum(execution_times_query_7) / len(execution_times_query_7)
average_time_query_7 = round(average_time_query_7, 2)
print(f"Average execution time for query 7: {average_time_query_7} seconds")
```

✓ 1m 53.9s

```
Average execution time for query 7: 11.4 seconds
```

- performances.ipynb file

II. Description of Architecture

Our application use Streamlit

Application Components

- **app_user.py:**
  - The main functionalities could include querying the MongoDB database for specific data, displaying results to the user
- **app_admin.py:**
  - There are features for performance monitoring, such as tracking query execution times, database load or statistics.
- **app_analyst.py:**
  - This script offers an interactive interface for data analysts and business users to execute complex MongoDB queries. It includes customizable query parameters for manual input or dropdown selections.
  - Combines end-user queries and data analyst queries. End-user queries can be modified and come with options. Data analyst queries display certain statistics of the dataset

MongoDB Database Configuration

- The MongoDB setup is configured for robust data handling and querying. The scripts include connection configurations to the MongoDB instance.
- The database is structured with multiple collections, each optimized for specific types of queries and data storage.

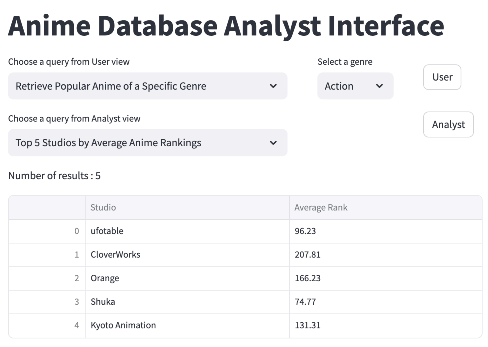
Integration and Workflow

- The applications (app_user.py and app_admin.py) interface with the MongoDB database, facilitating data retrieval, manipulation, and administration.
- There might be a flow of data from MongoDB to these applications, enabling real-time data processing and presentation.

III. Code Organization

- **Import Statements:** The script begins with importing necessary libraries such as **streamlit** for the web application interface, **pymongo** for MongoDB interaction, and **pandas** for data manipulation.
- **MongoDB Connection Configuration:** It establishes a connection to the MongoDB server and selects the animeDB database, setting up the foundation for database operations.
- **Utility Function:** A formatting function **format_without_commas** is defined to format numerical data, improving data readability in output.
- **Database Query Functions:** Several functions (**execute_query_1** to **execute_query_8**) are defined to execute specific queries on the MongoDB database. These functions are tailored to perform various analytical tasks like fetching popular anime by genre, retrieving data based on studio, and analyzing anime popularity over time, among others.
- **Streamlit Interface Setup:** The script uses Streamlit to create a user-friendly web interface. It includes layout definitions and input components (like select boxes and buttons) to interact with users.
- **Dynamic Query Execution:** Based on user selections, appropriate query functions are called to fetch and display results. This dynamic approach allows for a flexible and interactive user experience.
- **Results Presentation:** The script formats and presents the query results in a tabular format using Streamlit, making the data easily understandable and accessible to the end-users.

This structure demonstrates a well-organized and functional approach to building an analytical application with interactive features and efficient data processing capabilities.



Anime Database Analyst Interface

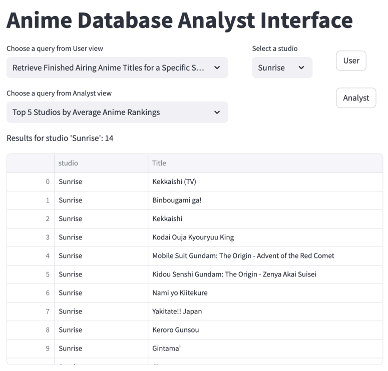
Choose a query from User view: Retrieve Popular Anime of a Specific Genre

Select a genre: Action

Choose a query from Analyst view: Top 5 Studios by Average Anime Rankings

Number of results : 5

	Studio	Average Rank
0	ufotable	96.23
1	CloverWorks	207.81
2	Orange	166.23
3	Shuka	74.77
4	Kyoto Animation	131.31



Anime Database Analyst Interface

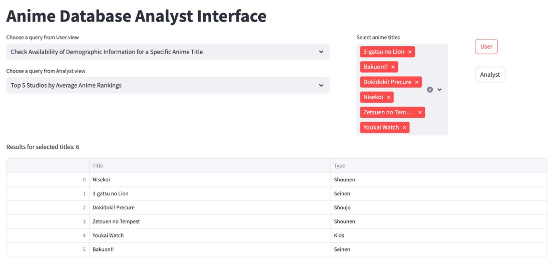
Choose a query from User view: Retrieve Finished Airing Anime Titles for a Specific S…

Select a studio: Sunrise

Choose a query from Analyst view: Top 5 Studios by Average Anime Rankings

Results for studio 'Sunrise': 14

	studio	Title
0	Sunrise	Kekkaishi (TV)
1	Sunrise	Binbougami ga!
2	Sunrise	Kekkaishi
3	Sunrise	Kodai Ouja Kyouryuu King
4	Sunrise	Mobile Suit Gundam: The Origin - Advent of the Red Comet
5	Sunrise	Kidou Senshi Gundam: The Origin - Zenya Akai Suisei
6	Sunrise	Nami yo Kiitekure
7	Sunrise	Yakitate!! Japan
8	Sunrise	Keroro Gunsou
9	Sunrise	Gintama'



Anime Database Analyst Interface

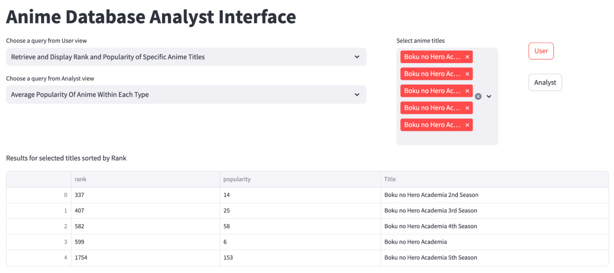
Choose a query from User view: Check Availability of Demographic Information for a Specific Anime Title

Choose a query from Analyst view: Top 5 Studios by Average Anime Rankings

Select anime titles: 3-gatsu no Lion, Bakuon!!, Dokidoki! Precure, Nisekoi, Zetsuen no Tem…, Youkai Watch

Results for selected titles: 6

	Title	Type
0	Nisekoi	Shounen
1	3-gatsu no Lion	Seinen
2	Dokidoki! Precure	Shoujo
3	Zetsuen no Tempest	Shounen
4	Youkai Watch	Kids
5	Bakuon!!	Seinen



Anime Database Analyst Interface

Choose a query from User view: Retrieve and Display Rank and Popularity of Specific Anime Titles

Choose a query from Analyst view: Average Popularity Of Anime Within Each Type

Select anime titles: Boku no Hero Ac…, Boku no Hero Ac…, Boku no Hero Ac…, Boku no Hero Ac…, Boku no Hero Ac…

Results for selected titles sorted by Rank

	rank	popularity	Title
0	337	14	Boku no Hero Academia 2nd Season
1	407	25	Boku no Hero Academia 3rd Season
2	582	58	Boku no Hero Academia 4th Season
3	599	6	Boku no Hero Academia
4	1754	153	Boku no Hero Academia 5th Season

IV. Code Source

- performances.ipynb file : performances.ipynb
- Other files in zip file