

INSTITUT SUPÉRIEUR D'ÉLECTRONIQUE DE PARIS (ISEP)



II.2414 ADVANCED DATABASES AND BIG DATA

**BIG DATA PROJECT (Data
Lake and Data Pipeline)**

ABUBAKAR UMAR ELNAFATY

PROF. THIBAUT DE BROCA

JUNE 2023

Synopsis

The main goal of the 'Big Data project' is to create a simple end-to-end data architecture that spans the entire data lifecycle, encompassing data ingestion, transformation, and exposition. Within this project, you have the freedom to choose the specific data you wish to retrieve and determine the desired output that will generate value from the input data.

However, it is essential to adhere to a Datalake architecture, as it plays a crucial role in organizing the data effectively, ensuring a streamlined data pipeline, and facilitating data sharing among stakeholders. By following the Datalake architecture principles, we can achieve a well-structured and efficient data ecosystem

Theme for the Data

The central focus of this project revolves around gathering movies, weather data, and forecast information from multiple sources. The primary goal is to conduct a comprehensive analysis of the current top-rated, upcoming, and trending movies, considering their popularity, vote counts, release dates, and weather conditions. By incorporating weather forecast data, this analysis aims to provide valuable insights for individuals planning to visit a cinema.

By visualizing the cinema data alongside weather forecasts, stakeholders such as consumers, researchers, statisticians, and data scientists can make more informed decisions about their moviegoing experiences. The integration of weather information allows users to consider factors like temperature, precipitation, and other weather-related conditions that can impact their cinema experience.

For example, if someone prefers to enjoy outdoor activities before or after watching a movie, they can check the weather forecast to determine the best time to visit the cinema. If it's expected to rain heavily, they might opt for an indoor movie theater instead of an outdoor cinema. On the other hand, if the weather forecast indicates clear skies and pleasant temperatures, they may choose an outdoor cinema for a unique and enjoyable experience.

Moreover, the combination of cinema data and weather forecasts enables users to apply filters and preferences based on their specific moviegoing desires. They can filter movies based on genre, release date, popularity, and user ratings, and further refine their choices by considering weather conditions. For instance, individuals who prefer to watch horror movies during rainy days can easily identify suitable options by cross-referencing the cinema data with the weather forecast.

Furthermore, this enhanced visualization empowers individuals to plan their cinema outings more effectively by taking into account both movie-related factors and weather conditions. It provides a comprehensive understanding of the movie landscape and allows users to make informed decisions, ensuring an enjoyable and tailored cinema experience

Data Sources

The retrieved data would be gathered in the data lake as raw data. Data sources include:

1. The Movie Database TMDB API
2. Weather Api

A community-built movie and television database is called The Movie Database (TMDB) API. The documentation for the API is available at <https://developer.themoviedb.org/docs/getting-started>, and it includes instructions on how to connect to our application and fetch data via the API. We have fetched 3 different dataset from the api, the top rated movies, the upcoming movies and trending movies in the cinema. We saved the data as a json file. While the weather api documentation can be access via <https://www.weatherapi.com/docs>, we fetched the current weather and forecast weather of Paris

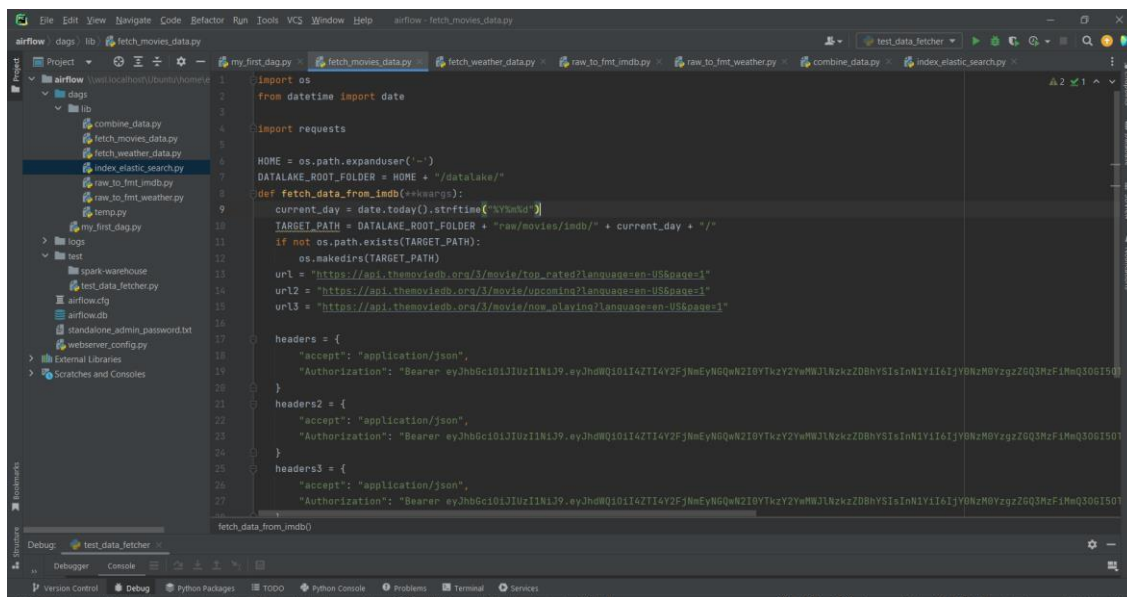
The image is a screenshot of a code editor, likely VS Code, showing a Python script named 'fetch_movies_data.py'. The script is part of an Airflow DAG. The code imports 'os' and 'datetime' to get the current date. It defines a base path for the data lake. A function 'fetch_data_from_tmdb' is defined, which takes a date as input and constructs three URLs to fetch top-rated, upcoming, and now-playing movies from the TMDB API. The script uses 'requests' to make these calls with specific headers for authentication. The fetched data is then saved as JSON files in the data lake. The editor's interface includes a file explorer on the left, a terminal at the bottom, and a debug console.

Figure 1 shows how to access the movie database API.

The Weather API serves as a valuable source of raw data for this project, providing real-time weather forecasts (for the next 7 days) and current weather information for Paris. By integrating this data into the project's data lake, stakeholders can explore the relationship between weather conditions and cinema trends, gaining insights into moviegoer behavior and enabling informed decision-making in Paris.

