Title of Project: Movie Analyzer

Application description:

Our application filters and calculates data from our movie database to answer specific user queries regarding various movies, from a list of possible actions. These functions are in line with what a streaming service, video site, or social media app might do. This application would appreciate NoSQL deployment, because the dataset being used can hold data of variable or unpredictable length, such as an array of various genres.

Use cases:

1. The user can enter movies that they would rate very highly, and the program will suggest what other "similar" users would also rate highly that isn't included in the user's favorites list.
2. The user can enter a movie, and the program will return a list of movies with overlapping user tags, i.e., one or more of the same user defined tags.
3. The user can enter a tag (or tags), specify if they want all of the tags to match or one of the tags to match, and the program will return a list of movies with all of those user tags or one of those tags, respectively.
4. The user can enter a movie, and the program will return the average rating across all users who have rated it, along with any tags that have been applied by more than one user (or another filter on tags).
5. The user can enter whether they want a list of the best movies or the worst movies, and the program will return all of the top-rated movies or all of the worst rated movies across the dataset, within a certain threshold.
6. The user can enter a genre and return a list of movies having the specified genre
7. The user can enter their userID and return a list of movies they have rated
8. The user can enter a movieID and return a list of tags associated with the respective movieID
9. The user can enter a character and will return a list of movies starting with the associated character
10. The user can enter 'most tagged' and return the document with movieID that is most tagged
11. To see the average ratings for each movie genre (a movie can count as multiple genres).
12. Find highest rated movies within a user-specified movie genre.
13. To search for a movie that contains a user-specified string inside the movie title.
14. To search for movies that have a rating within a user-specified range (i.e. user wants movies that have a rating between 3 to 5; application returns movies that have ratings between 3 to 5)

15. Search for movies sorted by the highest number of ratings (highest number of users that put a rating on that movie).

Dataset description:

This dataset is a collection of 100,000 ratings and 3,600 tags applied to 9,000 movies by 600 users. The data was gathered from users of the movielens.org website. This data is used for a variety of systems relating to streaming and content recommendation services. For example, a Netflix-style application may recommend new films to the user based on what similar users had watched in the past.

How to wrangle our data set:

We used a website (https://www.convertcsv.com/csv-to-json.htm) to convert the CSV files into JSON files, before we use mongoimport to import the files into our database. We choose to import the "ratings", "movies", and "tags" json files as three tables, and then we join the tables together into one table by the field "movieId", keeping fields which are not the timestamp, and the result will be our final dataset. In addition, we used a python program to convert the 'genre' field from a string to a list of string/JSON array.

High level language: Python
Driver: pymongo /other dependencies

Schema description: one collection; we plan to make an index on the field ratings, since we believe the field will be used frequently in queries. Fields: userId(Integer), movieId(Integer), title(String), genres(String), rating(Integer), tag(String)

MongoShell queries and Screenshots: Test queries on the shell for at least 5 functions

Query 1:
Collection: ratings
Database: testdb

```
#get all documents with a rating of 5 (or close enough)
db.ratings.find( { rating: { $gt: 4.7 } } );
```

