DS4300 HW 3

February 18, 2025

1 Homework Assignment 03

1.1 DS 4300 - Spring 2025

- **EC Due Date**: Feb 16, 2025 @ 11:59pm
- **Regular Due Date**: Feb 18, 2025 @ 11:59pm
- Upload to GradeScope (no question/solutions to Match)

[1]:
```python
# Set up your connection to Mongo DB here.
!pip install pymongo
import pymongo
from bson.json_util import dumps

# I'm not sure why I can't use a username & password but something goes wrong␣
↪with the connection everytime I try to incorporate them
uri = "mongodb://localhost:27017/"
client = pymongo.MongoClient(uri)
```

```
Requirement already satisfied: pymongo in ./opt/anaconda3/lib/python3.8/site-
packages (4.10.1)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in
./opt/anaconda3/lib/python3.8/site-packages (from pymongo) (2.6.1)
```

1.2 Directions:

- Use the mflix sample database to prepare a pymongo query each of the following prompts.
- Be sure to print the results of your query using the dumps function.

[2]:
```python
mflixdb = client.mlfix
```

1.2.1 Question 1:

Give the street, city, and zipcode of all theaters in Massachusetts.

[46]:
```python
MA_theaters = mflixdb.theaters.aggregate([
    {"$match": {"location.address.state": "MA"}},
    {"$project": {"_id": 0, "location.address.street1": 1, "location.address.
↪city": 1, "location.address.zipcode": 1}}
])
MA_theaters = list(MA_theaters)
print(MA_theaters[:10])
# print(dumps(MA_theaters, indent=4)) # not using dumps because the output is␣
↪super long
```

```
[{'location': {'address': {'street1': '162 Santilli Hwy', 'city': 'Everett',
'zipcode': '02149'}}}, {'location': {'address': {'street1': '14 Allstate Rd',
'city': 'Dorchester', 'zipcode': '02125'}}}, {'location': {'address':
{'street1': '280 School St', 'city': 'Mansfield', 'zipcode': '02048'}}},
{'location': {'address': {'street1': '208 Fortune Blvd', 'city': 'Milford',
'zipcode': '01757'}}}, {'location': {'address': {'street1': '33 Orchard Hill
Park Dr', 'city': 'Leominster', 'zipcode': '01453'}}}, {'location': {'address':
{'street1': '2 Galleria Mall Dr', 'city': 'Taunton', 'zipcode': '02780'}}},
{'location': {'address': {'street1': '84 Middlesex Turnpike', 'city':
'Burlington', 'zipcode': '01803'}}}, {'location': {'address': {'street1': '250
Granite Street', 'city': 'Braintree', 'zipcode': '02184'}}}, {'location':
{'address': {'street1': '999 South Washington Street', 'city': 'North
Attleboro', 'zipcode': '02760'}}}, {'location': {'address': {'street1': '100
CambridgeSide Place', 'city': 'Cambridge', 'zipcode': '02141'}}}]
```

1.2.2 Question 2:

How many theaters are there in each state? Order the output in alphabetical order by 2-character state code.

[45]:
```
count_theaters = mflixdb.theaters.aggregate([
    {"$group": {"_id": "$location.address.state", "theater_count": {"$sum":␣
↪1}}},
    {"$sort": {"_id": 1}}
])

count_theaters = list(count_theaters)
print(count_theaters[:10])
```

```
[{'_id': 'AK', 'theater_count': 4}, {'_id': 'AL', 'theater_count': 19}, {'_id':
'AR', 'theater_count': 16}, {'_id': 'AZ', 'theater_count': 26}, {'_id': 'CA',
'theater_count': 169}, {'_id': 'CO', 'theater_count': 26}, {'_id': 'CT',
'theater_count': 21}, {'_id': 'DC', 'theater_count': 3}, {'_id': 'DE',
'theater_count': 5}, {'_id': 'FL', 'theater_count': 111}]
```

1.2.3 Question 3:

How many movies are in the Comedy genre?

[13]:
```
comedy_movies_count = mflixdb.movies.count_documents({"genres": {"$in":␣
↪["Comedy"]}})
print(dumps(comedy_movies_count, indent=4))
```

```
6532
```

1.2.4 Question 4:

What movie has the longest run time? Give the movie's title and genre(s).

