YouTube Analysis End-To-End Data Engineering Project using Python and AWS

PROBLEM DEFINITION:
The YouTube Data Analysis project addresses the challenges associated with harnessing insights from the vast and diverse landscape of YouTube data. The issues encompass the need to handle, organize, and extract meaningful information from the ever-expanding pool of videos, channels, and user interactions. It is essential for a company to perform Data Analysis, so that it can predict how well their YouTube video would do, or find ways to deploy more customer-centric marketing strategies by learning how to categorize their videos based on various factors like views, comments etc.

OBJECTIVE:
The objective of this project is to harness the wealth of information within the Kaggle dataset, focused on daily popular YouTube videos across various regions. Our aim is to design and implement a robust data management and analysis system, which presents an opportunity to explore patterns, preferences, and variations in video popularity within the YouTube platform, and to gain deeper insights into the dynamics of YouTube trends and audience interactions across diverse geographical locations.

DATASET:
https://www.kaggle.com/datasets/datasnaek/youtube-new
This dataset offers a rich repository of information, allowing for detailed analyses of YouTube trends, user engagement, and content categorization across diverse regions and over an extended period.

The Kaggle dataset in question comprises CSV files containing statistical information about daily popular YouTube videos spanning several months. Each day, up to 200 trending videos are documented for various locations, with each region having its own dedicated file. The dataset provides a comprehensive set of details for each video, including the video title, channel title, publication time, tags, views, likes, dislikes, description, and comment count. One distinctive feature is the inclusion of a category_id field, and this field varies depending on the geographical region. The category_id information is stored in a JSON file linked to each specific region, offering insights into the thematic categorization of trending videos in different locations.

ANALYSIS AND STRATEGIC GOALS:

- Which category videos were most liked, viewed or commented upon by the audience? Analyze the distribution of trending videos across different categories to uncover the most prevalent content genres and their respective engagement levels.

- Region-wise analysis of likes, views, or comments of different categories of videos. Compare and contrast trends and user behaviors across different regions, highlighting cultural and regional nuances in content consumption.

- Explore relationships between views, likes, dislikes, comments, and the time of publication to discern patterns in audience engagement.

- Examine temporal trends, including daily and seasonal patterns, to identify optimal times for content publication.

- Identify frequently used tags and keywords in trending videos to understand the language and themes that resonate with audiences.

CLOUD STORAGE:

Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud, with more than 200 fully featured services available from data centers globally. AWS offers reliable, scalable, and inexpensive cloud computing services.

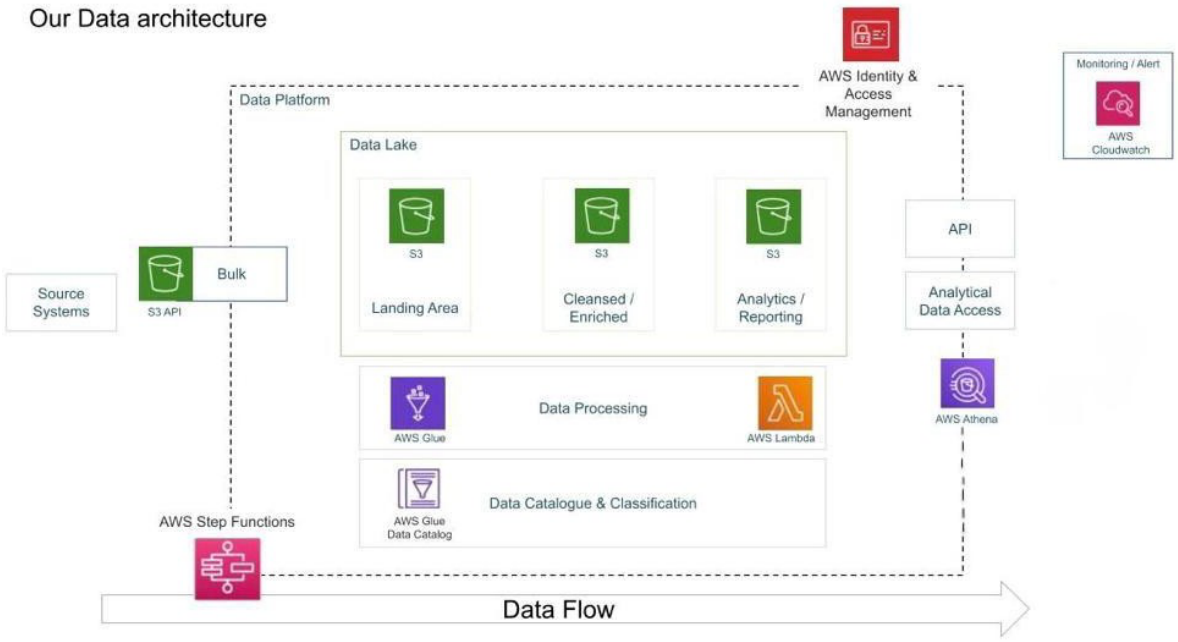
Amazon Web Services used in this project:

- AWS S3: Amazon Simple Storage Service (Amazon S3) is a storage service that offers high scalability, data availability, security, and performance. It stores the data as objects within buckets and is readily available for integration with thousands of applications. An object is a file and any metadata that describes the file. A bucket is a container for objects.
- AWS IAM: Amazon Identity and Access Management is a service to securely control access to AWS Services. It is used to manage permissions for different roles under various scenarios. IAM is used to control who is authenticated (signed in) and authorized (has permissions) to use resources.
- AWS Glue: It is a scalable, serverless data integration service that can be used to discover, prepare, and combine data for application development, analytics, and machine learning. AWS Glue consolidates major data integration capabilities into a single service. It also provides DataOps tools to effortlessly author, run jobs, and implement business workflows.
- AWS Lambda: It is a compute service that provides a runtime environment letting you run code without provisioning or managing servers. The code is written in Lambda functions and is set to trigger as per the use case and is scaled as per the demand.
- AWS Athena: It is a query service that allows easy analysis of data stored directly in Amazon S3 using standard SQL. Amazon Athena also makes it easy to interactively run data analytics using Apache Spark without having to plan for, configure, or manage resources.

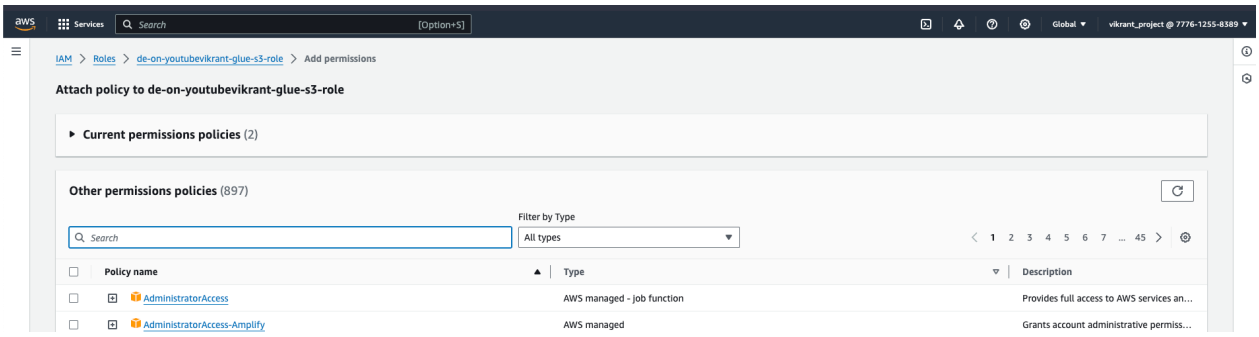
To create dashboards, we will use:

Tableau: Tableau is a data visualization tool that helps businesses analyze data. It's used to create dashboards and visualizations that can be shared with stakeholders.