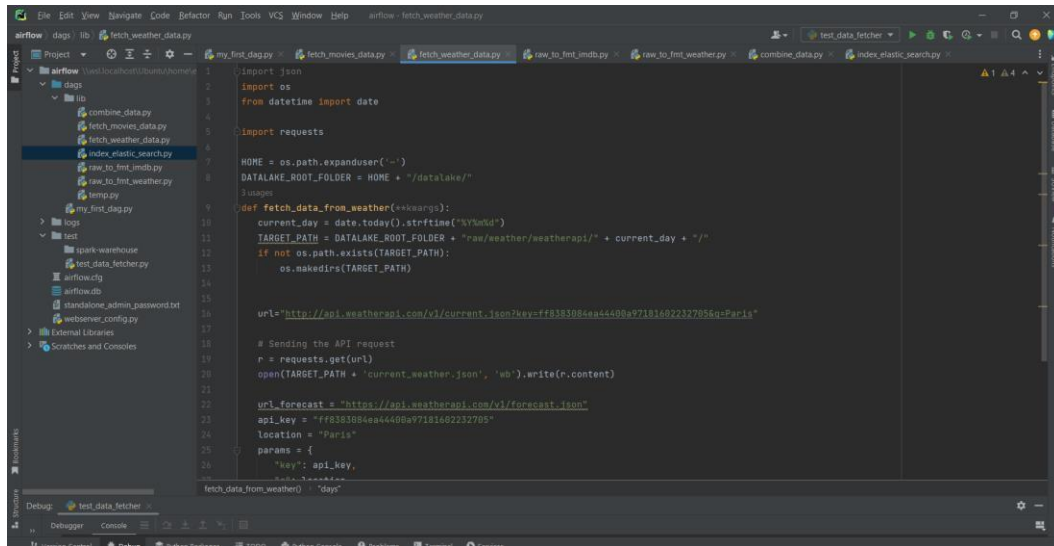


Figure 2: Connection to weather API

Data Lake Organization

We designed our data lake with a structure that adheres to the following naming convention: Each route will look like this:

`/[{layer}]/[{group}]/[{TableName}]/[{date}]/filename`

Layers can include: - **raw** which is the first phase of the data and is straight from the source system, the data is prepared for use by the Datalake in the second stage, which is *formatted*, and *usage* is the final stage before

Group: we use *movies* to store movies file and *weather* to save weather files

Table Name: An item in this folder will always have this name and a similar schema, in raw we use *imdb* for the movies and *weatherapi* for weather.

FileName: `toprated_movies.json`, `upcoming_movies.json`, `trending_movies.json`, `current_weather.json` and `forecast.json` are the file names for the raw data.

Date: should be formatted as `YYYYMMDD`, for instance, `20230611` (11th June 2022).

For formatted file: `toprated_movies.snappy.parquet`, `trending_movies.snappy.parquet`, `upcoming_movies.snappy.parquet`, `current_weather.snappy.parquet` and `forecast.snappy.parquet`.

