Movies Analysis

Elnaz Dehkharghani

Introduction:

Data is required everywhere to draw meaningful insights about any business. Afterall a useful and insightful analysis of data cannot begin unless the data becomes available. And it is known to all, how important it is to find interesting patterns and visualize the data to gain success in any field. But the flow of data from one place to other is not as simple as it sounds to be. Handling data is treacherous as so many things can go wrong while the transportation. Data can get corrupted, duplicated or accidentally deleted which might increase the latency in order to meet the performance requirements. We all know how difficult it is to work with Big data and imagining a process to extract, transform, aggregate, validate etc. as a manual process can be seen as unrealistic approach.

So here we have a data pipeline, which solves this issue at some level by eliminating many manual steps from the process and enable a smooth and automated flow of data from one place to another. It views all the data as streams of data and give us the flexibility to store the data with any schema.

Chosen dataset:

We chose movie dataset for building our pipeline. There are two main sources of data.

- API Call
  We have used the TMDB API [1] calls to fetch data for various movies.
- Web Scrapping
  We have scraped the data from the "The Numbers" [2] website.

Motivation:

Films can be an amazing source of entertaining people due to their key plot messages and unforgettable acting performance in some master pieces. However, they are also a significant part of the business and as such, their main goal is to be profitable. That's why understanding the patterns in successful movies is a crucial step for assisting businesses in establishing the basis of more successful movies. While there are lot of data available on internet about movies provided by IMDB, TMDB, Rotten tomatoes and what not, do we ever wonder what is helping these movies becoming a massive hit? Is there a pattern we can find in the success story of all the movies? We all know that we all have evolved over time, so does our taste in everything, so it might be interesting to see how the love for different kind of movies is changing over time.

Therefore, some of the most interesting questions to answer are:

- What are the most relevant features in successful movies?
- Does budget determine the success of a movie?
- What is the people's satisfaction regarding movies with more budget?
- If the popularity of a movie is helping it earn more profit?

Data Sources:

- Movies API (discover): Json format result to get information about movies.
- Movies API (movie): Json format result to get detailed information about a specific movie
- Movies Website (To determine the budget and revenue): Tabular format result to get the budget and revenue of a specific movie

Tools:

- Python: For calling API requests, manipulate and visualize data
- Beautiful Soup: For web scraping budget and revenue data.
- Apache Kafka: Data Integration tool and data preprocessing
- MongoDB: For storing unstructured results as documents.

Methodology:

We can divide our project broadly in three different steps.

- Ingesting data from different data sources.
- Streaming data from the different data sources to Mongo DB through Apache Kafka.
- Visualizing the unstructured data stored

Steps:

The following diagram describe the necessary steps to answer our business questions.

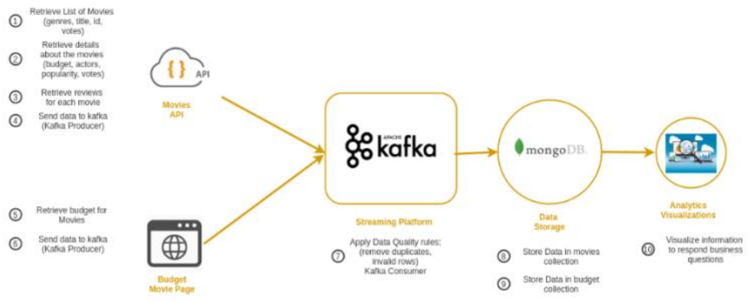

① Retrieve List of Movies (genres, title, id, votes)
② Retrieve details about the movies (budget, actors, popularity, votes)
③ Retrieve reviews for each movie
④ Send data to kafka (Kafka Producer)
API
Movies API
⑤ Retrieve budget for Movies
⑥ Send data to kafka (Kafka Producer)
Budget Movie Page
Apache kafka
Streaming Platform
⑦ Apply Data Quality rules: (remove duplicates, invalid rows) Kafka Consumer
mongoDB
Data Storage
⑧ Store Data in movies collection
⑨ Store Data in budget collection
Analytics Visualizations
⑩ Visualize information to respond business questions

Figure 1.1 General architecture of the system. Steps are indicated for each stage.

Difficulties:

- Get Kafka up and running: Lack of knowledge in this tool and complications in Kafka configuration.
- Not a clear understanding in Kafka architecture (consumers and producers) to exchange messages for data processing.
- Kafka connectors configuration to enable data transfer directly to MongoDB.
- Setting up Docker images and modify their configuration for Kafka Confluent.

Decisions:

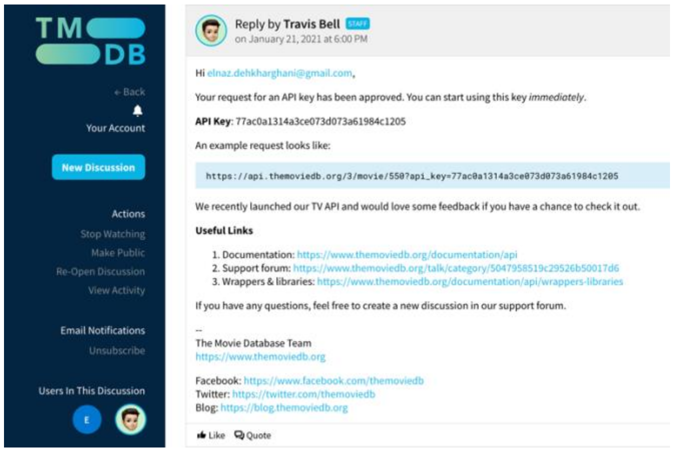
- Discard Kafka Confluent after having difficulties in installing it and configuring its Mongo Connector as a Docker image. Run the code locally.
- Originally text analysis was considered in our movies' reviews from Reddit, but due to the complexity of the topic and the deadline of the project we decided to put it aside for now.

Data Ingestion:

We are writing scripts in python to fetch data from two data sources.

Requesting the Movie API:

In order to receive an API key from the "The MovieDB" website, we needed to create an account. After that, in the API tab or in the provided email address the API Key will be appeared.



Reply by Travis Bell STAFF
on January 21, 2021 at 6:00 PM

Hi elnaz.dehkharghani@gmail.com,

Your request for an API key has been approved. You can start using this key *immediately*.

API Key: 77ac0a1314a3ce073d073a61984c1205

An example request looks like:

```
https://api.themoviedb.org/3/movie/550?api_key=77ac0a1314a3ce073d073a61984c1205
```

We recently launched our TV API and would love some feedback if you have a chance to check it out.

Useful Links

1. Documentation: https://www.themoviedb.org/documentation/api
2. Support forum: https://www.themoviedb.org/talk/category/5047958519c29526b50017d6
3. Wrappers & libraries: https://www.themoviedb.org/documentation/api/wrappers-libraries

If you have any questions, feel free to create a new discussion in our support forum.

--
The Movie Database Team
https://www.themoviedb.org

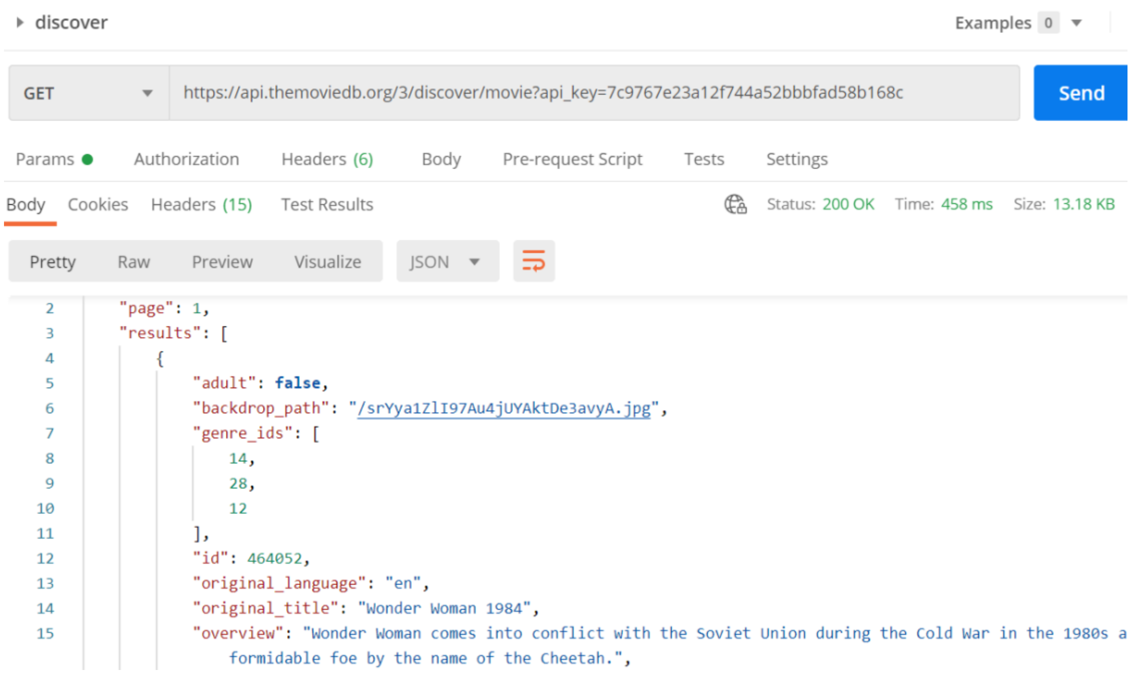
Facebook: https://www.facebook.com/themoviedb
Twitter: https://twitter.com/themoviedb
Blog: https://blog.themoviedb.org

Figure 1.2 Requesting Movies API key to access data.

First, one deals with calling the TMDB (the movie data base) API. We are basically calling two of its API requests. Calling both the APIs require an API key we already received in the previous step.

- **GET**/discover/movie

  Discover API discovers movies by different types of data. It gives an overview about title of the movie, release date, popularity, vote count. We can also apply filters for release data, title, page value etc. to narrow down the results. An example of the API call can be seen in the below figure.

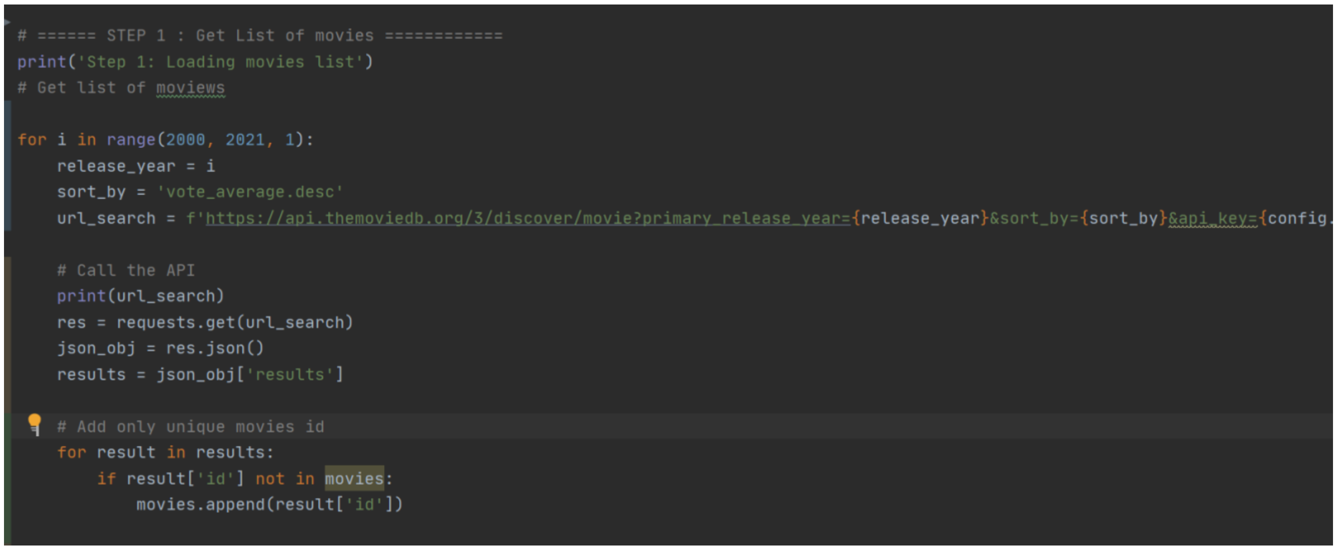


```
https://api.themoviedb.org/3/discover/movie?api_key=7c9767e23a12f744a52bbbfad58b168c
```

```
    "page": 1,
    "results": [
        {
            "adult": false,
            "backdrop_path": "/srYya1ZlI97Au4jUYAktDe3avyA.jpg",
            "genre_ids": [
                14,
                28,
                12
            ],
            "id": 464052,
            "original_language": "en",
            "original_title": "Wonder Woman 1984",
            "overview": "Wonder Woman comes into conflict with the Soviet Union during the Cold War in the 1980s a
                formidable foe by the name of the Cheetah.",
```

Figure 1.3 Example of the /discover/movie API call.

We have stored the data from the year 2000 till 2021. Using this API, we are storing the ids of all the movies that were released in this duration as the id field is the unique identifier here which we can use in the next API call.
Only the unique ids are being stored to avoid duplicates in data.



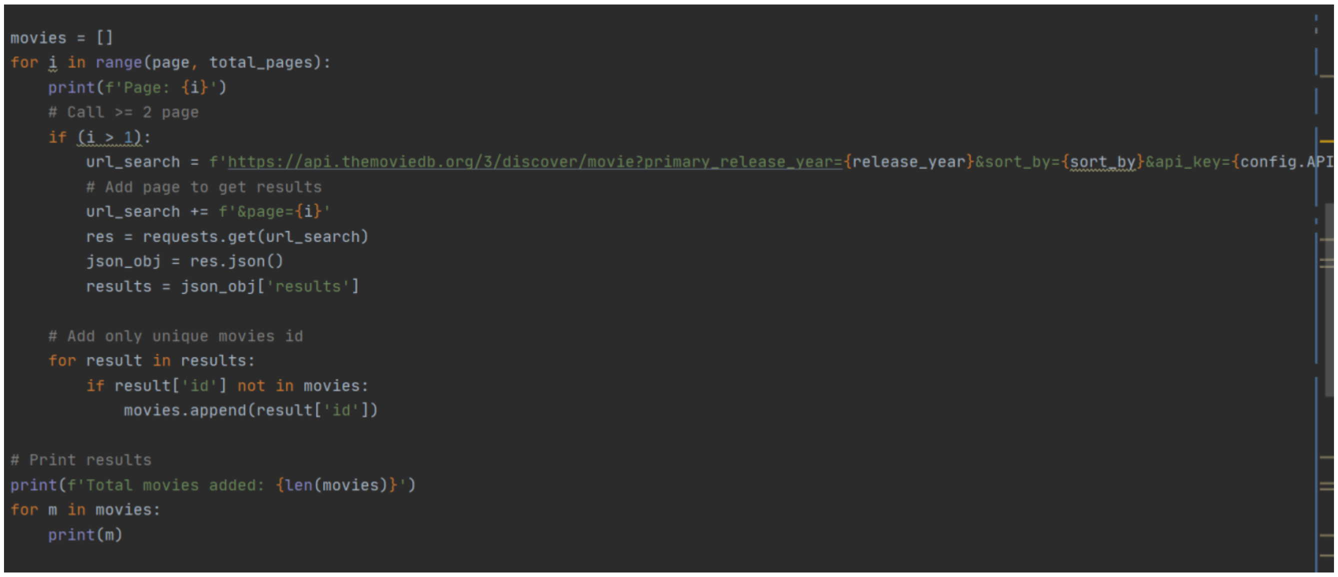
```
# ====== STEP 1 : Get List of movies ============
print('Step 1: Loading movies list')
# Get list of moviews

for i in range(2000, 2021, 1):
    release_year = i
    sort_by = 'vote_average.desc'
    url_search = f'https://api.themoviedb.org/3/discover/movie?primary_release_year={release_year}&sort_by={sort_by}&api_key={config.

    # Call the API
    print(url_search)
    res = requests.get(url_search)
    json_obj = res.json()
    results = json_obj['results']

    # Add only unique movies id
    for result in results:
        if result['id'] not in movies:
            movies.append(result['id'])
```

Figure 1.4 Remove duplicated movies ids after calling /discover/movie API.

- GET/movie/{movie_id}

After we have all the ids of the movies released in the year 2000 till 2021, we can call the above API with the movie id. It will give a more detailed response for the movies which will include information like budget, production companies, production countries, revenue etc.

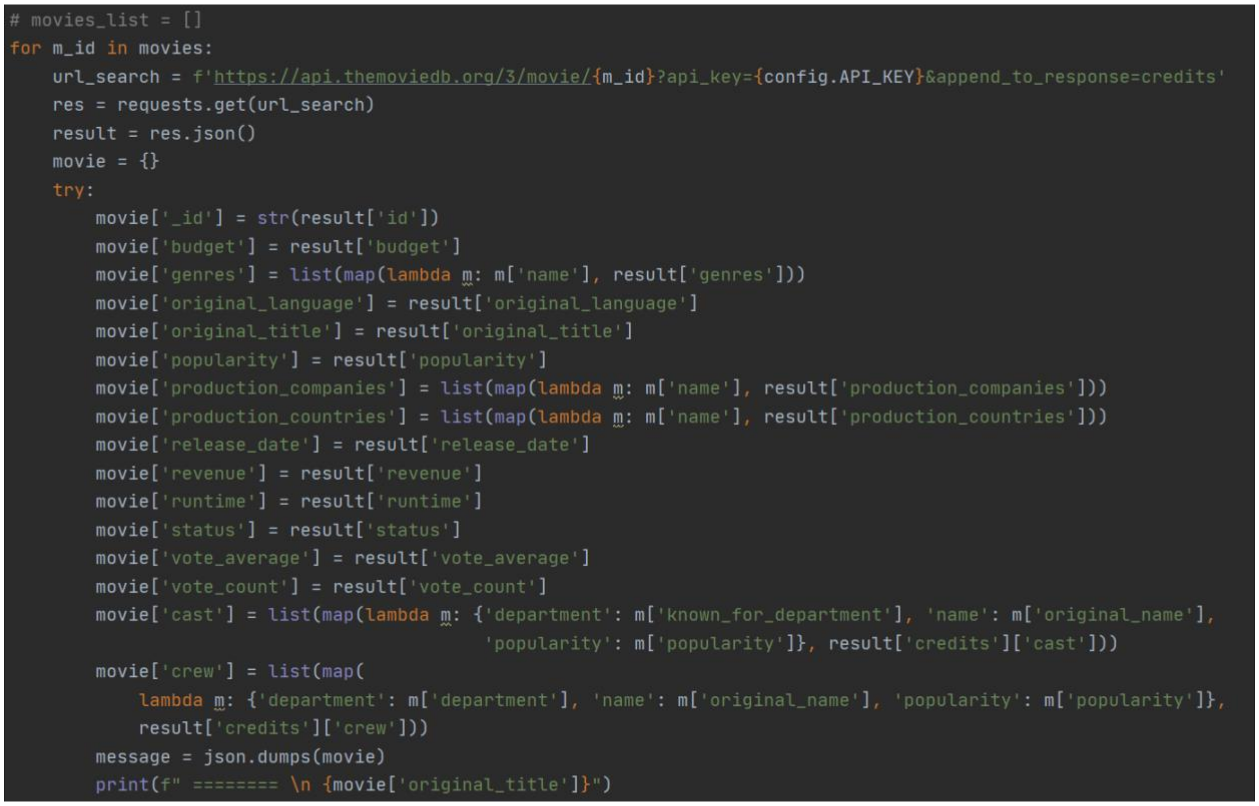
There is also an optional query parameter `append_to_response` which helps us get information like cast and crew members who took part in that movie.



```python
movies = []
for i in range(page, total_pages):
    print(f'Page: {i}')
    # Call >= 2 page
    if (i > 1):
        url_search = f'https://api.themoviedb.org/3/discover/movie?primary_release_year={release_year}&sort_by={sort_by}&api_key={config.API
        # Add page to get results
        url_search += f'&page={i}'
        res = requests.get(url_search)
        json_obj = res.json()
        results = json_obj['results']

    # Add only unique movies id
    for result in results:
        if result['id'] not in movies:
            movies.append(result['id'])

# Print results
print(f'Total movies added: {len(movies)}')
for m in movies:
    print(m)
```

Figure 1.5 Obtain movie details by calling /movie/movie_id API.

Results of both the APIs were stored as a JSON object. Also, we thought of just keeping the useful fields for sending data to Kafka for visualization and not sending the entire response. The fields we sent to Kafka were as follows:



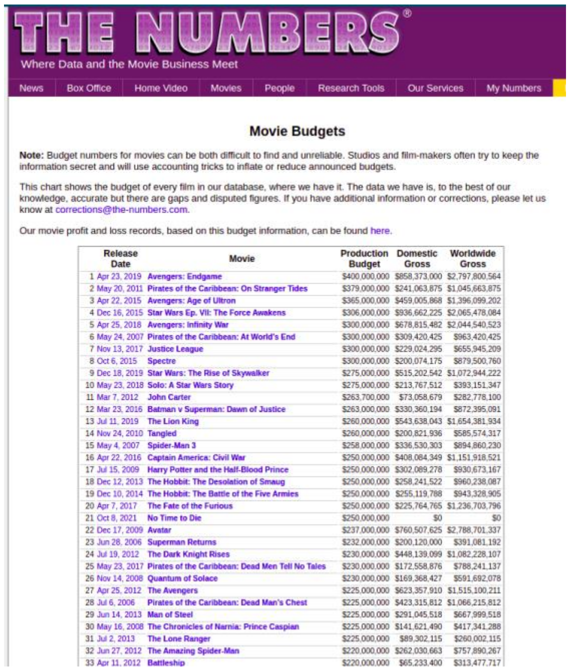
```python
# movies_list = []
for m_id in movies:
    url_search = f'https://api.themoviedb.org/3/movie/{m_id}?api_key={config.API_KEY}&append_to_response=credits'
    res = requests.get(url_search)
    result = res.json()
    movie = {}
    try:
        movie['_id'] = str(result['id'])
        movie['budget'] = result['budget']
        movie['genres'] = list(map(lambda m: m['name'], result['genres']))
        movie['original_language'] = result['original_language']
        movie['original_title'] = result['original_title']
        movie['popularity'] = result['popularity']
        movie['production_companies'] = list(map(lambda m: m['name'], result['production_companies']))
        movie['production_countries'] = list(map(lambda m: m['name'], result['production_countries']))
        movie['release_date'] = result['release_date']
        movie['revenue'] = result['revenue']
        movie['runtime'] = result['runtime']
        movie['status'] = result['status']
        movie['vote_average'] = result['vote_average']
        movie['vote_count'] = result['vote_count']
        movie['cast'] = list(map(lambda m: {'department': m['known_for_department'], 'name': m['original_name'],
                                            'popularity': m['popularity']}, result['credits']['cast']))
        movie['crew'] = list(map(
            lambda m: {'department': m['department'], 'name': m['original_name'], 'popularity': m['popularity']},
            result['credits']['crew']))
        message = json.dumps(movie)
        print(f" ======== \n {movie['original_title']}")
```

Figure 1.6 Fields of interest of each movie to send to Kafka.