ARCHITECTURE DIAGRAM:

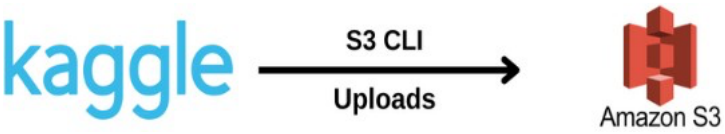


Step 1: Creating an Admin IAM user (to separate from root)
On the AWS Console, logged in to the root account. Then Navigated to the AWS IAM page and clicked on Users and then created a new user. On the Create New User page, chose the option to get console access as well and set up login credentials accordingly. On the next window, provided AdministratorAccess to the User being created.



Once the user is created, logged out of the root account and logged in to the User created using the credentials created in the above steps.
Now, we created Access Key pairs for this user and use them to log in from AWS CLI.

Step 2 – Downloading and Uploading Dataset
Downloaded AWS CLI onto your local system.
AWS CLI download path: https://aws.amazon.com/cli/



We downloaded Structured and semi-structured CSV and JSON data categorized by region from Kaggle. Now we ingested the data to Amazon S3 buckets. There are two formats for each region namely csv and json files. We used AWS CLI to upload the data to S3 bucket to fast control of partitioning and making folders with commands.

```
vikrant@Vikrants-MacBook-Air ~ % aws configure
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
Default region name [us-east-1]: us-east-1
Default output format [None]:
vikrant@Vikrants-MacBook-Air ~ % aws s3 ls
2023-11-26 16:33:35 de-on-youtubevikrant-raw-useast1-dev
vikrant@Vikrants-MacBook-Air ~ % cd downloads
```

```
vikrant@Vikrants-MacBook-Air data % aws s3 cp . s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics_reference_data/ --recursive --exclude "*" --include "*.json"
zsh: segmentation fault  aws s3 cp .  --recursive --exclude "*" --include "*.json"
vikrant@Vikrants-MacBook-Air data % aws s3 ls
vikrant@Vikrants-MacBook-Air data % aws s3 cp . s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics_reference_data/ --recursive --exclude "*" --include "*.json"
upload: ./US_category_id.json to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics_reference_data/US_category_id.json
upload: ./KR_category_id.json to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics_reference_data/KR_category_id.json
upload: ./CA_category_id.json to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics_reference_data/CA_category_id.json
upload: ./GB_category_id.json to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics_reference_data/GB_category_id.json
upload: ./FR_category_id.json to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics_reference_data/FR_category_id.json
upload: ./DE_category_id.json to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics_reference_data/DE_category_id.json
upload: ./MX_category_id.json to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics_reference_data/MX_category_id.json
upload: ./IN_category_id.json to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics_reference_data/IN_category_id.json
upload: ./JP_category_id.json to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics_reference_data/JP_category_id.json
upload: ./RU_category_id.json to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics_reference_data/RU_category_id.json
```

```
vikrant@Vikrants-MacBook-Air data % aws s3 cp CAvideos.csv s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=ca/
upload: ./CAvideos.csv to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=ca/CAvideos.csv
vikrant@Vikrants-MacBook-Air data % aws s3 cp DEvideos.csv s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=de/
aws s3 cp FRvideos.csv s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=fr/
aws s3 cp GBvideos.csv s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=gb/
aws s3 cp INvideos.csv s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=in/
aws s3 cp JPvideos.csv s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=jp/
aws s3 cp KRvideos.csv s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=kr/
aws s3 cp MXvideos.csv s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=mx/
aws s3 cp RUvideos.csv s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=ru/
aws s3 cp USvideos.csv s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=us/
upload: ./DEvideos.csv to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=de/DEvideos.csv
upload: ./FRvideos.csv to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=fr/FRvideos.csv
upload: ./GBvideos.csv to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=gb/GBvideos.csv
upload: ./INvideos.csv to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=in/INvideos.csv
upload: ./JPvideos.csv to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=jp/JPvideos.csv
upload: ./KRvideos.csv to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=kr/KRvideos.csv
upload: ./MXvideos.csv to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=mx/MXvideos.csv
upload: ./RUvideos.csv to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=ru/RUvideos.csv
upload: ./USvideos.csv to s3://de-on-youtubevikrant-raw-useast1-dev/youtube/raw_statistics/region=us/USvideos.csv
vikrant@Vikrants-MacBook-Air data % ls
CA_category_id.json     DEvideos.csv            GB_category_id.json     INvideos.csv            KR_category_id.json     MXvideos.csv            US_category_id.json
CAvideos.csv            FR_category_id.json     GBvideos.csv            JP_category_id.json     KRvideos.csv            RU_category_id.json     USvideos.csv
DE_category_id.json     FRvideos.csv            IN_category_id.json     JPvideos.csv            MX_category_id.json     RUvideos.csv
```

Step 3 - Transforming the .json data to parquet format

Created a Schema based on the data (.json files) using AWS Glue Crawler For the first step, we did crawl over the folder containing .json files.

To create a Crawler :

1. Go to AWS Glue and chose Crawlers from the sidebar menu.

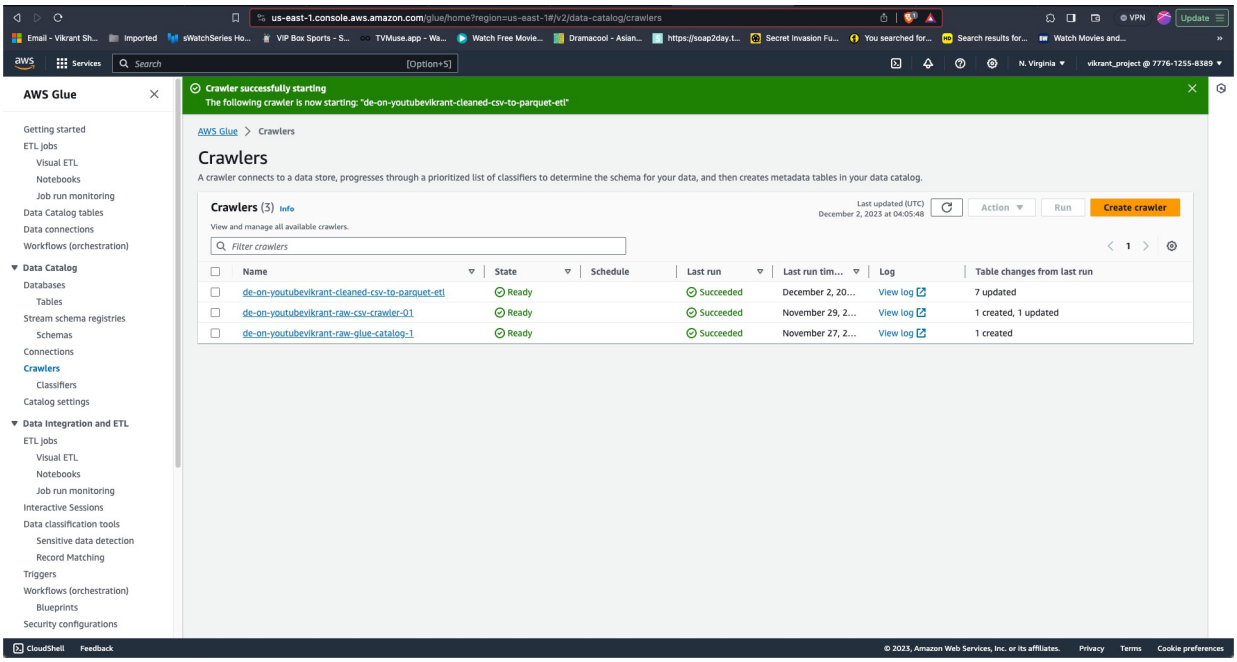
2. Clicked on the Create Crawler button and followed the steps of the crawler creation.