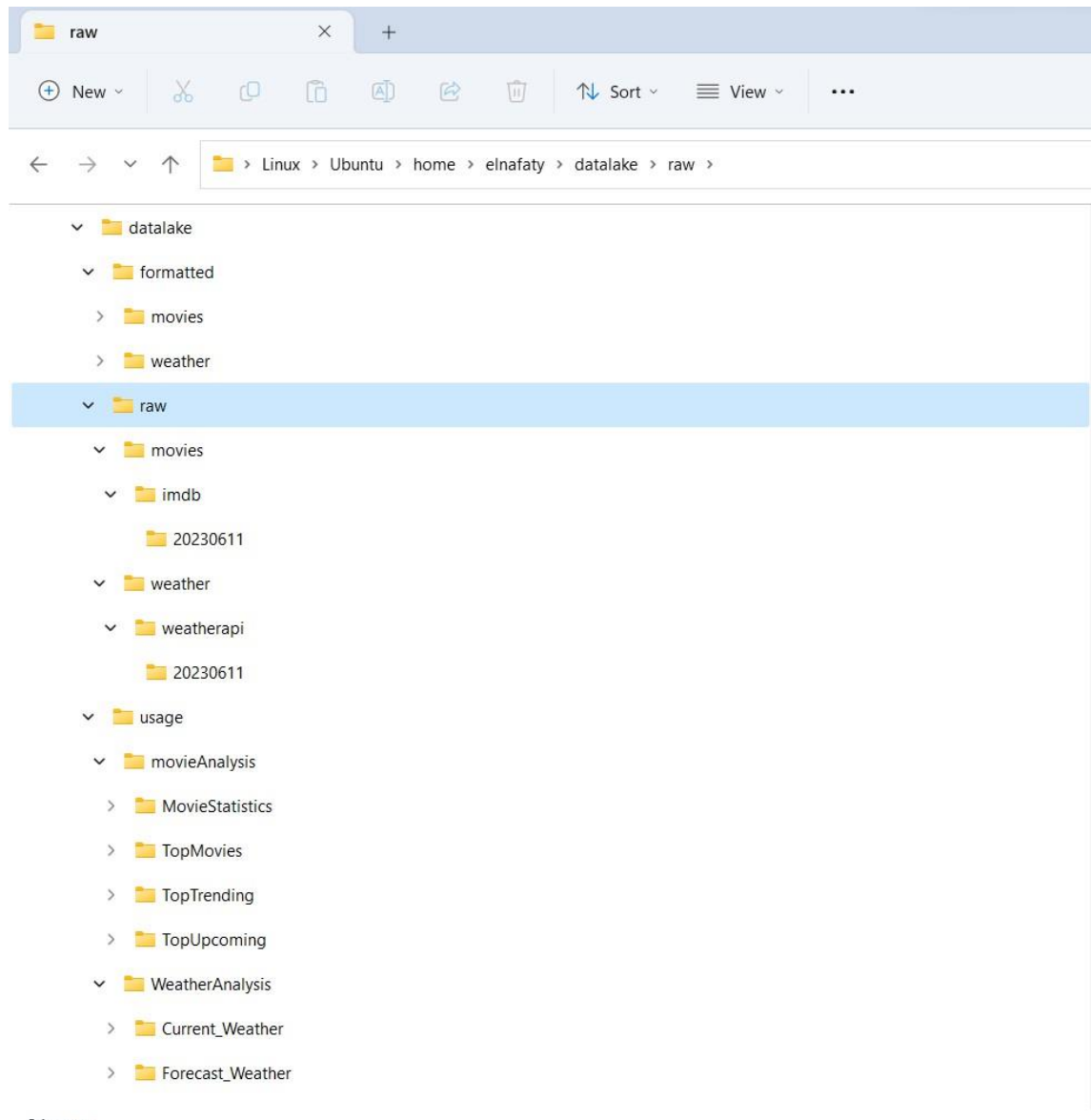


Figure 3: Datalake folder structure

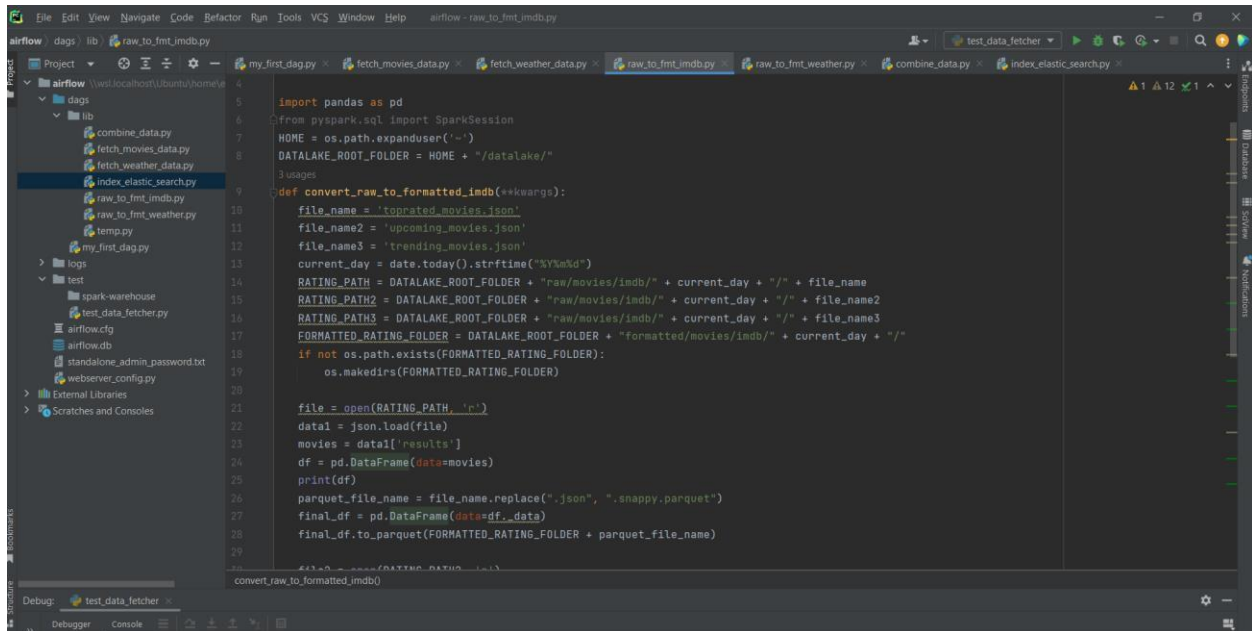
Data Pipeline

The data pipeline plays a crucial role in transferring data from its source to a designated destination, such as a data warehouse. Throughout this journey, the data undergoes transformations and optimizations, ultimately arriving in a state that allows for analysis and the development of valuable business insights. The data pipeline encompasses the essential steps of aggregating, organizing, and moving data effectively.

Data Pipeline Architecture

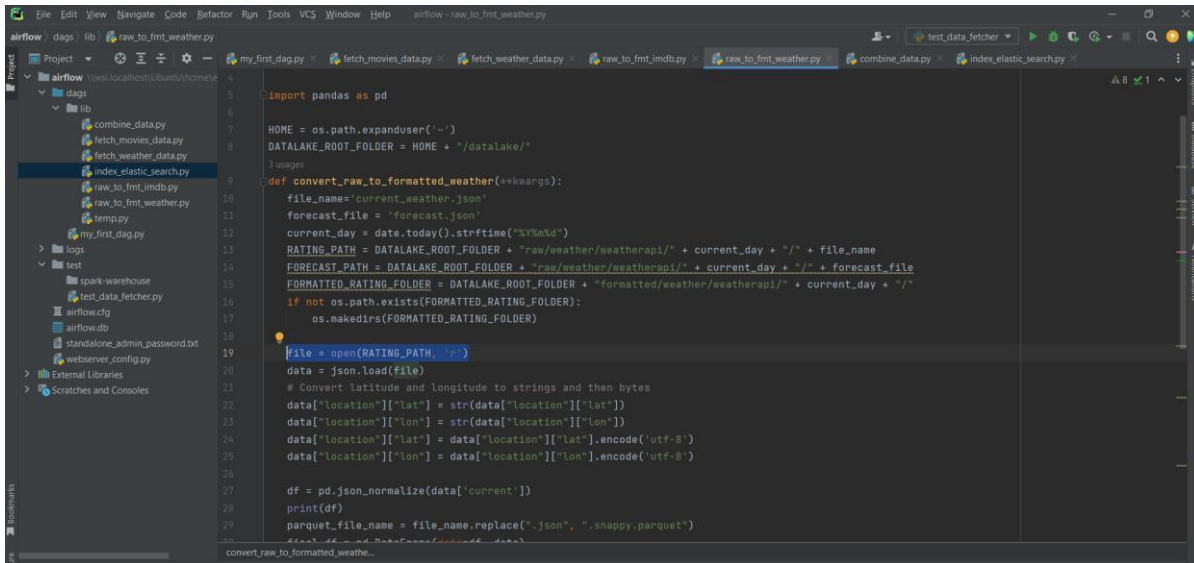
1. **Data Ingestion:** The source data is obtained through APIs. The data sources and ingestion process are detailed in Figure 1 and Figure 2 in the data source section. The collected data is then transferred to the raw folder in the data lake, where it can be further prepared for analysis. Python programming language is used for data ingestion in this project.
2. **Data Formatting:** In this step, the data is aggregated, cleansed, and manipulated to bring it to a standardized format suitable for analysis. Tools like Pandas are utilized to format the data like location, latitude and longitude for the weather. Figure 5 and Figure 6 illustrate the formatting process applied to data from different sources. The formatted data is then converted to the Parquet data format and compressed using snappy, which offers better processing and analysis capabilities.