[21]:
```python
runtimes = mflixdb.movies.aggregate([
    {"$sort": {"runtime": -1}},
    {"$limit": 1},
    {"$project": {"_id": 0,"title": 1, "genres": 1, "runtime": 1}}
])
print(dumps(longest, indent=4))
```

```
{
    "genres": [
        "Action",
        "Adventure",
        "Drama"
    ],
    "runtime": 1256,
    "title": "Centennial"
}
```

1.2.5 Question 5:

Which movies released after 2010 have a Rotten Tomatoes viewer rating of 3 or higher? Give the title of the movies along with their Rotten Tomatoes viewer rating score. The viewer rating score should become a top-level attribute of the returned documents. Return the matching movies in descending order by viewer rating.

[44]:
```python
by_rating = mflixdb.movies.aggregate([
    {"$match": {"year": {"$gt": 2010}, "tomatoes.viewer.rating": {"$gte": 3.
 ↪0}}},
    {"$sort": {"tomatoes.viewer.rating": -1}},
    {"$project": {"_id":0, "title": 1, "rating":"$tomatoes.viewer.rating"}},
])
by_rating = list(by_rating)
print(by_rating[:10])
```

```
[{'title': 'Scooby-Doo! Mask of the Blue Falcon', 'rating': 5}, {'title': 'The
Dead and the Living', 'rating': 5}, {'title': 'My Last Year with the Nuns',
'rating': 5}, {'title': 'Silenced', 'rating': 5}, {'title': "Rosemary's Baby",
'rating': 5}, {'title': 'Crocodile Gennadiy', 'rating': 5}, {'title': 'The
Dancer', 'rating': 5}, {'title': "Beethoven's Christmas Adventure", 'rating':
5}, {'title': 'Scattered Cloud', 'rating': 5}, {'title': 'The Color of Rain',
'rating': 5}]
```

1.2.6 Question 6:

How many movies released each year have a plot that contains some type of police activity (i.e., plot contains the word "police")? The returned data should be in ascending order by year.

[47]:
```python
count_police = mflixdb.movies.aggregate([
    {"$match": {"plot": {"$regex": "police"}}},
    {"$group": {"_id": "$year", "movie_count": {"$sum": 1}}},
    {"$sort": {"_id": 1}}
])
count_police = list(count_police)
print(count_police)
```

```
[{'_id': 1913, 'movie_count': 1}, {'_id': 1934, 'movie_count': 1}, {'_id': 1935,
'movie_count': 1}, {'_id': 1944, 'movie_count': 1}, {'_id': 1947, 'movie_count':
1}, {'_id': 1948, 'movie_count': 1}, {'_id': 1949, 'movie_count': 1}, {'_id':
1950, 'movie_count': 2}, {'_id': 1951, 'movie_count': 2}, {'_id': 1957,
'movie_count': 1}, {'_id': 1958, 'movie_count': 1}, {'_id': 1959, 'movie_count':
2}, {'_id': 1960, 'movie_count': 1}, {'_id': 1961, 'movie_count': 1}, {'_id':
1963, 'movie_count': 2}, {'_id': 1965, 'movie_count': 2}, {'_id': 1966,
'movie_count': 1}, {'_id': 1967, 'movie_count': 2}, {'_id': 1968, 'movie_count':
1}, {'_id': 1969, 'movie_count': 2}, {'_id': 1970, 'movie_count': 2}, {'_id':
1971, 'movie_count': 2}, {'_id': 1972, 'movie_count': 3}, {'_id': 1973,
'movie_count': 6}, {'_id': 1974, 'movie_count': 2}, {'_id': 1975, 'movie_count':
5}, {'_id': 1976, 'movie_count': 1}, {'_id': 1977, 'movie_count': 1}, {'_id':
1978, 'movie_count': 5}, {'_id': 1979, 'movie_count': 3}, {'_id': 1980,
'movie_count': 4}, {'_id': 1981, 'movie_count': 6}, {'_id': 1982, 'movie_count':
3}, {'_id': 1983, 'movie_count': 6}, {'_id': 1984, 'movie_count': 6}, {'_id':
1985, 'movie_count': 7}, {'_id': 1986, 'movie_count': 3}, {'_id': 1987,
'movie_count': 4}, {'_id': 1988, 'movie_count': 3}, {'_id': 1989, 'movie_count':
9}, {'_id': 1990, 'movie_count': 5}, {'_id': 1991, 'movie_count': 6}, {'_id':
1992, 'movie_count': 14}, {'_id': 1993, 'movie_count': 7}, {'_id': 1994,
'movie_count': 6}, {'_id': 1995, 'movie_count': 11}, {'_id': 1996,
'movie_count': 9}, {'_id': 1997, 'movie_count': 10}, {'_id': 1998,
'movie_count': 15}, {'_id': 1999, 'movie_count': 12}, {'_id': 2000,
'movie_count': 14}, {'_id': 2001, 'movie_count': 14}, {'_id': 2002,
'movie_count': 12}, {'_id': 2003, 'movie_count': 13}, {'_id': 2004,
'movie_count': 13}, {'_id': 2005, 'movie_count': 14}, {'_id': 2006,
'movie_count': 24}, {'_id': 2007, 'movie_count': 11}, {'_id': 2008,
'movie_count': 12}, {'_id': 2009, 'movie_count': 11}, {'_id': 2010,
'movie_count': 15}, {'_id': 2011, 'movie_count': 20}, {'_id': 2012,
'movie_count': 17}, {'_id': 2013, 'movie_count': 19}, {'_id': 2014,
'movie_count': 17}, {'_id': 2015, 'movie_count': 2}, {'_id': '1988è',
'movie_count': 1}]
```