3. Mentioned the data source to parse when prompted. We selected Data source as S3 bucket.

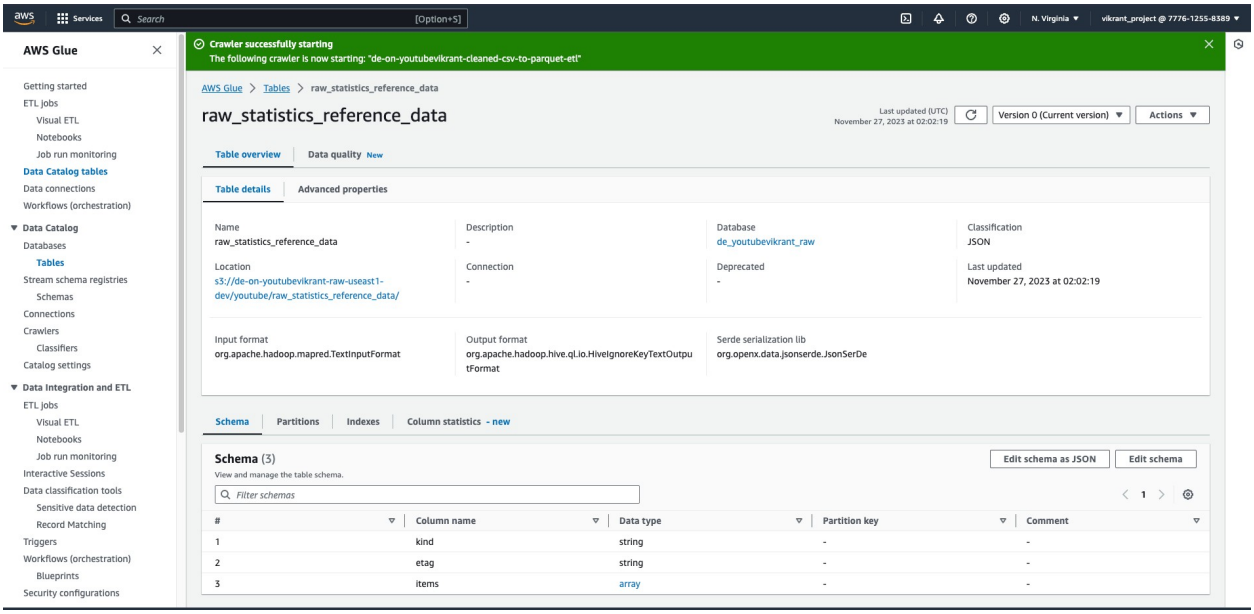
Since we have to access S3 from Glue, we need to set permissions for these resources. To do this, we created an IAM role for Glue, giving it Admin access, so that the S3 bucket data can be accessed from Glue.

4. Next we created an output database ie. the database that stores the results of the crawler run.

Now we created a AWS Glue Crawler as below.



Once the crawler gets created, we ran the crawler and generated the schema of the datastore it crawls over as a table.

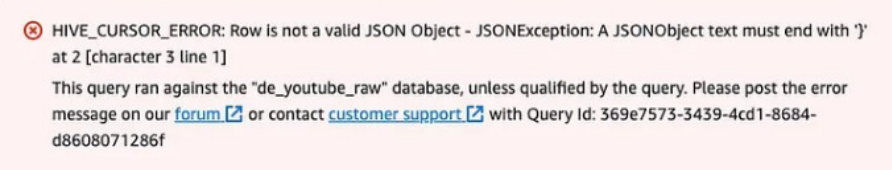


The schema as shown above can be accessed by clicking on the Tables tab of AWS Glue and choosing the table created by the crawler run.

Step 5: Querying the table created using AWS Athena

Using the AWS Athena editor and chose the Database (created in the above step) and tried to write a SQL query to see the data in table form and get results.

But below error appeared.

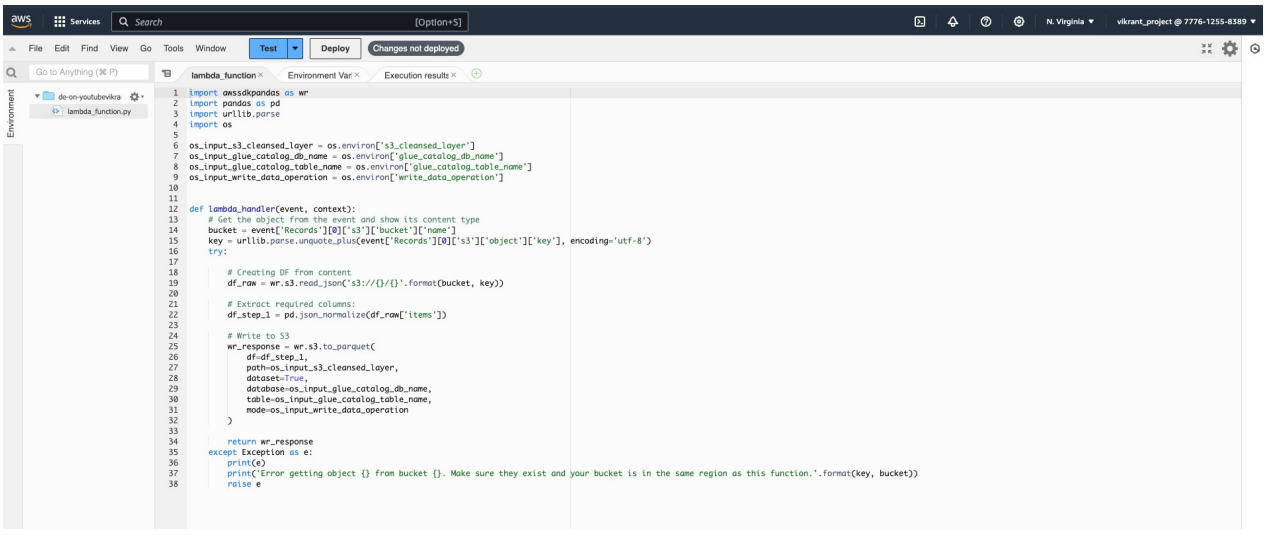


This error comes due to the fact that the items column of data is in array format and the Glue tools are unable to read this format of data. To rectify this error, we should convert this .json data into parquet format. To do this we will use an AWS service called Lambda.