Budget Movies

The data we received from the API had a lot of null values for the field budget and revenue. These fields were important parts of our visualization analysis. Though removing all those documents having null value for budget and revenue would have incurred a significant data loss.

We tried to gather data from a website for the missing values of budget and revenue. The source was "The Numbers where data and the movie business meet" [2]. We needed to do web data scraping to fulfil our requirement. The advantage is that the web page structure makes this process achievable, and the information can be easily obtained. A screenshot of the webpage is shown below:



Movie Budgets

Note: Budget numbers for movies can be both difficult to find and unreliable. Studios and film-makers often try to keep the information secret and will use accounting tricks to inflate or reduce announced budgets.

This chart shows the budget of every film in our database, where we have it. The data we have is, to the best of our knowledge, accurate but there are gaps and disputed figures. If you have additional information or corrections, please let us know at corrections@the-numbers.com.

Our movie profit and loss records, based on this budget information, can be found here.

	Release Date	Movie	Production Budget	Domestic Gross	Worldwide Gross
1	Apr 23, 2019	Avengers: Endgame	$400,000,000	$858,373,000	$2,797,800,564
2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$379,000,000	$241,063,875	$1,045,663,875
3	Apr 22, 2015	Avengers: Age of Ultron	$365,000,000	$459,005,868	$1,396,099,202
4	Dec 16, 2015	Star Wars Ep. VII: The Force Awakens	$306,000,000	$936,662,225	$2,065,478,084
5	Apr 25, 2018	Avengers: Infinity War	$300,000,000	$678,815,482	$2,044,540,523
6	May 24, 2007	Pirates of the Caribbean: At World's End	$300,000,000	$309,420,425	$963,420,425
7	Nov 13, 2017	Justice League	$300,000,000	$229,024,295	$655,945,209
8	Oct 6, 2015	Spectre	$300,000,000	$200,074,175	$879,500,760
9	Dec 18, 2019	Star Wars: The Rise of Skywalker	$275,000,000	$515,202,542	$1,072,944,222
10	May 23, 2018	Solo: A Star Wars Story	$275,000,000	$213,767,512	$393,151,347
11	Mar 7, 2012	John Carter	$263,700,000	$73,058,679	$282,778,100
12	Mar 23, 2016	Batman v Superman: Dawn of Justice	$263,000,000	$330,360,194	$872,395,091
13	Jul 11, 2019	The Lion King	$260,000,000	$543,638,043	$1,654,381,934
14	Nov 24, 2010	Tangled	$260,000,000	$200,821,936	$585,574,317
15	May 4, 2007	Spider-Man 3	$258,000,000	$336,530,303	$894,860,230
16	Apr 22, 2016	Captain America: Civil War	$250,000,000	$408,084,349	$1,151,918,521
17	Jul 15, 2009	Harry Potter and the Half-Blood Prince	$250,000,000	$302,089,278	$930,673,167
18	Dec 12, 2013	The Hobbit: The Desolation of Smaug	$250,000,000	$258,241,522	$960,238,087
19	Dec 10, 2014	The Hobbit: The Battle of the Five Armies	$250,000,000	$255,119,788	$943,328,905
20	Apr 7, 2017	The Fate of the Furious	$250,000,000	$225,764,765	$1,236,703,796
21	Oct 8, 2021	No Time to Die	$250,000,000	$0	$0
22	Dec 17, 2009	Avatar	$237,000,000	$760,507,625	$2,788,701,337
23	Jun 28, 2006	Superman Returns	$232,000,000	$200,120,000	$391,081,192
24	Jul 19, 2012	The Dark Knight Rises	$230,000,000	$448,139,099	$1,082,228,107
25	May 23, 2017	Pirates of the Caribbean: Dead Men Tell No Tales	$230,000,000	$172,558,876	$788,241,137
26	Nov 14, 2008	Quantum of Solace	$230,000,000	$169,368,427	$591,692,078
27	Apr 25, 2012	The Avengers	$225,000,000	$623,357,910	$1,515,100,211
28	Jul 6, 2006	Pirates of the Caribbean: Dead Man's Chest	$225,000,000	$423,315,812	$1,066,215,812
29	Jun 14, 2013	Man of Steel	$225,000,000	$291,045,518	$667,999,518
30	May 16, 2008	The Chronicles of Narnia: Prince Caspian	$225,000,000	$141,621,490	$417,341,288
31	Jul 2, 2013	The Lone Ranger	$225,000,000	$89,302,115	$260,002,115
32	Jun 27, 2012	The Amazing Spider-Man	$220,000,000	$262,030,663	$757,890,267
33	Apr 11, 2012	Battleship	$220,000,000	$65,233,400	$313,477,717

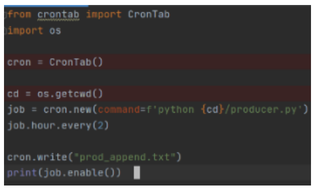
Figure 1.7 List of movies from the webpage to scrap

The general steps for this web scrapping are listed as follow:

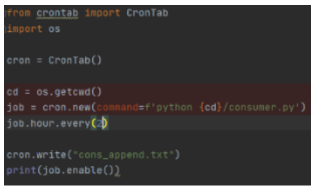
1. Use splinter and beautiful soup python modules inside a Kafka producer to access the html content of a single webpage. The movies are paginated like 101, 201, 301, etc.
2. For each page, get Title, Production budget, Domestic Gross and Worldwide Gross.
3. Send data to Kafka and process the information.

Data Source Automation:

We have automated the data sourcing process by using crontab library of python. The example of which can be seen in the underlined script. These scripts will run once every day at the second hour.



```
from crontab import CronTab
import os

cron = CronTab()

cd = os.getcwd()
job = cron.new(command=f'python {cd}/producer.py')
job.hour.every(2)

cron.write("prod_append.txt")
print(job.enable())
```



```
from crontab import CronTab
import os

cron = CronTab()

cd = os.getcwd()
job = cron.new(command=f'python {cd}/consumer.py')
job.hour.every(2)

cron.write("cons_append.txt")
print(job.enable())
```

Figure 1.8 Automation process for movies producer

Data Storage:

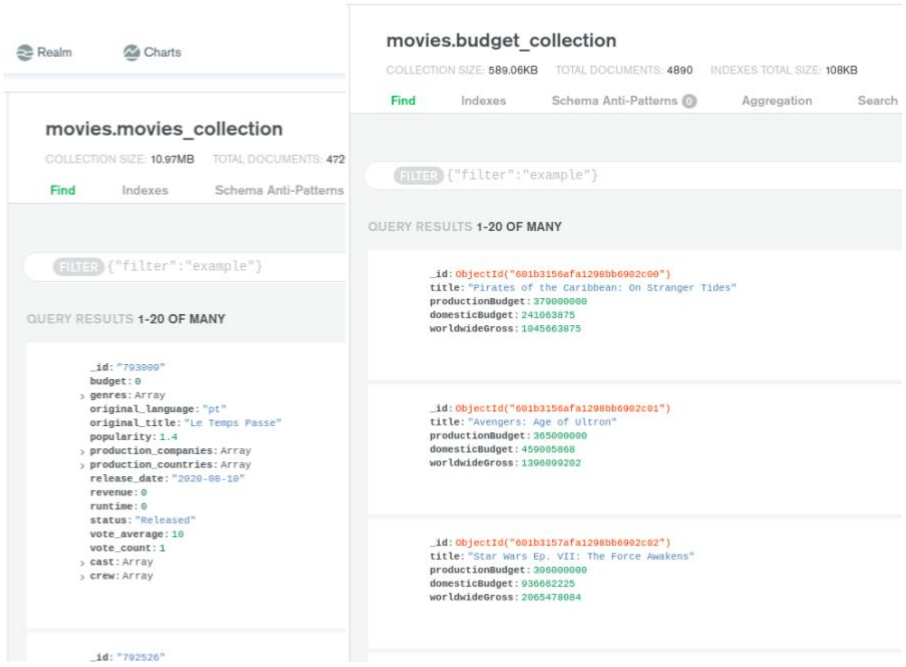
Mongo DB

As the majority of the information comes in JSON format, it is much easier to handle the storage if we use an unstructured data base like Mongo DB. In addition, the whole team wanted to learn this technology since it is more appealing to work in Big Data use cases.

Specifically, we used a python module called pymongo which allowed us to interact directly with the database from the Kafka consumers, reducing the difficulties with the connection between Kafka and Mongo DB rather than using a Kafka connector (as indicated in our difficulties).

To work simultaneously in the same database, we created a cluster in Mongo DB Atlas where the team could insert and retrieve information independently and develop different parts of the project. As we have two sources of information (movies API and budget movies) we defined 2 data collections: movies_collection and budget_collection.
Movies_collection => detailed movies information (4,729 movies as of Feb 7, 2021)
Budget_collection => financial movies information (4,890 movies as of Feb 7, 2021)

A screenshot of the collections is shown below:



movies.movies_collection

COLLECTION SIZE: 10.97MB TOTAL DOCUMENTS: 472

{"filter":"example"}

QUERY RESULTS 1-20 OF MANY

```
_id: "793009"
budget: 0
> genres: Array
original_language: "pt"
original_title: "Le Temps Passe"
popularity: 1.4
> production_companies: Array
> production_countries: Array
release_date: "2020-08-10"
revenue: 0
runtime: 0
status: "Released"
vote_average: 10
vote_count: 1
> cast: Array
> crew: Array

_id: "792526"
```

movies.budget_collection

COLLECTION SIZE: 589.06KB TOTAL DOCUMENTS: 4890 INDEXES TOTAL SIZE: 108KB

{"filter":"example"}

QUERY RESULTS 1-20 OF MANY

```
_id: ObjectId("601b3156afa1298bb6902c00")
title: "Pirates of the Caribbean: On Stranger Tides"
productionBudget: 379000000
domesticBudget: 241063875
worldwideGross: 1045663875

_id: ObjectId("601b3156afa1298bb6902c01")
title: "Avengers: Age of Ultron"
productionBudget: 365000000
domesticBudget: 459005868
worldwideGross: 1396099202

_id: ObjectId("601b3157afa1298bb6902c02")
title: "Star Wars Ep. VII: The Force Awakens"
productionBudget: 306000000
domesticBudget: 936662225
worldwideGross: 2065478084
```

Figure 1.9 Movies Collections in Mogo Atlas. Left Image - Detailed data related to movies. Right Image – Financial movies data

Kafka producers and consumers

Kafka was selected to successfully handle the data processing. As the Kafka Architecture suggests, we need to use Producers and Consumers to exchange messages of different topics, handle them and target them to their respective data storage. The following diagram shows how the producers and consumers were related:

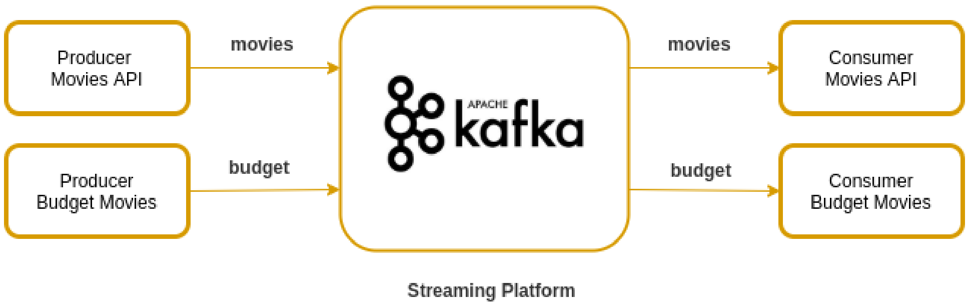


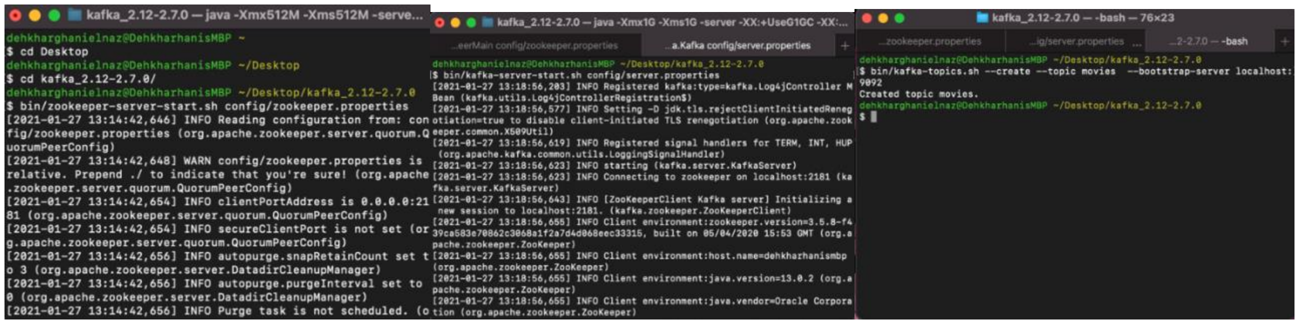
Figure 1.10 Kafka producers and consumers.

Kafka Configuration:

As mentioned before, we decided to work with Kafka, for this purpose we tried 2 approaches to setting up the environment: Locally and in Docker. But as we continued encountering difficulties in setting the MongoDB connector for Confluent, we decided to run it locally.

Local Environment:

Initially Kafka is downloaded from the mirror site. After downloading and unzipping the Kafka through the terminal you must go to the Kafka folder to run the zookeeper and the Kafka broker service to start all services in the correct order:



```
kafka_2.12-2.7.0 — java -Xmx512M -Xms512M -serve...
dehkharghanielnaz@DehkharhanisMBP ~
$ cd Desktop
dehkharghanielnaz@DehkharhanisMBP ~/Desktop
$ cd kafka_2.12-2.7.0/
dehkharghanielnaz@DehkharhanisMBP ~/Desktop/kafka_2.12-2.7.0
$ bin/zookeeper-server-start.sh config/zookeeper.properties
[2021-01-27 13:14:42,646] INFO Reading configuration from: con
fig/zookeeper.properties (org.apache.zookeeper.server.quorum.Q
uorumPeerConfig)
[2021-01-27 13:14:42,648] WARN config/zookeeper.properties is
relative. Prepend ./ to indicate that you're sure! (org.apache
.zookeeper.server.quorum.QuorumPeerConfig)
[2021-01-27 13:14:42,654] INFO clientPortAddress is 0.0.0.0:21
81 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-01-27 13:14:42,654] INFO secureClientPort is not set (or
g.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-01-27 13:14:42,656] INFO autopurge.snapRetainCount set t
o 3 (org.apache.zookeeper.server.DatadirCleanupManager)
[2021-01-27 13:14:42,656] INFO autopurge.purgeInterval set to
0 (org.apache.zookeeper.server.DatadirCleanupManager)
[2021-01-27 13:14:42,656] INFO Purge task is not scheduled. (o
```

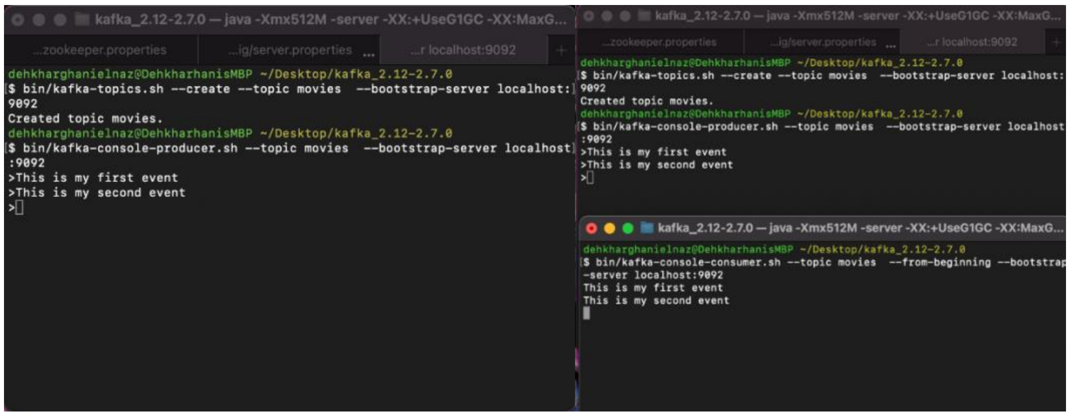
```
kafka_2.12-2.7.0 — java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:...
...eerMain config/zookeeper.properties    ...a.Kafka config/server.properties
dehkharghanielnaz@DehkharhanisMBP ~/Desktop/kafka_2.12-2.7.0
$ bin/kafka-server-start.sh config/server.properties
[2021-01-27 13:18:56,203] INFO Registered kafka:type=kafka.Log4jController M
Bean (kafka.utils.Log4jControllerRegistration$)
[2021-01-27 13:18:56,577] INFO Setting -D jdk.tls.rejectClientInitiatedReneg
otiation=true to disable client-initiated TLS renegotiation (org.apache.zook
eeper.common.X509Util)
[2021-01-27 13:18:56,619] INFO Registered signal handlers for TERM, INT, HUP
 (org.apache.kafka.common.utils.LoggingSignalHandler)
[2021-01-27 13:18:56,623] INFO starting (kafka.server.KafkaServer)
[2021-01-27 13:18:56,623] INFO Connecting to zookeeper on localhost:2181 (ka
fka.server.KafkaServer)
[2021-01-27 13:18:56,643] INFO [ZooKeeperClient Kafka server] Initializing a
 new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2021-01-27 13:18:56,655] INFO Client environment:zookeeper.version=3.5.8-f4
39ca583e70862c3068a1f2a7d4d068eec33315, built on 05/04/2020 15:53 GMT (org.a
pache.zookeeper.ZooKeeper)
[2021-01-27 13:18:56,655] INFO Client environment:host.name=dehkharhanismbp
(org.apache.zookeeper.ZooKeeper)
[2021-01-27 13:18:56,655] INFO Client environment:java.version=13.0.2 (org.a
pache.zookeeper.ZooKeeper)
[2021-01-27 13:18:56,655] INFO Client environment:java.vendor=Oracle Corpora
tion (org.apache.zookeeper.ZooKeeper)
```

```
kafka_2.12-2.7.0 — -bash — 76×23
...zookeeper.properties    ...ig/server.properties ...    ...2-2.7.0 — -bash
dehkharghanielnaz@DehkharhanisMBP ~/Desktop/kafka_2.12-2.7.0
$ bin/kafka-topics.sh --create --topic movies  --bootstrap-server localhost:
9092
Created topic movies.
dehkharghanielnaz@DehkharhanisMBP ~/Desktop/kafka_2.12-2.7.0
$
```

Figure 1.11 Kafka services running: Left Image – Zookeeper service. Middle Image – Broker service. Right Image – New topic created. All commands are in independent terminals.

Testing the Producer and Consumer:

A Kafka client communicates with the Kafka brokers via the network for writing (or reading) events. Once received, the brokers will store the events in a durable and fault-tolerant manner for as long as you need—even forever. Run the console producer client to write a few events into your topic. By default, each line you enter will result in a separate event being written to the topic.