ubuntu@ip-172-31-18-227: ~

```
> db.ratings.find({rating: {$gt: 4.7 }})
{ "_id" : ObjectId("6259e033c20e08d910da90fe"), "userId" : 1, "movieId" : 47, "rating" : 5, "timestamp" : 964983815 }
{ "_id" : ObjectId("6259e033c20e08d910da90ff"), "userId" : 1, "movieId" : 50, "rating" : 5, "timestamp" : 964982931 }
{ "_id" : ObjectId("6259e033c20e08d910da9101"), "userId" : 1, "movieId" : 101, "rating" : 5, "timestamp" : 964980868 }
{ "_id" : ObjectId("6259e033c20e08d910da9103"), "userId" : 1, "movieId" : 151, "rating" : 5, "timestamp" : 964984041 }
{ "_id" : ObjectId("6259e033c20e08d910da9104"), "userId" : 1, "movieId" : 157, "rating" : 5, "timestamp" : 964984100 }
{ "_id" : ObjectId("6259e033c20e08d910da9105"), "userId" : 1, "movieId" : 163, "rating" : 5, "timestamp" : 964983650 }
{ "_id" : ObjectId("6259e033c20e08d910da9106"), "userId" : 1, "movieId" : 216, "rating" : 5, "timestamp" : 964981208 }
{ "_id" : ObjectId("6259e033c20e08d910da9108"), "userId" : 1, "movieId" : 231, "rating" : 5, "timestamp" : 964981179 }
{ "_id" : ObjectId("6259e033c20e08d910da910a"), "userId" : 1, "movieId" : 260, "rating" : 5, "timestamp" : 964981680 }
{ "_id" : ObjectId("6259e033c20e08d910da910d"), "userId" : 1, "movieId" : 333, "rating" : 5, "timestamp" : 964981179 }
{ "_id" : ObjectId("6259e033c20e08d910da9110"), "userId" : 1, "movieId" : 362, "rating" : 5, "timestamp" : 964982588 }
{ "_id" : ObjectId("6259e033c20e08d910da9114"), "userId" : 1, "movieId" : 457, "rating" : 5, "timestamp" : 964981909 }
{ "_id" : ObjectId("6259e033c20e08d910da9117"), "userId" : 1, "movieId" : 527, "rating" : 5, "timestamp" : 964984002 }
{ "_id" : ObjectId("6259e033c20e08d910da911a"), "userId" : 1, "movieId" : 553, "rating" : 5, "timestamp" : 964984153 }
{ "_id" : ObjectId("6259e033c20e08d910da911e"), "userId" : 1, "movieId" : 596, "rating" : 5, "timestamp" : 964982838 }
{ "_id" : ObjectId("6259e033c20e08d910da911f"), "userId" : 1, "movieId" : 608, "rating" : 5, "timestamp" : 964982931 }
{ "_id" : ObjectId("6259e033c20e08d910da9121"), "userId" : 1, "movieId" : 661, "rating" : 5, "timestamp" : 964982838 }
{ "_id" : ObjectId("6259e033c20e08d910da9127"), "userId" : 1, "movieId" : 919, "rating" : 5, "timestamp" : 964982475 }
{ "_id" : ObjectId("6259e033c20e08d910da9128"), "userId" : 1, "movieId" : 923, "rating" : 5, "timestamp" : 964981529 }
{ "_id" : ObjectId("6259e033c20e08d910da9129"), "userId" : 1, "movieId" : 940, "rating" : 5, "timestamp" : 964982176 }
Type "it" for more
>
```

Query 2:

Collection: movies

Database: testdb

```
#get all genres for a movie
db.movies.aggregate(
... { $match: { title: { $eq: "GoldenEye (1995)"} } },
... { $project: { _id: 0, genres: 1 } }
... )
```

ubuntu@ip-172-31-30-203: ~

```
> db.movies.aggregate(
... ... { $match: { title: { $eq: "GoldenEye (1995)"} } },
... ... { $project: { _id: 0, genres: 1 } }
... ... )
{ "genres" : "Action|Adventure|Thriller" }
>
```

Query 3:

Collection: movies

Database: testdb

#get all movies for a genre

db.movies.find({"genres": { $eq: "Thriller" }}, { _id: 0 })

```
ubuntu@ip-172-31-30-203: ~
> db.movies.find( {"genres": { $eq: "Thriller" }}, { _id: 0 })
{ "movieId" : 132, "title" : "Jade (1995)", "genres" : "Thriller" }
{ "movieId" : 190, "title" : "Safe (1995)", "genres" : "Thriller" }
{ "movieId" : 240, "title" : "Hideaway (1995)", "genres" : "Thriller" }
{ "movieId" : 373, "title" : "Red Rock West (1992)", "genres" : "Thriller" }
{ "movieId" : 422, "title" : "Blink (1994)", "genres" : "Thriller" }
{ "movieId" : 457, "title" : "Fugitive, The (1993)", "genres" : "Thriller" }
{ "movieId" : 490, "title" : "Malice (1993)", "genres" : "Thriller" }
{ "movieId" : 540, "title" : "Sliver (1993)", "genres" : "Thriller" }
{ "movieId" : 662, "title" : "Fear (1996)", "genres" : "Thriller" }
{ "movieId" : 1055, "title" : "Shadow Conspiracy (1997)", "genres" : "Thriller" }
{ "movieId" : 1061, "title" : "Sleepers (1996)", "genres" : "Thriller" }
{ "movieId" : 1343, "title" : "Cape Fear (1991)", "genres" : "Thriller" }
{ "movieId" : 1718, "title" : "Stranger in the House (1997)", "genres" : "Thriller" }
{ "movieId" : 1798, "title" : "Hush (1998)", "genres" : "Thriller" }
{ "movieId" : 1892, "title" : "Perfect Murder, A (1998)", "genres" : "Thriller" }
{ "movieId" : 2118, "title" : "Dead Zone, The (1983)", "genres" : "Thriller" }
{ "movieId" : 2178, "title" : "Frenzy (1972)", "genres" : "Thriller" }
{ "movieId" : 2179, "title" : "Topaz (1969)", "genres" : "Thriller" }
{ "movieId" : 2180, "title" : "Torn Curtain (1966)", "genres" : "Thriller" }
{ "movieId" : 2185, "title" : "I Confess (1953)", "genres" : "Thriller" }
Type "it" for more
>
```