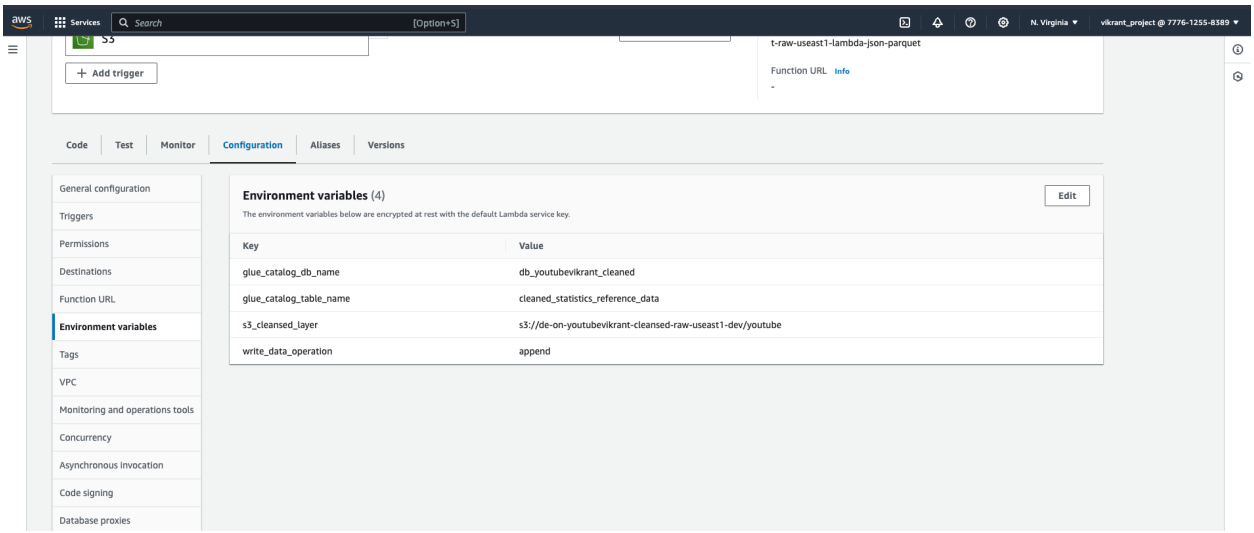
Step 6: Transforming the data using AWS Lambda

We created a function in AWS Lambda. This can be done easily by following the Create Function page. Specified Python as the runtime environment. We created an another IAM role, providing access to Lambda for S3 and Glue. Used Python code to convert the JSON data in the form of an array to a column (parquet) format. Now we wrote the dataframe as a parquet file onto the bucket and the folder specified by the environmental variables.

Python code:

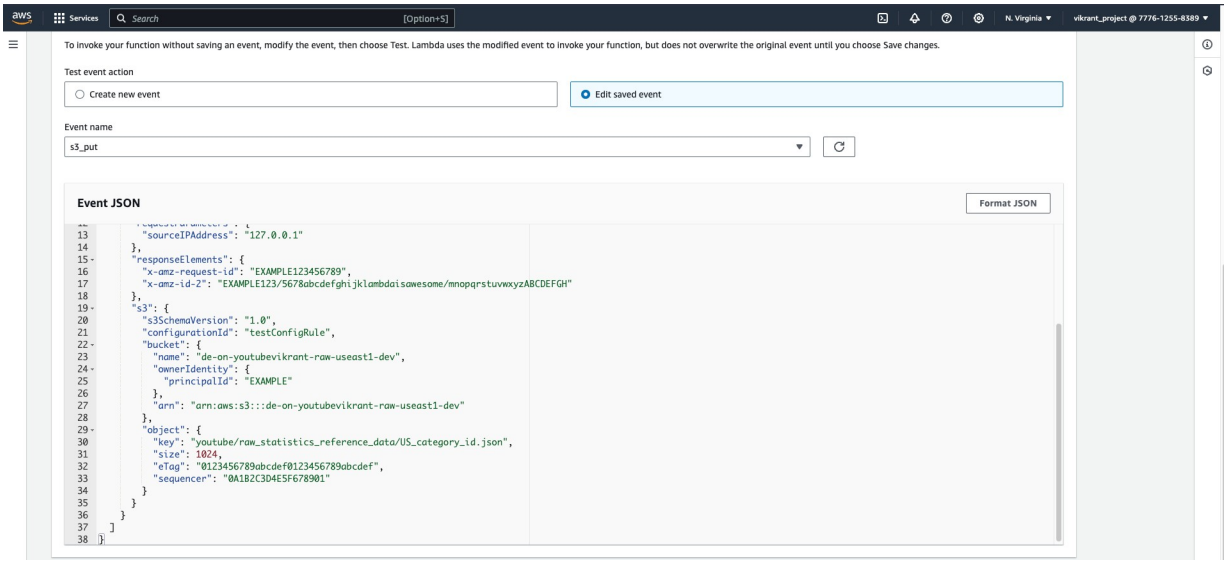


The values of these environmental variables can be set by going onto the Configuration tab and then the Environmental Variables as shown below.



Also created a new S3 bucket and copied its URI onto the variable.

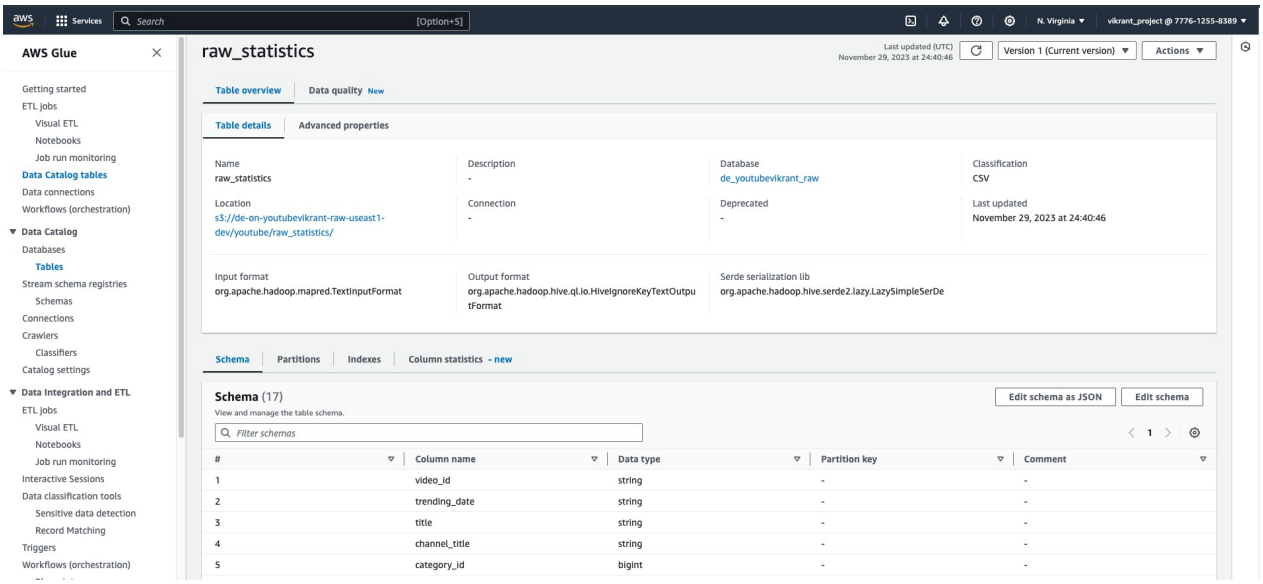
Once this is done, we tested the Lambda function. For this, we created s3_put template in the create test window and specified the bucket and the key in the test JSON provided by the template.



Here, we successfully converted nested JSON data into a column-based parquet format which is easily readable.

Step 7: Created a Schema based on the data (.csv files) using AWS Glue Crawler. Followed the same steps as were followed for creating the crawler for the .json files.

Once the crawler is created, on running it a table is created.