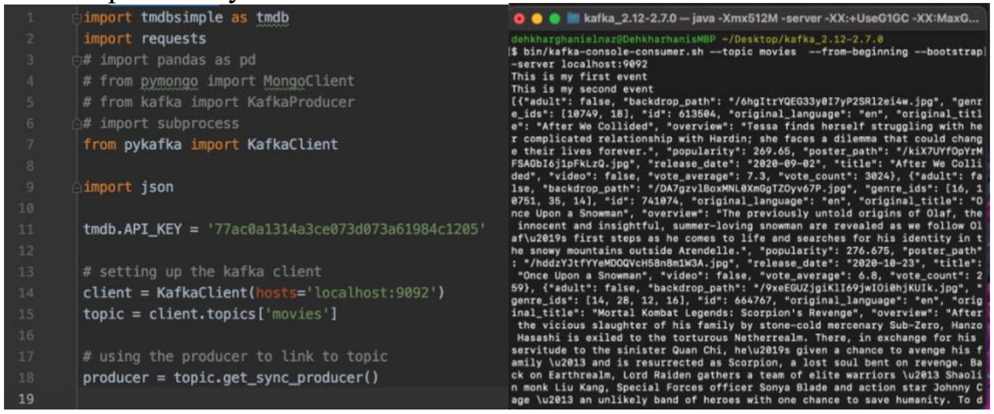
```
kafka_2.12-2.7.0 — java -Xmx512M -server -XX:+UseG1GC -XX:MaxG...
...zookeeper.properties    ...ig/server.properties ...    ...r localhost:9092
dehkharghanielnaz@DehkharhanisMBP ~/Desktop/kafka_2.12-2.7.0
$ bin/kafka-topics.sh --create --topic movies  --bootstrap-server localhost:
9092
Created topic movies.
dehkharghanielnaz@DehkharhanisMBP ~/Desktop/kafka_2.12-2.7.0
$ bin/kafka-console-producer.sh --topic movies  --bootstrap-server localhost
:9092
>This is my first event
>This is my second event
>
```

```
kafka_2.12-2.7.0 — java -Xmx512M -server -XX:+UseG1GC -XX:MaxG...
...zookeeper.properties    ...ig/server.properties ...    ...r localhost:9092
dehkharghanielnaz@DehkharhanisMBP ~/Desktop/kafka_2.12-2.7.0
$ bin/kafka-topics.sh --create --topic movies  --bootstrap-server localhost:
9092
Created topic movies.
dehkharghanielnaz@DehkharhanisMBP ~/Desktop/kafka_2.12-2.7.0
$ bin/kafka-console-producer.sh --topic movies  --bootstrap-server localhost
:9092
>This is my first event
>This is my second event
>
```

```
kafka_2.12-2.7.0 — java -Xmx512M -server -XX:+UseG1GC -XX:MaxG...
dehkharghanielnaz@DehkharhanisMBP ~/Desktop/kafka_2.12-2.7.0
$ bin/kafka-console-consumer.sh --topic movies  --from-beginning --bootstrap
-server localhost:9092
This is my first event
This is my second event
```

Figure 1.12 Example of sending and receiving a message in Kafka. Left Image – Producer. Right Image - Consumer

Now we do the same process by our movie API:

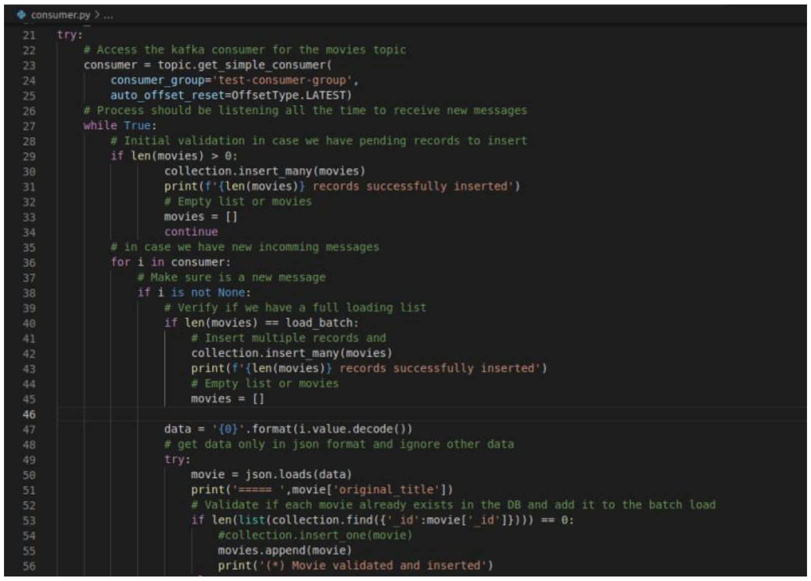


```python
import tmdbsimple as tmdb
import requests
# import pandas as pd
# from pymongo import MongoClient
# from kafka import KafkaProducer
# import subprocess
from pykafka import KafkaClient

import json

tmdb.API_KEY = '77ac0a1314a3ce073d073a61984c1205'

# setting up the kafka client
client = KafkaClient(hosts='localhost:9092')
topic = client.topics['movies']

# using the producer to link to topic
producer = topic.get_sync_producer()

```

```
kafka_2.12-2.7.0 — java -Xmx512M -server -XX:+UseG1GC -XX:MaxG...
dehkharghanielnaz@DehkharhanisMBP ~/Desktop/kafka_2.12-2.7.0
$ bin/kafka-console-consumer.sh --topic movies  --from-beginning --bootstrap
-server localhost:9092
This is my first event
This is my second event
[{"adult": false, "backdrop_path": "/6hgItrYQEG33y0I7yP2SRl2ei4w.jpg", "genr
e_ids": [10749, 18], "id": 613504, "original_language": "en", "original_titl
e": "After We Collided", "overview": "Tessa finds herself struggling with he
r complicated relationship with Hardin; she faces a dilemma that could chang
e their lives forever.", "popularity": 269.65, "poster_path": "/kiX7UYfOpYrM
FSAGbI6j1pFkLzQ.jpg", "release_date": "2020-09-02", "title": "After We Colli
ded", "video": false, "vote_average": 7.3, "vote_count": 3024}, {"adult": fa
lse, "backdrop_path": "/DA7gzvlBoxMNL0XmGgTZOyv67P.jpg", "genre_ids": [16, 1
0751, 35, 14], "id": 741074, "original_language": "en", "original_title": "O
nce Upon a Snowman", "overview": "The previously untold origins of Olaf, the
 innocent and insightful, summer-loving snowman are revealed as we follow Ol
af\u2019s first steps as he comes to life and searches for his identity in t
he snowy mountains outside Arendelle.", "popularity": 276.675, "poster_path"
: "/hddzYJtfYYeMDOQVcH58n8m1W3A.jpg", "release_date": "2020-10-23", "title":
 "Once Upon a Snowman", "video": false, "vote_average": 6.8, "vote_count": 2
59}, {"adult": false, "backdrop_path": "/9xeEGUZjgiKlI69jwIOi0hjKUIk.jpg", "
genre_ids": [14, 28, 12, 16], "id": 664767, "original_language": "en", "orig
inal_title": "Mortal Kombat Legends: Scorpion's Revenge", "overview": "After
 the vicious slaughter of his family by stone-cold mercenary Sub-Zero, Hanzo
 Hasashi is exiled to the torturous Netherrealm. There, in exchange for his
servitude to the sinister Quan Chi, he\u2019s given a chance to avenge his f
amily \u2013 and is resurrected as Scorpion, a lost soul bent on revenge. Ba
ck on Earthrealm, Lord Raiden gathers a team of elite warriors \u2013 Shaoli
n monk Liu Kang, Special Forces officer Sonya Blade and action star Johnny C
age \u2013 an unlikely band of heroes with one chance to save humanity. To d
```

Figure 1.13 Example of sending movies messages in Kafka.

After we read movies information in the Kafka consumer, we send the data directly to the Kafka Broker where the consumer performed a simple data pre-processing, including a validation in our DB to avoid duplicates.



```python
consumer.py > ...
try:
    # Access the kafka consumer for the movies topic
    consumer = topic.get_simple_consumer(
        consumer_group='test-consumer-group',
        auto_offset_reset=OffsetType.LATEST)
    # Process should be listening all the time to receive new messages
    while True:
        # Initial validation in case we have pending records to insert
        if len(movies) > 0:
                collection.insert_many(movies)
                print(f'{len(movies)} records successfully inserted')
                # Empty list or movies
                movies = []
                continue
        # in case we have new incomming messages
        for i in consumer:
            # Make sure is a new message
            if i is not None:
                # Verify if we have a full loading list
                if len(movies) == load_batch:
                    # Insert multiple records and
                    collection.insert_many(movies)
                    print(f'{len(movies)} records successfully inserted')
                    # Empty list or movies
                    movies = []

                data = '{0}'.format(i.value.decode())
                # get data only in json format and ignore other data
                try:
                    movie = json.loads(data)
                    print('===== ',movie['original_title'])
                    # Validate if each movie already exists in the DB and add it to the batch load
                    if len(list(collection.find({'_id':movie['_id']}))) == 0:
                        #collection.insert_one(movie)
                        movies.append(movie)
                        print('(*) Movie validated and inserted')
```

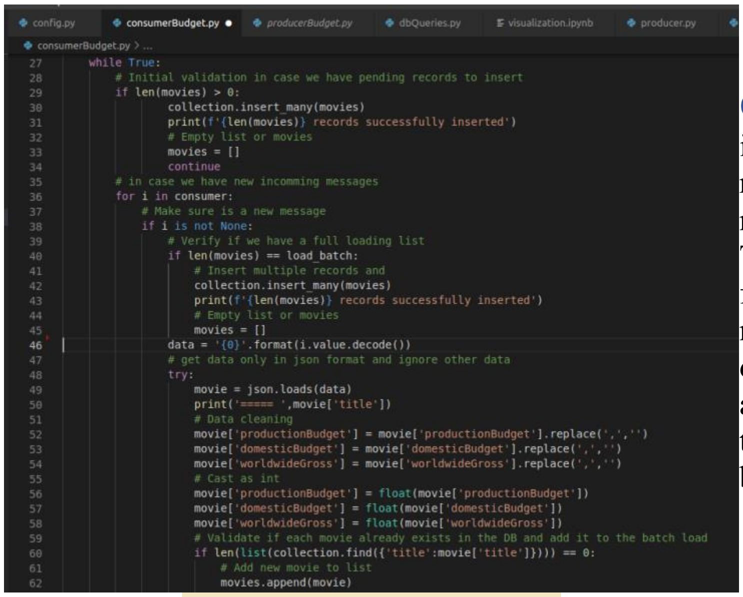
Image 1.14. Budget Consumer in python

Kafka Consumer (movies)

In order to store the “movies” topic information, a python consumer listens to all incoming messages from the producer and inserts a batch of movies into Mongo DB after validating whether that movie already exists or not. Part of the code is shown in the image below:

Kafka Consumer (budget)

In order to store the “budget” topic information, we wrote a python consumer which listens to all incoming messages from the producer and inserts a batch of movies with budgets into Mongo after a validation to determine whether that movie already exists or not. Part of the code is shown in the image below:

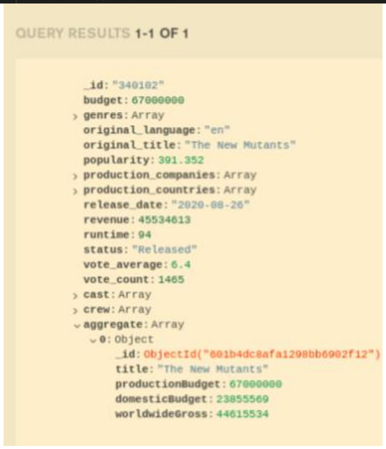


```
    while True:
        # Initial validation in case we have pending records to insert
        if len(movies) > 0:
                collection.insert_many(movies)
                print(f'{len(movies)} records successfully inserted')
                # Empty list or movies
                movies = []
                continue
        # in case we have new incomming messages
        for i in consumer:
            # Make sure is a new message
            if i is not None:
                # Verify if we have a full loading list
                if len(movies) == load_batch:
                    # Insert multiple records and
                    collection.insert_many(movies)
                    print(f'{len(movies)} records successfully inserted')
                    # Empty list or movies
                    movies = []
                data = '{0}'.format(i.value.decode())
                # get data only in json format and ignore other data
                try:
                    movie = json.loads(data)
                    print('===== ',movie['title'])
                    # Data cleaning
                    movie['productionBudget'] = movie['productionBudget'].replace(',','')
                    movie['domesticBudget'] = movie['domesticBudget'].replace(',','')
                    movie['worldwideGross'] = movie['worldwideGross'].replace(',','')
                    # Cast as int
                    movie['productionBudget'] = float(movie['productionBudget'])
                    movie['domesticBudget'] = float(movie['domesticBudget'])
                    movie['worldwideGross'] = float(movie['worldwideGross'])
                    # Validate if each movie already exists in the DB and add it to the batch load
                    if len(list(collection.find({'title':movie['title']}))) == 0:
                        # Add new movie to list
                        movies.append(movie)
```

Figure 1.15 Kafka consumer for the budget producer

Aggregated Collection

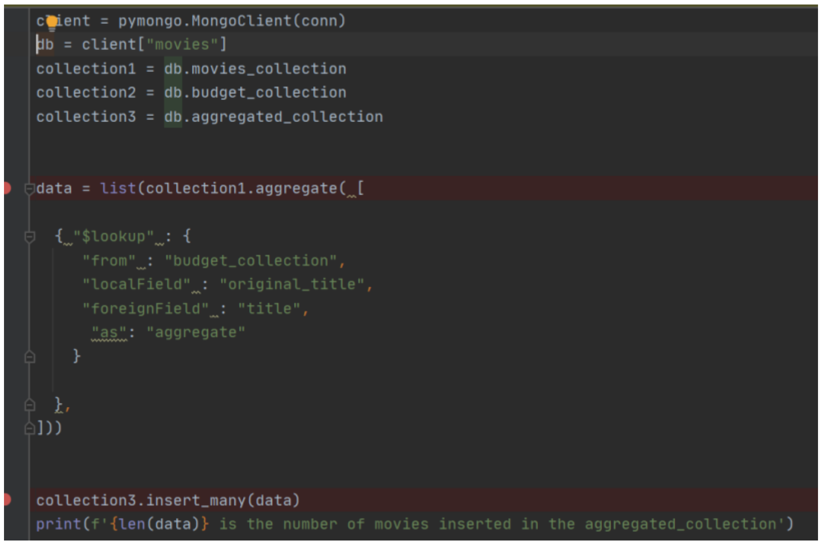
Since we have information about movies and financial results of some of them, it is possible to merge that information in another collection. This is mainly because not all the movies from the financial results appear in the movies collection. That is why we decided to create a new collection called aggregated_collection to store the data from the movies_collection joint with the budget_collection.



```
QUERY RESULTS 1-1 OF 1

    _id: "340102"
    budget: 67000000
  > genres: Array
    original_language: "en"
    original_title: "The New Mutants"
    popularity: 391.352
  > production_companies: Array
  > production_countries: Array
    release_date: "2020-08-26"
    revenue: 45534613
    runtime: 94
    status: "Released"
    vote_average: 6.4
    vote_count: 1465
  > cast: Array
  > crew: Array
  ⌄ aggregate: Array
     ⌄ 0: Object
          _id: ObjectId("601b4dc8afa1298bb6902f12")
          title: "The New Mutants"
          productionBudget: 67000000
          domesticBudget: 23855569
          worldwideGross: 44615534
```

Figure 1.16 Aggregated collection that combines data from movies and budget collection.

The python code that creates this collection is shown below: (We have applied the left join on the two collections, movies and budget_collection)



```
client = pymongo.MongoClient(conn)
db = client["movies"]
collection1 = db.movies_collection
collection2 = db.budget_collection
collection3 = db.aggregated_collection


data = list(collection1.aggregate( [

  { "$lookup" : {
      "from" : "budget_collection",
      "localField" : "original_title",
      "foreignField" : "title",
       "as": "aggregate"
    }

  },
]))

collection3.insert_many(data)
print(f'{len(data)} is the number of movies inserted in the aggregated_collection')
```

Figure 1.17 Python code to create a new aggregated collection.

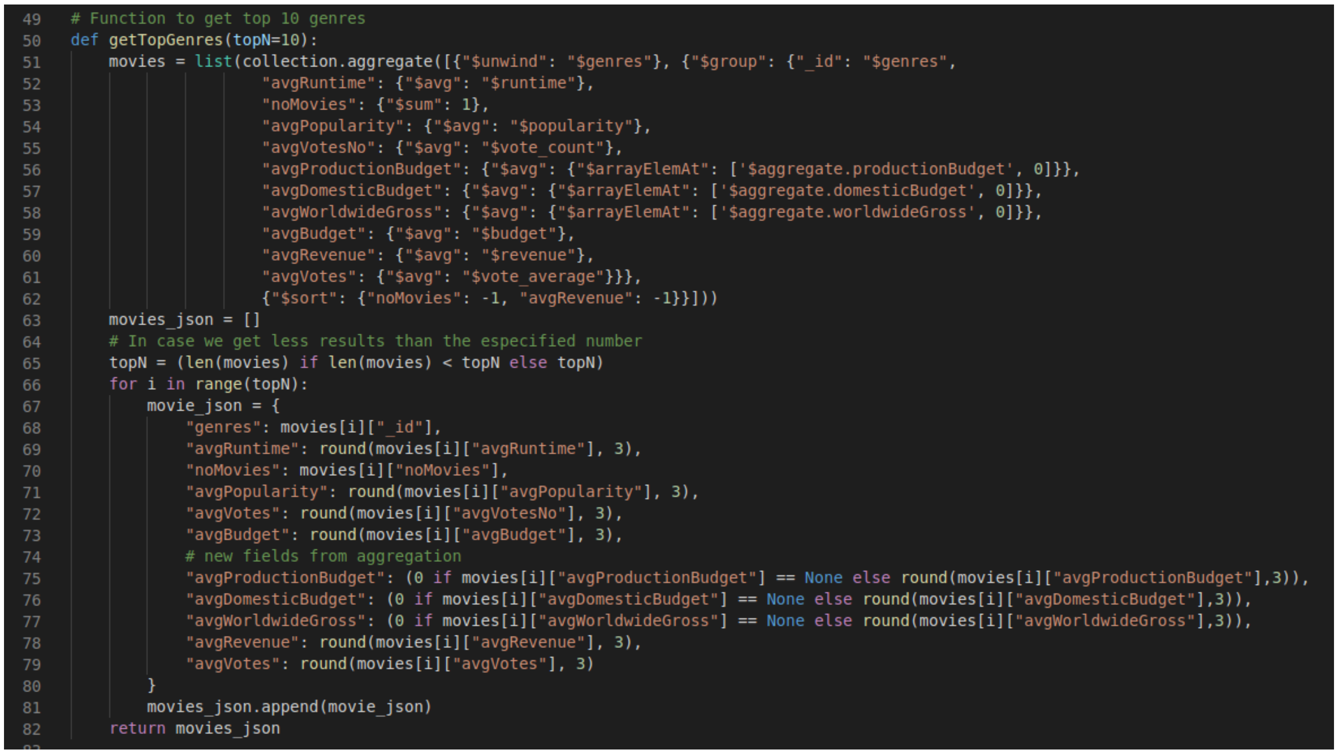
Access data in Mongo

One step before visualizing data is to define the approach of getting the data we need from Mongo. As we are using python as our main tool to manipulate data, we can build a custom API in which we define the functions we need to get the data in an appropriate format that pandas can process.

Therefore, the following functions retrieve results for only movies in English:

Custom API method	Explanation
getMoviesReleaseYear(topN=10)	Retrieves the top 10 movies based on their released year and highest vote average.
getTopGenres(topN=10)	Retrieves the top 10 genres based on the highest number of movies and highest average revenue.
getTopProductionCompanies(topN=10)	Retrieves the top 10 production companies based on the highest number of movies released and highest average revenue.
getTopMoreRevenue(topN=10)	Retrieves the top 10 movies based on their release year and highest revenue.
getTopVoteAvg(topN=10)	Retrieves top 10 movies based on their release year and highest vote average.
wholeData(topN=1000)	Retrieves the top 1,000 movies based on their release date and vote average.

An example of the python code for some of these functions is shown in the following picture:



```python
# Function to get top 10 genres
def getTopGenres(topN=10):
    movies = list(collection.aggregate([{"$unwind": "$genres"}, {"$group": {"_id": "$genres",
                                "avgRuntime": {"$avg": "$runtime"},
                                "noMovies": {"$sum": 1},
                                "avgPopularity": {"$avg": "$popularity"},
                                "avgVotesNo": {"$avg": "$vote_count"},
                                "avgProductionBudget": {"$avg": {"$arrayElemAt": ['$aggregate.productionBudget', 0]}},
                                "avgDomesticBudget": {"$avg": {"$arrayElemAt": ['$aggregate.domesticBudget', 0]}},
                                "avgWorldwideGross": {"$avg": {"$arrayElemAt": ['$aggregate.worldwideGross', 0]}},
                                "avgBudget": {"$avg": "$budget"},
                                "avgRevenue": {"$avg": "$revenue"},
                                "avgVotes": {"$avg": "$vote_average"}}},
                                {"$sort": {"noMovies": -1, "avgRevenue": -1}}]))
    movies_json = []
    # In case we get less results than the especified number
    topN = (len(movies) if len(movies) < topN else topN)
    for i in range(topN):
        movie_json = {
            "genres": movies[i]["_id"],
            "avgRuntime": round(movies[i]["avgRuntime"], 3),
            "noMovies": movies[i]["noMovies"],
            "avgPopularity": round(movies[i]["avgPopularity"], 3),
            "avgVotes": round(movies[i]["avgVotesNo"], 3),
            "avgBudget": round(movies[i]["avgBudget"], 3),
            # new fields from aggregation
            "avgProductionBudget": (0 if movies[i]["avgProductionBudget"] == None else round(movies[i]["avgProductionBudget"],3)),
            "avgDomesticBudget": (0 if movies[i]["avgDomesticBudget"] == None else round(movies[i]["avgDomesticBudget"],3)),
            "avgWorldwideGross": (0 if movies[i]["avgWorldwideGross"] == None else round(movies[i]["avgWorldwideGross"],3)),
            "avgRevenue": round(movies[i]["avgRevenue"], 3),
            "avgVotes": round(movies[i]["avgVotes"], 3)
        }
        movies_json.append(movie_json)
    return movies_json
```

Figure 1.18 Python code of getTopGenres() function. We use pymongo to access the data in our Mongo DB and perform some aggregations when needed

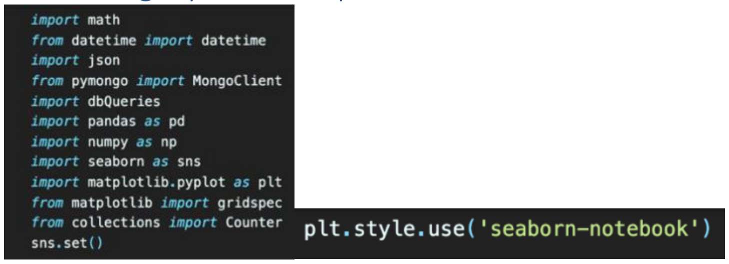
Data Visualization

In the upcoming stages we want to process our data to find the answers to the following questions:

1. Which movie has the highest and lowest profit, budget, revenue, and runtime in our dataset?
2. What are the top 10 movies in profit, budget, revenue, and runtime?
3. What are the top 30 movies in popularity?
4. What are the top 30 movies in vote_count?
6. What is the average budget, profit, popularity, etc for top 10 Genres?
7. Comparing the average Budget, Domestic Gross, Worldwide Gross for top Genre in the single plot?
8. What is the average budget, profit, popularity, etc for top 10 Companies?
9. Comparing the average Budget, Domestic Gross, Worldwide Gross for top companies in the single plot?
10. What are the key features of the movie success rate for our future modeling?