1.2.7 Question 7:

What is the average number of imdb votes per year for movies released between 1970 and 2000 (inclusive)? Make sure the results are ordered by year.

```
[48]: avg_votes = mflixdb.movies.aggregate([
          {"$match": {"year": {"$gte": 1970, "$lte": 2000}}},
          {"$group": {"_id": {"release year": "$year"}, "Avg Votes": {"$avg": "$imdb.
       ↪votes"}}},
          {"$sort": {"_id": 1}}
      ])
      avg_votes = list(avg_votes)
      print(avg_votes)
```

```
[{'_id': {'release year': 1970}, 'Avg Votes': 4786.925}, {'_id': {'release
year': 1971}, 'Avg Votes': 8528.462264150943}, {'_id': {'release year': 1972},
'Avg Votes': 13582.685950413223}, {'_id': {'release year': 1973}, 'Avg Votes':
14478.785714285714}, {'_id': {'release year': 1974}, 'Avg Votes': 17602.0},
{'_id': {'release year': 1975}, 'Avg Votes': 19615.00934579439}, {'_id':
{'release year': 1976}, 'Avg Votes': 14259.76724137931}, {'_id': {'release
year': 1977}, 'Avg Votes': 14210.393442622952}, {'_id': {'release year': 1978},
'Avg Votes': 9992.5703125}, {'_id': {'release year': 1979}, 'Avg Votes':
15729.419847328245}, {'_id': {'release year': 1980}, 'Avg Votes':
16487.155688622755}, {'_id': {'release year': 1981}, 'Avg Votes':
12193.797619047618}, {'_id': {'release year': 1982}, 'Avg Votes':
15729.898305084746}, {'_id': {'release year': 1983}, 'Avg Votes':
15521.664596273293}, {'_id': {'release year': 1984}, 'Avg Votes':
19255.838383838385}, {'_id': {'release year': 1985}, 'Avg Votes':
15590.121693121693}, {'_id': {'release year': 1986}, 'Avg Votes':
19503.78947368421}, {'_id': {'release year': 1987}, 'Avg Votes':
18074.545045045044}, {'_id': {'release year': 1988}, 'Avg Votes':
16892.521912350596}, {'_id': {'release year': 1989}, 'Avg Votes':
18821.879310344826}, {'_id': {'release year': 1990}, 'Avg Votes':
22580.324444444443}, {'_id': {'release year': 1991}, 'Avg Votes':
19933.7731092437}, {'_id': {'release year': 1992}, 'Avg Votes':
19883.774074074074}, {'_id': {'release year': 1993}, 'Avg Votes':
23896.525547445257}, {'_id': {'release year': 1994}, 'Avg Votes':
38413.69836065574}, {'_id': {'release year': 1995}, 'Avg Votes':
25700.217741935485}, {'_id': {'release year': 1996}, 'Avg Votes':
18431.299754299755}, {'_id': {'release year': 1997}, 'Avg Votes':
26613.216400911162}, {'_id': {'release year': 1998}, 'Avg Votes':
23481.639376218325}, {'_id': {'release year': 1999}, 'Avg Votes':
30182.335922330098}, {'_id': {'release year': 2000}, 'Avg Votes':
25951.459552495697}]
```