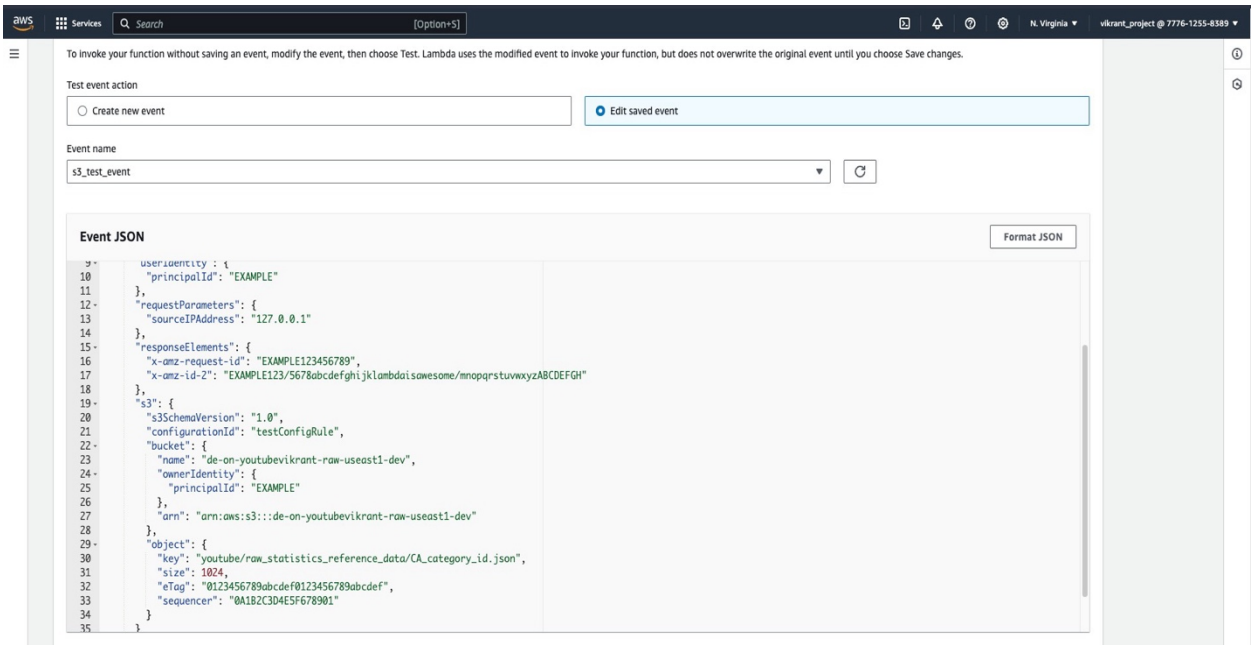
Step 8: Joining the two tables, csv and json table data

Initially, an attempt to perform a join operation between two tables, "raw_statistics" and "cleaned_statistics_reference_data," resulted in an error due to a schema mismatch in column datatypes. The suggested fix involved transforming the CSV data to align the column types. A cast operation was initially proposed in the query, but it was deemed expensive for large datasets.

```
SELECT * FROM "db_youtubevikrant_cleaned"."raw_statistics" a

INNER JOIN "db_youtubevikrant_cleaned"."cleaned_statistics_reference_data" b ON a.category_id=cast(b.id as int)
```

To address this, we changed the schema of the "cleaned_statistics_reference_data" table using the Glue Console, specifically by modifying the datatype of the "id" column to bigint. However, changing the schema in the Glue catalog did not automatically update the metadata in the Parquet files, leading to errors in subsequent join queries. To resolve this, the recommended action was to delete the Parquet file created during the Lambda function test and then rerun the test event. This resulted in the creation of a new Parquet file in the cleaned bucket.

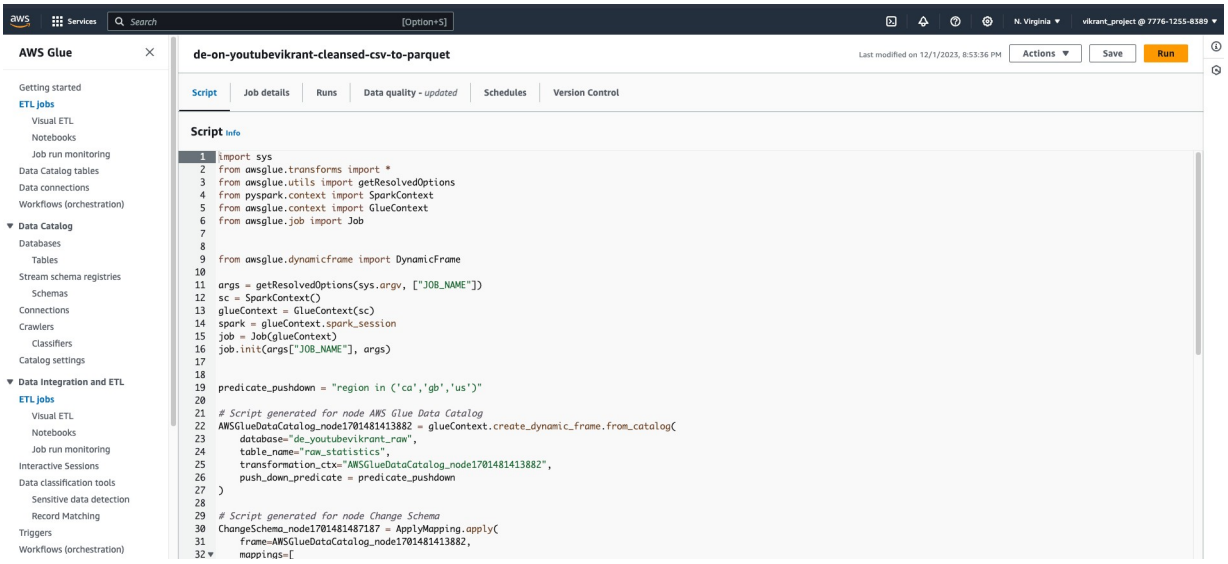


After these steps, when running an Athena query again, no errors were encountered, and the join operation between the tables was successfully executed.

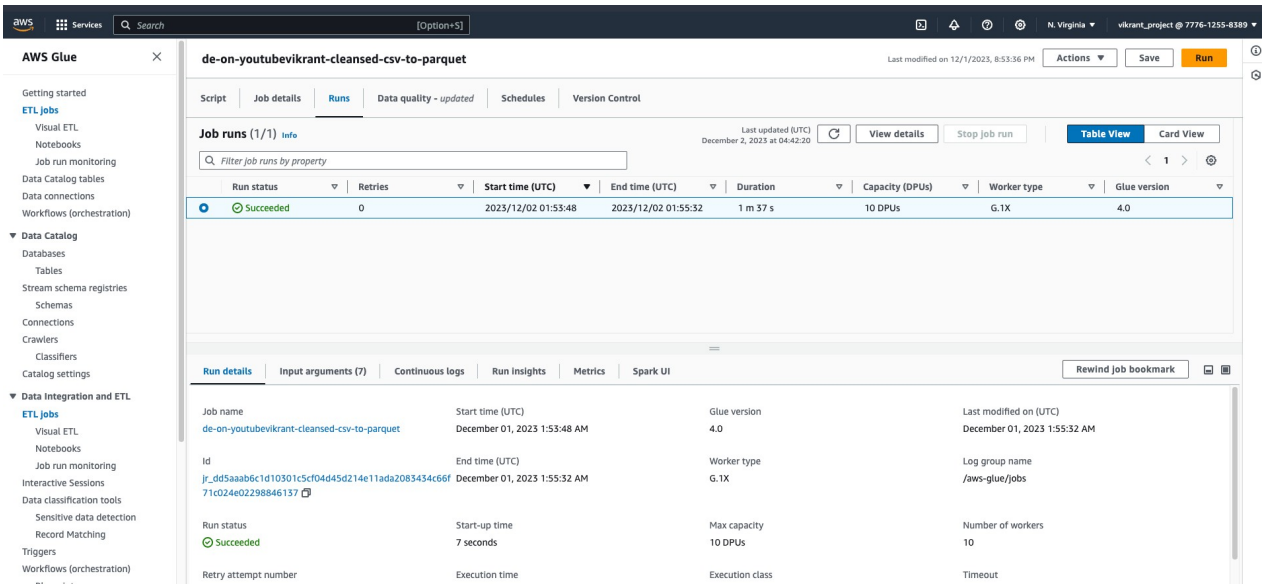
Step 9: Using Glue ETL to transform data

To initiate this transformation, we accessed the Glue Console and navigated to the "ETL Jobs" option in the sidebar, opting for the "Spark Script Editor" to create a new job. After specifying the IAM role for S3 and Glue access in the "Job details" tab, we selected "Spark" as the job type, utilizing Python as the programming language.