Until now, with the help of Kafka we could ingest our data from two different sources, calling API and web scraping and then store them in the Atlas MongoDB. Therefore, it is time to visualize our data to have a good insight into our collected data and find the answers to the above questions. For this purpose, we need NumPy, Pandas, Seaborn, and Matplotlib libraries. Beforehand to avoid any incorrect result in our data analysis we need to do one of the most important tasks: data cleaning. This step usually takes too much time and effort of any data scientist and most of the times it takes more than 70 percent of his time.

Importing Libraries and selecting style for our plots:



```python
import math
from datetime import datetime
import json
from pymongo import MongoClient
import dbQueries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import gridspec
from collections import Counter
sns.set()
```

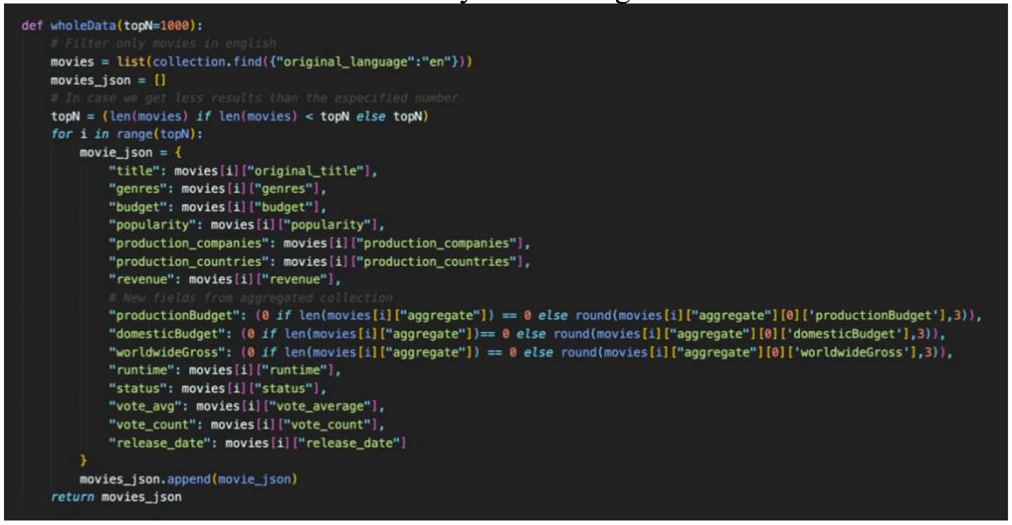
```python
plt.style.use('seaborn-notebook')
```

Figure 1.19 Elements used in the project. Left Image – python modules. Right Image – Style for plots.

First Part (Visualization – Investigating on Movies):

Loading Whole Data from Mongodb:

To have access to our MongoDB we need to write functions in a separate file called dbQueries.py which is an API to make movies data available to anyone who wants to use it. We mainly used pymongo module to make a client in order to fetch our data directly from MongoDB:



```python
def wholeData(topN=1000):
    # Filter only movies in english
    movies = list(collection.find({"original_language":"en"}))
    movies_json = []
    # In case we get less results than the especified number
    topN = (len(movies) if len(movies) < topN else topN)
    for i in range(topN):
        movie_json = {
            "title": movies[i]["original_title"],
            "genres": movies[i]["genres"],
            "budget": movies[i]["budget"],
            "popularity": movies[i]["popularity"],
            "production_companies": movies[i]["production_companies"],
            "production_countries": movies[i]["production_countries"],
            "revenue": movies[i]["revenue"],
            # New fields from aggregated collection
            "productionBudget": (0 if len(movies[i]["aggregate"]) == 0 else round(movies[i]["aggregate"][0]['productionBudget'],3)),
            "domesticBudget": (0 if len(movies[i]["aggregate"])== 0 else round(movies[i]["aggregate"][0]['domesticBudget'],3)),
            "worldwideGross": (0 if len(movies[i]["aggregate"]) == 0 else round(movies[i]["aggregate"][0]['worldwideGross'],3)),
            "runtime": movies[i]["runtime"],
            "status": movies[i]["status"],
            "vote_avg": movies[i]["vote_average"],
            "vote_count": movies[i]["vote_count"],
            "release_date": movies[i]["release_date"]
        }
        movies_json.append(movie_json)
    return movies_json
```

Figure 1.20 - Whole Data API code implementation.

The wholeData function has access to 1,000 of our aggregated data (aggregated_collection). These data are aggregated from two other collections by "Left Join" of (budget_collection and movies_collection) in the Atlas MongoDB. For our analysis, we are only considering movies in English language.

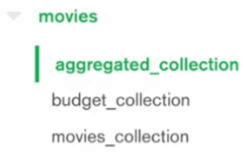


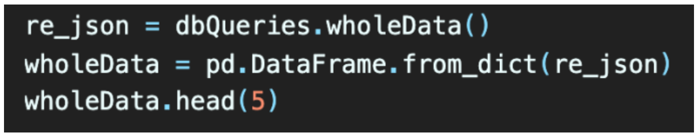
Figure 1.21 - MongoDb collections

Before going ahead, we need to know our data set:

title: The Official Title of the movie.
genres: A stringified list of dictionaries that list out all the genres associated with the movie.
budget: The budget of the movie in dollars.
popularity: The Popularity Score assigned by TMDB.
production_companies: A stringified list of production companies involved with the making of the movie.
production_countries: A stringified list of countries where the movie was shot/produced in.
revenue: The total revenue of the movie in dollars.

Production budget: The budget of the movie in dollars. (Added by web scraping from another source)
Domestic Gross: The total revenue of the movie in dollars in domestic area. (Added by web scraping from another source)
Worldwide Gross: The total revenue of the movie in dollars in worldwide. (Added by web scraping from another source)
runtime: The runtime of the movie in minutes.
status: The status of the movie (Released, To Be Released, Announced, etc.)
vote_average: The average rating of the movie.
release_date: Theatrical Release Date of the movie.

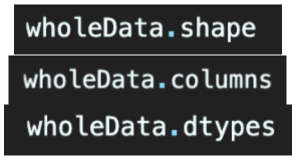
Now we import and call this wholedata function in our Jupyter notebook for further analysis, and then convert our JSON data to a data Frame object. In case to have an initial insight and make sure that our data is loaded properly we use the head() function.



```
re_json = dbQueries.wholeData()
wholeData = pd.DataFrame.from_dict(re_json)
wholeData.head(5)
```

Figure 1.22 - Transform API result into dataframe in python

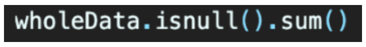
Before doing any data cleaning we would have a quick look at the number of rows, columns, column names, and their data types.



```
wholeData.shape
wholeData.columns
wholeData.dtypes
```

Figure 1.23 - python commands to understand the data structure

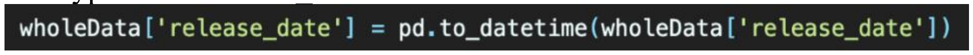
To check how many rows have null values we use this function.



```
wholeData.isnull().sum()
```

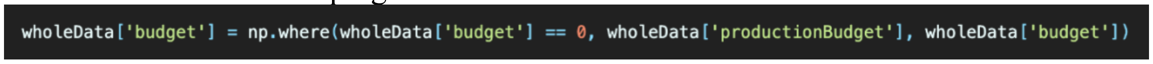
Data cleaning:

Changing the data type of the release_date column to datetime format.



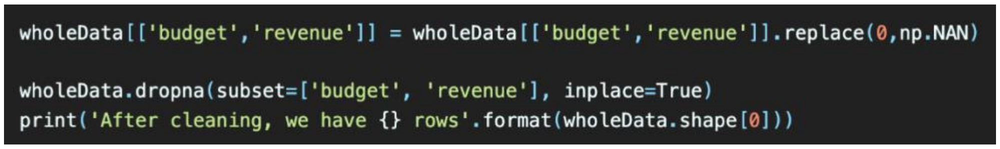
```
wholeData['release_date'] = pd.to_datetime(wholeData['release_date'])
```

Since some values in the budget column are missing, we can fill them with the productionBudget column that is fetched from the web scraping source.



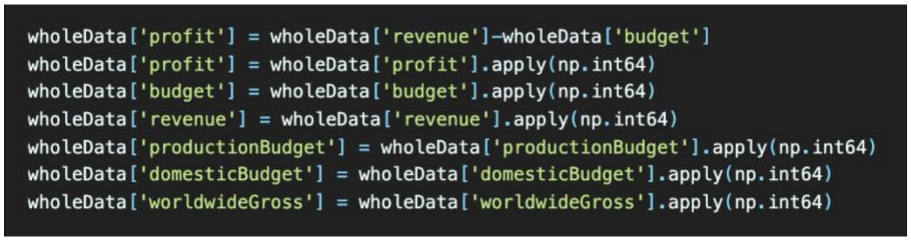
```
wholeData['budget'] = np.where(wholeData['budget'] == 0, wholeData['productionBudget'], wholeData['budget'])
```

After that, if still there are some missing values in the rows, we replace those 0 values in both revenue and budget columns with NaN values and then drop them.



```
wholeData[['budget','revenue']] = wholeData[['budget','revenue']].replace(0,np.NAN)

wholeData.dropna(subset=['budget', 'revenue'], inplace=True)
print('After cleaning, we have {} rows'.format(wholeData.shape[0]))
```

In this step for further analysis, we need to add a new column as Profit for each of the movies. We do this by subtracting the revenue column by budget column. Also, to change the datatype of some columns to int64 we apply the Numpy.int64 function.

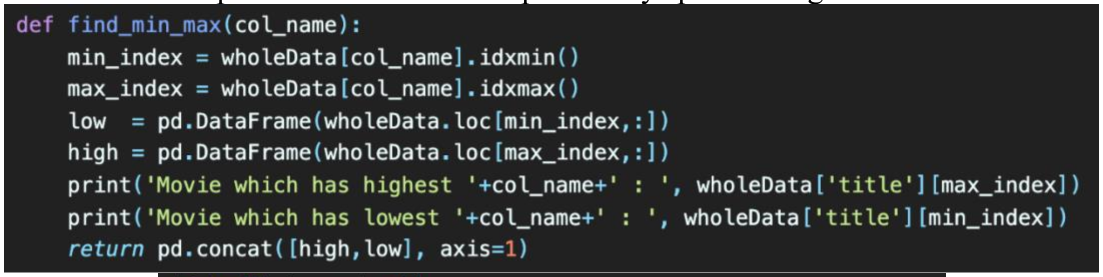


```
wholeData['profit'] = wholeData['revenue']-wholeData['budget']
wholeData['profit'] = wholeData['profit'].apply(np.int64)
wholeData['budget'] = wholeData['budget'].apply(np.int64)
wholeData['revenue'] = wholeData['revenue'].apply(np.int64)
wholeData['productionBudget'] = wholeData['productionBudget'].apply(np.int64)
wholeData['domesticBudget'] = wholeData['domesticBudget'].apply(np.int64)
wholeData['worldwideGross'] = wholeData['worldwideGross'].apply(np.int64)
```

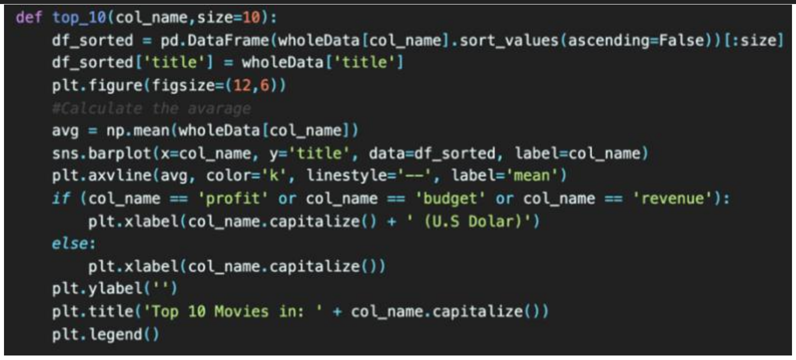
Data Analysis:

By using the describe() function, we have a quick summary of statistics related to the DataFrame columns. This function gives the mean, std and IQR values.

Before going ahead, we need to define two different functions: one for finding the minimum and maximum values of a given column and the other for finding the first top ten for that specific column name. In the first function by idxmin() and idxmax() we get the id of the minimum and maximum values in the column. After that by loc[] we can filter our data frame to catch only these two rows by all their columns then we concatenate them together in a single data frame. For the second function we need to sort the values in the descending order and just define a size as given parameter to catch results. Also, with Numpy we applied the mean function to demonstrate where the mean value is in the x-axis. At the end with seaborn we will plot the result with that previously specified figure size.



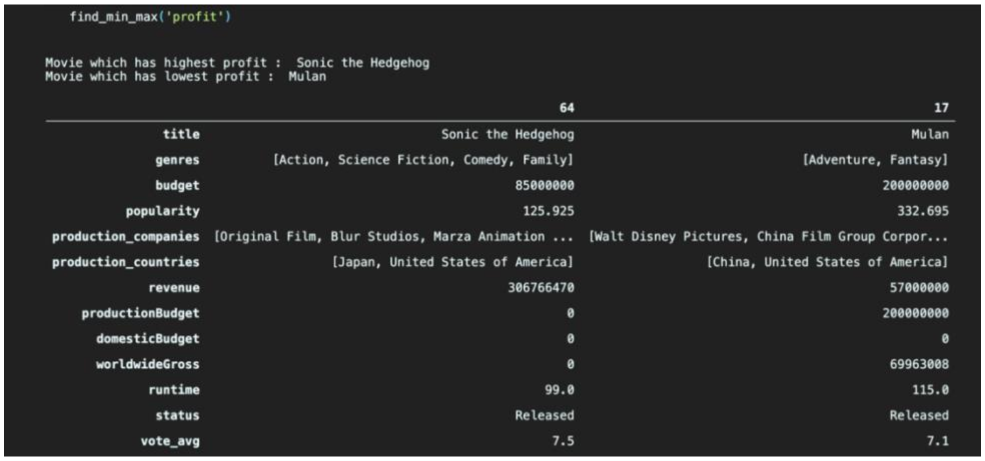
```
def find_min_max(col_name):
    min_index = wholeData[col_name].idxmin()
    max_index = wholeData[col_name].idxmax()
    low  = pd.DataFrame(wholeData.loc[min_index,:])
    high = pd.DataFrame(wholeData.loc[max_index,:])
    print('Movie which has highest '+col_name+' : ', wholeData['title'][max_index])
    print('Movie which has lowest '+col_name+' : ', wholeData['title'][min_index])
    return pd.concat([high,low], axis=1)
```



```
def top_10(col_name,size=10):
    df_sorted = pd.DataFrame(wholeData[col_name].sort_values(ascending=False))[:size]
    df_sorted['title'] = wholeData['title']
    plt.figure(figsize=(12,6))
    #Calculate the avarage
    avg = np.mean(wholeData[col_name])
    sns.barplot(x=col_name, y='title', data=df_sorted, label=col_name)
    plt.axvline(avg, color='k', linestyle='--', label='mean')
    if (col_name == 'profit' or col_name == 'budget' or col_name == 'revenue'):
        plt.xlabel(col_name.capitalize() + ' (U.S Dolar)')
    else:
        plt.xlabel(col_name.capitalize())
    plt.ylabel('')
    plt.title('Top 10 Movies in: ' + col_name.capitalize())
    plt.legend()
```

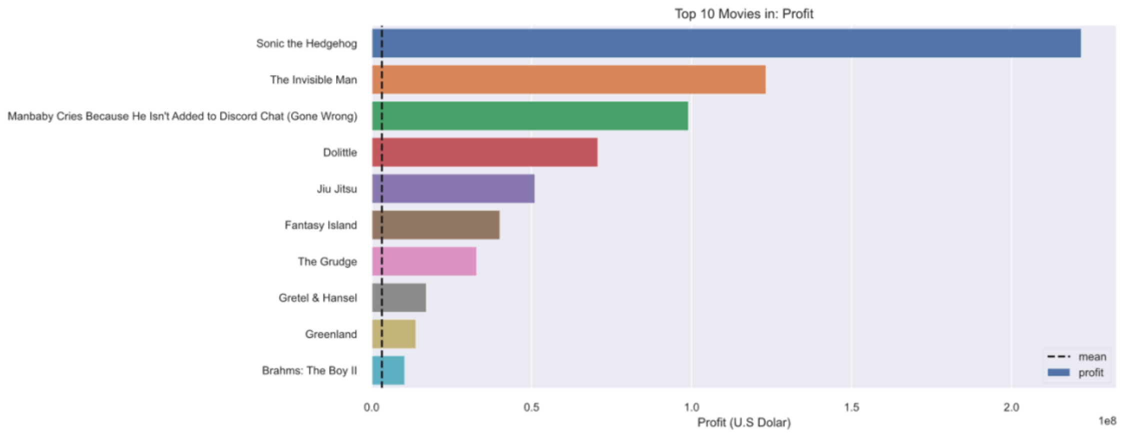
We will apply these two functions on the following metrics: profit, budget, revenue, and runtime and the illustrated results will be as below. Since we are fetching data in a real time streaming by Kafka messenger through MovieDB API, results will be changed at any time that the program is running.

Results for the movies with highest and lowest Profit:

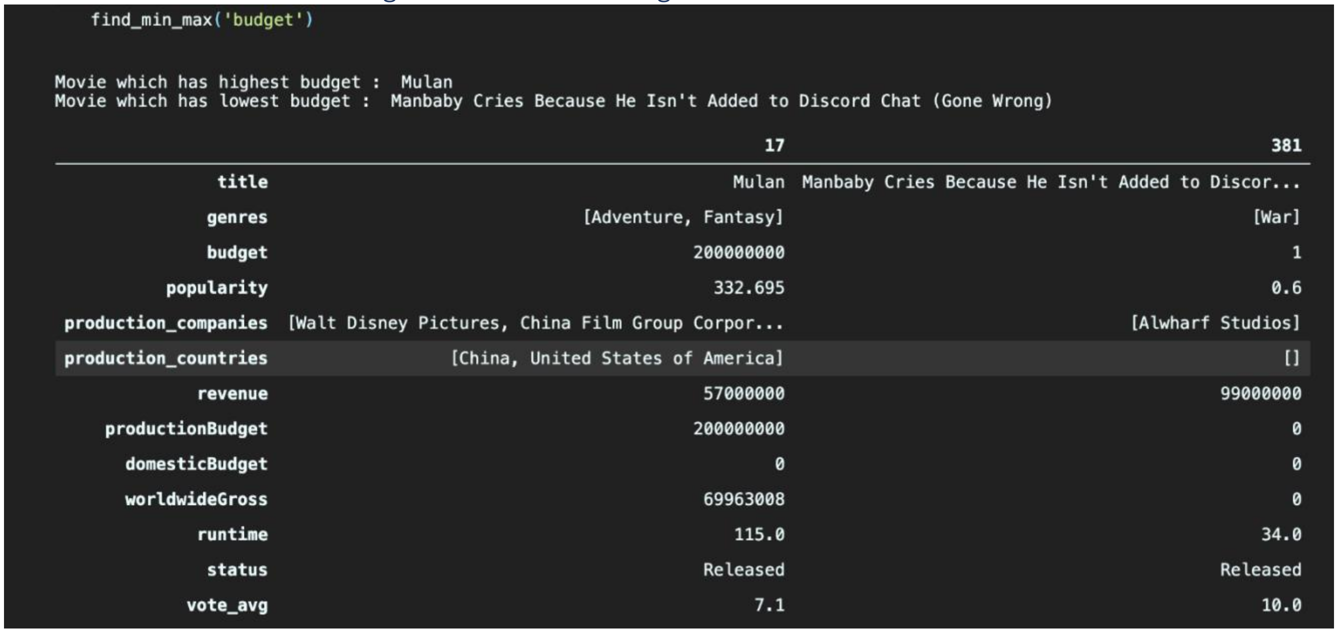


```
find_min_max('profit')

Movie which has highest profit :  Sonic the Hedgehog
Movie which has lowest profit :  Mulan

                                                          64                                        17
                  title                    Sonic the Hedgehog                                     Mulan
                 genres   [Action, Science Fiction, Comedy, Family]                  [Adventure, Fantasy]
                 budget                              85000000                                 200000000
             popularity                               125.925                                   332.695
   production_companies  [Original Film, Blur Studios, Marza Animation ...  [Walt Disney Pictures, China Film Group Corpor...
   production_countries              [Japan, United States of America]     [China, United States of America]
                revenue                             306766470                                  57000000
       productionBudget                                     0                                 200000000
         domesticBudget                                     0                                         0
         worldwideGross                                     0                                  69963008
                runtime                                  99.0                                     115.0
                 status                              Released                                  Released
               vote_avg                                   7.5                                       7.1
```

Results for the top 10 movies in Profit:

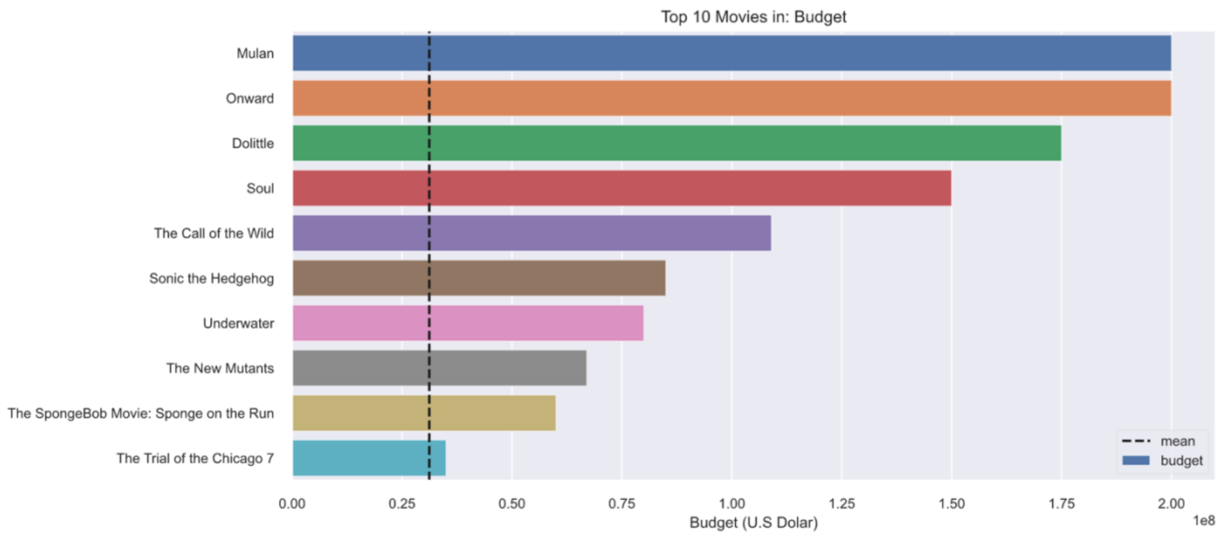


Results for the movies with highest and lowest Budget:

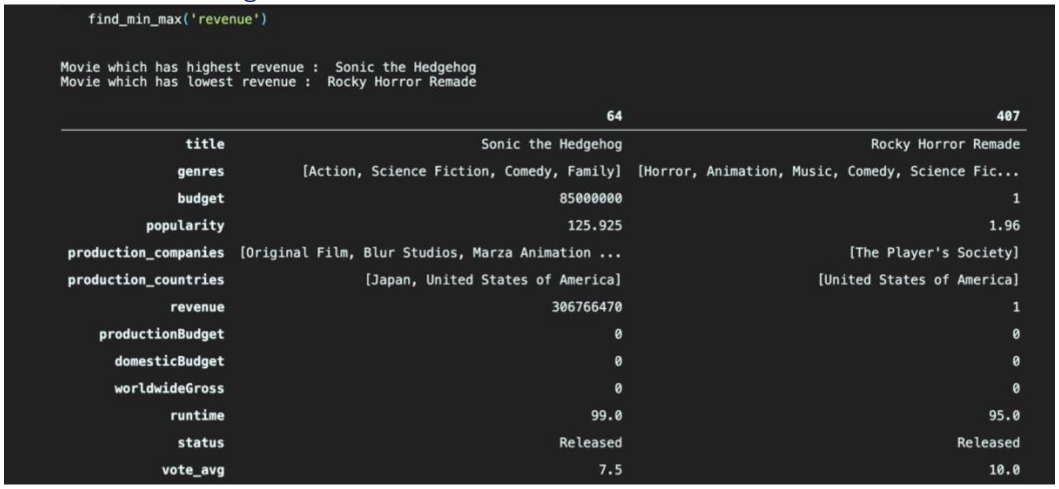


```
find_min_max('budget')

Movie which has highest budget :  Mulan
Movie which has lowest budget :  Manbaby Cries Because He Isn't Added to Discord Chat (Gone Wrong)

                                                            17                                          381
                  title                                  Mulan  Manbaby Cries Because He Isn't Added to Discor...
                 genres                   [Adventure, Fantasy]                                          [War]
                 budget                              200000000                                              1
             popularity                                332.695                                            0.6
   production_companies  [Walt Disney Pictures, China Film Group Corpor...                    [Alwharf Studios]
   production_countries              [China, United States of America]                                     []
                revenue                               57000000                                       99000000
       productionBudget                              200000000                                              0
         domesticBudget                                      0                                              0
         worldwideGross                               69963008                                              0
                runtime                                  115.0                                           34.0
                 status                               Released                                       Released
               vote_avg                                    7.1                                           10.0
```

Results for the top 10 movies in Budget:

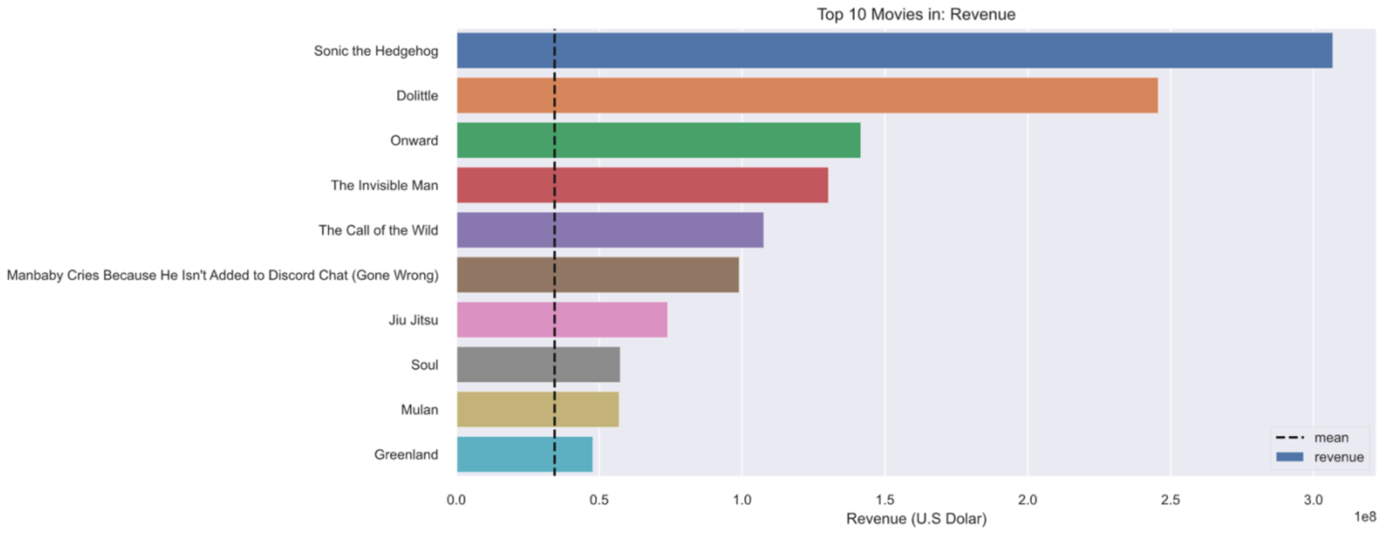


Results for the movies with highest and lowest Revenue:

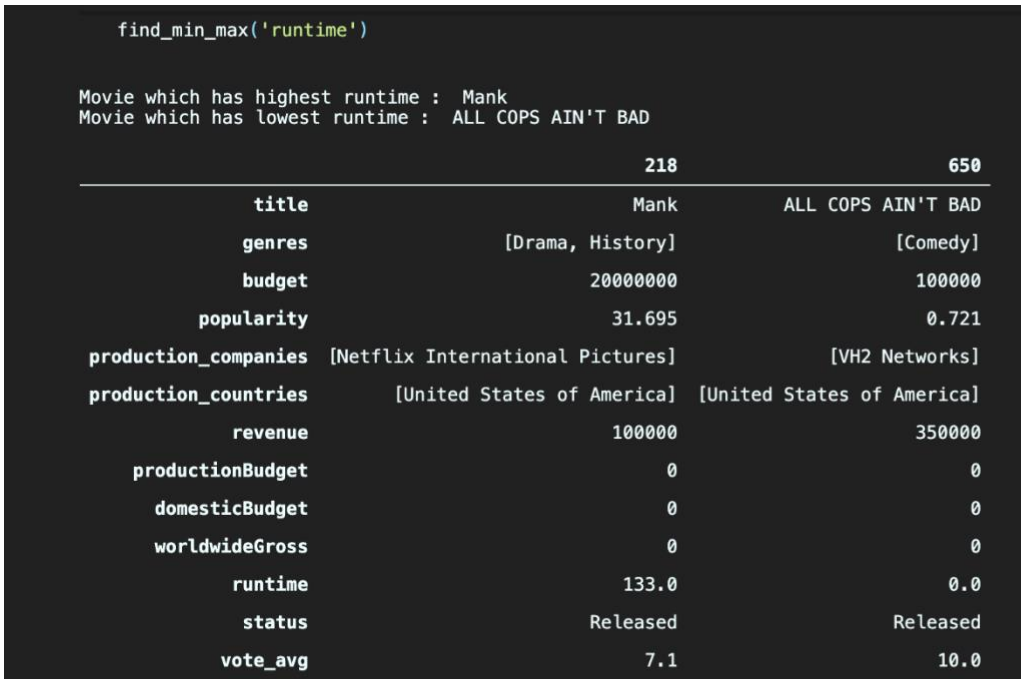


```
    find_min_max('revenue')


Movie which has highest revenue :  Sonic the Hedgehog
Movie which has lowest revenue :  Rocky Horror Remade

                                                          64                                                   407
title                                      Sonic the Hedgehog                                  Rocky Horror Remade
genres                [Action, Science Fiction, Comedy, Family]  [Horror, Animation, Music, Comedy, Science Fic...
budget                                               85000000                                                    1
popularity                                            125.925                                                 1.96
production_companies  [Original Film, Blur Studios, Marza Animation ...                     [The Player's Society]
production_countries               [Japan, United States of America]                    [United States of America]
revenue                                             306766470                                                    1
productionBudget                                            0                                                    0
domesticBudget                                              0                                                    0
worldwideGross                                              0                                                    0
runtime                                                  99.0                                                 95.0
status                                               Released                                             Released
vote_avg                                                  7.5                                                 10.0
```

Results for the top 10 movies in Revenue:

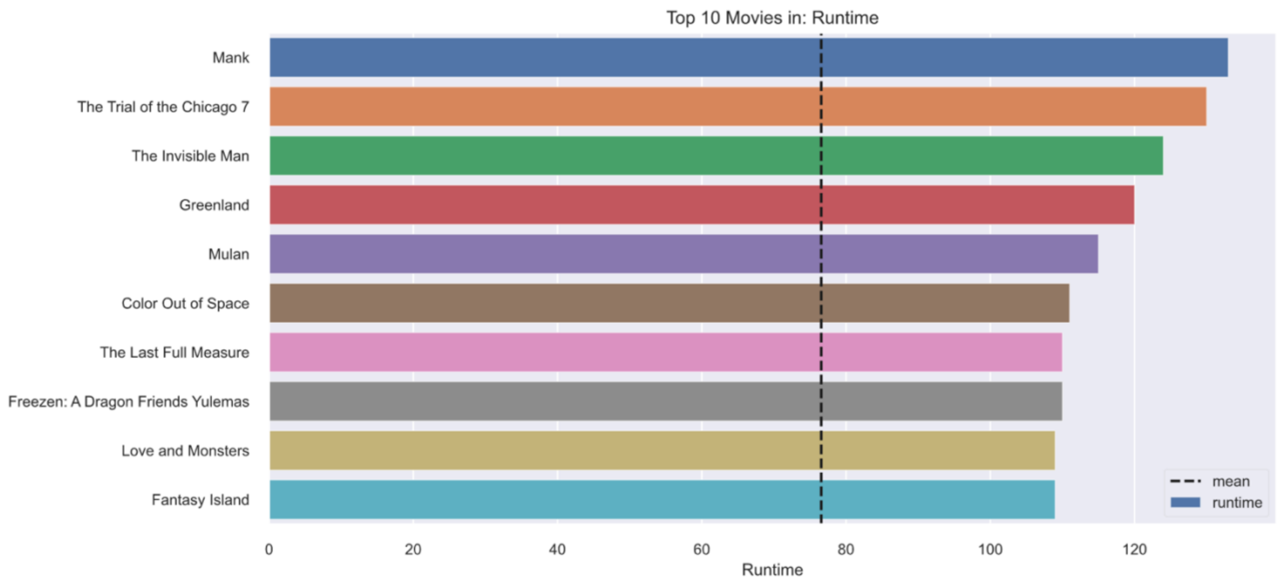


Results for the movies with highest and lowest Runtime:

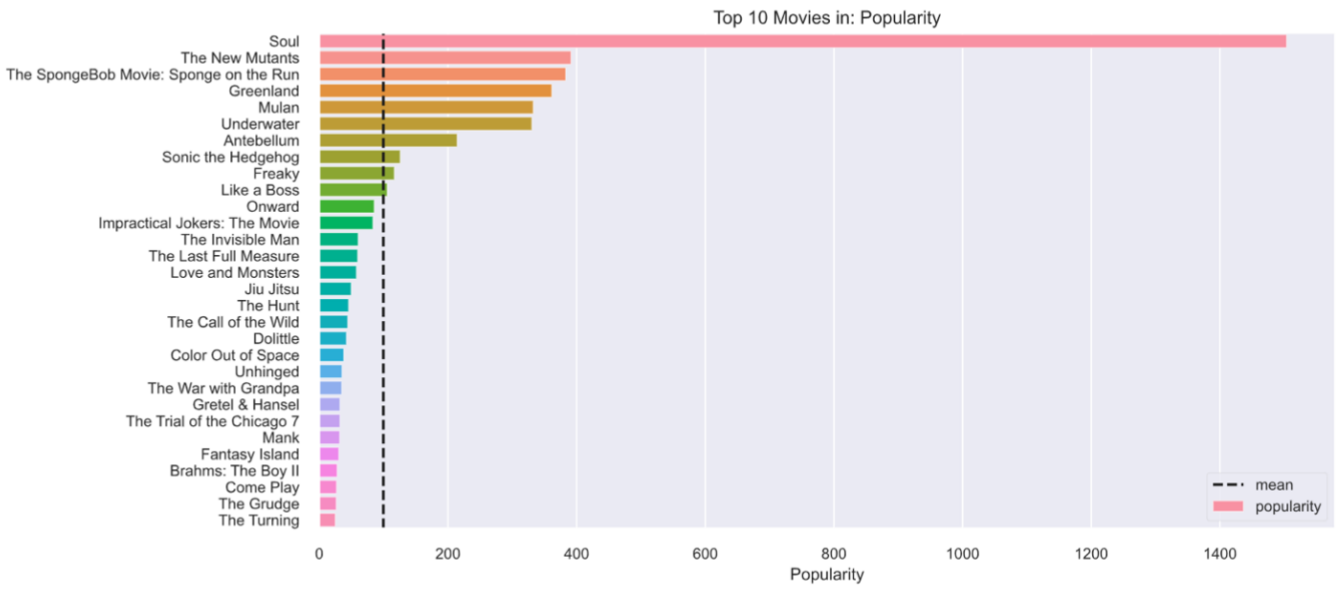


```
   find_min_max('runtime')

Movie which has highest runtime :  Mank
Movie which has lowest runtime :  ALL COPS AIN'T BAD

                                            218                  650
                   title                   Mank   ALL COPS AIN'T BAD
                  genres      [Drama, History]             [Comedy]
                  budget               20000000               100000
              popularity                 31.695                0.721
    production_companies [Netflix International Pictures]   [VH2 Networks]
    production_countries [United States of America] [United States of America]
                 revenue                 100000               350000
        productionBudget                      0                    0
          domesticBudget                      0                    0
          worldwideGross                      0                    0
                 runtime                  133.0                  0.0
                  status               Released             Released
                vote_avg                    7.1                 10.0
```

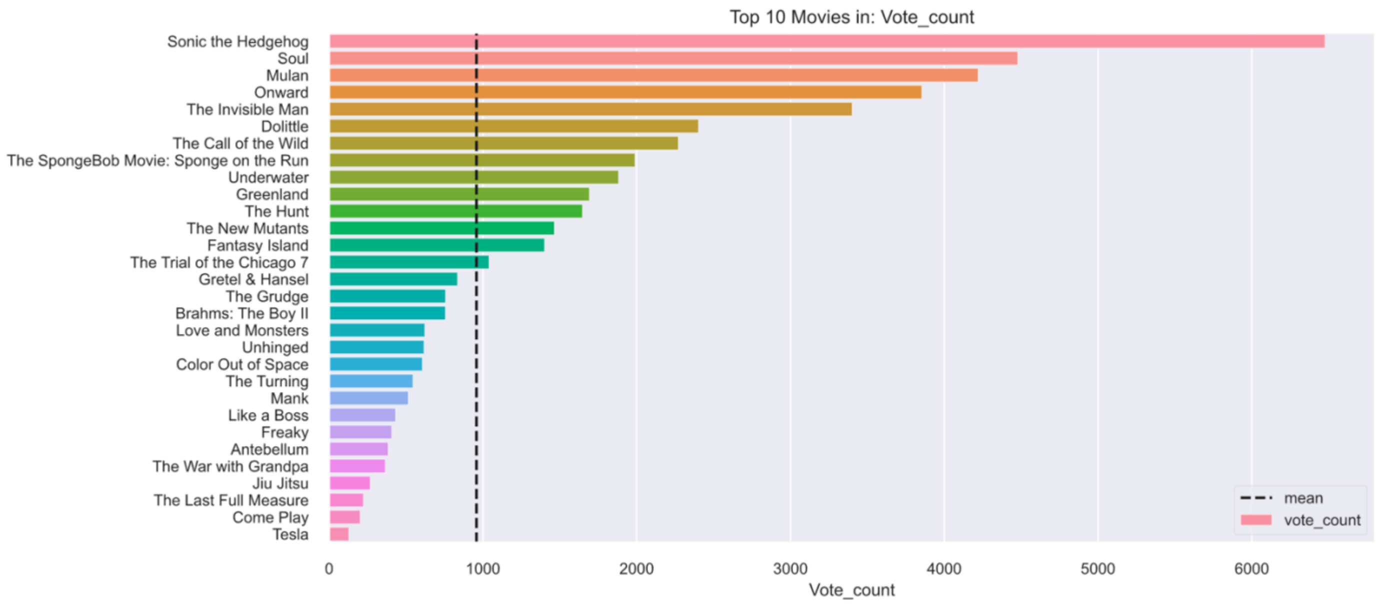
Results for the top 10 movies in Runtime:



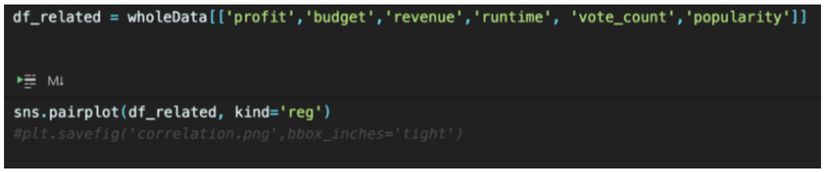
Results for the first top 30 movies in Popularity:



Results for the first top 30 movies in vote_count:

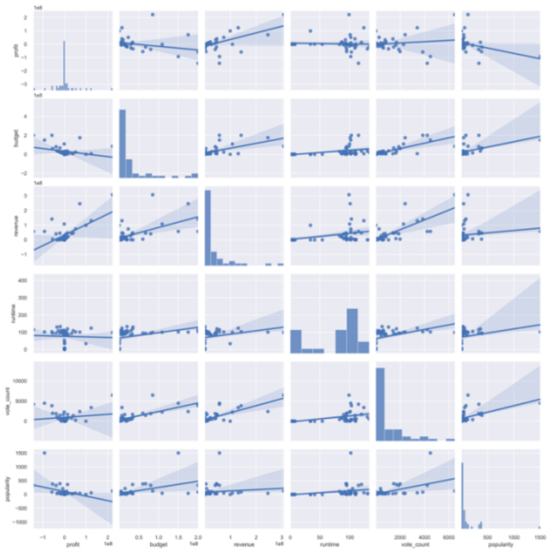


For plotting the pairwise relationships between given columns in a dataset, we have applied the seaborn.paiplot() function and gave a parameter as kind: reg to have regression plots.

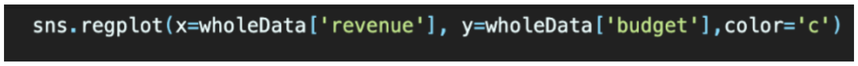


```
df_related = wholeData[['profit','budget','revenue','runtime', 'vote_count','popularity']]
```

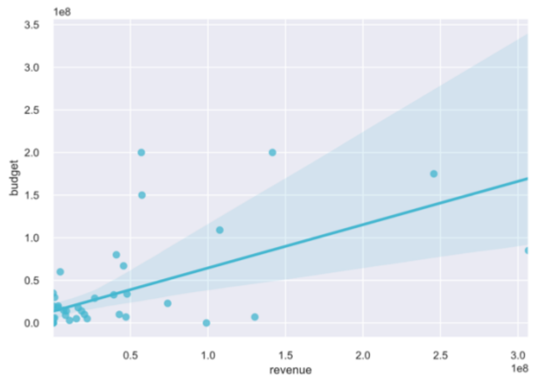
```
sns.pairplot(df_related, kind='reg')
#plt.savefig('correlation.png',bbox_inches='tight')
```



To visualize only one of these plots here for budget, revenue in a single figure we used this regplot() function of seaborn.



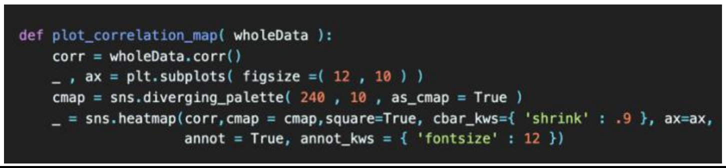
```
sns.regplot(x=wholeData['revenue'], y=wholeData['budget'],color='c')
```



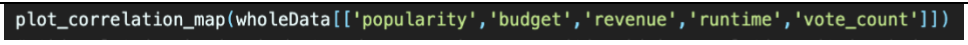
Results for key features of success rate a movie:

We defined a function to plot a heatmap by finding the correlation between given variables here as our columns of the data frame. A correlation plot used to investigate the dependency between multiple variables at the same time and to highlight the most correlated variables in a data frame.