Query 4:

#To search for a movie that contains a user-specified string inside the movie title.

db.movies.find({title: {$regex: 'Toy'}})

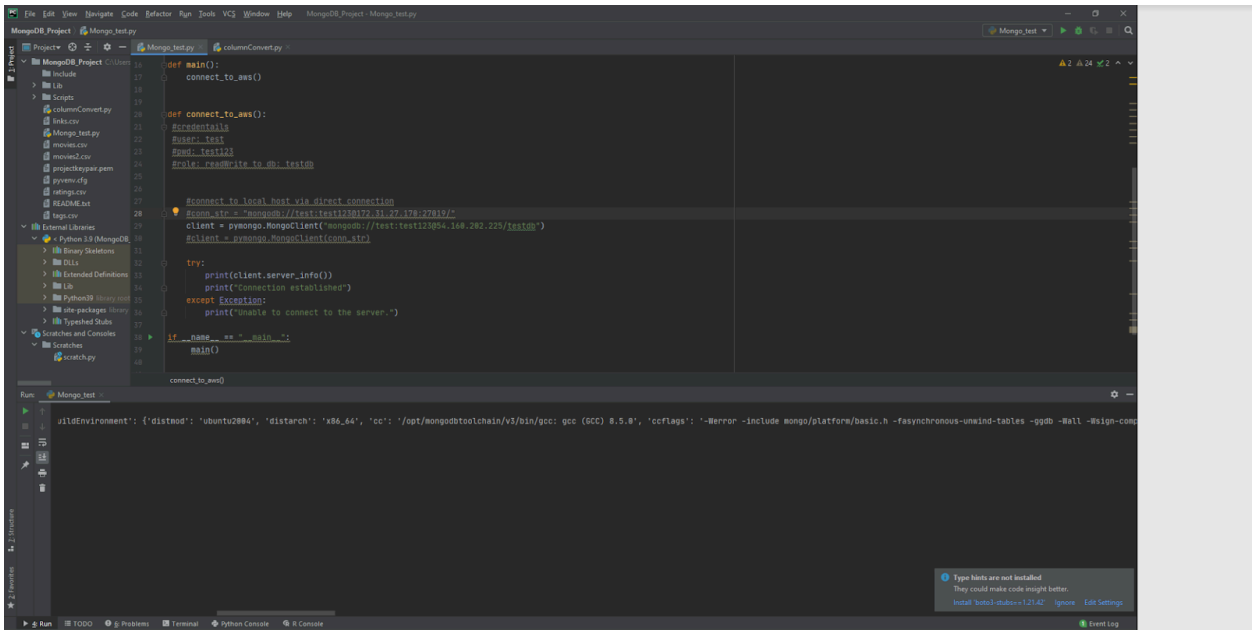
```
ubuntu@ip-172-31-30-203: ~
> db.movies.find({title: {$regex: 'Toy'}})
{ "_id" : ObjectId("6259ed2d2cbaa39cb846ee3a"), "movieId" : 1, "title" : "Toy Story (1995)", "genres" :
"Adventure|Animation|Children|Comedy|Fantasy" }
{ "_id" : ObjectId("6259ed2d2cbaa39cb846f40e"), "movieId" : 2017, "title" : "Babes in Toyland (1961)", "
genres" : "Children|Fantasy|Musical" }
{ "_id" : ObjectId("6259ed2d2cbaa39cb846f4c4"), "movieId" : 2253, "title" : "Toys (1992)", "genres" : "C
omedy|Fantasy" }{ "_id" : ObjectId("6259ed2d2cbaa39cb846f753"), "movieId" : 3086, "title" : "Babes in To
yland (1934)", "genres" : "Children|Comedy|Fantasy|Musical" }
{ "_id" : ObjectId("6259ed2d2cbaa39cb846f76d"), "movieId" : 3114, "title" : "Toy Story 2 (1999)", "genre
s" : "Adventure|Animation|Children|Comedy|Fantasy" }
{ "_id" : ObjectId("6259ed2d2cbaa39cb846fc45"), "movieId" : 4929, "title" : "Toy, The (1982)", "genres"
: "Comedy" }
{ "_id" : ObjectId("6259ed2d2cbaa39cb846fe33"), "movieId" : 5843, "title" : "Toy Soldiers (1991)", "genr
es" : "Action|Drama" }
{ "_id" : ObjectId("6259ed2d2cbaa39cb8470af5"), "movieId" : 78499, "title" : "Toy Story 3 (2010)", "genr
es" : "Adventure|Animation|Children|Comedy|Fantasy|IMAX" }
>
```

