École d'ingénieurs du numérique

Movie Data Analysis

Advanced Database and Big Data

Team Members:

XU Yiteng_62735

FAN Hanyue_62718

Summary

1. Brief Introduction

This project realizes fetching and calling API to obtain data from IMDb and TMDb websites, formatted the source data, and combines data from two different data sources. Then write the combined data into Elasticsearch, combine with Kibana, draw a chart, and finally form a dashboard.

2. Data Processing

2.1 Ingestion

a.Fetching data from IMDb

- We fetch data from the IMDb website and save it locally. The following modules are used:

1) The *date* class from the *datetime* module is imported to get the current date.
2) The *requests* module is imported to send HTTP requests and retrieve data from URLs.

- The utility function is then defined: $fetch\_data\_from\_imdb$, which takes the following parameters:

1) url: The URL of the IMDb dataset.
2) $data\_entity\_name$: A string representing the name of the data entity being fetched.
3) $**\ kwargs$: Additional keyword arguments (if any).

$TARGET\_PATH$ is constructed by combining $DATALAKE\_ROOT\_FOLDER$, raw, $data\_entity\_name$ and the current date. If $TARGET\_PATH$ does not exist, it is created using $os.makedirs()$**.** Then we get the data from IMDb: use $requests.get()$ to make a GET request to the specified url and store the response content in the r variable. We need to save the data we got, extract the filename from url by splitting the filename at the ' $/$ ' character and take the last part use $open\ (file\_path, wb).write(r.content)$. Save the data into datalake.

b.Fetching data from TMDb

- We fetch movie data from the Movie Database (TMDb) API and store it locally. The following modules are used:

1) The $requests$ module is imported to send HTTP requests and retrieve data from the API.
2) The $json$ module is imported to work with JSON data.
3) The $date$ class from the 'datetime' module is imported to get the current date.
4) The os module is imported to work with file paths and directories.

- The $fetch\_data\_from\_tmdb$ function is defined, which accepts the following arguments:

1) api_key: API key required to access TMDb API.
2) $kwargs$: additional keyword arguments (if any).

We use the $fetch\_data\_from\_tmdb$ function to fetch the data, and call the query_data_from_tmdb function, passing the api_key as a parameter.
In $query\_data\_from\_tmdb$, the URL of the top rated movie endpoint is set. API keys are passed as parameters in the params dictionary. Make a GET request to the TMDb API using $requests.get(url, params = params)$. If the response status code is 200 (indicating a successful request), use $response.json()$ to extract the JSON data from the response. The function returns the "result" key from the JSON data containing movie details.
We store the data using the $store\_tmdb\_data$ function, passing the retrieved movie data as a parameter.
In $store\_tmdb\_data$, use $date.today().strftime("\%Y\%m\%d")$ to get the current date in YYYYMMDD format. The target directory path is constructed by combining DATALAKE_ROOT_FOLDER, $raw/tmdb/MovieTopRated$ and the current date. If the target directory does not exist, it is created using $os.makedirs()$.
Movie data is stored in a JSON file using $json.dump(movies, f, indent = 4)$.

It is worth noting that we have applied for a valid TMDb API key.

2.2 Formatting

a.IMDb data formatted

We read the raw IMDb dataset file in TSV format, convert it to a Data Frame using pandas, and save the Data Frame as a Parquet file in the formatted IMDb dataset folder. Parquet files provide a more efficient and compressed storage format for data.
We read the raw IMDb dataset file into a Data Frame using $pd.read\_csv()$, specifying the delimiter as '\t' (tab-separated values). Use $df.to\_parquet()$ to convert DataFrame to Parquet file format.

b.TMDb data formatted

We read the TMDb raw dataset file in JSON format, convert it to a DataFrame using pandas, and save the DataFrame as a Parquet file in the formatted dataset folder. Parquet files provide a more efficient and compressed storage format for data, in this case using the "snappy" compression algorithm.
Use $df.to\_parquet()$ to convert DataFrame to Parquet file format. Parquet files are saved in the formatted dataset folder, and the file name is to replace $.json$ in filename with $.snappy.parquet$. Parquet files are compressed using the $snappy$ compression algorithm.