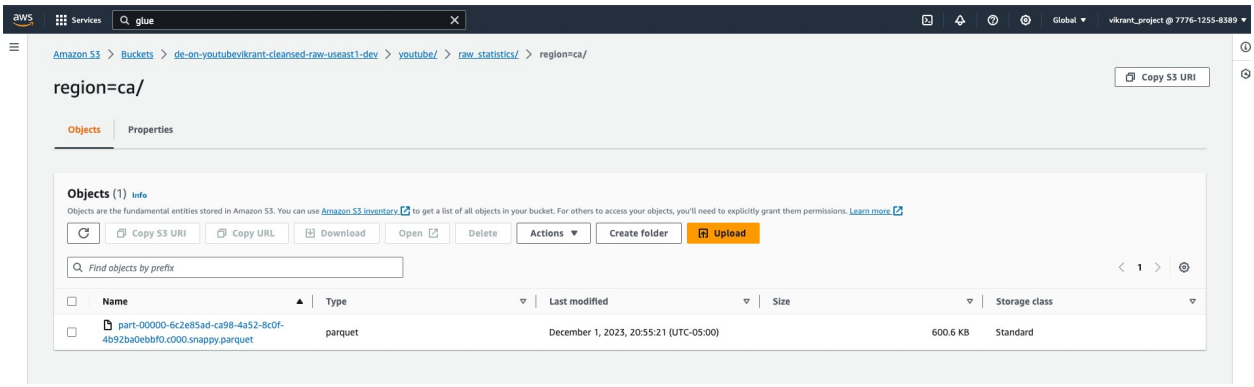
We wrote the pyspark code for this transformation.

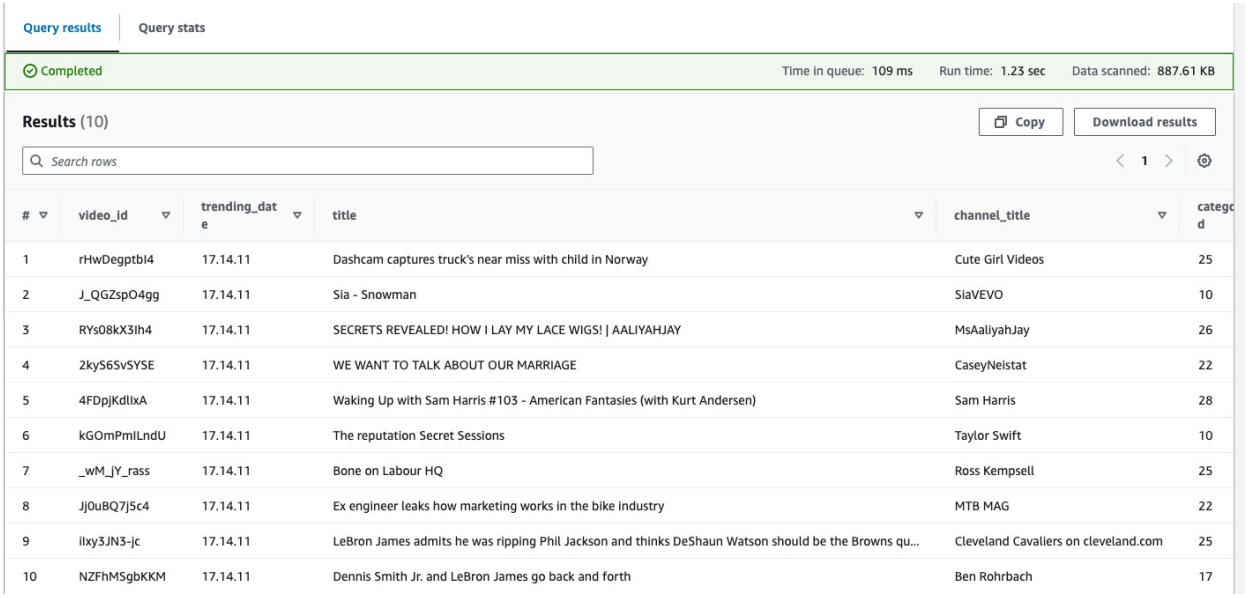


Job ran successfully.



The source data for this transformation was the "raw_statistics" data, generated through the Glue Crawler on CSV data. Notably, the destination for the cleaned data was defined, and a filter was implemented to include data only from English-title regions, such as Canada, UK, and US, aiming to prevent errors due to special characters. Upon configuring these settings, we executed the Spark job, leading to the successful observation that the destination bucket and folder now housed all the cleaned files in the desired Parquet format.





Query results | Query stats

Completed — Time in queue: 109 ms — Run time: 1.23 sec — Data scanned: 887.61 KB

Results (10) — Copy — Download results

Search rows — 1

#	video_id	trending_date	title	channel_title	categoryId
1	rHwDegptbI4	17.14.11	Dashcam captures truck's near miss with child in Norway	Cute Girl Videos	25
2	J_QGZspO4gg	17.14.11	Sia - Snowman	SiaVEVO	10
3	RYs08kX3Ih4	17.14.11	SECRETS REVEALED! HOW I LAY MY LACE WIGS! | AALIYAHJAY	MsAaliyahJay	26
4	2kyS6SvSYSE	17.14.11	WE WANT TO TALK ABOUT OUR MARRIAGE	CaseyNeistat	22
5	4FDpjKdlIxA	17.14.11	Waking Up with Sam Harris #103 - American Fantasies (with Kurt Andersen)	Sam Harris	28
6	kGOmPmILndU	17.14.11	The reputation Secret Sessions	Taylor Swift	10
7	_wM_jY_rass	17.14.11	Bone on Labour HQ	Ross Kempsell	25
8	Jj0uBQ7j5c4	17.14.11	Ex engineer leaks how marketing works in the bike industry	MTB MAG	22
9	ilxy3JN3-jc	17.14.11	LeBron James admits he was ripping Phil Jackson and thinks DeShaun Watson should be the Browns qu...	Cleveland Cavaliers on cleveland.com	25
10	NZFhMSgbKKM	17.14.11	Dennis Smith Jr. and LeBron James go back and forth	Ben Rohrbach	17

Step 10: Crawl the cleaned .csv data created in the above step and create a table.

A crawler was established with the data source set to the folder containing the cleaned Parquet files. The crawler was then executed, generating a table from the cleaned data. Subsequently, the completion of this step enabled the ability to query the data using Athena.

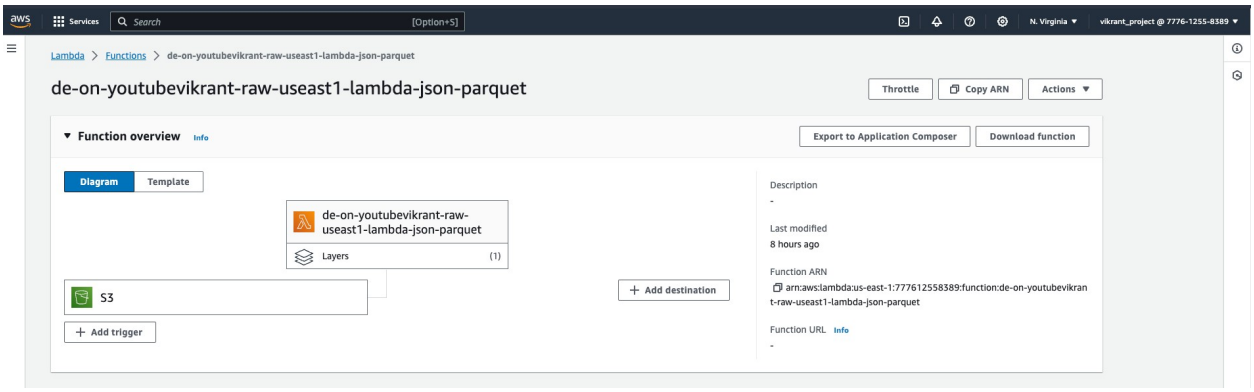
Step 11:

The focus was on implementing a trigger for the Lambda function established in Step 6. Originally, the Lambda function only generated a cleaned output file when manually invoked through a test case.

