YouTube Ad Placement Analytics

Kirill Nartov

Abstract

The objective of this project is to design a cloud-based YouTube analytics platform using Amazon Web Services (AWS). Our goal is to develop a cloud storage solution for managing the raw data extracted from the YouTube API, using Amazon S3 as a repository for all raw data. We will be using AWS Glue to build a data catalog and automate the extraction, transformation, and loading (ETL) process. For handling Data Transformation, we will be using Lambda. We will also leverage AWS's query tool, Athena, to efficiently interact with and analyze the data stored in S3.

At the end of the day, we deploy an interactive web dashboard with use of Amazon QuickSigh, a tool for the Data Visualization and creating Dashboards. This dashboard will empower companies to explore trends, identify popular video types, and make data-driven decisions regarding optimal ad placements, helping them target their audience more effectively and maximize advertising ROI.

Introduction

1.1. Problem Statement

It’s a common headache for any business to decide which communication channels are best for reaching their target audience and maximizing the impact of their advertising campaigns. With YouTube being the second most visited website globally, it is one of the biggest platforms for businesses to connect with potential customers. However, selecting the right video content or channels to place these ads is not that straightforward. There are many factors influencing this, such as popularity, user engagement, trends, category, views and many more. By leveraging AWS, we aim to build an interactive dashboard that can enable businesses to analyze YouTube trends, understand viewer behavior, and optimize their ad allocations. Our dashboard will be a supportive tool for small businesses when they aim to decide what types of videos/channels are the most efficient for ad allocation. Thus, at the end of the day, this tool could help them bring more customers.

1.2. Objectives

The objective of our project are next:

- Create a mechanism for extracting a raw data from the YouTube API and putting this raw data directly on cloud;
- Streamline the extraction, transformation, and loading (ETL) process of converting data into a query-friendly format;
- To design a dashboard for YouTube analytics.
    - This tool would empower companies to make data-driven decisions regarding optimal ad placements, helping them target their audience more effectively and maximize advertising ROI.

1.3. Relevance to Cloud Computing

This project aligns with Cloud Computing due to use of variety of web and AWS services (such as API, S3, Glue, Lambda, Athena, etc.) and important concepts (e.g., databases, data catalogs, layers for Lambda functions, JSON format, Tabular format, etc.) that were needed for accomplishing this project.

2. Background and Related Work

For such projects, it's recommended to build a data lake [3] that passes data through different stages of preprocessing (e.g., 'raw data', 'transformed data', 'enriched data') since it increases the transparency and flexibility of projects.

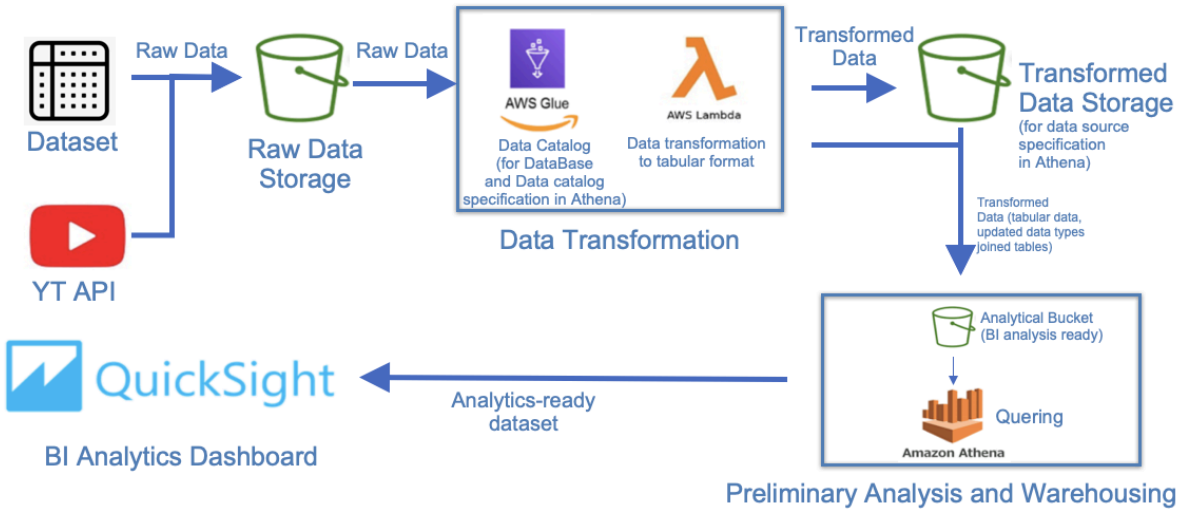
The following key technologies are used for this project:

- **S3** as repositories for the data;
- **Lambda** for handling the Data Transformation steps;
- **AWS Glue** for building a data catalog and for ETL process;
- **QuickSigh** for the Data Visualization and Dashboards.

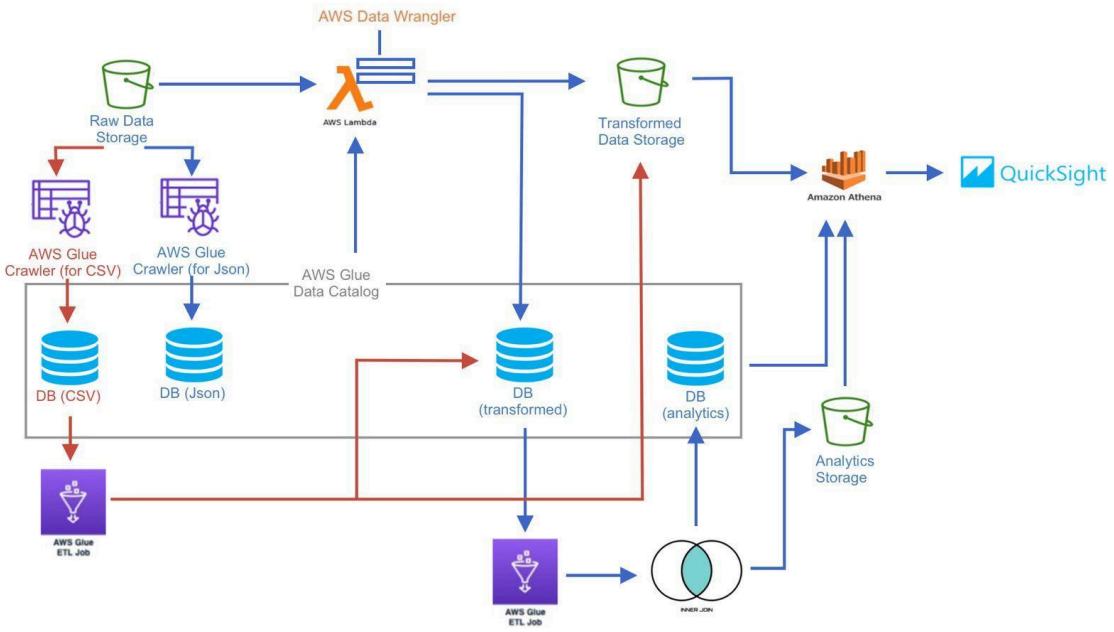
3. System Design and Implementation

3.1. System Architecture

Here is a high-level diagram of the architecture:



Here is more granular diagram of the architecture:



3.2. Technologies Used

The following technologies are used for this project:

- **YouTube API** for enriching the initial dataset with up-to-date data;
- **S3** as repositories for the raw data and the transformed data;
- **AWS CLI** for populating S3 with raw data;
- **IAM** for managing a role/access policy;

- **Lambda** for handling the Data Transformation step as the data is being converted to a query-friendly tabular format and triggering this procedure every time our S3 bucket receives new input;
- **AWS Glue (Crawler)** for building a data catalog for the data, for updating data types of fields in the dataset;
- **AWS Glue (Studio)** for joining data tables as a part of ETL process;
- **AWS Athena** for running queries against a data-catalog in Glue;
- **QuickSigh** for the Data Visualization and Dashboards;

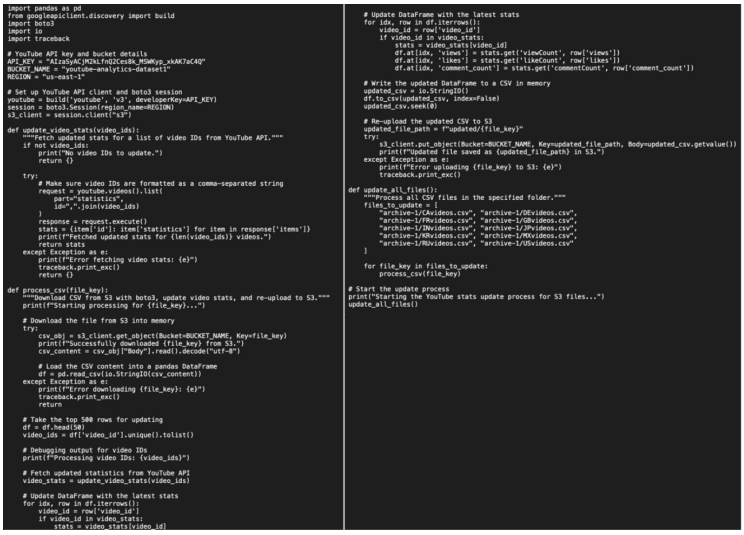
3.3. Implementation Details

We have implemented our system by taking the following steps:

- Step 1: Data Receivening
- Step 2: Data Transformation
    - Converting .json into Parquet
    - Adding .csv files in to the raw DB
    - Updating Data Types in Parquet file
    - Converting .csv to parquet
    - Joining the Tables
- Step 3: Querying

Let's walk through the steps.