By following these steps, the data is transformed and prepared in a way that optimizes its usability for subsequent analysis and visualization.



```
airflow - raw_to_fmt_imdb.py
4
5 import pandas as pd
6 from pyspark.sql import SparkSession
7 HOME = os.path.expanduser('~')
8 DATALAKE_ROOT_FOLDER = HOME + "/datalake/"
9
10 def convert_raw_to_formatted_imdb(**kwargs):
11     file_name = 'toprated_movies.json'
12     file_name2 = 'upcoming_movies.json'
13     file_name3 = 'trending_movies.json'
14     current_day = date.today().strftime("%Y%m%d")
15     RATING_PATH = DATALAKE_ROOT_FOLDER + "raw/movies/imdb/" + current_day + "/" + file_name
16     RATING_PATH2 = DATALAKE_ROOT_FOLDER + "raw/movies/imdb/" + current_day + "/" + file_name2
17     RATING_PATH3 = DATALAKE_ROOT_FOLDER + "raw/movies/imdb/" + current_day + "/" + file_name3
18     FORMATTED_RATING_FOLDER = DATALAKE_ROOT_FOLDER + "Formatted/movies/imdb/" + current_day + "/"
19     if not os.path.exists(FORMATTED_RATING_FOLDER):
20         os.makedirs(FORMATTED_RATING_FOLDER)
21
22     file = open(RATING_PATH, 'r')
23     data1 = json.load(file)
24     movies = data1['results']
25     df = pd.DataFrame(data=movies)
26     print(df)
27     parquet_file_name = file_name.replace(".json", ".snappy.parquet")
28     final_df = pd.DataFrame(data=df, data)
29     final_df.to_parquet(FORMATTED_RATING_FOLDER + parquet_file_name)
30
31 # Run the function
32 convert_raw_to_formatted_imdb()
```

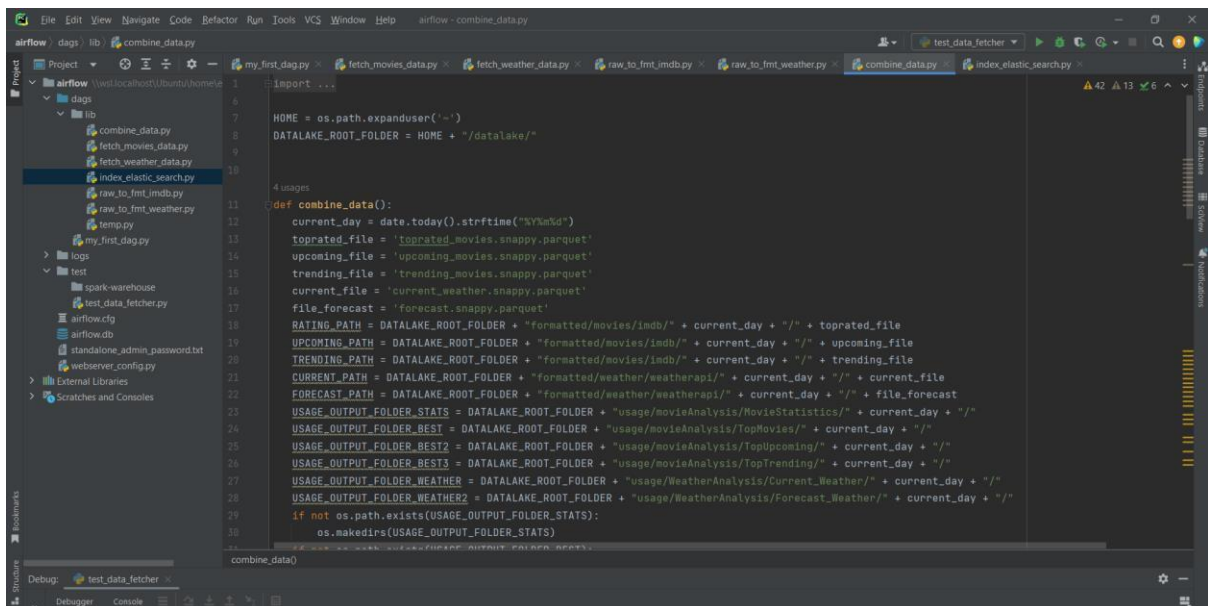
Figure 5: Formatting data from imdb with Panda



```
1 import pandas as pd
2
3 HOME = os.path.expanduser('~')
4 DATALAKE_ROOT_FOLDER = HOME + "/datalake/"
5
6 3 usages
7
8 def convert_raw_to_formatted_weather(**kwargs):
9     file_name = 'current_weather.json'
10    forecast_file = 'forecast.json'
11    current_day = date.today().strftime("%Y%m%d")
12    RATING_PATH = DATALAKE_ROOT_FOLDER + "raw/weather/weatherapi/" + current_day + "/" + file_name
13    FORECAST_PATH = DATALAKE_ROOT_FOLDER + "raw/weather/weatherapi/" + current_day + "/" + forecast_file
14    FORMATTED_RATING_FOLDER = DATALAKE_ROOT_FOLDER + "formatted/weather/weatherapi/" + current_day + "/"
15    if not os.path.exists(FORMATTED_RATING_FOLDER):
16        os.makedirs(FORMATTED_RATING_FOLDER)
17
18    file = open(RATING_PATH, 'r')
19    data = json.load(file)
20    # Convert latitude and longitude to strings and then bytes
21    data["location"]["lat"] = str(data["location"]["lat"])
22    data["location"]["lon"] = str(data["location"]["lon"])
23    data["location"]["lat"] = data["location"]["lat"].encode('utf-8')
24    data["location"]["lon"] = data["location"]["lon"].encode('utf-8')
25
26    df = pd.json_normalize(data['current'])
27    print(df)
28    parquet_file_name = file_name.replace(".json", ".snappy.parquet")
29    # Save the DataFrame as a snappy.parquet file
30    convert_raw_to_formatted_weather...
```