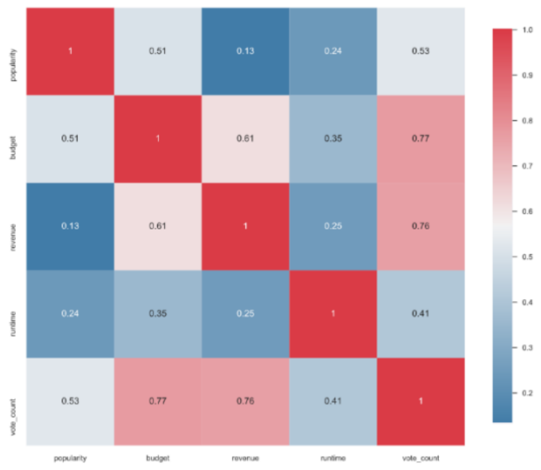
# sns.diverging_palette: Make a diverging palette between two HUSL colors.
# as_cmapbool: If True, return a matplotlib.colors.Colormap.
# Sns.heatmap: Plot rectangular data as a color-encoded matrix.
# cbar: Whether to draw a colorbar.
# ax: Axes in which to draw the plot, otherwise use the currently-active Axes
# annot: If True, write the data value in each cell.
# annot_kws: Keyword arguments for matplotlib.axes.Axes.text() when annot is True.
# Square: If True, set the Axes aspect to "equal" so each cell will be square-shaped.



```
def plot_correlation_map( wholeData ):
    corr = wholeData.corr()
    _ , ax = plt.subplots( figsize =( 12 , 10 ) )
    cmap = sns.diverging_palette( 240 , 10 , as_cmap = True )
    _ = sns.heatmap(corr,cmap = cmap,square=True, cbar_kws={ 'shrink' : .9 }, ax=ax,
                    annot = True, annot_kws = { 'fontsize' : 12 })
```



```
plot_correlation_map(wholeData[['popularity','budget','revenue','runtime','vote_count']])
```



	popularity	budget	revenue	runtime	vote_count
popularity	1	0.51	0.13	0.24	0.53
budget	0.51	1	0.61	0.35	0.77
revenue	0.13	0.61	1	0.25	0.76
runtime	0.24	0.35	0.25	1	0.41
vote_count	0.53	0.77	0.76	0.41	1

Interpretation of this plot:

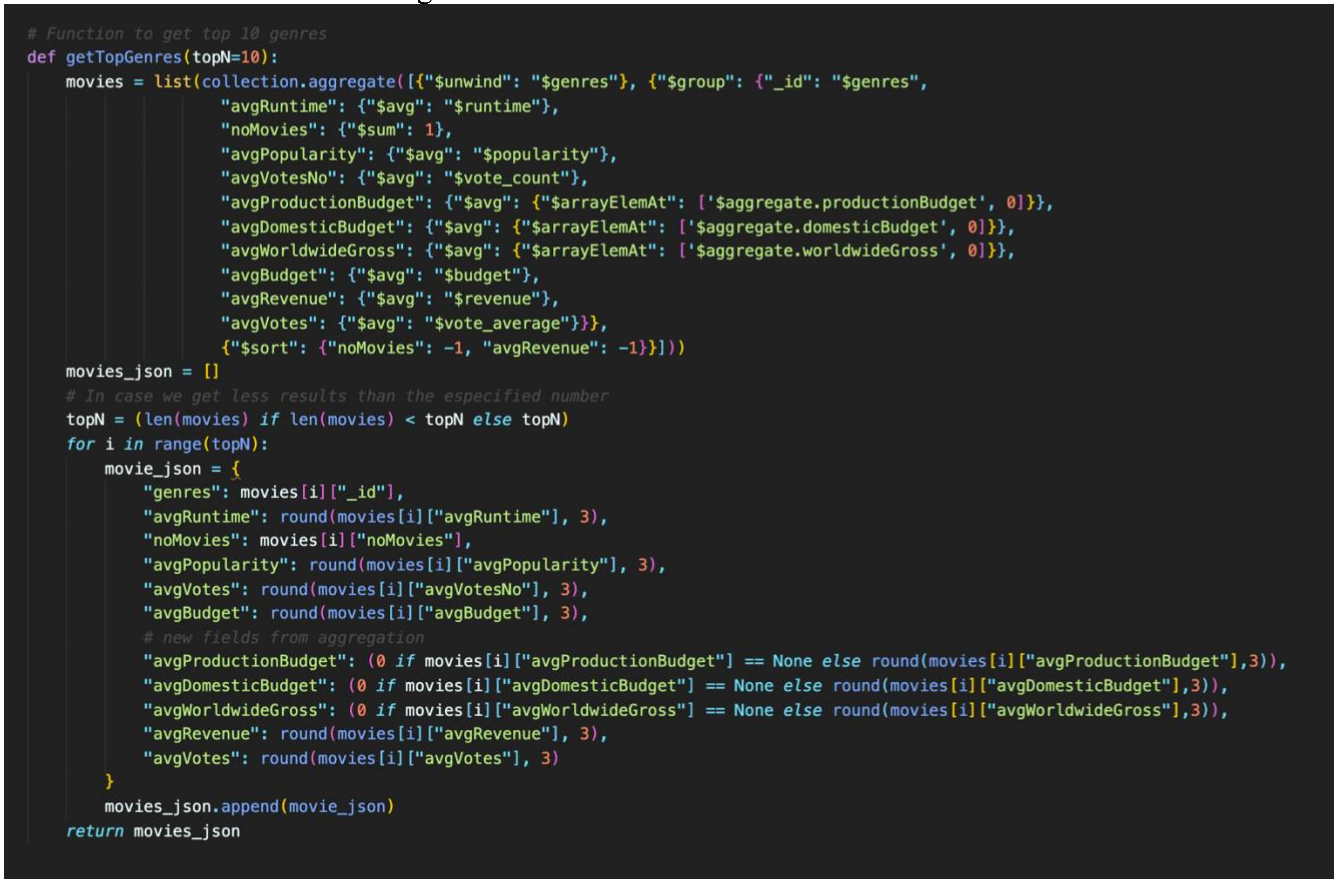
If we have a Negative correlation it means that features increase or decrease in values in opposite directions on a number line. If one feature increases, the other feature decreases. But on the other hand, Positive correlation means features decrease or increase in values together. If two features have a correlation of one, then the features are duplicates.

As an example, since this plot is showing budget and revenue or budget and vote counts have a positive high correlation, it is obvious that with more investment would have more revenues and vote counts. A higher correlation of features with the target feature indicates the possible higher importance of the feature for any future modeling.

Second Part: (Visualization – Investigating on Genres):

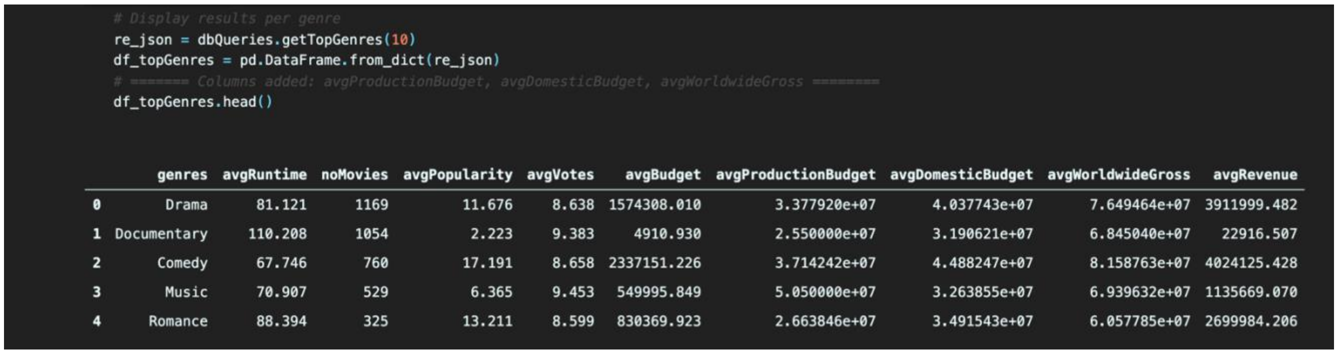
Loading Only Ten Top Genre From MongoDB:

Our API can return the top ten genres from our MongoDB collection. The method works as follows: Since the genres for each movie are stored as an array first, we need to apply $unwind to deconstruct them. After that with $group we can group these genres by their ids. For the other values in the document, we calculate their averages and finally sort them base on the number of movies and average revenues, we iterate until to find these ten genres and the result will be saved in a list:



```python
# Function to get top 10 genres
def getTopGenres(topN=10):
    movies = list(collection.aggregate([{"$unwind": "$genres"}, {"$group": {"_id": "$genres",
                "avgRuntime": {"$avg": "$runtime"},
                "noMovies": {"$sum": 1},
                "avgPopularity": {"$avg": "$popularity"},
                "avgVotesNo": {"$avg": "$vote_count"},
                "avgProductionBudget": {"$avg": {"$arrayElemAt": ['$aggregate.productionBudget', 0]}},
                "avgDomesticBudget": {"$avg": {"$arrayElemAt": ['$aggregate.domesticBudget', 0]}},
                "avgWorldwideGross": {"$avg": {"$arrayElemAt": ['$aggregate.worldwideGross', 0]}},
                "avgBudget": {"$avg": "$budget"},
                "avgRevenue": {"$avg": "$revenue"},
                "avgVotes": {"$avg": "$vote_average"}}},
                {"$sort": {"noMovies": -1, "avgRevenue": -1}}]))
    movies_json = []
    # In case we get less results than the especified number
    topN = (len(movies) if len(movies) < topN else topN)
    for i in range(topN):
        movie_json = {
            "genres": movies[i]["_id"],
            "avgRuntime": round(movies[i]["avgRuntime"], 3),
            "noMovies": movies[i]["noMovies"],
            "avgPopularity": round(movies[i]["avgPopularity"], 3),
            "avgVotes": round(movies[i]["avgVotesNo"], 3),
            "avgBudget": round(movies[i]["avgBudget"], 3),
            # new fields from aggregation
            "avgProductionBudget": (0 if movies[i]["avgProductionBudget"] == None else round(movies[i]["avgProductionBudget"],3)),
            "avgDomesticBudget": (0 if movies[i]["avgDomesticBudget"] == None else round(movies[i]["avgDomesticBudget"],3)),
            "avgWorldwideGross": (0 if movies[i]["avgWorldwideGross"] == None else round(movies[i]["avgWorldwideGross"],3)),
            "avgRevenue": round(movies[i]["avgRevenue"], 3),
            "avgVotes": round(movies[i]["avgVotes"], 3)
        }
        movies_json.append(movie_json)
    return movies_json
```

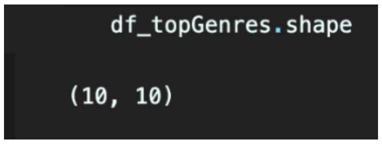
Now with calling this function and applying the head() method we would be sure of our data loading:



```
# Display results per genre
re_json = dbQueries.getTopGenres(10)
df_topGenres = pd.DataFrame.from_dict(re_json)
# ======= Columns added: avgProductionBudget, avgDomesticBudget, avgWorldwideGross ========
df_topGenres.head()
```

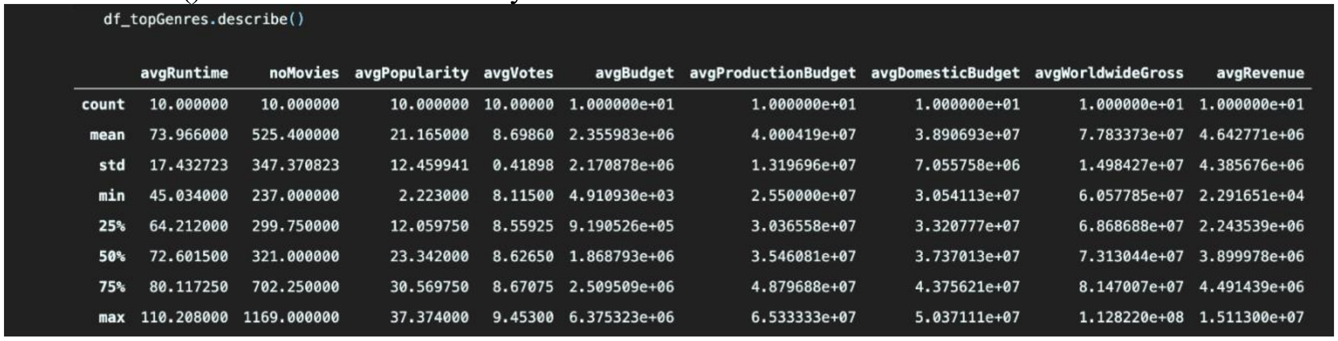
```
        genres  avgRuntime  noMovies  avgPopularity  avgVotes    avgBudget  avgProductionBudget  avgDomesticBudget  avgWorldwideGross   avgRevenue
0        Drama      81.121      1169         11.676     8.638  1574308.010         3.377920e+07       4.037743e+07       7.649464e+07  3911999.482
1  Documentary     110.208      1054          2.223     9.383     4910.930         2.550000e+07       3.190621e+07       6.845040e+07    22916.507
2       Comedy      67.746       760         17.191     8.658  2337151.226         3.714242e+07       4.488247e+07       8.158763e+07  4024125.428
3        Music      70.907       529          6.365     9.453   549995.849         5.050000e+07       3.263855e+07       6.939632e+07  1135669.070
4      Romance      88.394       325         13.211     8.599   830369.923         2.663846e+07       3.491543e+07       6.057785e+07  2699984.206
```

We see here number of columns and rows:



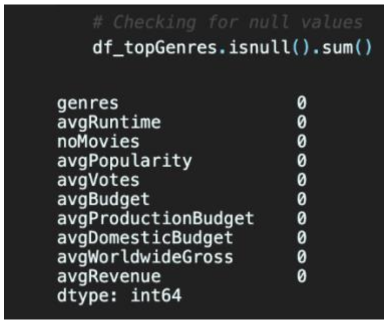
```
df_topGenres.shape

(10, 10)
```

With describe() function find a summary of statistics.



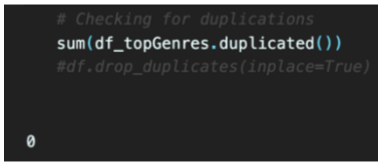
```
df_topGenres.describe()

       avgRuntime     noMovies  avgPopularity  avgVotes     avgBudget  avgProductionBudget  avgDomesticBudget  avgWorldwideGross    avgRevenue
count   10.000000    10.000000      10.000000  10.00000  1.000000e+01         1.000000e+01       1.000000e+01       1.000000e+01  1.000000e+01
mean    73.966000   525.400000      21.165000   8.69860  2.355983e+06         4.000419e+07       3.890693e+07       7.783373e+07  4.642771e+06
std     17.432723   347.370823      12.459941   0.41898  2.170878e+06         1.319696e+07       7.055758e+06       1.498427e+07  4.385676e+06
min     45.034000   237.000000       2.223000   8.11500  4.910930e+03         2.550000e+07       3.054113e+07       6.057785e+07  2.291651e+04
25%     64.212000   299.750000      12.059750   8.55925  9.190526e+05         3.036558e+07       3.320777e+07       6.868688e+07  2.243539e+06
50%     72.601500   321.000000      23.342000   8.62650  1.868793e+06         3.546081e+07       3.737013e+07       7.313044e+07  3.899978e+06
75%     80.117250   702.250000      30.569750   8.67075  2.509509e+06         4.879688e+07       4.375621e+07       8.147007e+07  4.491439e+06
max    110.208000  1169.000000      37.374000   9.45300  6.375323e+06         6.533333e+07       5.037111e+07       1.128220e+08  1.511300e+07
```

Calculate number of null values:



```
# Checking for null values
df_topGenres.isnull().sum()

genres                 0
avgRuntime             0
noMovies               0
avgPopularity          0
avgVotes               0
avgBudget              0
avgProductionBudget    0
avgDomesticBudget      0
avgWorldwideGross      0
avgRevenue             0
dtype: int64
```

Checking for any duplicate existence:

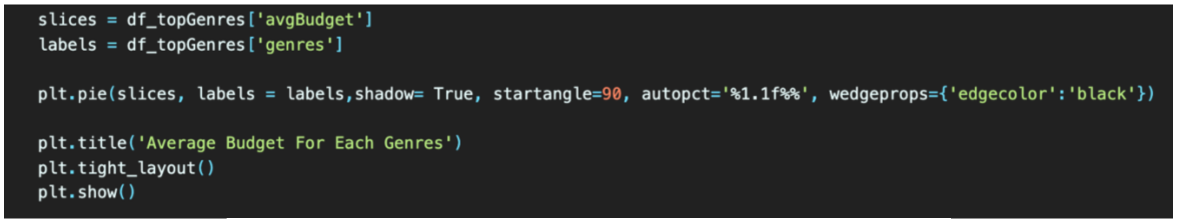


```
# Checking for duplications
sum(df_topGenres.duplicated())
#df.drop_duplicates(inplace=True)

0
```

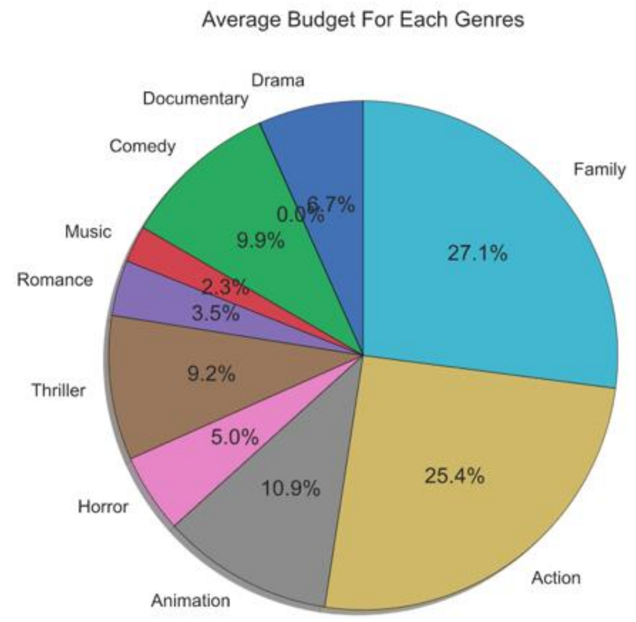
Data Cleaning:

Data cleaning and imputing the average profit column in the data frame would be quite same with which we had before on our previous whole data therefor to avoid any repetition we did not write this part here.

Plotting the Average budget for these 10 top Genre with a piechart.

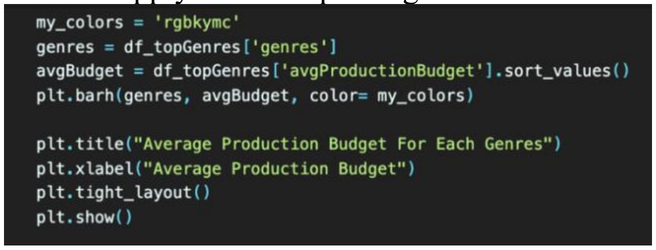


```
slices = df_topGenres['avgBudget']
labels = df_topGenres['genres']

plt.pie(slices, labels = labels,shadow= True, startangle=90, autopct='%1.1f%%', wedgeprops={'edgecolor':'black'})

plt.title('Average Budget For Each Genres')
plt.tight_layout()
plt.show()
```

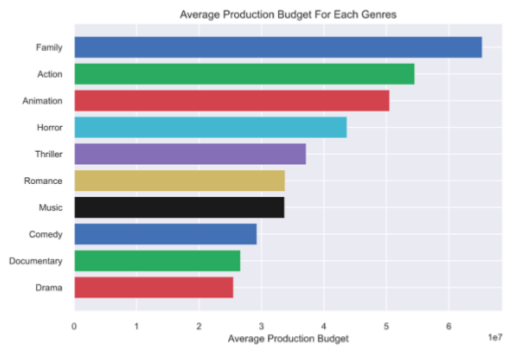


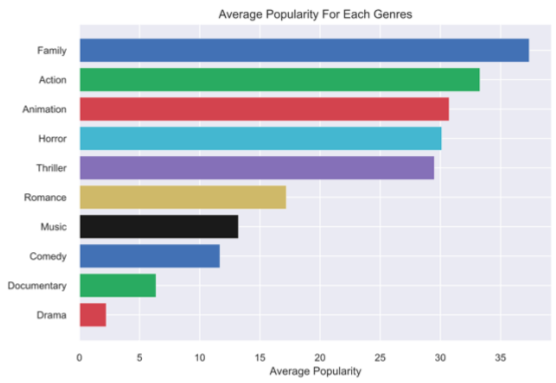
Results for the average of Profit, Budget and Popularity for each top 10 Genres:

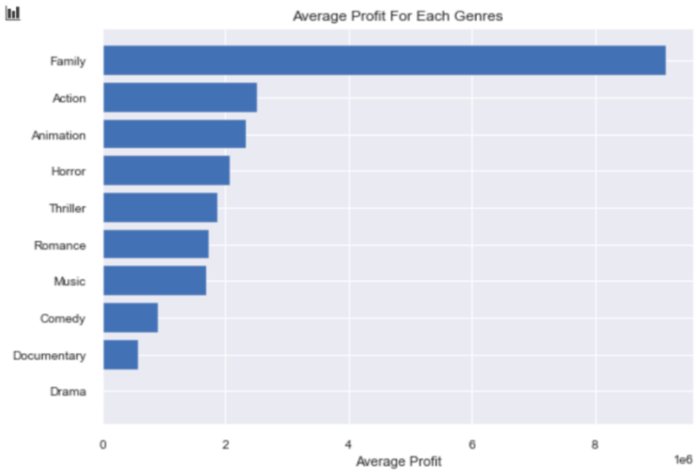
Also, for the other columns we can apply the same plotting.



```
my_colors = 'rgbkymc'
genres = df_topGenres['genres']
avgBudget = df_topGenres['avgProductionBudget'].sort_values()
plt.barh(genres, avgBudget, color= my_colors)

plt.title("Average Production Budget For Each Genres")
plt.xlabel("Average Production Budget")
plt.tight_layout()
plt.show()
```