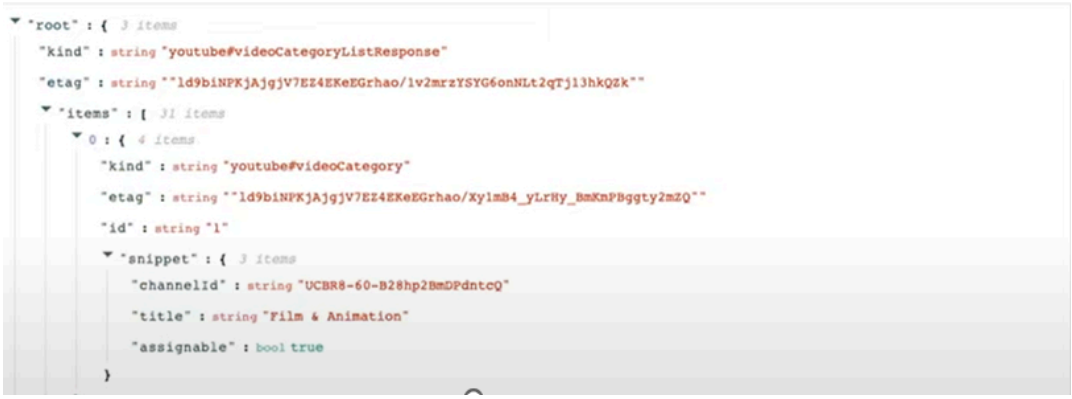
Step 1: Data Receivening
We have an initial dataset [4] containing two files: json and csv. A .Json file maps ids from the .csv file to human readable category id's.
Aside from use of the dataset, we decided to update videos in this dataset with up-to-date statistics data. We have created a script that was taking an up-to-date data from YouTube API and use CLI to upload it to the S3.
Here is the screenshot of this script:



```
import pandas as pd
from googleapiclient.discovery import build
import boto3
import io
import traceback

# YouTube API key and bucket details
API_KEY = "AIzaSyACjM2kLfnQ2esBk_MSWKyp_xAK7aC4Q"
BUCKET_NAME = "youtube-analytics-dataset1"
REGION = "us-east-1"

# Set up YouTube API client and boto3 session
youtube = build('youtube', 'v3', developerKey=API_KEY)
session = boto3.Session(region_name=REGION)
s3_client = session.client("s3")

def update_video_stats(video_ids):
    """Fetch updated stats for a list of video IDs from YouTube API."""
    if not video_ids:
        print("No video IDs to update.")
        return {}

    try:
        # Make sure video IDs are formatted as a comma-separated string
        request = youtube.videos().list(
            part="statistics",
            id=",".join(video_ids)
        )
        response = request.execute()
        stats = {item['id']: item['statistics'] for item in response['items']}
        print(f"Fetched updated stats for {len(video_ids)} videos.")
        return stats
    except Exception as e:
        print(f"Error fetching video stats: {e}")
        traceback.print_exc()
        return {}

def process_csv(file_key):
    """Download CSV from S3 with boto3, update video stats, and re-upload to S3."""
    print(f"Starting processing for {file_key}...")

    # Download the file from S3 into memory
    try:
        csv_obj = s3_client.get_object(Bucket=BUCKET_NAME, Key=file_key)
        print(f"Successfully downloaded {file_key} from S3.")
        csv_content = csv_obj["Body"].read().decode("utf-8")

        # Load the CSV content into a pandas DataFrame
        df = pd.read_csv(io.StringIO(csv_content))
    except Exception as e:
        print(f"Error downloading {file_key}: {e}")
        traceback.print_exc()
        return

    # Take the top 500 rows for updating
    df = df.head(50)
    video_ids = df['video_id'].unique().tolist()

    # Debugging output for video IDs
    print(f"Processing video IDs: {video_ids}")

    # Fetch updated statistics from YouTube API
    video_stats = update_video_stats(video_ids)

    # Update DataFrame with the latest stats
    for idx, row in df.iterrows():
        video_id = row['video_id']
        if video_id in video_stats:
            stats = video_stats[video_id]
            df.at[idx, 'views'] = stats.get('viewCount', row['views'])
            df.at[idx, 'likes'] = stats.get('likeCount', row['likes'])
            df.at[idx, 'comment_count'] = stats.get('commentCount', row['comment_count'])

    # Write the updated DataFrame to a CSV in memory
    updated_csv = io.StringIO()
    df.to_csv(updated_csv, index=False)
    updated_csv.seek(0)

    # Re-upload the updated CSV to S3
    updated_file_path = f"updated/{file_key}"
    try:
        s3_client.put_object(Bucket=BUCKET_NAME, Key=updated_file_path, Body=updated_csv.getvalue())
        print(f"Updated file saved as {updated_file_path} in S3.")
    except Exception as e:
        print(f"Error uploading {file_key} to S3: {e}")
        traceback.print_exc()

def update_all_files():
    """Process all CSV files in the specified folder."""
    files_to_update = [
        "archive-1/CAvideos.csv", "archive-1/DEvideos.csv",
        "archive-1/FRvideos.csv", "archive-1/GBvideos.csv",
        "archive-1/INvideos.csv", "archive-1/JPvideos.csv",
        "archive-1/KRvideos.csv", "archive-1/MXvideos.csv",
        "archive-1/RUvideos.csv", "archive-1/USvideos.csv"
    ]

    for file_key in files_to_update:
        process_csv(file_key)

# Start the update process
print("Starting the YouTube stats update process for S3 files...")
update_all_files()
```

Step 2: Data Transformation (converting .json into Parquet)

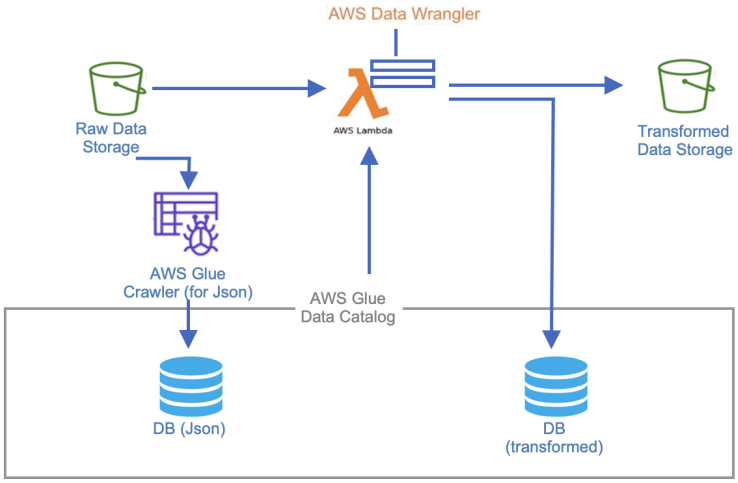
So, why do we need format converting? The thing is our end goal is to query data with Athena and create a dashboard out of the gained data. However, Athena understands .json data in the format containing one pair “key:value” per, but our data is not in the appropriate format (it contains 3 pairs per block):




```
"root" : {  3 items
  "kind" : string "youtube#videoCategoryListResponse"
  "etag" : string ""ld9biNPKjAjgjV7EZ4EKeEGrhao/1v2mrzYSYG6onNLt2qTjl3hkQZk""
  "items" : [  31 items
    0 : {  4 items
      "kind" : string "youtube#videoCategory"
      "etag" : string ""ld9biNPKjAjgjV7EZ4EKeEGrhao/Xy1mB4_yLrHy_BmKmPBggty2mZQ""
      "id" : string "1"
      "snippet" : {  3 items
        "channelId" : string "UCBR8-60-B28hp2BmDPdntcQ"
        "title" : string "Film & Animation"
        "assignable" : bool true
      }
```

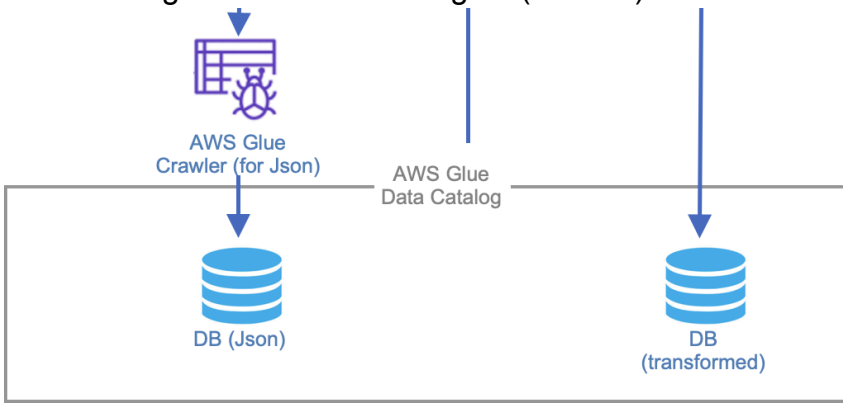

Apache Parquet format is a suitable option. Hence we will be converting .json to Parquet format, using Lambda function.