2.3 Combination

We perform data manipulation and merging on the IMDb and TMDb datasets. The following modules are imported:
1) The $pandas$ library is imported and assigned the alias pd for data processing and analysis.
2) The $matplotlib.pyplot$ module is imported and assigned the alias plt for data visualization.
3) The $seaborn$ library is imported and assigned the alias sns for enhanced data visualization.
4) The os module is imported to work with file paths and directories.

a.Merging IMDb data

We use $pd.read\_parquet()$ to load the IMDb data with rating information and raw header information into a DataFrame $imdb\_rating\_data$. The two IMDb DataFrames are then merged based on the common column $tconst$ using $merge()$, and the merged data is assigned to the DataFrame $imdb\_rating\_title\_data$.

b.Merging TMDb data

We use $pd.read\_parquet()$ to load the TMDb data with rating information and raw title information into DataFrame $imdb\_rating\_data$ and DataFrame $tmdb\_title\_data$ respectively. The two TMDb dataframes are then vertically concatenated using $pd.concat()$ and assigned to the DataFrame $tmdb\_rating\_title\_data$.

c.Combining IMDb and TMDb data

We have renamed the “original_title” column in the TMDb DataFrame to $originalTitle$ to match IMDb's $originalTitle$ using $rename()$. Then select specific columns from the TMDb DataFrame \`tmdb_rating_title_data\` using the column index and assign the result to the same DataFrame, filtering out unwanted columns. The “vote_average”, “vote_count” and “title” columns in the TMDb DataFrame were renamed to “averageRating”, “num”. Votes' and “primaryTitle” respectively use $rename()$ to match the column names of the IMDb DataFrame.
In the compositing operation, we vertically concatenate the IMDb DataFrame $imdb\_rating\_title\_data$ and the TMDb DataFrame $tmdb\_rating\_title\_data$ using $pd.concat()$. The result is assigned to DataFrame $combine\_twoAPI$.

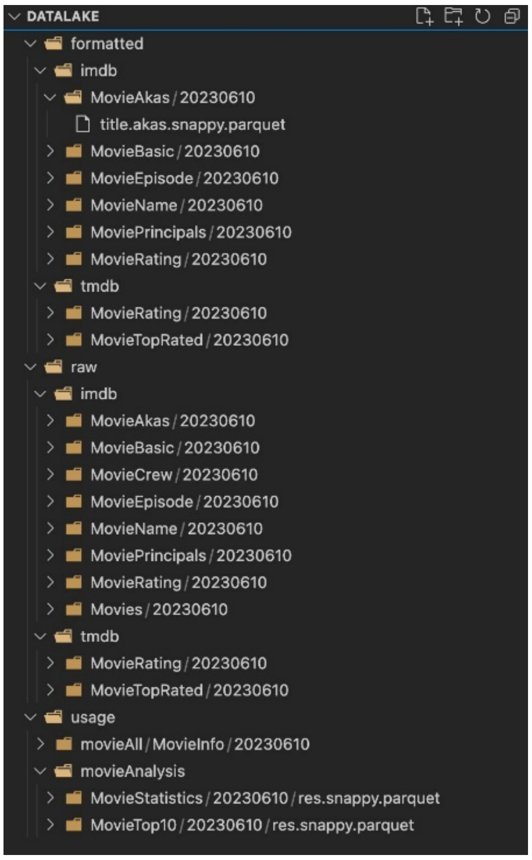
Finally, we save the combined DataFrame $combine\_twoAPI$ to a Parquet file in the output folder using $to\_parquet()$. The filename is specified as "combine_twoAPI.snappy.parquet" and compressed using $snappy$.

d.Spark Using

We use PySpark to combine and analyse movie rating data. The os module for filesystem operations and the $SQLContext$ module from $pyspark.sql$ for use with $Spark\ SQL$ are imported. We create a $SparkContext$ with the application name "CombineData" and create a $SQLContext$ from the $SparkContext$. We created a function combine_data. This function reads data from the Parquet file located on RATING_PATH and registers it as a temporary table named "ratings" using the registerTempTable method. Call the show method to print the contents of the DataFrame $df\_ratings$.
A SQL query is then executed using the $sqlContext.sql$ method to calculate statistics for the Average Rating column of the Ratings table. The resulting DataFrame is assigned to the variable stats_df. Then we call the show method to print the contents of the $stats\_df$ DataFrame.
Finally, another SQL query is executed to select the top 10 movies based on their average rating, but only include movies with more than 5000 votes. The resulting DataFrame is assigned to the variable top10_df. Call the show method to print the contents of the top10_df DataFrame.
This function saves the $stats\_df$ DataFrame as a Parquet file in the USAGE_OUTPUT_FOLDER_STATS directory, overwriting any existing data. The file is saved with the name "res.snappy.parquet".