However, for real-world applications, it was crucial for the function to be triggered automatically whenever a raw file was added to the S3 bucket. The solution to this requirement involved configuring triggers on Lambda functions.

To add the trigger:

Accessed the Lambda function to which the trigger needed to be applied.
Clicked on the "Add trigger" button.



Once this setup was complete, the addition of any .json files to the designated bucket and prefix folder in the trigger configuration triggered the Lambda function automatically. Consequently, this resulted in the creation of clean files in the specified destination bucket and folder.

Creating the Analytics Bucket and Table

Step 12: Using AWS Glue ETL to join the two cleaned tables created and generate final data.

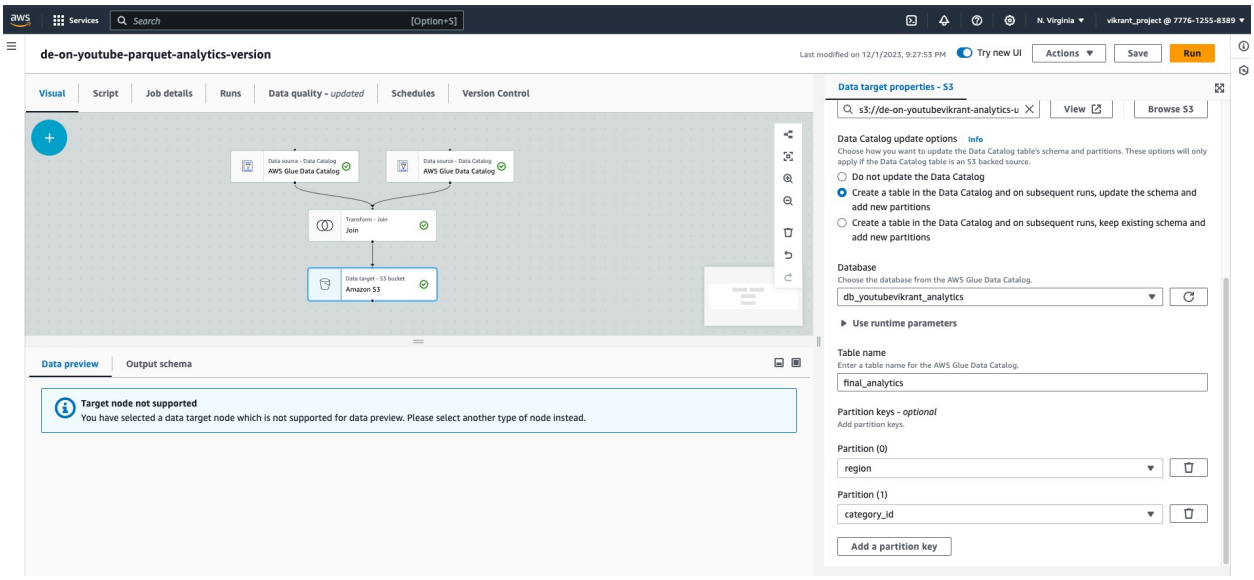
we created a final table, containing a join between the two clean tables that we created.

Step 13:
We created a Glue ETL job using the Visual with a source and target.

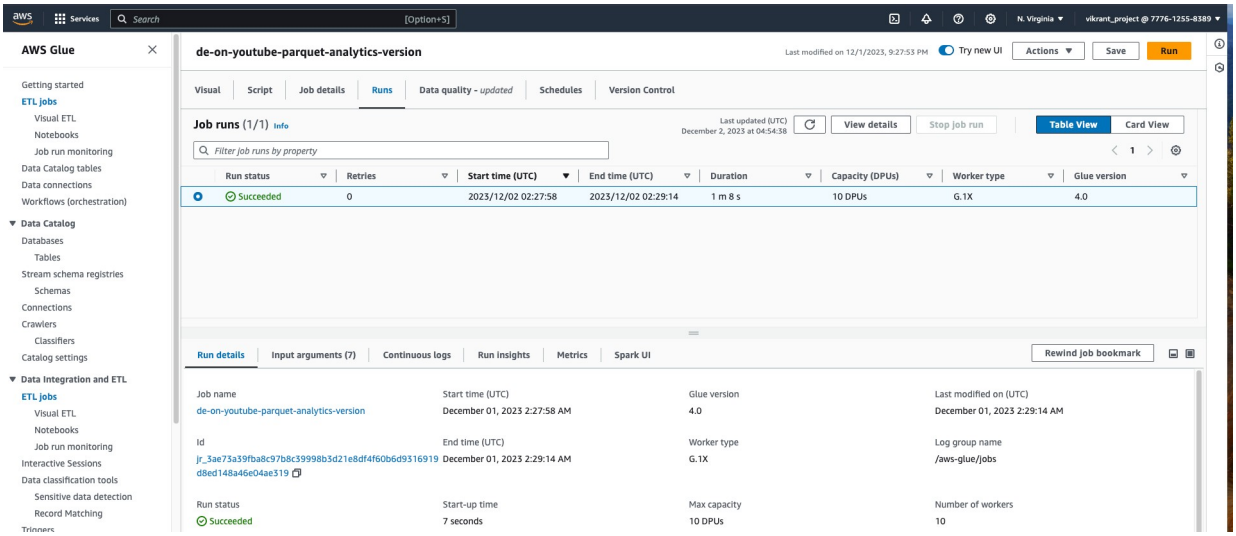
First, two S3 source nodes were added, representing folders containing cleaned data in both .json and .csv formats stored in Parquet format.
Details such as folder locations and configurations were specified in the node details pane adjacent to the drawing board.

Subsequently, a Join Transform node was introduced, and in the details pane, the type of join and the column on which the join should be performed were specified. Both S3 source nodes were added as node parents to this Join Transform node.