Here is the initial flow of the converting:

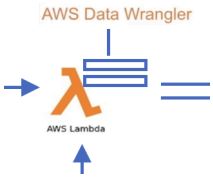


Let’s dive into it.

First, we create data catalog and DB with AWS glue (crawler):



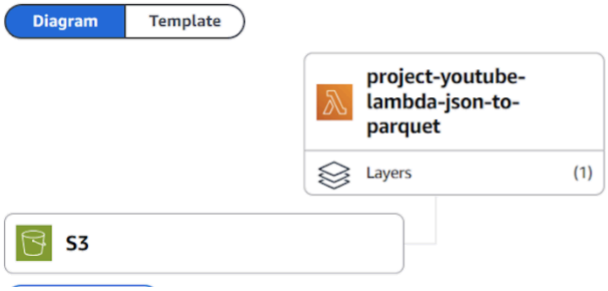
Second, we do creating Lambda with use of AWS Data Wrangler:



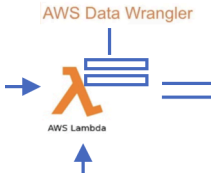
with this code:

```python
# Creating DF from content
df_raw = wr.s3.read_json('s3://{}/{}'.format(bucket, key))

# Extract required columns:
df_step_1 = pd.json_normalize(df_raw['items'])

# Write to S3
wr_response = wr.s3.to_parquet(
    df=df_step_1,
    path=os_input_s3_cleansed_layer,
    dataset=True,
    database=os_input_glue_catalog_db_name,
    table=os_input_glue_catalog_table_name,
    mode=os_input_write_data_operation
)
```

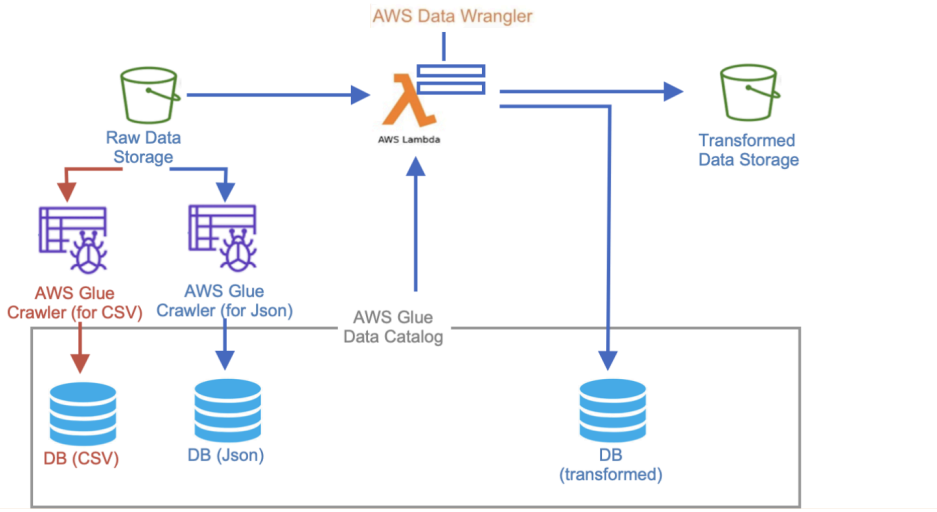
Third, we implement S3 based trigger for our Lambda, such that if new data is added to the bucket, it should run the the lambda function:



This still happens in this part of the flow:



Step 2: Data Transformation (adding .csv files in to the raw DB)
After that we do all these steps again, but for csv file (highlighted with red color):



Step 2: Data Transformation (updating Data Types in Parquet file)
Here we change data types in the cleaned parquet file(which was created from the lambda):

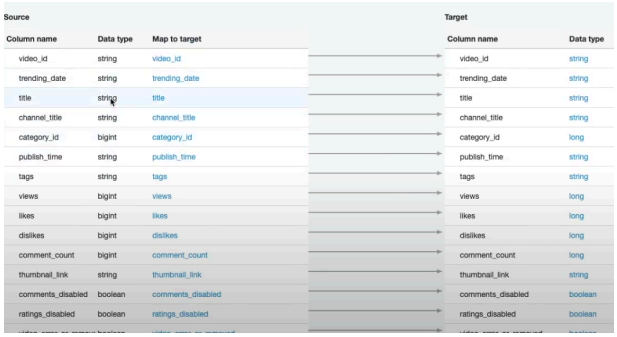
Schema (6)

View and manage the table schema.

Filter schemas

#	Column name	Data type	Partitior
1	kind	string	-
2	etag	string	-
3	id	bigint	-
4	snippet_channel_id	string	-
5	snippet_title	string	-
6	snippet_assignable	boolean	-

Step 2: Data Transformation (converting .csv to parquet)
This part of Data Transformation includes transforming the .csv files present the raw database to .parquet files and save them in the new transformed database and s3 bucket:

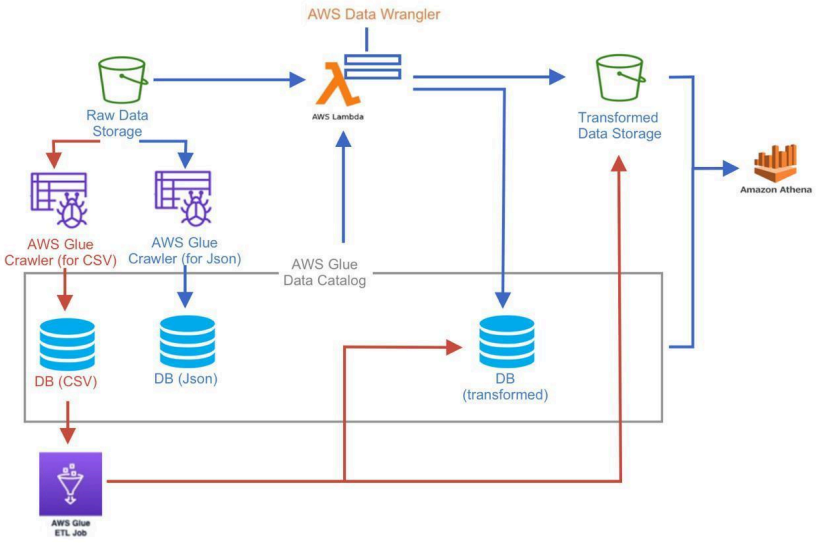


For this, we do the next:

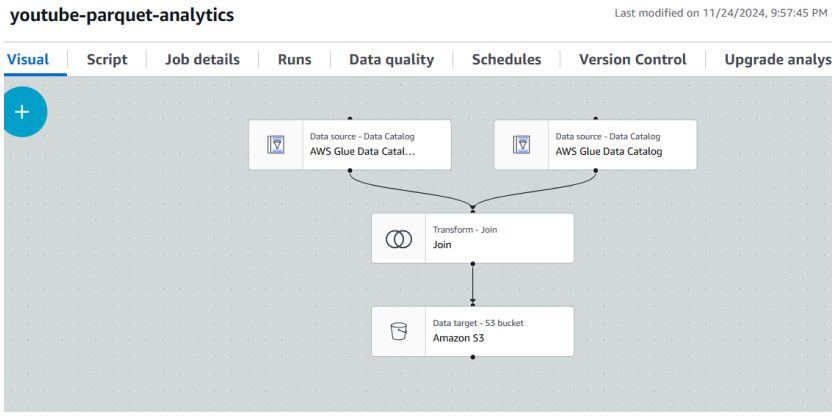
- Create a glue job to convert .csv to Parquet format.

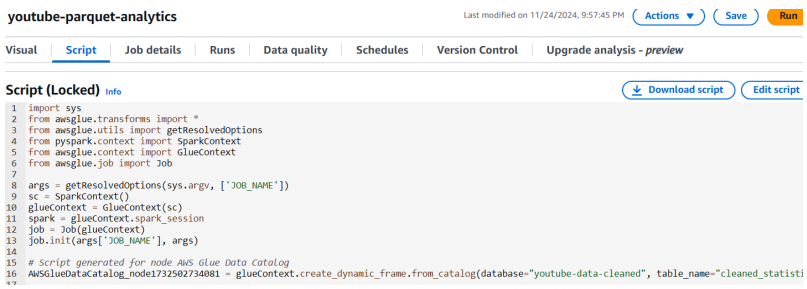
- Transform the data types in the target database. (Made changes such as long was converted to Bigint etc.) and modified schema.
- Optimize the job so that all files are written at once and “Partition” is created (Partitioned by region).