1.2.8 Question 8:

What distinct movie languages are represented in the database? You only need to provide the list of languages.

```
[51]: languages = mflixdb.movies.aggregate([
          {"$unwind": "$languages"},
          {"$group": {"_id": "$languages"}},
```

```python
])

languages = [dic["_id"] for dic in languages]
print(languages)
```

```
['Ladakhi', 'Yiddish', 'Klingon', 'Nyanja', 'Oriya', 'Breton', 'Aymara',
'Kannada', 'Cornish', 'Kazakh', 'Manipuri', 'Ladino', 'Hassanya', 'Scottish
Gaelic', 'Nenets', 'Kirghiz', 'Romanian', 'Tulu', 'Luxembourgish', 'Southern
Sotho', 'Gallegan', 'Pawnee', 'Japanese Sign Language', 'Russian Sign Language',
'Nama', 'Nepali', 'Polish', 'Shanxi', 'Ryukyuan', 'Dzongkha', 'Acholi', 'Irish',
'German Sign Language', 'Kurdish', 'Hungarian', 'Faroese', 'Latvian', 'Italian',
'Maori', 'Ungwatsi', 'Karajè', 'Min Nan', 'Mapudungun', 'Urdu', 'Georgian',
'Slovenian', 'Tswana', 'Norse', 'Hebrew', 'Uzbek', 'Basque', 'Egyptian
(Ancient)', 'Occitan', 'Fulah', 'Tatar', 'Dutch', 'Indian Sign Language',
'Dyula', 'Tonga', 'Kabuverdianu', 'Inupiaq', 'Assamese', 'Swedish', 'Quechua',
'Maltese', 'Shanghainese', 'More', 'Spanish Sign Language', 'Thai', 'Quenya',
'Croatian', 'English', 'Mohawk', 'Scanian', 'Sindarin', 'Wolof', 'Macedonian',
'Cheyenne', 'Portuguese', 'Burmese', 'Japanese', 'Chechen', 'Nyaneka',
'Turkish', 'Maya', 'Visayan', 'Algonquin', 'Romany', 'Samoan', 'Fur', 'Sign
Languages', 'Cree', ' Ancient (to 1453)', 'Jola-Fonyi', 'Peul', 'Kuna', 'Hindi',
'Mandarin', 'Sinhalese', 'Purepecha', 'Serbo-Croatian', 'Albanian', 'Telugu',
'Bambara', 'Apache languages', 'Low German', 'Tok Pisin', 'Shona', 'Turkmen',
'Norwegian', 'Kikuyu', 'Creoles and pidgins', 'Mongolian', 'Gujarati',
'Aidoukrou', 'Uighur', 'Berber languages', 'Bengali', 'Aboriginal',
'Rajasthani', 'Xhosa', 'Spanish', 'Afrikaans', 'Tamil', 'Greenlandic',
'Tarahumara', 'British Sign Language', 'Catalan', 'Mende', 'Assyrian Neo-
Aramaic', ' Old', 'Slovak', 'Old English', 'Ukrainian', 'Tupi', 'Amharic',
'Navajo', 'Latin', 'Finnish', 'Zulu', 'Sardinian', 'Gumatj', 'Shoshoni',
'Persian', 'Dinka', 'Washoe', 'Hmong', 'Creole', 'Somali', 'Swiss German',
'Tuvinian', 'Lingala', 'Filipino', 'Konkani', 'Armenian', 'Athapascan
languages', 'Flemish', 'Azerbaijani', 'Russian', 'Swahili', 'Guarani',
'Sanskrit', 'Scots', 'Vietnamese', 'German', 'Songhay', 'Lao', 'Arapaho',
'Chinese', 'Tzotzil', 'Malayalam', 'Inuktitut', 'Icelandic', 'Esperanto',
'Tajik', 'French Sign Language', 'Hawaiian', 'Kinyarwanda', 'Awadhi', 'Welsh',
'Czech', 'Sioux', 'Hokkien', 'Brazilian Sign Language', 'Balinese', 'Tagalog',
'Frisian', 'Estonian', 'Korean Sign Language', 'Korean', 'Middle English',
'Abkhazian', 'Syriac', 'Dari', 'Haitian', 'Panjabi', 'Tibetan', 'Eastern
Frisian', 'Cantonese', 'Tigrigna', 'Sicilian', 'Aramaic', 'Belarusian', 'Crow',
'Masai', 'Serbian', 'French', 'Marathi', 'Bosnian', 'Corsican', 'American Sign
Language', 'Lithuanian', 'Bhojpuri', 'Bulgarian', 'Greek', 'Malay', 'Ibo',
'Indonesian', 'Saami', 'Pushto', 'Mari', 'Danish', 'Yoruba', 'Arabic', 'Kabyle',
'Ewe', 'Hakka', 'North American Indian', 'Nahuatl', 'Neapolitan', 'Khmer']
```