Query 5:

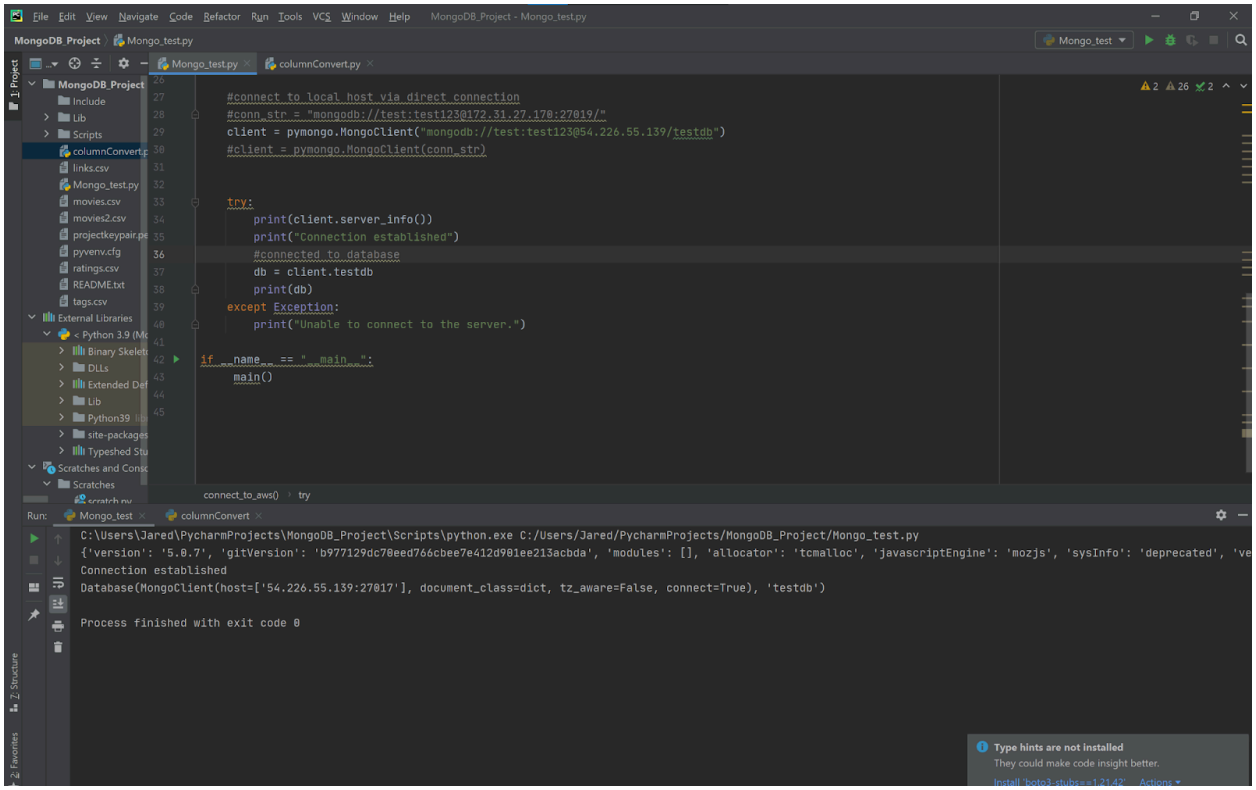
```
#get all genres for a movie
db.movies.aggregate(
... { $match: { title: { $eq: "Toy Story (1995)"} } },
... { $project: { _id: 0, genres: 1 } }
... )
```

```
ubuntu@ip-172-31-30-203: ~
> db.movies.aggregate(
... ... { $match: { title: { $eq: "Toy Story (1995)"} } },
... ... { $project: { _id: 0, genres: 1 } }
... ... )
{ "genres" : "Adventure|Animation|Children|Comedy|Fantasy" }
>
```