That is, we end up having this flow:



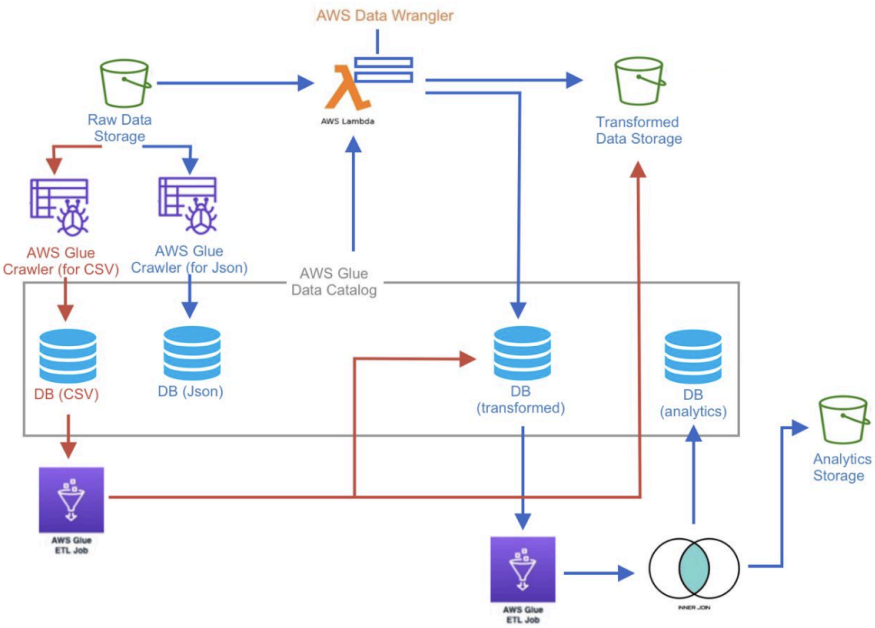
Step 2: Data Transformation (joining the Tables)
Now that the .parquet files are placed in a transformed bucket, we can go ahead and perform a join operation on the parquet files created by the .json and .csv files. We create a new job in AWS glue, which performs a join on these 2 tables. We gave the partition keys as region and ID:





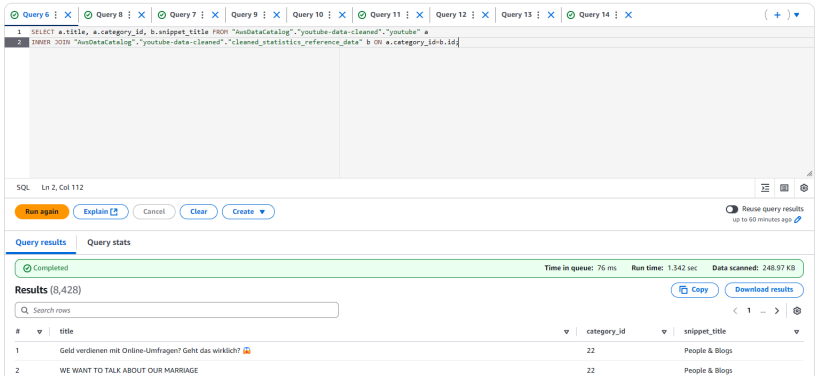
We create a new bucket called “Analytics Bucket”. This bucket stored the output of the join.

That is, this is our flow for now:

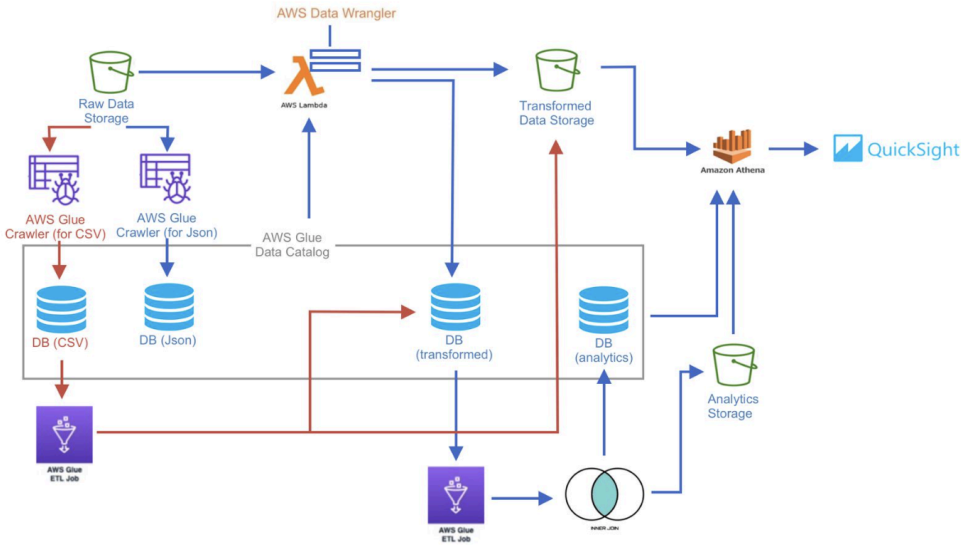


Step 3: Querying

Then, we do querying the Data with Athena:



That is, after we are done with querying and connecting Athena, we end up having this final flow:



4. Demo Details

4.1. Demo Setup

- **Hardware Used:**
  - **Local Machine**: Our Local i7 machine was used for development for creating, testing, and debugging AWS Lambda packages and scripts.
  - **AWS CLI**: The AWS Command Line Interface (CLI) was used on the local machine to interact with AWS services for tasks such as uploading data to S3 and triggering workflows.
- **Software Used:**
  - **Python 3.8**: Python 3.8 was used for developing Lambda functions and creating the necessary ETL scripts. The choice of version was due to compatibility with AWS Data Wrangler.
- **Manually Creating the Lambda Package:**

To prepare the Lambda function for transforming data, we created the package manually as follows:

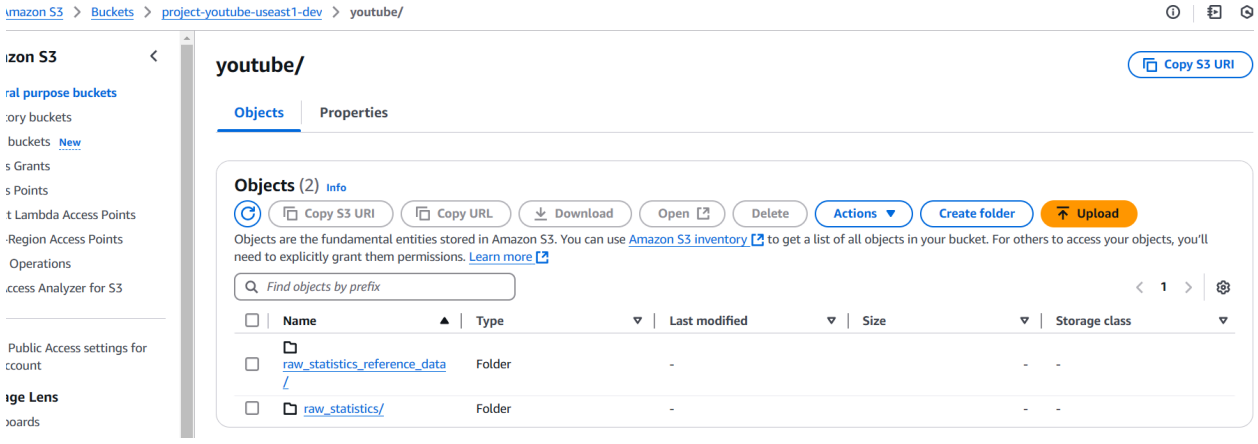
1. **Dependencies Installation**: Installed all required Python libraries, including AWS Data Wrangler, in a local environment.
2. **Environment Setup**: Used a local virtual environment to ensure version consistency and prevent conflicts with other Python projects.
3. **Packaging Process**:
    - All necessary dependencies and scripts were bundled into a single ZIP file.
    - Special attention was given to compatibility, ensuring the package worked with AWS Lambda's Python 3.8 runtime environment.
4. **Uploading to AWS**: The ZIP package was manually uploaded to Lambda via the AWS Management Console.

- **Services Used:**

- **S3 Bucket**: Served as the primary storage repository for raw, transformed, and analytical datasets.
- **Lambda**: Handled data transformations, such as converting JSON to Parquet and triggering ETL processes automatically.
- **AWS Glue**: Enabled ETL operations, including creating a data catalog, updating data schemas, and joining datasets.
- **AWS Athena**: Provided querying capabilities for the transformed datasets stored in S3.
- **IAM**: Managed roles and permissions to ensure secure access to AWS services.
- **QuickSight**: Used for visualizing data and building an interactive analytics dashboard.

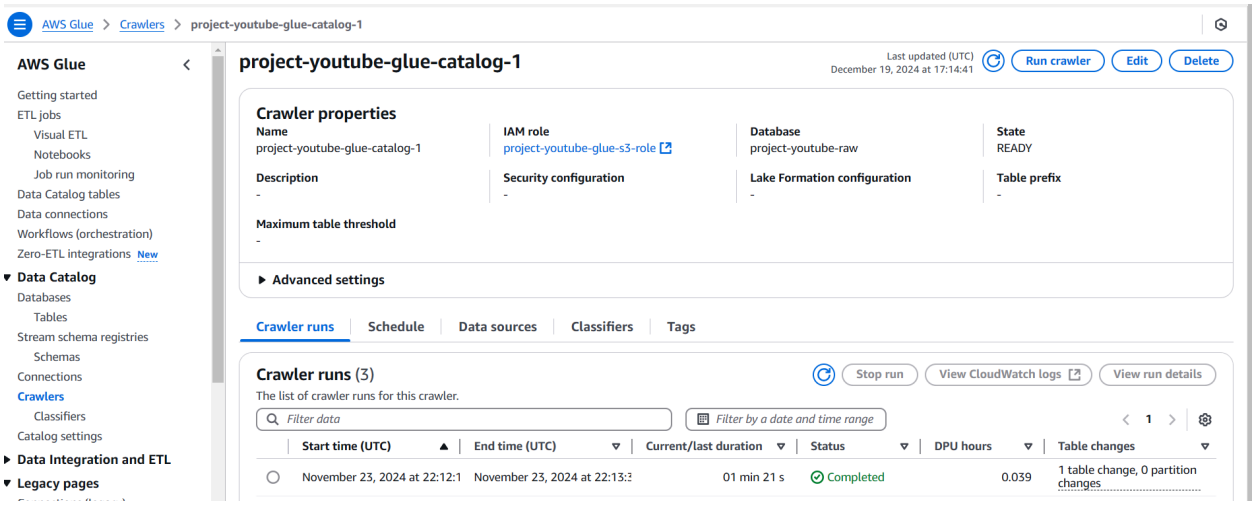
4.2. Demo Walkthrough

The project starts with storing the data files (.json and .csv files) in the raw s3 bucket:
Initial S3 Bucket:



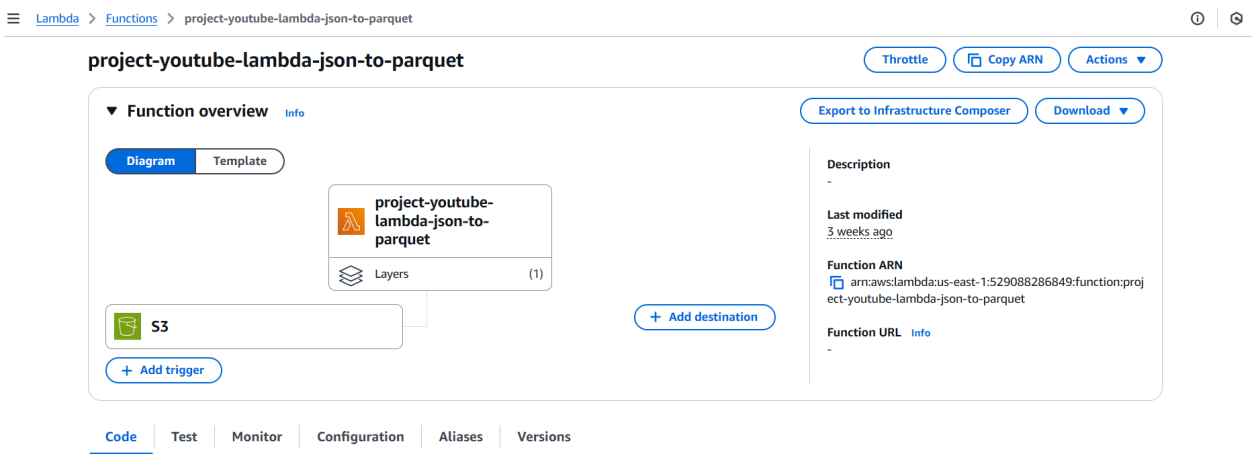
Here Raw data files which are in .csv and .json format are stored

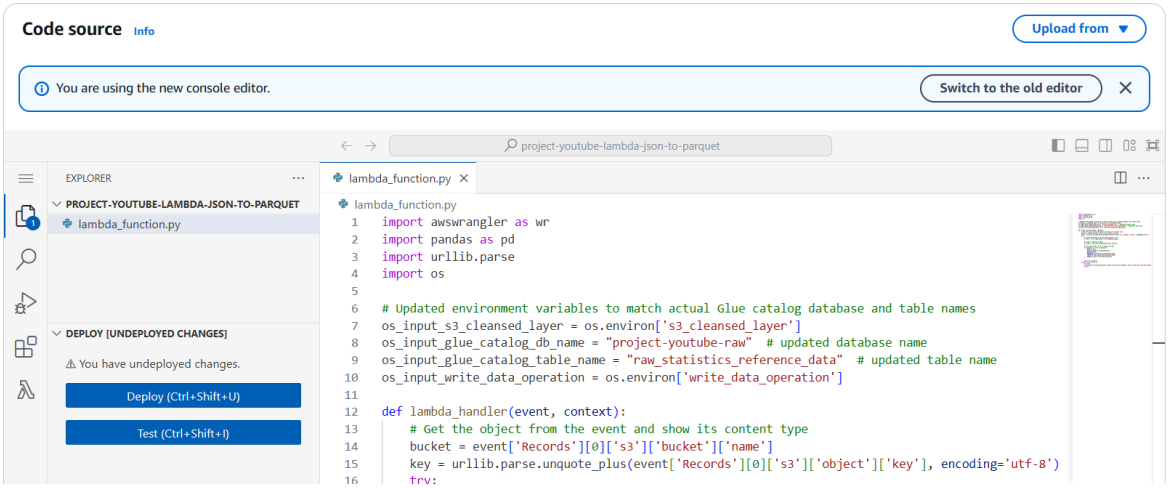
AWS GLUE JOB for creating raw database.



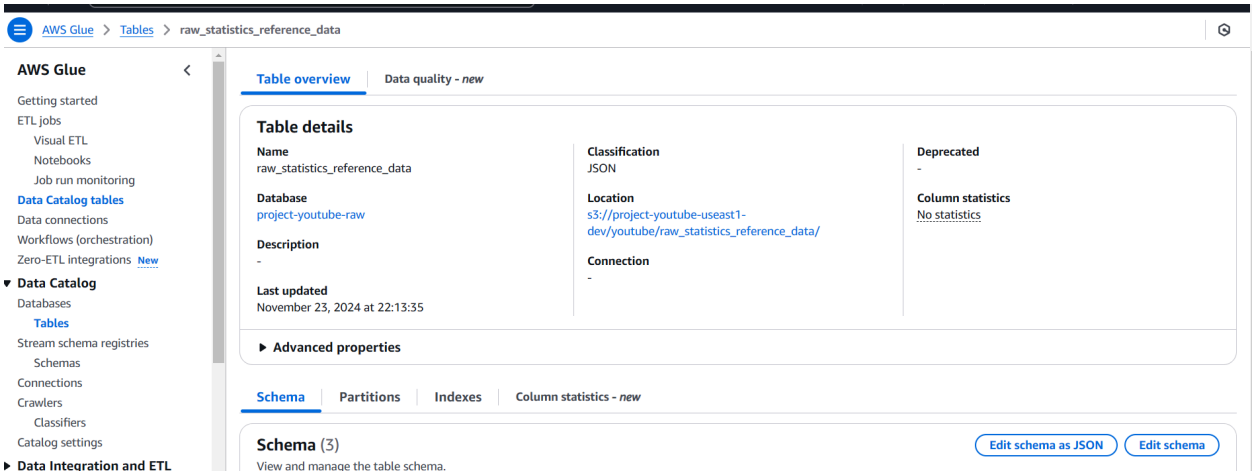
.

AWS Lambda :