Figure 6: Formatting weather data with Panda

- Data Combination:** Apache Spark is utilized in this step to combine the data from different data sources and performed sql analysis. This combined data becomes the foundation for further analysis. It is indexed and made accessible through the Elasticsearch and Kibana to visualize the data, we use the cloud version of the elasticsearch and kibana

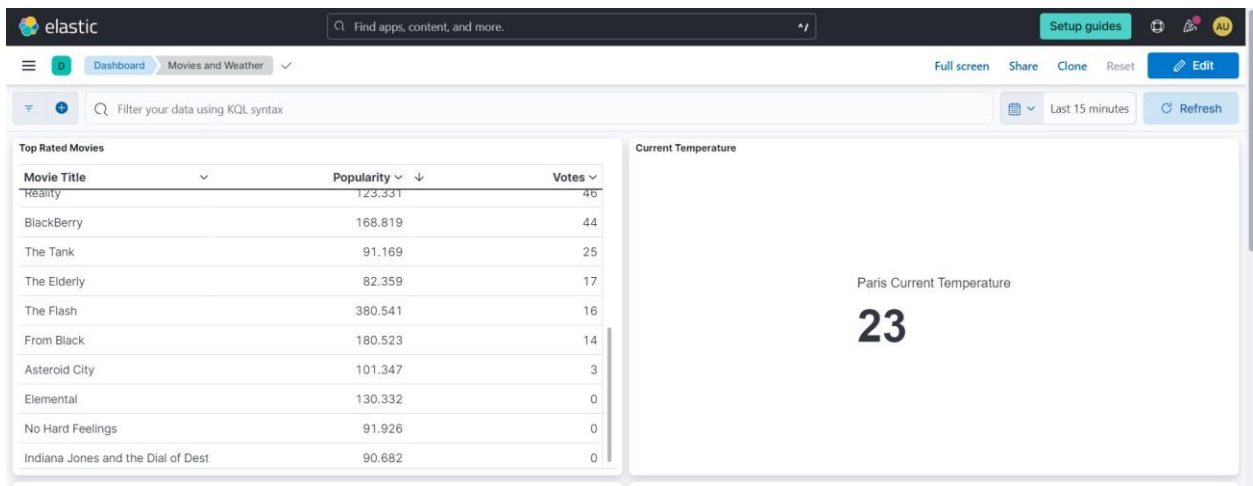


```
1 import ...
2
3 HOME = os.path.expanduser('~')
4 DATALAKE_ROOT_FOLDER = HOME + "/datalake/"
5
6 4 usages
7
8 def combine_data():
9     current_day = date.today().strftime("%Y%m%d")
10    toprated_file = 'toprated_movies.snappy.parquet'
11    upcoming_file = 'upcoming_movies.snappy.parquet'
12    trending_file = 'trending_movies.snappy.parquet'
13    current_file = 'current_weather.snappy.parquet'
14    file_forecast = 'forecast.snappy.parquet'
15    RATING_PATH = DATALAKE_ROOT_FOLDER + "formatted/movies/indb/" + current_day + "/" + toprated_file
16    UPCOMING_PATH = DATALAKE_ROOT_FOLDER + "formatted/movies/indb/" + current_day + "/" + upcoming_file
17    TRENDING_PATH = DATALAKE_ROOT_FOLDER + "formatted/movies/indb/" + current_day + "/" + trending_file
18    CURRENT_PATH = DATALAKE_ROOT_FOLDER + "formatted/weather/weatherapi/" + current_day + "/" + current_file
19    FORECAST_PATH = DATALAKE_ROOT_FOLDER + "formatted/weather/weatherapi/" + current_day + "/" + file_forecast
20    USAGE_OUTPUT_FOLDER_STATS = DATALAKE_ROOT_FOLDER + "usage/movieAnalysis/MovieStatistics/" + current_day + "/"
21    USAGE_OUTPUT_FOLDER_BEST2 = DATALAKE_ROOT_FOLDER + "usage/movieAnalysis/TopUpcoming/" + current_day + "/"
22    USAGE_OUTPUT_FOLDER_BEST3 = DATALAKE_ROOT_FOLDER + "usage/movieAnalysis/TopTrending/" + current_day + "/"
23    USAGE_OUTPUT_FOLDER_WEATHER = DATALAKE_ROOT_FOLDER + "usage/WeatherAnalysis/CurrentWeather/" + current_day + "/"
24    USAGE_OUTPUT_FOLDER_WEATHER2 = DATALAKE_ROOT_FOLDER + "usage/WeatherAnalysis/ForecastWeather/" + current_day + "/"
25    if not os.path.exists(USAGE_OUTPUT_FOLDER_STATS):
26        os.makedirs(USAGE_OUTPUT_FOLDER_STATS)
27
28    combine_data()
```