End-to-end connection from application to MongoDB (screenshot):



End-to-End connection with Mongos:
Public IP used: 54.226.55.139
Private IP: 172.31.27.170:27017
Description: establishes connection to Mongos instance in AWS using a user's credentials and prints a successful connection message, server and database information to the console.



Output:
{'version': '5.0.7', 'gitVersion': 'b977129dc70eed766cbee7e412d901ee213acbda', 'modules': [], 'allocator': 'tcmalloc', 'javascriptEngine': 'mozjs', 'sysInfo': 'deprecated', 'versionArray': [5, 0, 7, 0], 'openssl': {'running': 'OpenSSL 1.1.1f 31 Mar 2020', 'compiled': 'OpenSSL 1.1.1f 31 Mar 2020'}, 'buildEnvironment': {'distmod': 'ubuntu2004', 'distarch': 'x86_64', 'cc': '/opt/mongodbtoolchain/v3/bin/gcc: gcc (GCC) 8.5.0', 'ccflags': '-Werror -include mongo/platform/basic.h -fasynchronous-unwind-tables -ggdb -Wall -Wsign-compare -Wno-unknown-pragmas -Winvalid-pch -fno-omit-frame-pointer -fno-strict-aliasing -O2 -march=sandybridge -mtune=generic -mprefer-vector-width=128 -Wno-unused-local-typedefs -Wno-unused-function -Wno-deprecated-declarations -Wno-unused-const-variable -Wno-unused-but-set-variable -Wno-missing-braces -fstack-protector-strong -Wa,--nocompress-debug-sections -fno-builtin-memcmp', 'cxx': '/opt/mongodbtoolchain/v3/bin/g++: g++ (GCC) 8.5.0', 'cxxflags': '-Woverloaded-virtual -Wno-maybe-uninitialized -fsized-deallocation -std=c++17', 'linkflags': '-Wl,--fatal-warnings -pthread -Wl,-z,now -fuse-ld=gold -fstack-protector-strong -Wl,--no-threads -Wl,--build-id -Wl,--hash-style=gnu -Wl,-z,noexecstack -Wl,--warn-execstack -Wl,-z,relro -Wl,--compress-debug-sections=none -Wl,-z,origin -Wl,--enable-new-dtags', 'target_arch': 'x86_64', 'target_os': 'linux', 'cppdefines': 'SAFEINT_USE_INTRINSICS 0 PCRE_STATIC NDEBUG _XOPEN_SOURCE 700 _GNU_SOURCE _FORTIFY_SOURCE 2 BOOST_THREAD_VERSION 5 BOOST_THREAD_USES_DATETIME BOOST_SYSTEM_NO_DEPRECATED

BOOST_MATH_NO_LONG_DOUBLE_MATH_FUNCTIONS
BOOST_ENABLE_ASSERT_DEBUG_HANDLER
BOOST_LOG_NO_SHORTHAND_NAMES BOOST_LOG_USE_NATIVE_SYSLOG
BOOST_LOG_WITHOUT_THREAD_ATTR ABSL_FORCE_ALIGNED_ACCESS'}, 'bits':
64, 'debug': False, 'maxBsonObjectSize': 16777216, 'ok': 1.0, '$clusterTime': {'clusterTime':
Timestamp(1650152814, 1), 'signature': {'hash':
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00',
'keyId': 0}}, 'operationTime': Timestamp(1650152814, 1)}
Connection established
Database(MongoClient(host=['54.226.55.139:27017'], document_class=dict, tz_aware=False,
connect=True), 'testdb')
Collection(Database(MongoClient(host=['54.226.55.139:27017'], document_class=dict,
tz_aware=False, connect=True), 'testdb'), 'tests')