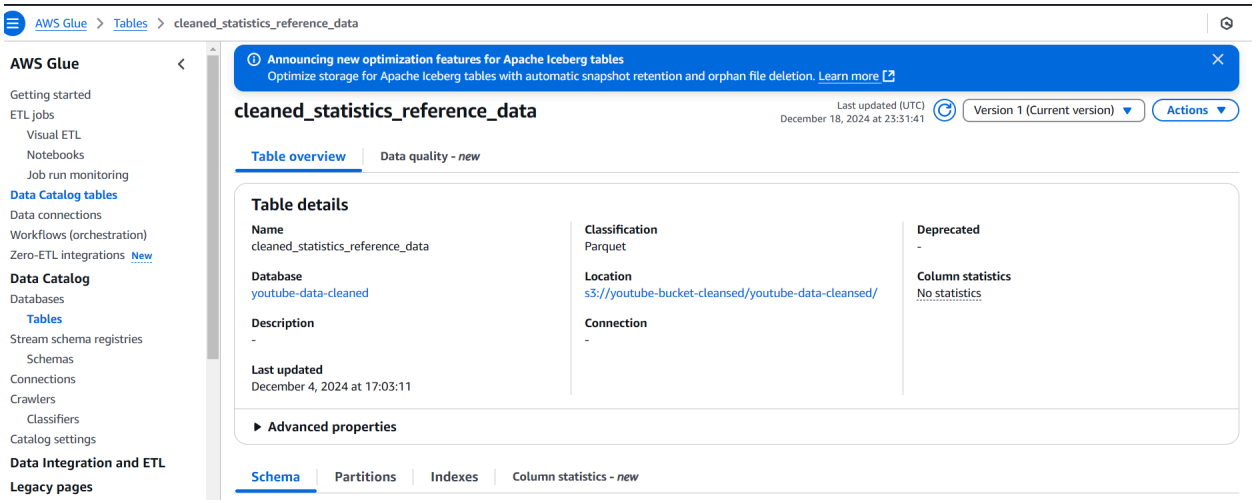




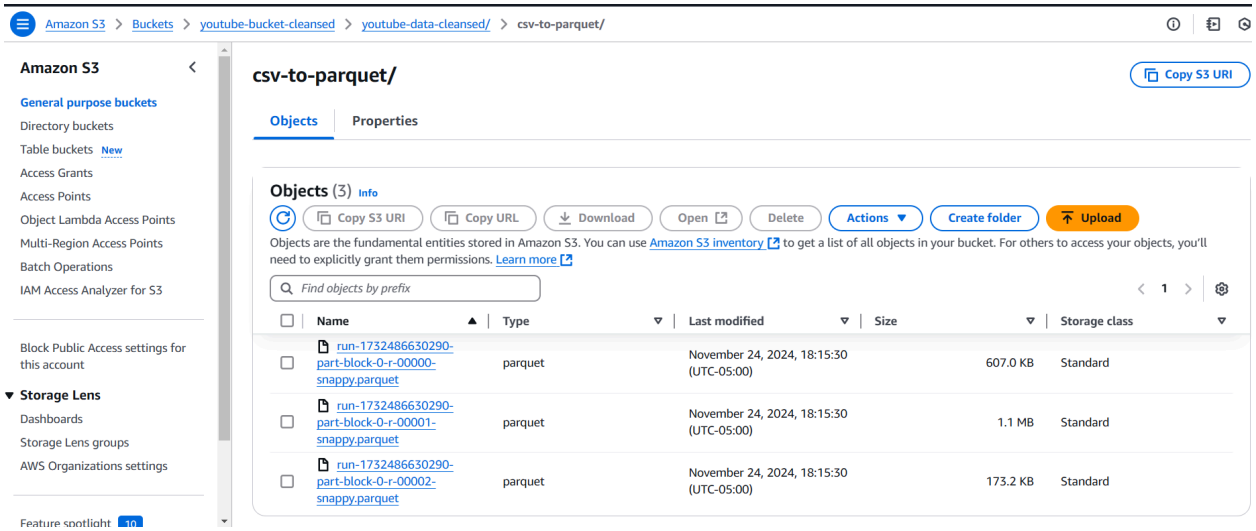
The lambda code is used to convert .csv to parquet format. Here we manually created a layer for Lambda which was called the AWS Data Wrangler layer.
The lambda stores the values in the s3 cleaned bucket and creates a database for the raw data in AWS Glue. Here we also added the s3 trigger to bucket.



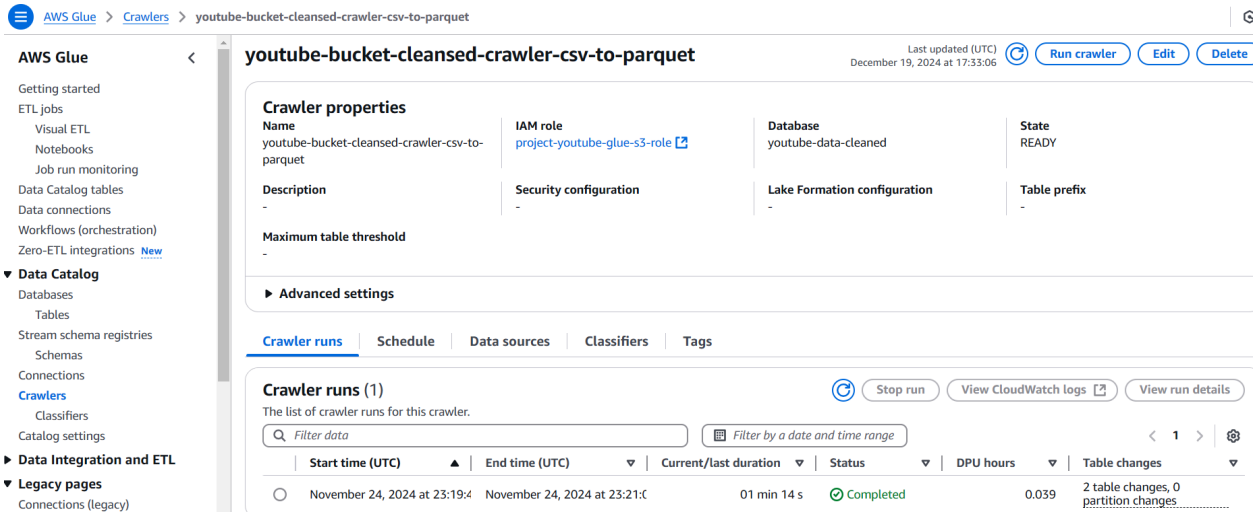
Created a cleaned version for the database, where we changed the data types



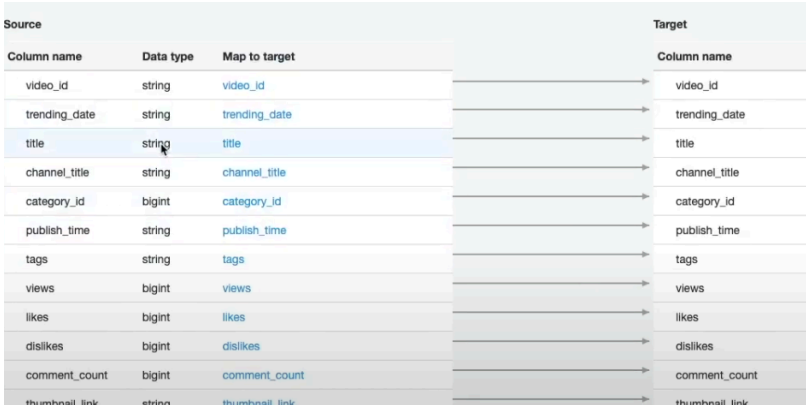
Used lambda function to convert .csv to parquet file and stored the new files in a new bucket called youtube-bucket-cleaned.



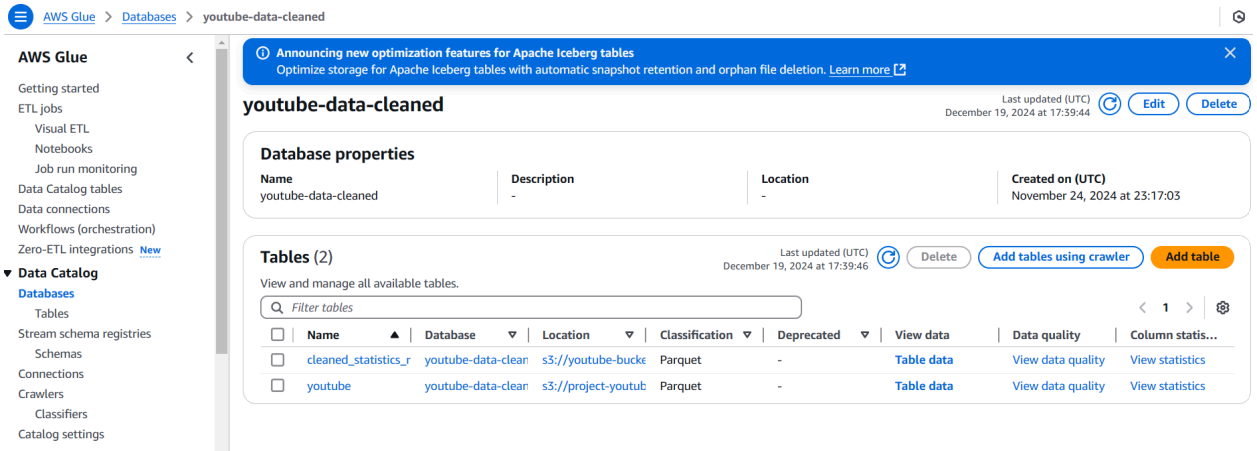
Next, we created a glue crawler to create a database out from the s3 bucket “youtube-bucket-cleaned”.



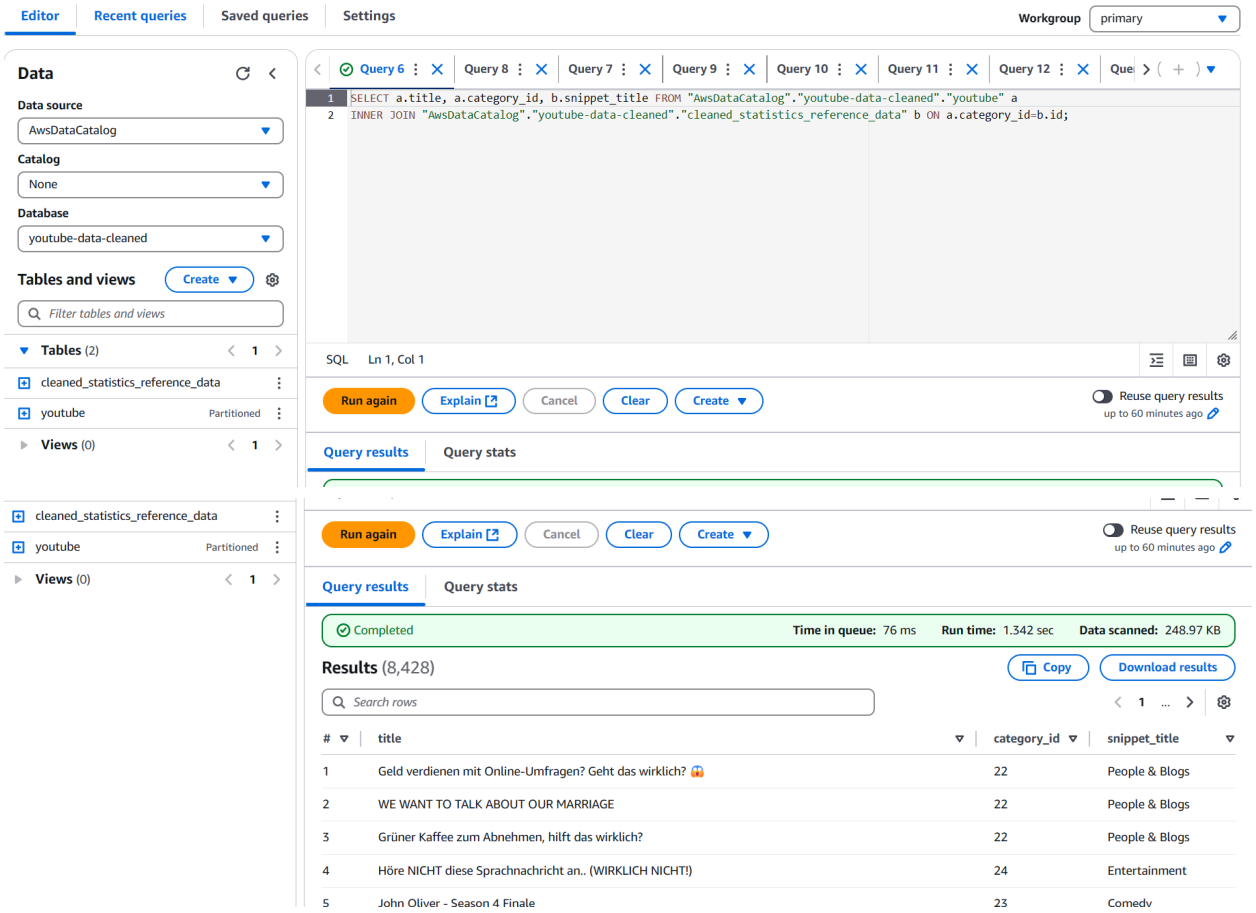
Changed the data types of the columns:



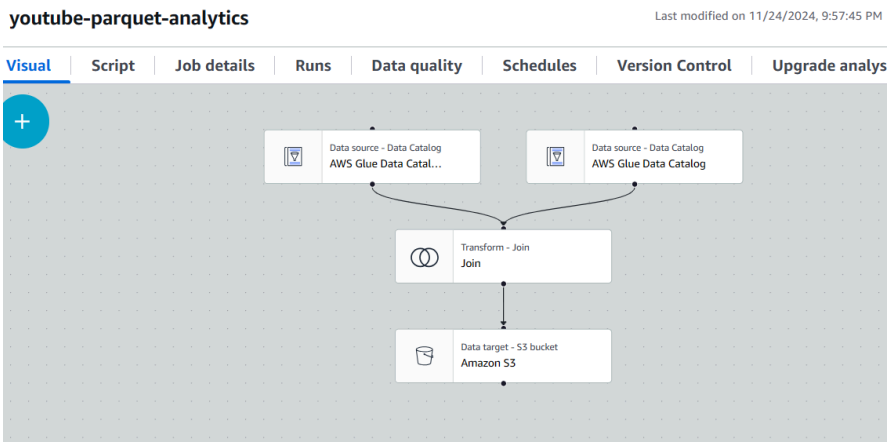
Cleaned Database:



Querying through **AWS ATHENA**:



Creating a **GLUE ETL JOB** to join the 2 parquet files created and saving the output to a new bucket called analytical bucket. Also created a new database youtube-parquet-analytics.



youtube-parquet-analytics

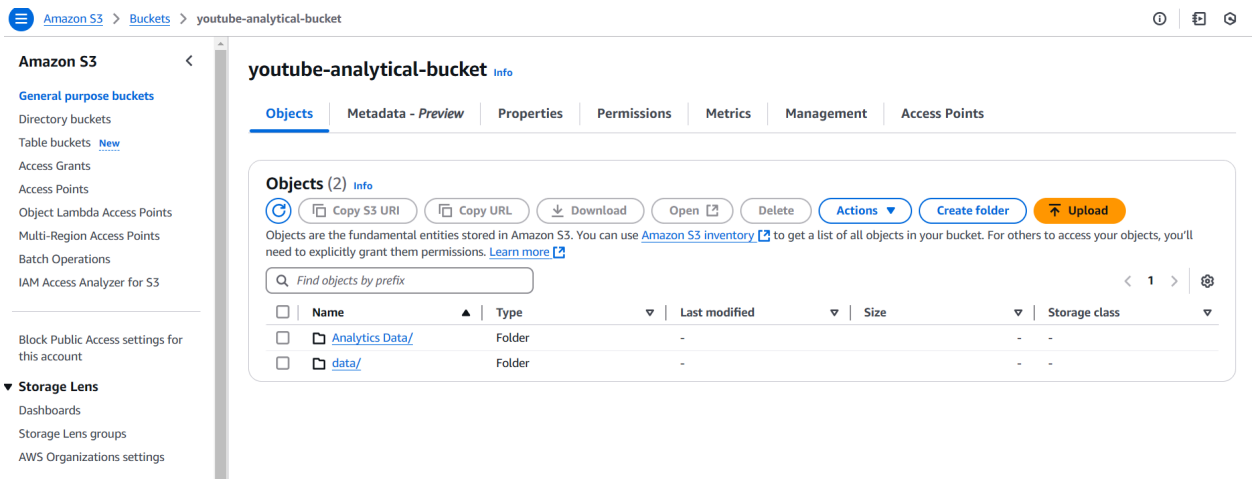
Last modified on 11/24/2024, 9:57:45 PM Actions Save Run

Visual | Script | Job details | Runs | Data quality | Schedules | Version Control | Upgrade analysis - *preview*