2.4 Datalake

We follow the structure of datalake to store source data, formatted data and combined data, as shown in the figure below.



```
DATALAKE
  formatted
    imdb
      MovieAkas / 20230610
        title.akas.snappy.parquet
      MovieBasic / 20230610
      MovieEpisode / 20230610
      MovieName / 20230610
      MoviePrincipals / 20230610
      MovieRating / 20230610
    tmdb
      MovieRating / 20230610
      MovieTopRated / 20230610
  raw
    imdb
      MovieAkas / 20230610
      MovieBasic / 20230610
      MovieCrew / 20230610
      MovieEpisode / 20230610
      MovieName / 20230610
      MoviePrincipals / 20230610
      MovieRating / 20230610
      Movies / 20230610
    tmdb
      MovieRating / 20230610
      MovieTopRated / 20230610
  usage
    movieAll / MovieInfo / 20230610
    movieAnalysis
      MovieStatistics / 20230610 / res.snappy.parquet
      MovieTop10 / 20230610 / res.snappy.parquet
```

Figure1. Datalake Structure

3. Dashboard

3.1 Indexing data into Elasticsearch

We index data into Elasticsearch using the Elasticsearch Python client.
The following modules are imported:
The *Elasticsearch* class from the *elasticsearch* module is imported to establish a connection with Elasticsearch.
The *pandas* library is imported and assigned the alias *pd* for data manipulation.
The *ssl* module is imported to configure SSL/TLS settings.

a.Configuring SSL/TLS context

We use $ssl.create\_default\_context()$ to create the SSL context $ssl\_context.check\_hostname = False$ to disable SSL hostname verification. Finally disable SSL certificate verification with $ssl\_context.verify\_mode = ssl.CERT\_NONE$.

b.Establishing a connection to Elasticsearch

During this process, an instance of the *Elasticsearch* class is created and assigned to the variable es, connecting to the *Elasticsearch* server running on localhost at port 9200. HTTP authentication is provided using username “elastic” and password “sKzLFRnpvFbd2s6***”. Finally, a connection is established using HTTPS with the specified SSL context.

c.Retrieving Elasticsearch server information

We call the $es.info()$ method to retrieve information about the Elasticsearch server. The $pretty = True$ parameter formats the output in a human-readable format.

d.Converting the DataFrame to a list of dictionaries

We use $to\_dict(orient = 'records')$ on the *combine_twoAPI* DataFrame containing the combined data from IMDb and TMDb to convert to a list of dictionaries. Each dictionary in the list represents a document to be indexed in Elasticsearch.

e.Indexing documents in Elasticsearch

We iterate over each document in the data list. And each document is filtered on averageRating greater than or equal to 6.0 and *numVotes* greater than 10000.
For filtered documents, only the required fields ($averageRating, numVotes, primaryTitle, and\ originalTitle$) are indexed in Elasticsearch using the $es.index()$ method.

3.2 Use kibana to draw icons

We use Kibana to search, view, and interact with data in the Elasticsearch index. Then a dashboard is generated, which contains three charts, as shown in the figure below

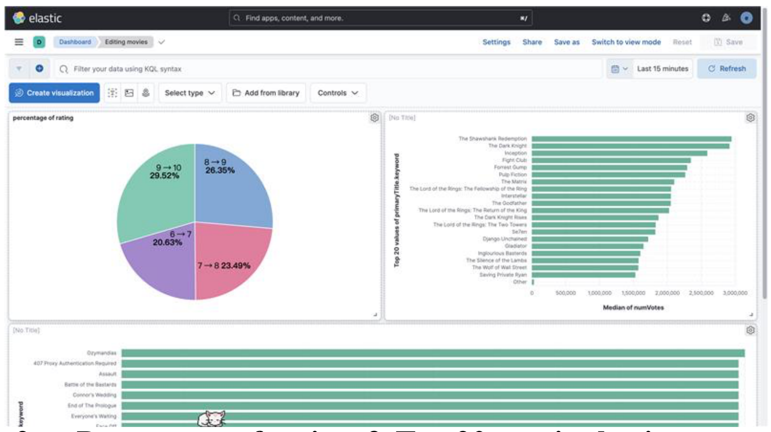


Figure2. Percentage of rating & Top 20 movies basic on *numVotes*

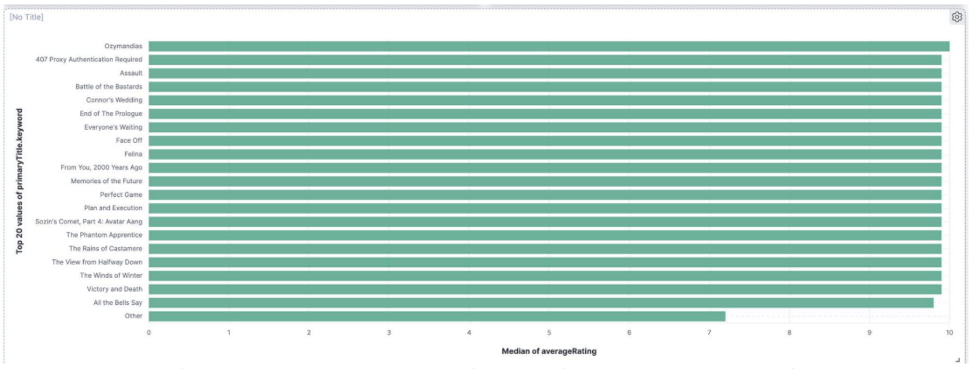


Figure3. Top 20 movies basic on averageRating