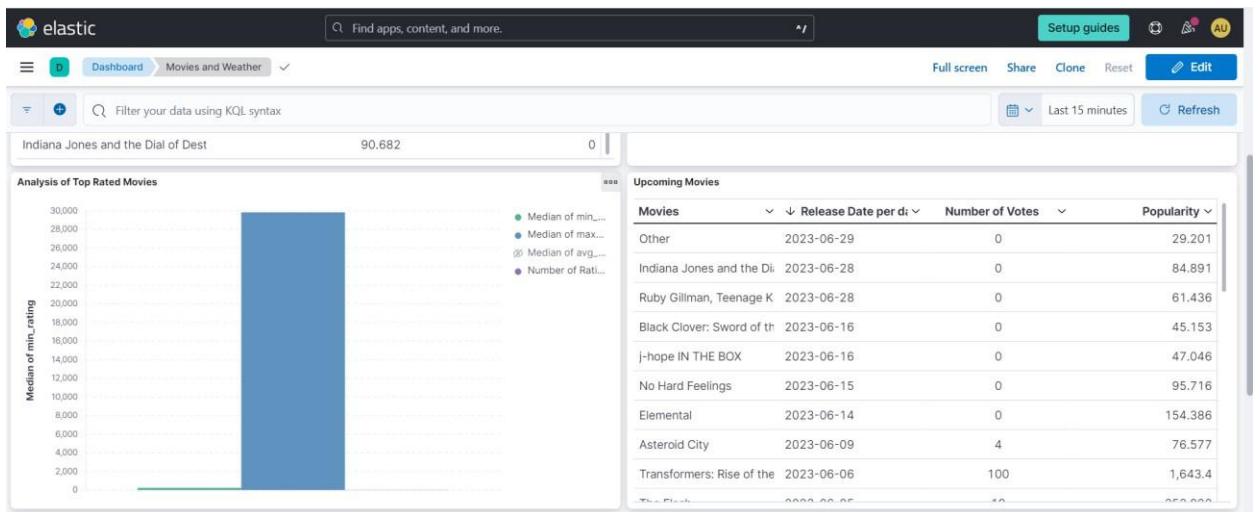
Figure 7: Data combination with Apache Spark


```
36
37 analysis_table = pq.read_table(ANALYSIS_PATH)
38 df_analysis = analysis_table.to_pandas()
39 for _, row in df_analysis.iterrows():
40     document = row.to_dict()
41     es.index(index=analysis_index, body=document)
42
43 upcoming_table = pq.read_table(UPCOMING_PATH)
44 df_upcoming = upcoming_table.to_pandas()
45 for _, row in df_upcoming.iterrows():
46     document = row.to_dict()
47     es.index(index=upcoming_index, body=document)
48
49 trending_table = pq.read_table(TRENDING_PATH)
50 df_trending = trending_table.to_pandas()
51 for _, row in df_trending.iterrows():
52     document = row.to_dict()
53     es.index(index=trending_index, body=document)
54
55 current_table = pq.read_table(CURRENT_PATH)
56 df_current = current_table.to_pandas()
57 for _, row in df_current.iterrows():
58     document = row.to_dict()
59     es.index(index=current_index, body=document)
60
61 forecast_table = pq.read_table(FORECAST_PATH)
62 df_forecast = forecast_table.to_pandas()
63 for _, row in df_forecast.iterrows():
64     document = row.to_dict()
65     es.index(index=forecast_index, body=document)
```

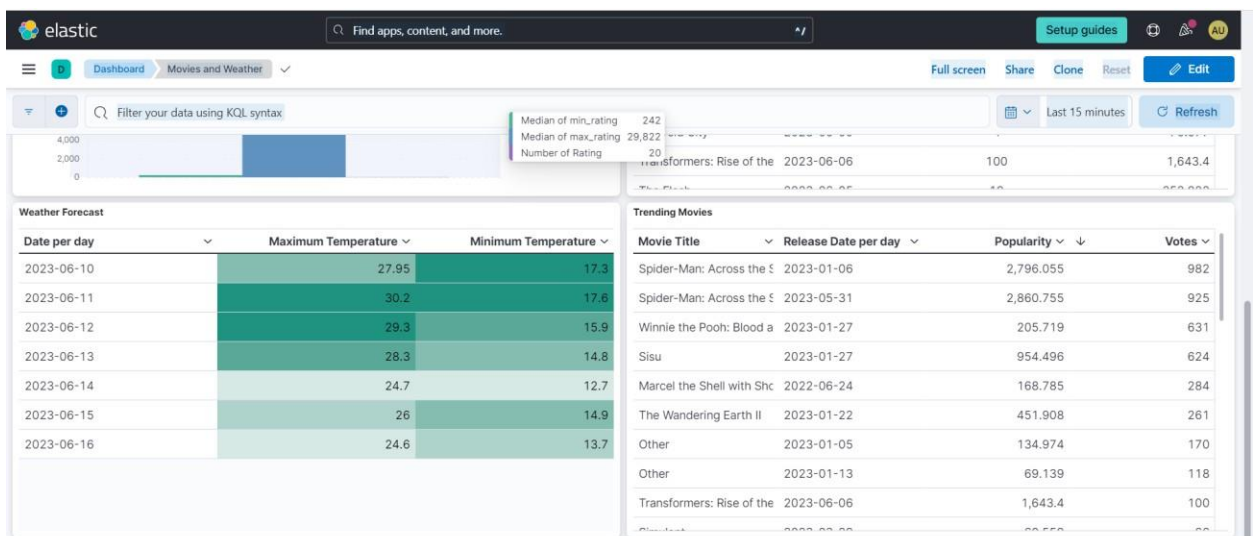
- Usage:** In this project, the prepared data is deployed to production systems, specifically Elasticsearch and Kibana. These tools serve as the backbone for analytics and visualization based on the data indexed. Multiple dashboards are created to cater to various analysis requirements and provide valuable insights. Users of the data, such as consumers, researchers, and statisticians can leverage the visually appealing and informative dashboards to perform a range of analyses. For example, if someone prefers to enjoy outdoor activities before or after watching a movie, they can check the weather forecast to determine the best time to visit the cinema. If it's expected to rain heavily, they might opt for an indoor movie theater instead of an outdoor cinema. On the other hand, if the weather forecast indicates clear skies and pleasant temperatures, they may choose an outdoor cinema for a unique and enjoyable experience. The versatility of Elasticsearch and Kibana empowers users to derive actionable insights from the data and make informed decisions across various domains.