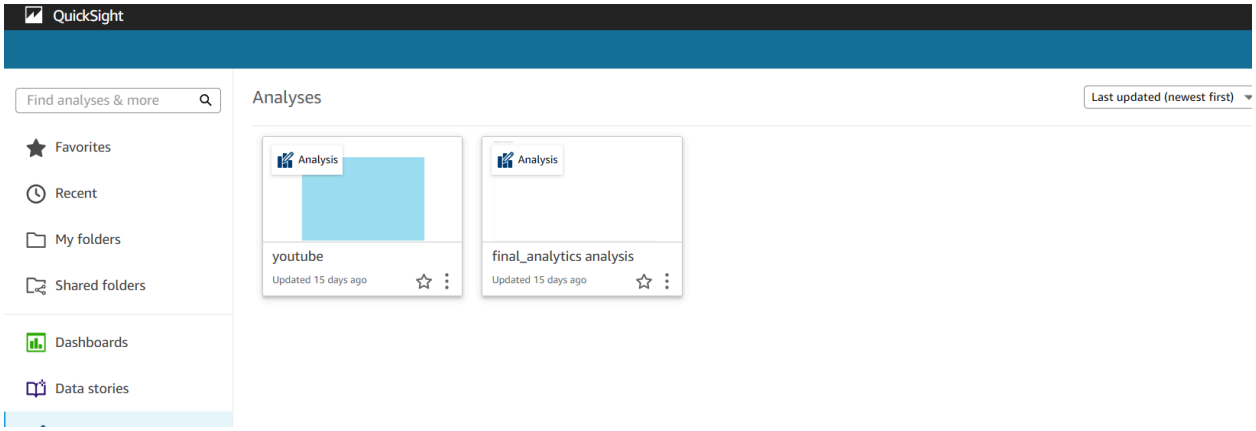
Script (Locked) Info

Download script Edit script

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

# Script generated for node AWS Glue Data Catalog
AWSGlueDataCatalog_node1732502734081 = glueContext.create_dynamic_frame.from_catalog(database="youtube-data-cleaned", table_name="cleaned_statisti
```



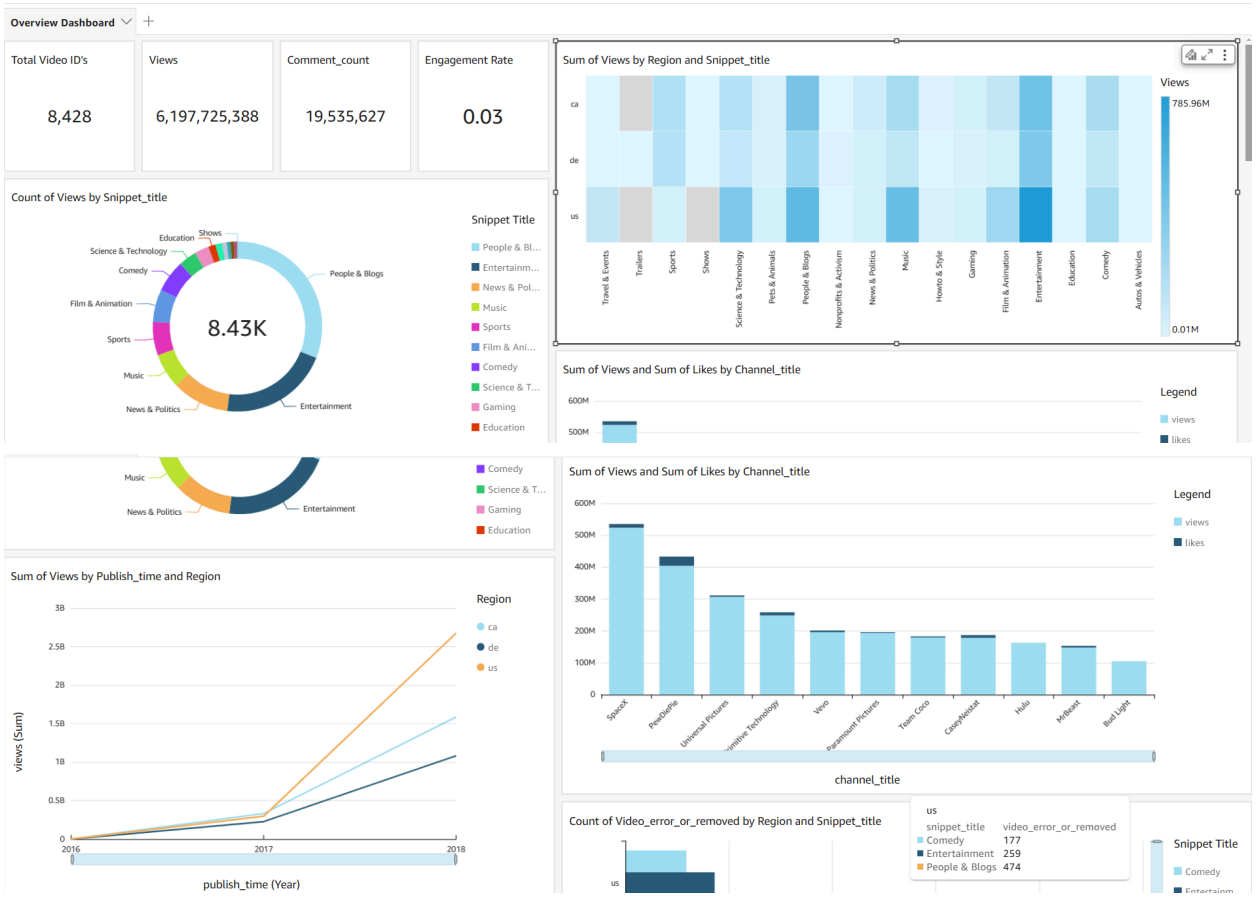
Used Quicksight for analytics. We connected Quicksight to AWS Athena

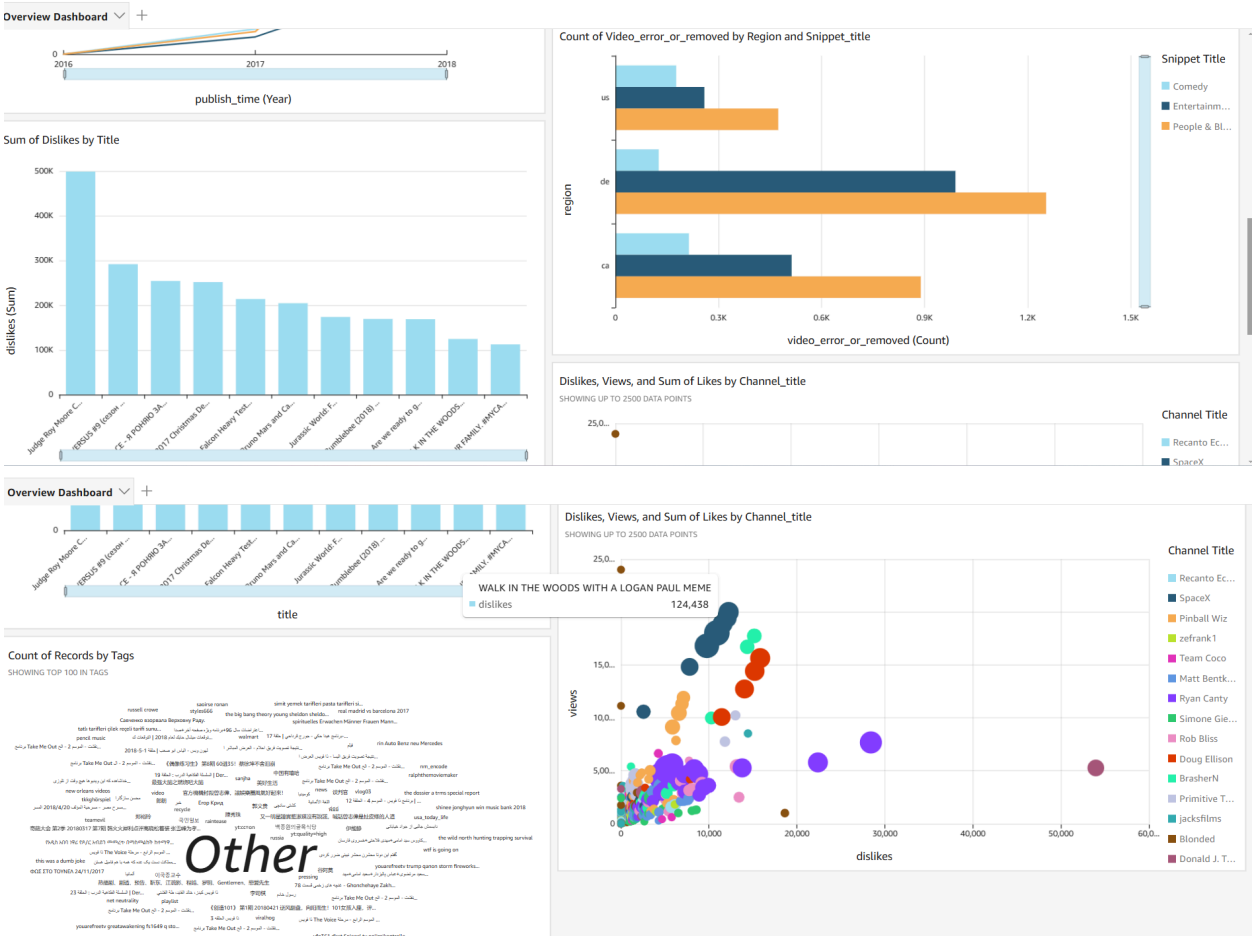


4.3. Expected Outcomes

Link to the Dashboard (it requires all users to create an account before the start of use): https://us-east-1.quicksight.aws.amazon.com/sn/dashboards/4e5f0d59-4fa3-430e-9c9b-9e1da61a0ea1

The outcome of our project is a functioning dashboard that shows YouTube statistics analytics. Here are the screenshots of this dashboard:





This dashboards offers the same interactive functionality as other common BI tools (e.g., Tableau, PowerBI) which includes such features as update of content based on click event, filtering and selecting of specific data points and etc.

5. Challenges and Limitations

- Dataset Creation using Youtube API:
    - We had a template dataset which we got from kaggle, and we decided to create a python script which would update the rows of the template Dataset with current data. We performed this using YouTube API which works in Json format. We uploaded the template dataset and created a script which we ran from Local CLI to update the rows, we encountered multiple errors at first and later when the code was fixed,we realized that updating 25000 rows will not be feasible as it would take many hours and we would even encounter costs for the YouTube API, so we just updated the top 500 rows in each table;
- Converting JSON to “Parquet” format (query-friendly tabular format):

    - There was an AWS layer called Data Wrangler which could convert JSON to Parquet format, but it was not present in AWS anymore. So we created a manual layer package and installed all the dependencies and created a zip file which we manually uploaded as a Lambda Layer. There were still many issues related to the python version as the AWS wrangler package does not work on python versions above 3.8. We downgraded the python version and finally we were able to fix the issue;
- Challenges with Lambda function:
    - There were several general errors requiring resolving while developing and testing the Lambda function that converts data into a tabular, query-friendly format, e.g., time-our error and AWS Glue permission restrictions;
- Updating data types of fields in a tabular data:
    - Parquet files are generated with headers, so we couldn't just change data types of fields in AWS glue, we also needed to delete an old file in S3 bucket as a first step;
- Creating analytical bucket:
    - Faced issue with creating joins (data_type issue) and determining what should be the best partion_keys for the analytical bucket which could help us in analyzing better.

6. Conclusions and Future Work

6.1. Conclusions

In conclusion, we have created an analytical marketing tool in a form of a web dashboard. Here are some findings from the analytics with this tool:

- Top Viewed Categories: The categories "People and Blogs" and "Entertainment" recorded the highest viewership, followed closely by "News" and "Music";

- Best Ad Placement Categories (US): In the United States, "Shows" and "Trailers" were the most effective categories for ad placements based on viewership data;

- Limited Ad Placement Value: Categories like "Education" and "Science & Technology" were less effective for ad placements, except when the advertised product or service was directly relevant to these fields;

- Viewership Trend: Video views experienced an exponential increase starting in 2017;

- Content Removal by Region: Denmark emerged as the region with the highest intolerance, with the most videos taken down;

- Content Removal in the US: In the United States, the "Comedy" category had the highest percentage of videos removed;

- Low Ad Engagement Channels: Although channels like Paramount and Hulu achieved high viewership rates, their engagement levels were low, making them less ideal for ad placements.

6.2. Future Work

Our future scope includes the next:

- Making an automatic update of youtube statistics data from API on a schedule base;

- Expand the list of updating video IDs from 500 to 1000 by optimizing the requests schedule;

- Enriching the dashboard with information from the regions that were not currently included;

References

1. https://developers.google.com/youtube/v3/getting-started
2. https://aws.amazon.com/blogs/big-data/
3. https://aws.amazon.com/blogs/big-data/build-a-data-lake-foundation-with-aws-glue-and-amazon-s3/
4. https://www.kaggle.com/datasets/datasnaek/youtube-new
5. https://aws.amazon.com/quicksight/features/