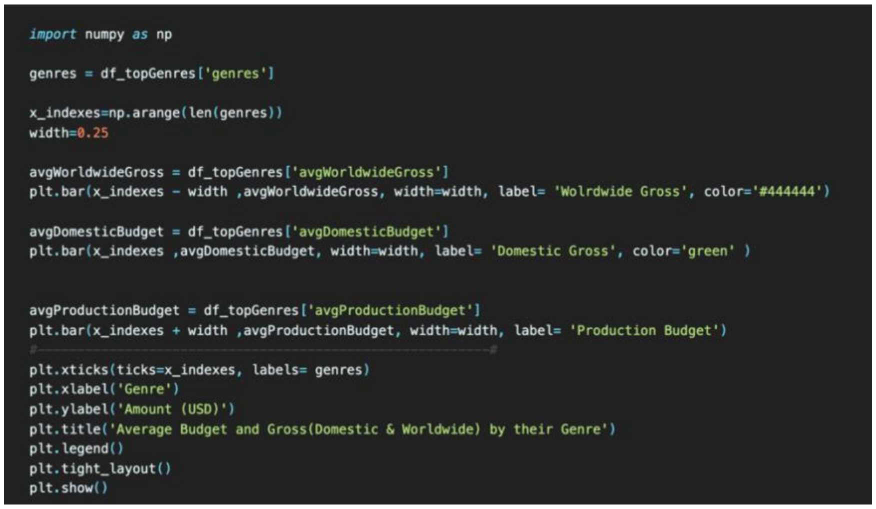




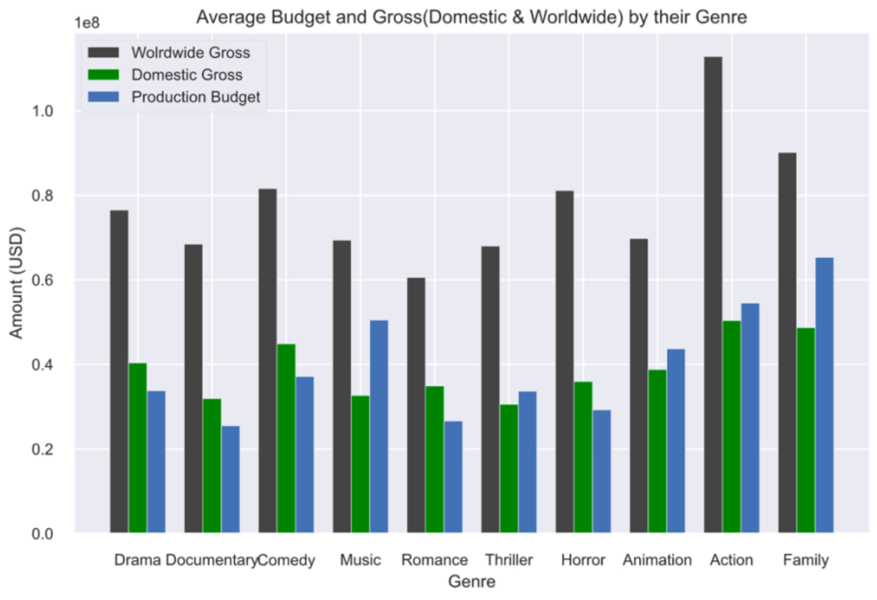
Family is the genre where the more investment occurs in English movies due to its well-known popularity among people of any age. As a result, it is the most profitable genre in the industry in comparison with Action and Animation that are also well-recognized lately.

Result for comparing the average of Budget, Domestic Gross and Worldwide Gross for these top ten Genres in a single plot:

In this stage we want to visualize all three average "investment for a specific Genre" and its Gross in both worldwide and Domestic area in one figure:



```
import numpy as np

genres = df_topGenres['genres']

x_indexes=np.arange(len(genres))
width=0.25

avgWorldwideGross = df_topGenres['avgWorldwideGross']
plt.bar(x_indexes - width ,avgWorldwideGross, width=width, label= 'Wolrdwide Gross', color='#444444')

avgDomesticBudget = df_topGenres['avgDomesticBudget']
plt.bar(x_indexes ,avgDomesticBudget, width=width, label= 'Domestic Gross', color='green' )


avgProductionBudget = df_topGenres['avgProductionBudget']
plt.bar(x_indexes + width ,avgProductionBudget, width=width, label= 'Production Budget')
#------------------------------------------------------------#
plt.xticks(ticks=x_indexes, labels= genres)
plt.xlabel('Genre')
plt.ylabel('Amount (USD)')
plt.title('Average Budget and Gross(Domestic & Worldwide) by their Genre')
plt.legend()
plt.tight_layout()
plt.show()
```

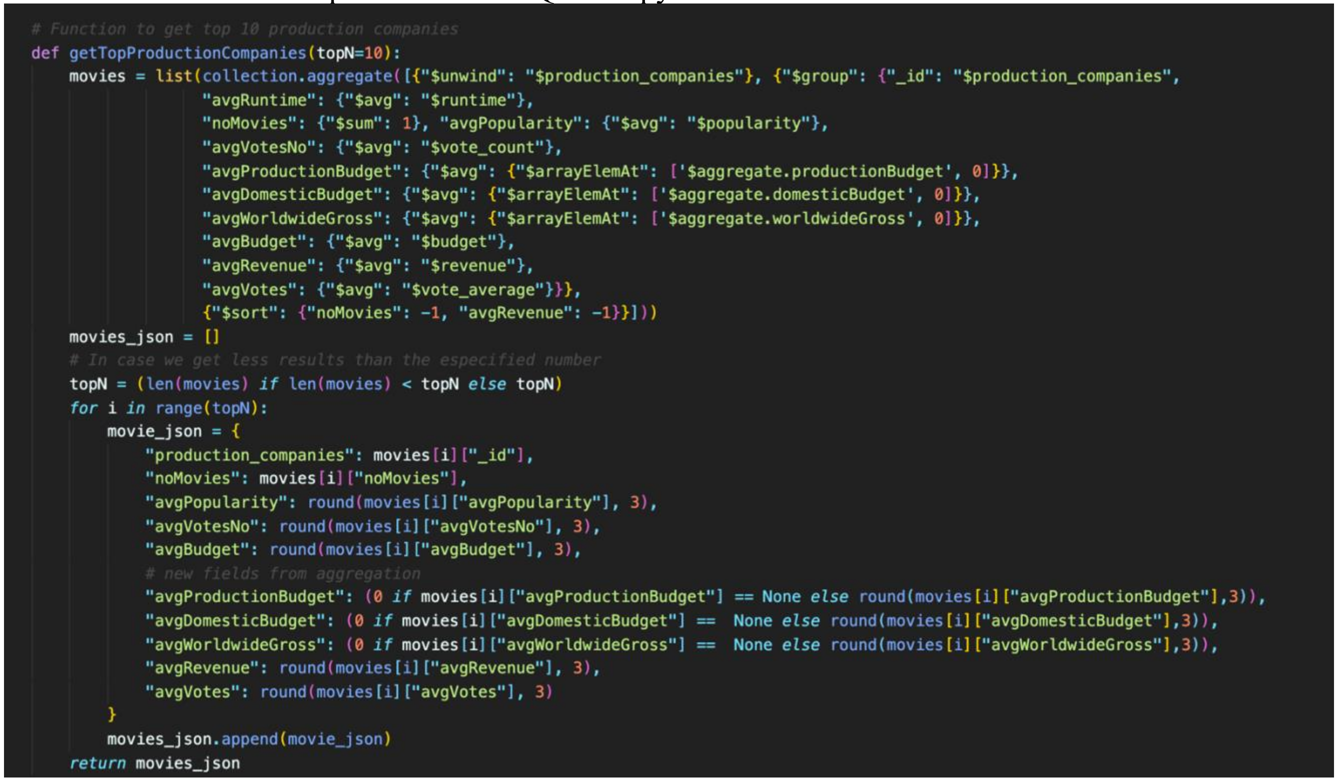


In this plot we can see two main profitable genres: Action and Family. However, whereas the investment on the family Genre is more than Action, the later earns more Gross revenue than the former.

Third Part: (Visualization – Investigating on companies):

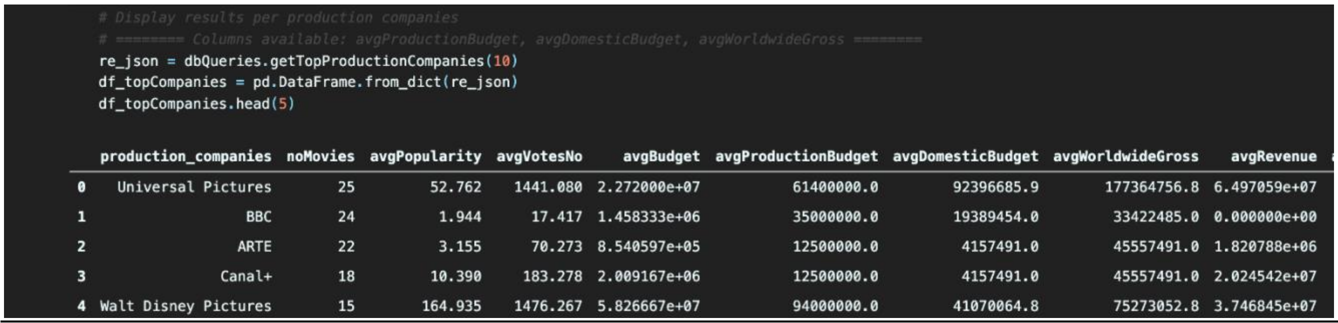
Loading Only Ten Top Company From Mongodb:

To have access to our mongodb and aggregate only ten top companies in a mongodb collection we need to write a function in a separate file as dbQueries.py as below:



```python
# Function to get top 10 production companies
def getTopProductionCompanies(topN=10):
    movies = list(collection.aggregate([{"$unwind": "$production_companies"}, {"$group": {"_id": "$production_companies",
                "avgRuntime": {"$avg": "$runtime"},
                "noMovies": {"$sum": 1}, "avgPopularity": {"$avg": "$popularity"},
                "avgVotesNo": {"$avg": "$vote_count"},
                "avgProductionBudget": {"$avg": {"$arrayElemAt": ['$aggregate.productionBudget', 0]}},
                "avgDomesticBudget": {"$avg": {"$arrayElemAt": ['$aggregate.domesticBudget', 0]}},
                "avgWorldwideGross": {"$avg": {"$arrayElemAt": ['$aggregate.worldwideGross', 0]}},
                "avgBudget": {"$avg": "$budget"},
                "avgRevenue": {"$avg": "$revenue"},
                "avgVotes": {"$avg": "$vote_average"}}},
                {"$sort": {"noMovies": -1, "avgRevenue": -1}}]))
    movies_json = []
    # In case we get less results than the especified number
    topN = (len(movies) if len(movies) < topN else topN)
    for i in range(topN):
        movie_json = {
            "production_companies": movies[i]["_id"],
            "noMovies": movies[i]["noMovies"],
            "avgPopularity": round(movies[i]["avgPopularity"], 3),
            "avgVotesNo": round(movies[i]["avgVotesNo"], 3),
            "avgBudget": round(movies[i]["avgBudget"], 3),
            # new fields from aggregation
            "avgProductionBudget": (0 if movies[i]["avgProductionBudget"] == None else round(movies[i]["avgProductionBudget"],3)),
            "avgDomesticBudget": (0 if movies[i]["avgDomesticBudget"] ==  None else round(movies[i]["avgDomesticBudget"],3)),
            "avgWorldwideGross": (0 if movies[i]["avgWorldwideGross"] ==  None else round(movies[i]["avgWorldwideGross"],3)),
            "avgRevenue": round(movies[i]["avgRevenue"], 3),
            "avgVotes": round(movies[i]["avgVotes"], 3)
        }
        movies_json.append(movie_json)
    return movies_json
```

Just like previous steps checks for any duplicates, null values or changing data types and then plot our results. For avoiding any repetition, we did not write these data cleaning process here and just showing the results.

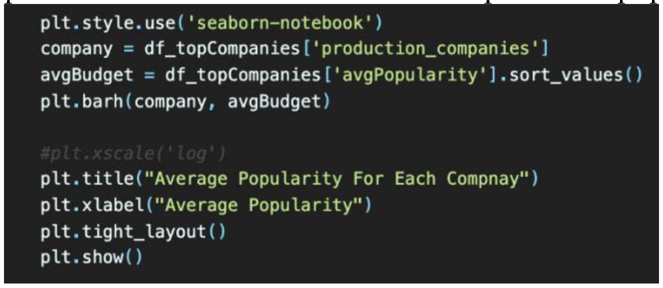


```
# Display results per production companies
# ======== Columns available: avgProductionBudget, avgDomesticBudget, avgWorldwideGross ========
re_json = dbQueries.getTopProductionCompanies(10)
df_topCompanies = pd.DataFrame.from_dict(re_json)
df_topCompanies.head(5)
```

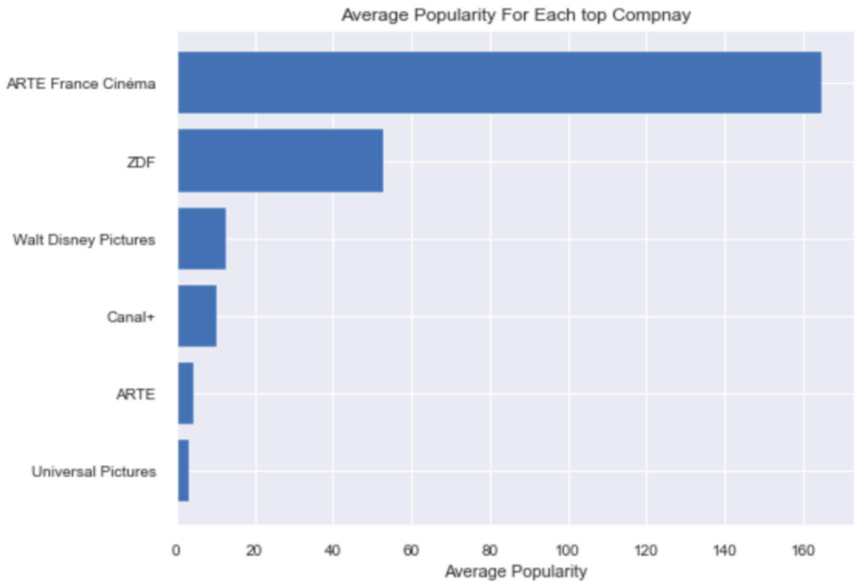
```
  production_companies  noMovies  avgPopularity  avgVotesNo     avgBudget  avgProductionBudget  avgDomesticBudget  avgWorldwideGross    avgRevenue
0    Universal Pictures        25         52.762    1441.080  2.272000e+07           61400000.0         92396685.9        177364756.8  6.497059e+07
1                   BBC        24          1.944      17.417  1.458333e+06           35000000.0         19389454.0         33422485.0  0.000000e+00
2                  ARTE        22          3.155      70.273  8.540597e+05           12500000.0          4157491.0         45557491.0  1.820788e+06
3                Canal+        18         10.390     183.278  2.009167e+06           12500000.0          4157491.0         45557491.0  2.024542e+07
4 Walt Disney Pictures        15        164.935    1476.267  5.826667e+07           94000000.0         41070064.8         75273052.8  3.746845e+07
```

Results for the average of Popularity for each of these top companies:

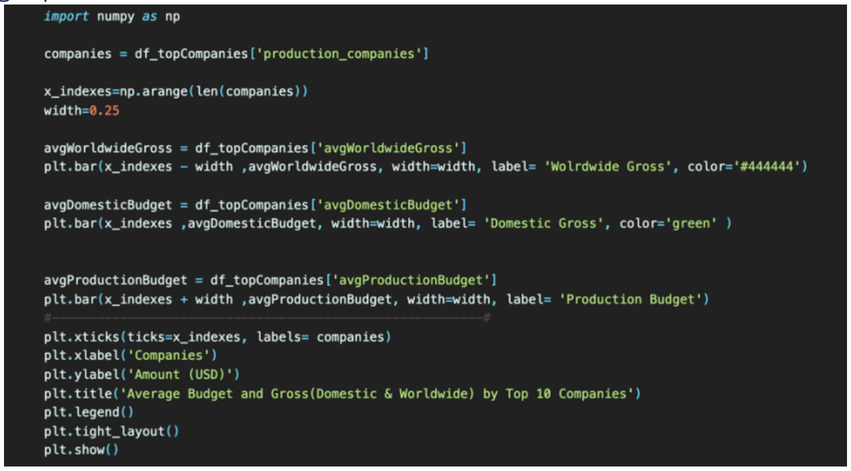
Here we plot these top companies based on their order of average popularity and we do the same for the average production budget and profit and number of movies they have released, and we see if each company invest more and produce more movies will earn more profit and popularity.



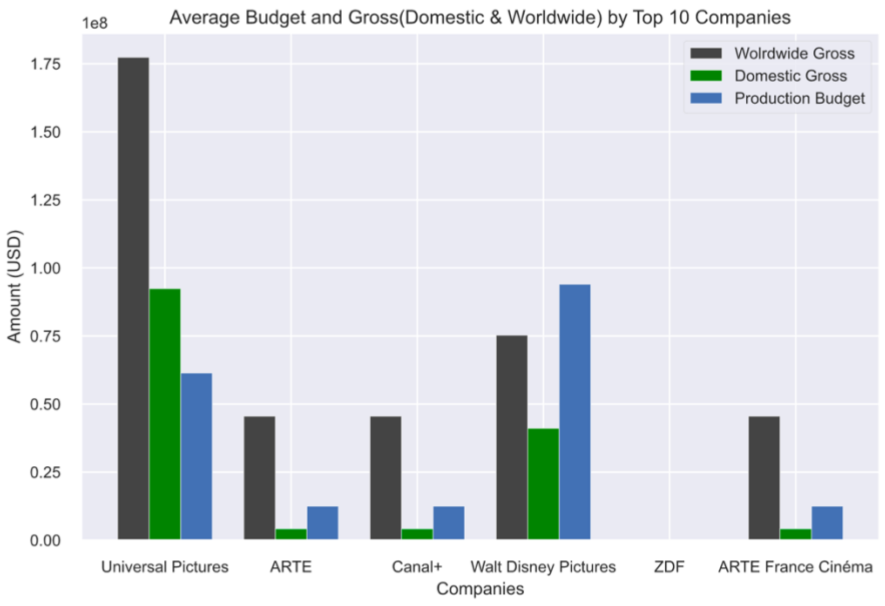
```
plt.style.use('seaborn-notebook')
company = df_topCompanies['production_companies']
avgBudget = df_topCompanies['avgPopularity'].sort_values()
plt.barh(company, avgBudget)

#plt.xscale('log')
plt.title("Average Popularity For Each Compnay")
plt.xlabel("Average Popularity")
plt.tight_layout()
plt.show()
```



Results for comparing the average of Budget, Domestic Gross and Worldwide Gross for these top companies in a single plot:



```
import numpy as np

companies = df_topCompanies['production_companies']

x_indexes=np.arange(len(companies))
width=0.25

avgWorldwideGross = df_topCompanies['avgWorldwideGross']
plt.bar(x_indexes - width ,avgWorldwideGross, width=width, label= 'Wolrdwide Gross', color='#444444')

avgDomesticBudget = df_topCompanies['avgDomesticBudget']
plt.bar(x_indexes ,avgDomesticBudget, width=width, label= 'Domestic Gross', color='green' )


avgProductionBudget = df_topCompanies['avgProductionBudget']
plt.bar(x_indexes + width ,avgProductionBudget, width=width, label= 'Production Budget')
#------------------------------------------------------------#
plt.xticks(ticks=x_indexes, labels= companies)
plt.xlabel('Companies')
plt.ylabel('Amount (USD)')
plt.title('Average Budget and Gross(Domestic & Worldwide) by Top 10 Companies')
plt.legend()
plt.tight_layout()
plt.show()
```



Here also we see that despite of Walt Disney invests more than Universal Pictures, Universal Pictures is still earning more gross revenue than its competitor.

Conclusion:

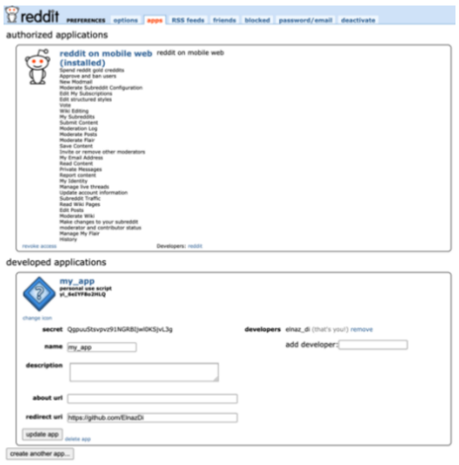
In the end, by visualizing our data we can understand that one of the key features for the movie's success would be a higher investment in the budget to get more votes between people and as the result be more popular to achieve more revenue. We also see that key features for the popularity of the genres or success of the companies to have more profit could be the number of movies released or their higher average investment. During the project we got a good opportunity to play around data with the help of functionalities like Kafka and Mongo DB. Python gave us a lot of flexibility with libraries like pandas and matplotlib to better visualize our data. But we did find there is a scope for **future enhancements** as following:

- Aggregation with flask or Metabase for better visualization.
- We can use docker instead of local machine to set this up.