Top Rated Movies & Current Temperature



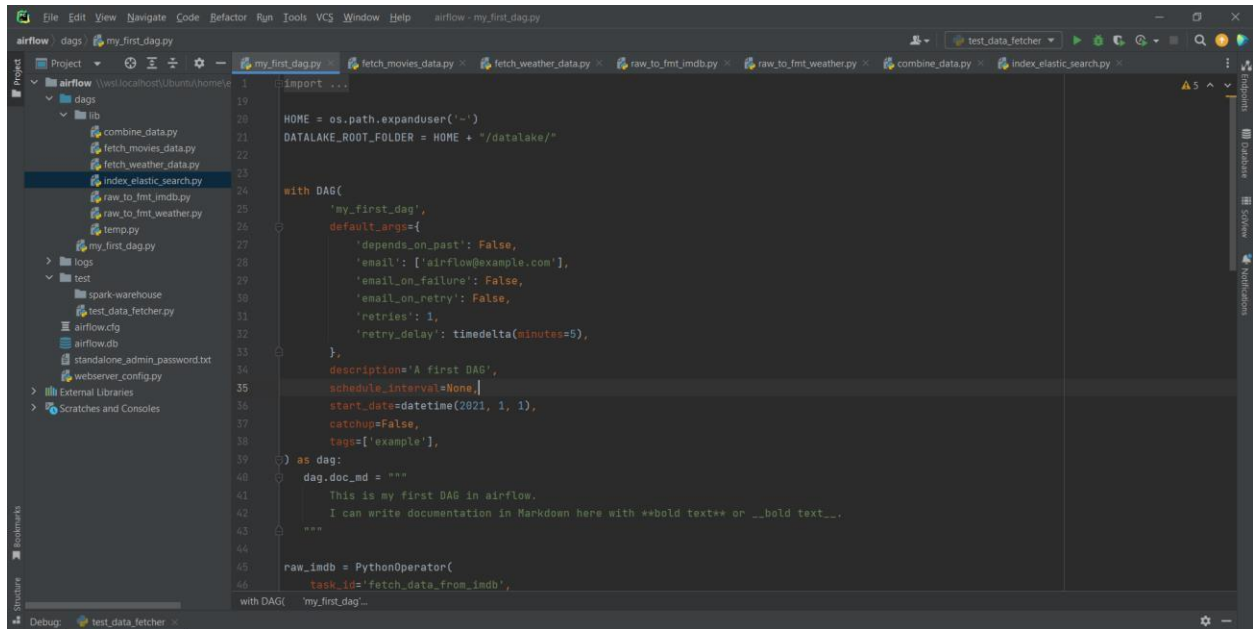
Analysis of Top Rated Movies & Upcoming Movies



Weather Forecast & Trending Movies

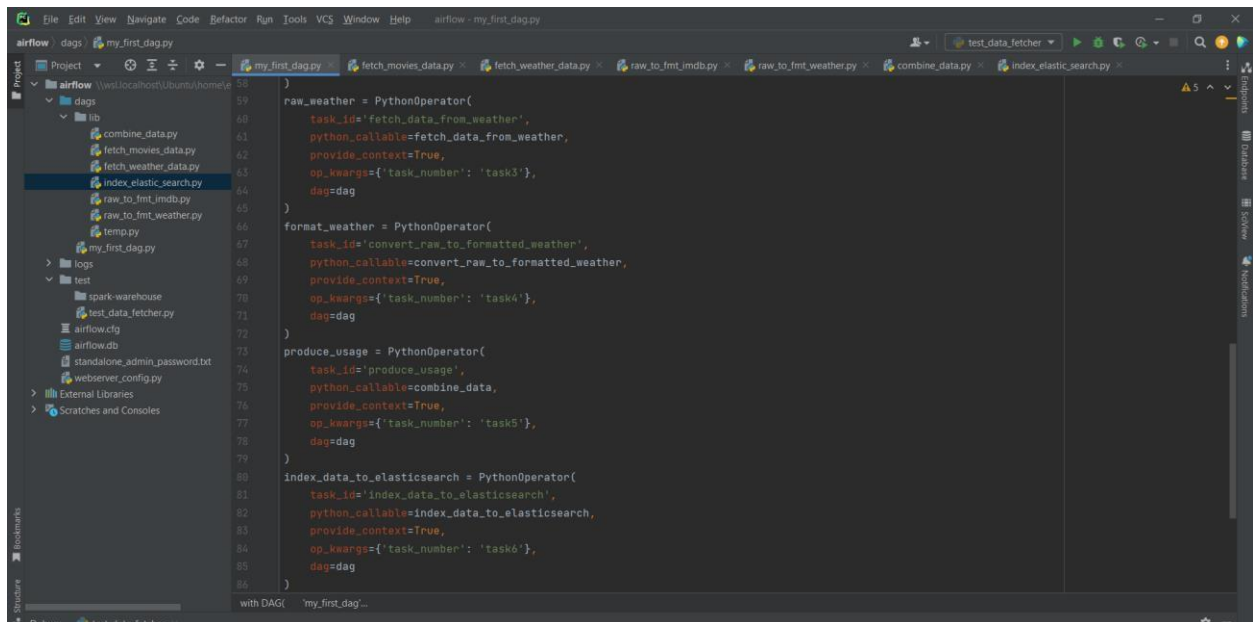
Directed Acyclic Graphs - DAGs

We use Apache Airflow to create DAGs (Directed Acyclic Graphs) to manage workflow orchestration of our data. The tasks and dependencies are defined in python (Figure 10 and 11) and Airflow manages the scheduling and execution. The DAG that reflects our pipeline architecture is shown in Figure 12.



```
1 import sys
2
3 HOME = os.path.expanduser('~')
4 DATA_LAKE_ROOT_FOLDER = HOME + "/datalake/"
5
6
7 with DAG(
8     'my_first_dag',
9     default_args={
10         'depends_on_past': False,
11         'email': ['airflow@example.com'],
12         'email_on_failure': False,
13         'email_on_retry': False,
14         'retries': 1,
15         'retry_delay': timedelta(minutes=5),
16     },
17     description='A first DAG',
18     schedule_interval=None,
19     start_date=datetime(2021, 1, 1),
20     catchup=False,
21     tags=['example'],
22 ):
23     # Example of a task
24     raw_imdb = PythonOperator(
25         task_id='fetch_data_from_imdb',
26         python_callable=fetch_data_from_imdb,
27         provide_context=True,
28         op_kwargs={'task_number': 'task1'},
29         dag=dag
30     )
```