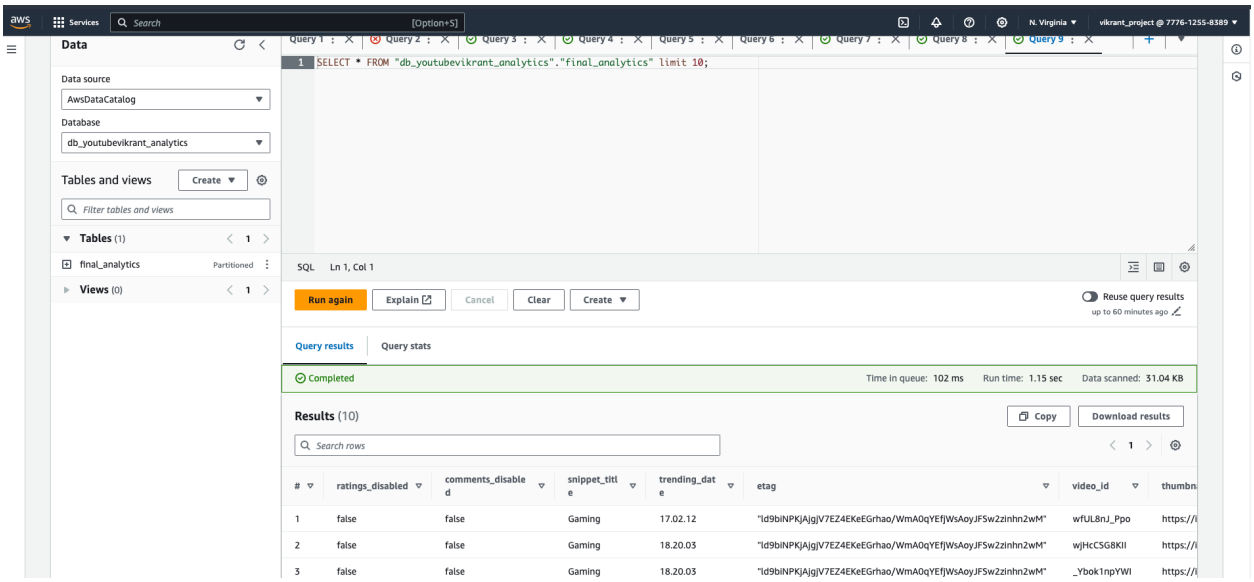
Finally, an S3 target was incorporated, necessitating the creation of an S3 bucket to store the joined data. During this process, a database named "db_youtubevikrant_analytics" was created to store the resulting table from job runs. The name for the table was specified, and two desired partition keys were added; Region and category_id.



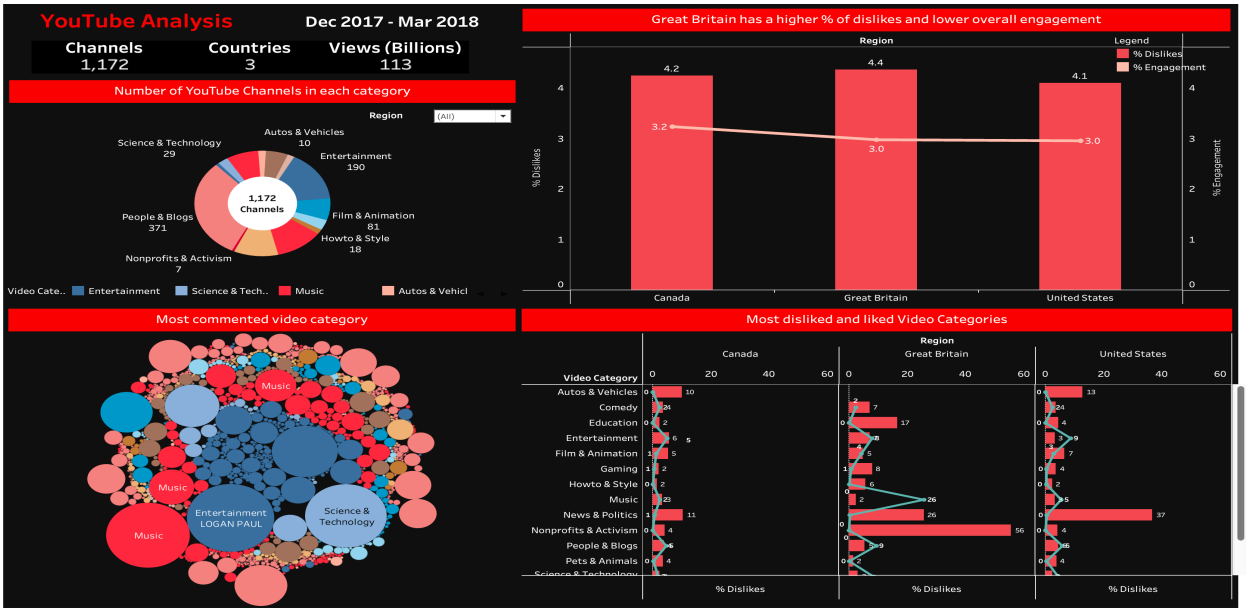
Successfully job run:

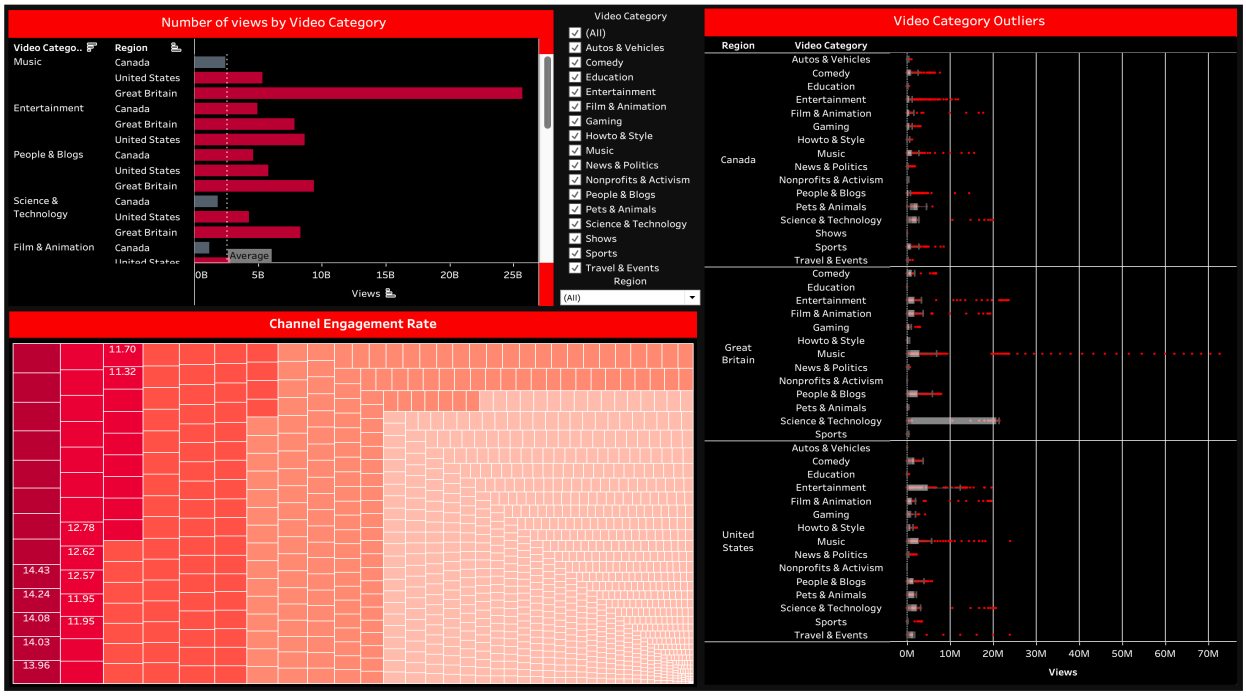


The data gets written on the S3 bucket mentioned in the target node and a table is created in the db_analytics database. This table can be queried using Athena.



Now that we have created the final data, we move on to visualization using Tableau.





VISUALIZATION ANALYSIS:

- People & Blogs is the most popular video category across the Canada, United States and Great Britain, followed by Music.
- Entertainment, Music and Science & Technology are the most commented Video Categories.
- Great Britain has highest percentage of dislikes and lower percentage of overall Engagement when compared with Canada and United States.
- The most disliked categories are News&Politics and NonProfits & Activism. The most liked categories: Travel & Events and Music.
- Music tends to have the highest views in Great Britain followed by Entertainment.
- Music tends to have the highest outliers (as high as ~72m Views). Entertainment came 2nd (highest outlier at ~24m).
- Wil Aime, Weest and exurb1a channels have the highest Engagement rates across the 3 countries.
- Only few video categoris like Music, Entertainment , People & Blogs were able to cross the average view count in United States and Great Britain.