Additional Resources

Requesting the Reddit API:

The Reddit website do not provide developers by any API Key instead it gives auth, therefor for this purpose you also need to create an account in the reddit website. Then in the app link by entering your application name, an optional link and selecting the script for the personal use the secret key and the client id will be appear and ready for use.



authorized applications

reddit on mobile web (installed) reddit on mobile web

Spend reddit gold creddits
Approve and ban users
New Modmail
Moderate Subreddit Configuration
Edit My Subscriptions
Edit structured styles
Vote
Wiki Editing
My Subreddits
Submit Content
Moderation Log
Moderate Posts
Moderate Flair
Save Content
Invite or remove other moderators
My Email Address
Read Content
Private Messages
Report content
My Identity
Manage live threads
Update account information
Subreddit Traffic
Read Wiki Pages
Edit Posts
Moderate Wiki
Make changes to your subreddit
moderator and contributor status
Manage My Flair
History

revoke access   Developers: reddit

developed applications

my_app
personal use script
yl_6eIYFBo2HLQ

change icon

secret QgpuuStsvpvz91NGRBIjwl0KSjvL3g   developers elnaz_di (that's you!) remove

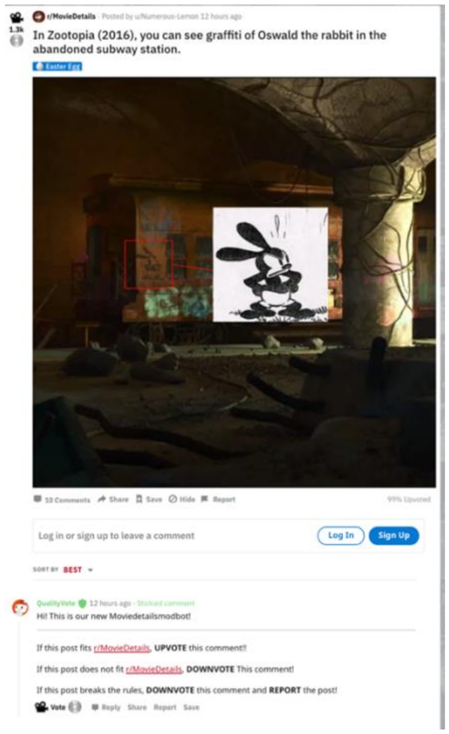
name my_app   add developer:

description

about url

redirect url https://github.com/ElnazDi

In the reddit we have some subreddits, that everybody can join to his favorite one. For this project we consider working on one of these subreddits named as: MovieDetails to not only receive the title of the subreddit but also receive both comments and replies by their unique ids, we see here that the title of this subreddit and all its comments are fetching into our python program:


r/MovieDetails · Posted by u/Numerous-Lemon 12 hours ago

1.3k

In Zootopia (2016), you can see graffiti of Oswald the rabbit in the abandoned subway station.

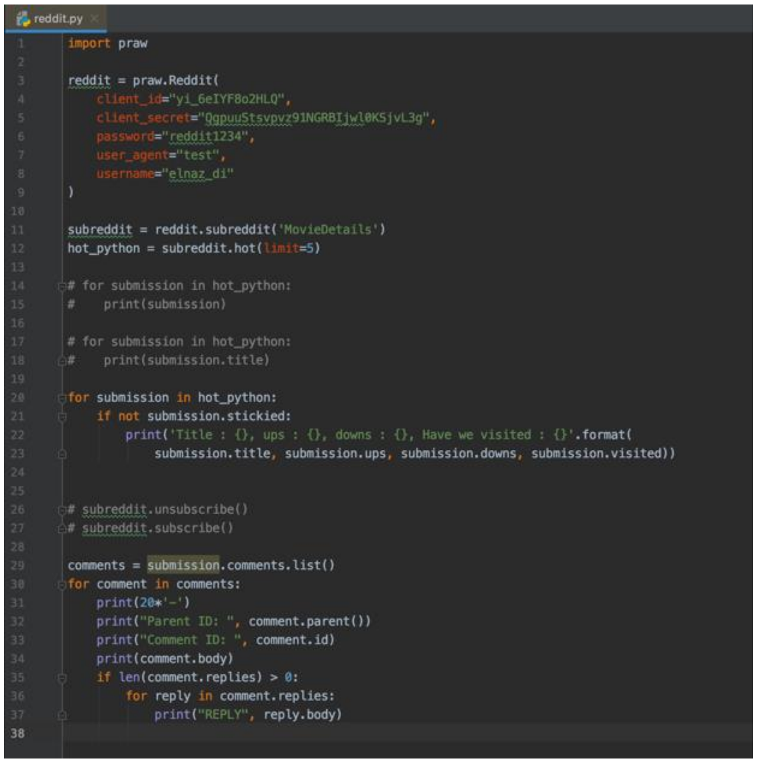
QualityVote 12 hours ago · Stickied comment

Hi! This is our new Moviedetailsmodbot!

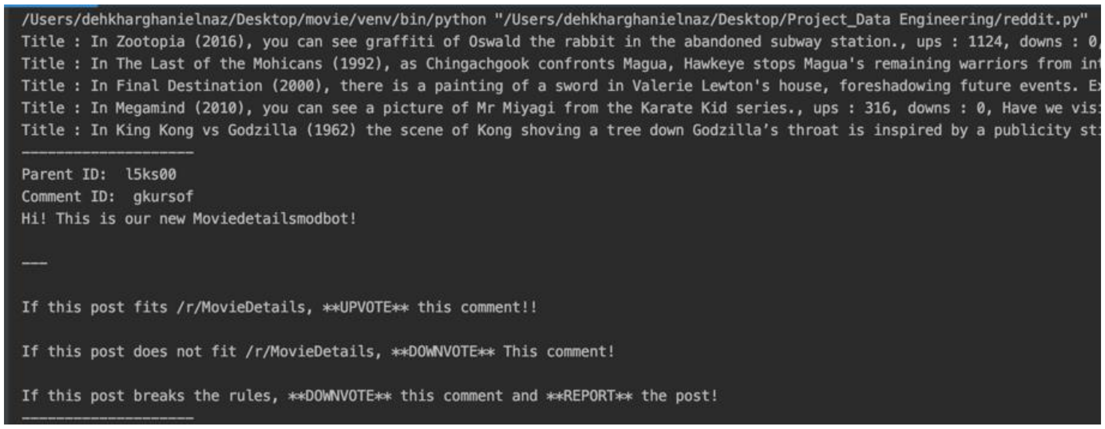
If this post fits r/MovieDetails, **UPVOTE** this comment!!

If this post does not fit r/MovieDetails, **DOWNVOTE** This comment!

If this post breaks the rules, **DOWNVOTE** this comment and **REPORT** the post!



```python
import praw

reddit = praw.Reddit(
    client_id="yi_6eIYF8o2HLQ",
    client_secret="QgpuuStsvpvz91NGRBIjwl0KSjvL3g",
    password="reddit1234",
    user_agent="test",
    username="elnaz_di"
)

subreddit = reddit.subreddit('MovieDetails')
hot_python = subreddit.hot(limit=5)

# for submission in hot_python:
#    print(submission)

# for submission in hot_python:
#    print(submission.title)

for submission in hot_python:
    if not submission.stickied:
        print('Title : {}, ups : {}, downs : {}, Have we visited : {}'.format(
            submission.title, submission.ups, submission.downs, submission.visited))


# subreddit.unsubscribe()
# subreddit.subscribe()

comments = submission.comments.list()
for comment in comments:
    print(20*'-')
    print("Parent ID: ", comment.parent())
    print("Comment ID: ", comment.id)
    print(comment.body)
    if len(comment.replies) > 0:
        for reply in comment.replies:
            print("REPLY", reply.body)

```

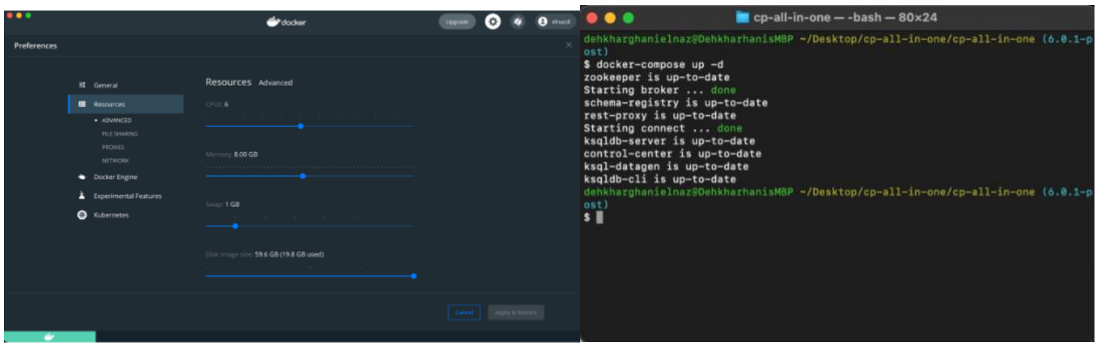


```
/Users/dehkharghanielnaz/Desktop/movie/venv/bin/python "/Users/dehkharghanielnaz/Desktop/Project_Data Engineering/reddit.py"
Title : In Zootopia (2016), you can see graffiti of Oswald the rabbit in the abandoned subway station., ups : 1124, downs : 0,
Title : In The Last of the Mohicans (1992), as Chingachgook confronts Magua, Hawkeye stops Magua's remaining warriors from in
Title : In Final Destination (2000), there is a painting of a sword in Valerie Lewton's house, foreshadowing future events. E
Title : In Megamind (2010), you can see a picture of Mr Miyagi from the Karate Kid series., ups : 316, downs : 0, Have we vis
Title : In King Kong vs Godzilla (1962) the scene of Kong shoving a tree down Godzilla’s throat is inspired by a publicity st
--------------------
Parent ID:  l5ks00
Comment ID:  gkursof
Hi! This is our new Moviedetailsmodbot!

---

If this post fits /r/MovieDetails, **UPVOTE** this comment!!

If this post does not fit /r/MovieDetails, **DOWNVOTE** This comment!

If this post breaks the rules, **DOWNVOTE** this comment and **REPORT** the post!
--------------------
```

Docker Configuration- Apache Kafka using Confluent Platform:

Step 1:

For using Apache Kafka through docker you need to have installed Docker then clone a confluent repository into your local system: https://github.com/confluentinc/cp-all-in-one Before using docker, you must be aware to set the memory from the default 2.00 GB to 8.00 GB to have a stable environment for working with Kafka Confluent. Then you can start Confluent by running a command in the terminal:


Preferences

Resources Advanced

CPUs: 6

Memory: 8.00 GB

Swap: 1 GB

Disk image size: 59.6 GB (19.8 GB used)

```
cp-all-in-one — -bash — 80×24
dehkharghanielnaz@DehkharhanisMBP ~/Desktop/cp-all-in-one/cp-all-in-one (6.0.1-p
ost)
$ docker-compose up -d
zookeeper is up-to-date
Starting broker ... done
schema-registry is up-to-date
rest-proxy is up-to-date
Starting connect ... done
ksqldb-server is up-to-date
control-center is up-to-date
ksql-datagen is up-to-date
ksqldb-cli is up-to-date
dehkharghanielnaz@DehkharhanisMBP ~/Desktop/cp-all-in-one/cp-all-in-one (6.0.1-p
ost)
$ 
```

Figure 1.14 Starting Confluent in Docker. Left Image – set RAM to 8 GB. Right Image – starting Confluent.

here, you will see that Confluent is up and running in the Docker dashboard as well in the terminal:

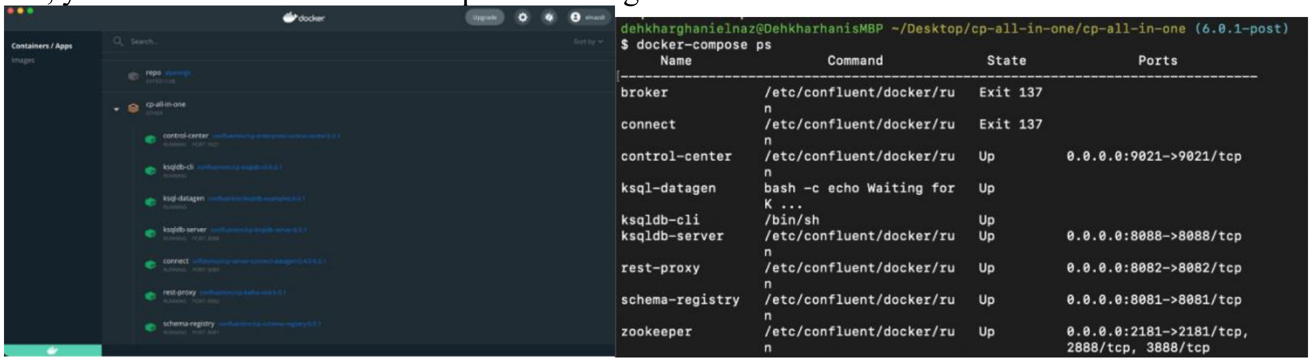
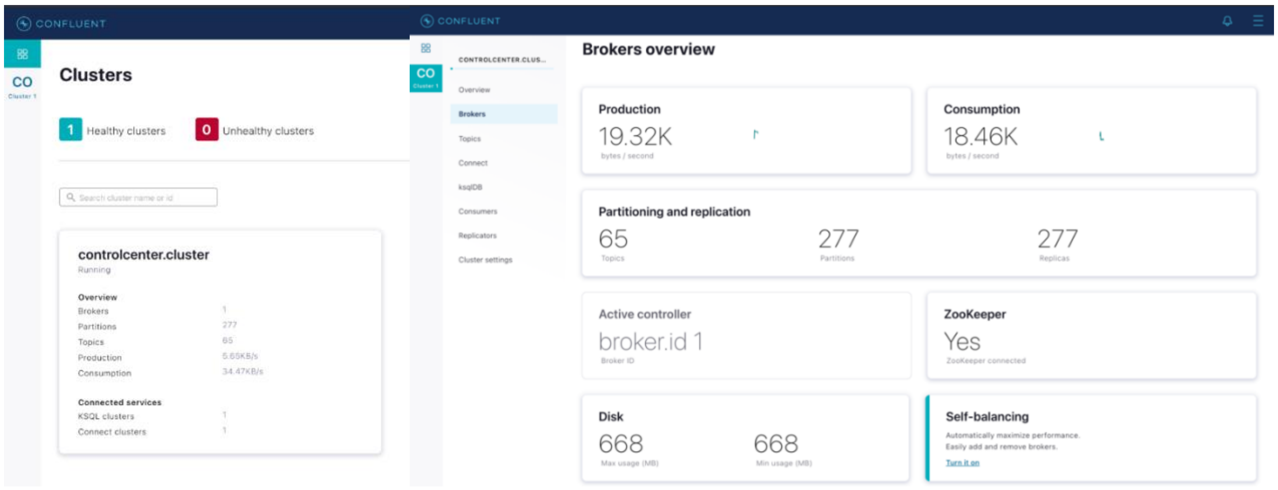

```
dehkharghanielnaz@DehkharhanisMBP ~/Desktop/cp-all-in-one/cp-all-in-one (6.0.1-post)
$ docker-compose ps
     Name                 Command              State                Ports
------------------------------------------------------------------------------------
broker            /etc/confluent/docker/ru   Exit 137
                  n
connect           /etc/confluent/docker/ru   Exit 137
                  n
control-center    /etc/confluent/docker/ru   Up         0.0.0.0:9021->9021/tcp
                  n
ksql-datagen      bash -c echo Waiting for   Up
                  K ...
ksqldb-cli        /bin/sh                    Up
ksqldb-server     /etc/confluent/docker/ru   Up         0.0.0.0:8088->8088/tcp
                  n
rest-proxy        /etc/confluent/docker/ru   Up         0.0.0.0:8082->8082/tcp
                  n
schema-registry   /etc/confluent/docker/ru   Up         0.0.0.0:8081->8081/tcp
                  n
zookeeper         /etc/confluent/docker/ru   Up         0.0.0.0:2181->2181/tcp,
                  n                                     2888/tcp, 3888/tcp
```

Figure 1.15 Confluent running. Left Image – Docker dashboard. Right Image – Terminal to see available services.

Step 2:

Create Kafka topics:

Now by using the Confluent Control Center we can create Kafka topics. Confluent Control Center provides the functionality for building and monitoring production data pipelines and event streaming applications. We Open the Control Center web interface at http://localhost:9021.


Clusters

1 Healthy clusters 0 Unhealthy clusters

controlcenter.cluster
Running

Overview
Brokers 1
Partitions 277
Topics 65
Production 5.65KB/s
Consumption 34.47KB/s

Connected services
KSQL clusters 1
Connect clusters 1

Brokers overview

Production 19.32K bytes / second

Consumption 18.46K bytes / second

Partitioning and replication
65 Topics
277 Partitions
277 Replicas

Active controller broker.id 1 Broker ID

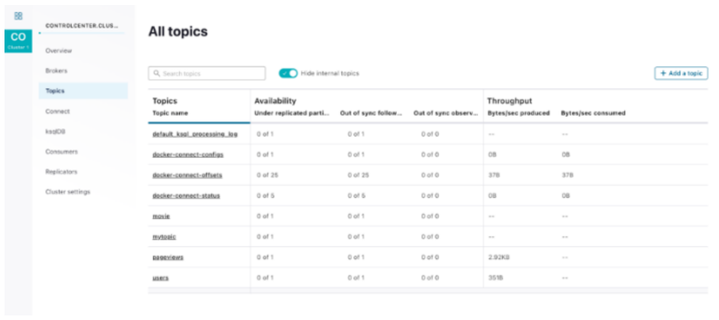
ZooKeeper Yes ZooKeeper connected

Disk
668 Max usage (MB)
668 Min usage (MB)

Self-balancing
Automatically maximize performance.
Easily add and remove brokers.
Turn it on

Figure 1.16 Confluent up and running. Left Image – cluster checkup. Right Image – Confluent dashboard

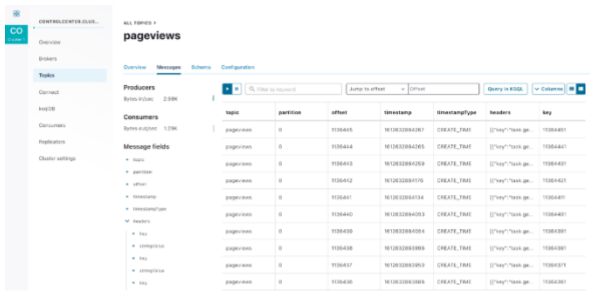
We continued our job with Confluent by adding topics, send and receiving message via python but since we could not stablish the MongoDB connector for this confluent we stopped here and continued our job with Kafka locally as an alternative solution.



All topics

Search topics | Hide internal topics | + Add a topic

Topics	Availability			Throughput	
Topic name	Under replicated parti...	Out of sync follow...	Out of sync observ...	Bytes/sec produced	Bytes/sec consumed
default_ksql_processing_log	0 of 1	0 of 1	0 of 0	--	--
docker-connect-configs	0 of 1	0 of 1	0 of 0	0B	0B
docker-connect-offsets	0 of 25	0 of 25	0 of 0	37B	37B
docker-connect-status	0 of 5	0 of 5	0 of 0	0B	0B
movie	0 of 1	0 of 1	0 of 0	--	--
mytopic	0 of 1	0 of 1	0 of 0	--	--
pageviews	0 of 1	0 of 1	0 of 0	2.92KB	--
users	0 of 1	0 of 1	0 of 0	351B	--



ALL TOPICS >

pageviews

Producers
Bytes in/sec 2.98K

Consumers
Bytes out/sec 1.29K

topic	partition	offset	timestamp	timestampType	headers	key
pageviews	0	1136445	1612632884267	CREATE_TIME	[{"key":"task.ge...	11364451
pageviews	0	1136444	1612632884265	CREATE_TIME	[{"key":"task.ge...	11364441
pageviews	0	1136443	1612632884259	CREATE_TIME	[{"key":"task.ge...	11364431
pageviews	0	1136442	1612632884176	CREATE_TIME	[{"key":"task.ge...	11364421
pageviews	0	1136441	1612632884134	CREATE_TIME	[{"key":"task.ge...	11364411
pageviews	0	1136440	1612632884093	CREATE_TIME	[{"key":"task.ge...	11364401
pageviews	0	1136439	1612632884054	CREATE_TIME	[{"key":"task.ge...	11364391
pageviews	0	1136438	1612632883999	CREATE_TIME	[{"key":"task.ge...	11364381
pageviews	0	1136437	1612632883953	CREATE_TIME	[{"key":"task.ge...	11364371
pageviews	0	1136436	1612632883885	CREATE_TIME	[{"key":"task.ge...	11364361

Figure 1.17 Topics in Confluent. Left Image – View all topics. Right Image – Create new topic.

Resources:

[1] “The numbers”, 1997-2021, Nash Information Service, LLC. https://www.the-numbers.com/movie/budgets/all
[2] “The Movie DB” (TMDB), 2008-2021, Community movie and TV database, https://www.themoviedb.org/
[3] Bat, A. “Investigating TMDB Movie Dataset”, 2019, https://medium.com/my-data-camp-journey/investigating-tmdb-movie-datasets-4ee04c263915
[4] Waskon, M. “seaborn: statistical data visualization”, 2012-2020, https://seaborn.pydata.org/index.html
[5] Richardson, L. “Beautiful Soup”, (Last modification: October 3, 2020), https://www.crummy.com/software/BeautifulSoup/
[6] Cobrateam Revision, “Splinter”, 2021, https://splinter.readthedocs.io/en/latest/
[7] Apache Sofwtare Foundation, “Apache Kafka”, 2017, https://kafka.apache.org/quickstart
[8] Confluent Inc. “Quick Start for Apache Kafka using Confluent Platform (Docker)”, 2021, https://docs.confluent.io/platform/current/quickstart/ce-docker-quickstart.html?utm_medium=sem&utm_source=google&utm_campaign=ch.sem_br.brand_tp.prs_tgt.confluent-brand_mt.mbm_rgn.emea_lng.eng_dv.all&utm_term=%2Bconfluent%20%2Bdocker&creative=&device=c&placement=&gclid=Cj0KCQiAmL-ABhDFARIsAKywVacHBIG8vBFffXpz12sVf0HtIgO7TqnrTfCQro5nwcMjHbhwqKvJPwEaAtZ7EALw_wcB
[9] MongoDB, “PyMongo 1.11.3 Documentation”, 2018-2021, https://pymongo.readthedocs.io/en/stable/index.html