Figure 10: DAG Python Code i



```
31
32
33 raw_weather = PythonOperator(
34     task_id='fetch_data_from_weather',
35     python_callable=fetch_data_from_weather,
36     provide_context=True,
37     op_kwargs={'task_number': 'task2'},
38     dag=dag
39 )
40
41 format_weather = PythonOperator(
42     task_id='convert_raw_to_formatted_weather',
43     python_callable=convert_raw_to_formatted_weather,
44     provide_context=True,
45     op_kwargs={'task_number': 'task3'},
46     dag=dag
47 )
48
49 produce_usage = PythonOperator(
50     task_id='produce_usage',
51     python_callable=combine_data,
52     provide_context=True,
53     op_kwargs={'task_number': 'task4'},
54     dag=dag
55 )
56
57 index_data_to_elasticsearch = PythonOperator(
58     task_id='index_data_to_elasticsearch',
59     python_callable=index_data_to_elasticsearch,
60     provide_context=True,
61     op_kwargs={'task_number': 'task5'},
62     dag=dag
63 )
64
65 with DAG(
66     'my_first_dag',
67     default_args={
68         'depends_on_past': False,
69         'email': ['airflow@example.com'],
70         'email_on_failure': False,
71         'email_on_retry': False,
72         'retries': 1,
73         'retry_delay': timedelta(minutes=5),
74     },
75     description='A first DAG',
76     schedule_interval=None,
77     start_date=datetime(2021, 1, 1),
78     catchup=False,
79     tags=['example'],
80 ):
81     raw_imdb = PythonOperator(
82         task_id='fetch_data_from_imdb',
83         python_callable=fetch_data_from_imdb,
84         provide_context=True,
85         op_kwargs={'task_number': 'task1'},
86         dag=dag
87     )
88     raw_weather = PythonOperator(
89         task_id='fetch_data_from_weather',
90         python_callable=fetch_data_from_weather,
91         provide_context=True,
92         op_kwargs={'task_number': 'task2'},
93         dag=dag
94     )
95     format_weather = PythonOperator(
96         task_id='convert_raw_to_formatted_weather',
97         python_callable=convert_raw_to_formatted_weather,
98         provide_context=True,
99         op_kwargs={'task_number': 'task3'},
100         dag=dag
101     )
102     produce_usage = PythonOperator(
103         task_id='produce_usage',
104         python_callable=combine_data,
105         provide_context=True,
106         op_kwargs={'task_number': 'task4'},
107         dag=dag
108     )
109     index_data_to_elasticsearch = PythonOperator(
110         task_id='index_data_to_elasticsearch',
111         python_callable=index_data_to_elasticsearch,
112         provide_context=True,
113         op_kwargs={'task_number': 'task5'},
114         dag=dag
115     )
```

Figure 11: DAG Python Code ii

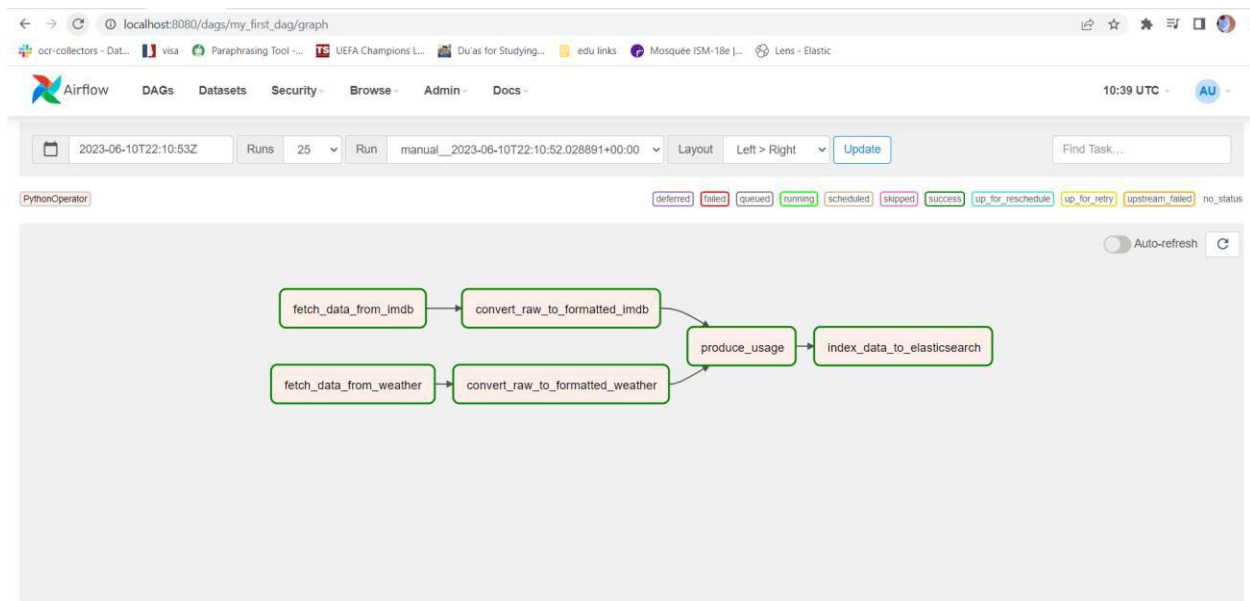


Figure 12 : DAG Airflow Graph View

Conclusion

Throughout the implementation of this project, we have gained extensive knowledge and valuable experience in various technologies. Despite facing challenges along the way, the outcome has been highly successful, providing us with in-depth learnings and honing our skills.

The dynamic and comprehensive visualization of cinema and weather data offers significant benefits to consumers, researchers, statisticians, data scientists, and others, empowering them to make well-informed decisions, perform comparisons, and apply filters based on their specific needs. By considering trending movies, upcoming releases, top-rated films, user ratings, and weather forecasts for the upcoming week, stakeholders can leverage these insights for powerful recommendations.

This project has allowed us to broaden our understanding of key concepts such as big data, data lakes, data transformation, visualization, streaming, data analytics, and workflow. We have become proficient in utilizing various tools and technologies, including Apache Airflow, Apache Spark, cloud-based Elasticsearch and Kibana, Python, Panda, Parquet, and JSON, among others.