Group Project- AWS Data Analytic Platform for The City of Vancouver

Gyanvi Bhardwaj (2320329)

Peng Wang (2306811)

Muhammad Zulqarnain Shahzad (2306553)

Haoran Xu (2317218)

University Canada West

BUSI653: Cloud Computing Technologies (HBD-WINTER25-03)

Dr Mahmood Mortazavi Dehkordi

March 05, 2025

Table of Contents

Introduction

City of Vancouver requires a Data Analytics Platform (DAP) to process and summarise bulk datasets with efficiency to make decisions. By utilising AWS services, scalability and accuracy can be ensured to understand city-based data.

The foundation of creating a data analytics platform for the City of Vancouver involves choosing relevant datasets followed by their preparatory processing. The project adopted four datasets namely Public Trees, 3-1-1 Contact Metrics, Business Licenses and Animal Control Inventory Register. The dataset filtering process aimed to maintain the right data amount. The data required profiling alongside cleaning steps which produced accurate data for analysis purposes.

The next step involved data preparation after which the data required cataloging. AWS S3 operated as a scalable storage platform that provided secure data access capabilities. AWS Glue performed data transformation after which it structured each dataset to allow analysis. The data formatting process as a part of this step ensured that information prepared for querying requirements correctly.

AWS Glue served as the platform to conduct data summary and analysis tasks after processing and storage took place. The data team implemented diverse analytical methods to obtain useful conclusions. The analysis of Public Trees data evaluated tree diameter versus height range, but the 3-1-1 Contact Metrics data supported standard service levels analysis. The analysis of business license records revealed worker-based trends and evaluated the Animal Control data for finding animal breed ratios and impoundment statistics.

The analysis generated valuable recommendations about municipal changes together with business enhancements that were derived from the statistical results. Data pattern assessments

from these analyses helped provide knowledge about environmental monitoring methods together with service efficiency improvements alongside policy effects. Through this structured process the City of Vancouver can create optimal public services which enable data-based decision-making for community welfare.

DAP Design and Implementation (Individual work – Gyanvi Bhardwaj)

Descriptive Analysis

Dataset selection

The selected dataset is sourced from the ***'Animal Control Inventory Register'*** under the theme of ***'Parks, Recreation and Pets'*** from the City of Vancouver data portal for the year 2024 (*Animal Control Inventory - Register*, 2024).

Link: Animal Control Inventory 2024

This dataset provides a comprehensive list of all the animals that have been reported to the City of Vancouver Animal Control Department and have been taken into their custody. It is one of the three interlinked datasets about animal reporting in Vancouver. The other related datasets include lost and found animals, and deceased animals datasets. It includes categorization based on various factors such as medical status, breed, colour and any known history of the animal.

Why the 'Animal-control-inventory-register' dataset?

I have chosen the 'Animal-Control-Inventory-Register' dataset due to its relevance in decision-making for the animal control department. The well-structured dataset consists of a comprehensive overview of the various animals that are in the custody of the City of Vancouver, differentiating them based on various attributes like breed, sex, colour, age category etc. By

utilizing this data, animal control officials can improve resource distribution, shelter capacity and trends in impoundment. By understanding breed or age categories, we can better evaluate animals that need urgent attention. By including operational details like source and status, we can reduce the number of stray animals and understand trends of impoundment. This relevance makes the dataset an invaluable tool for informing data-driven policies, improving animal welfare, and ensuring more efficient and responsive animal control practices.

Goal:

To analyze the distribution of animal breeds across age categories and identify the most recent impound date to make informed animal control strategies.

Analytical questions:

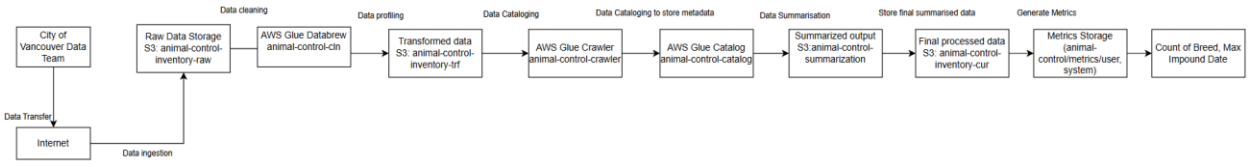
- What is the distribution of animal breeds across different age categories?
- What is the most recent date of animal impoundments?

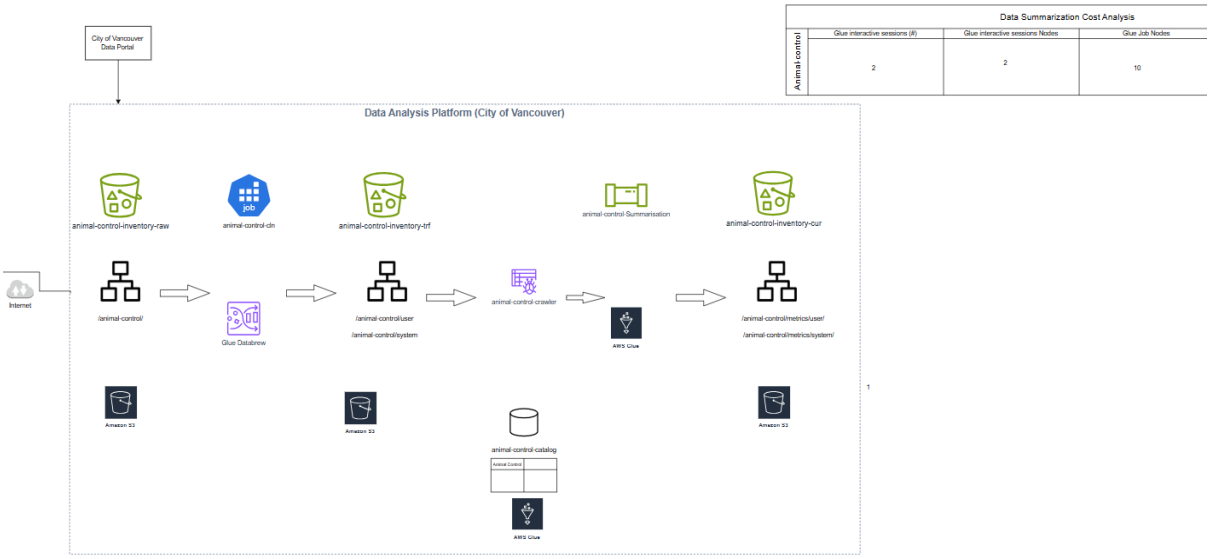
Descriptive Metric:

- Count of animals (by breed)
- Maximum impound date (most recent impoundment date)

Figure 1

Components of Data Analytics Platform for the City of Vancouver





Note. Components of Data Analytics Platform (DAP) for the City of Vancouver for Animal-control-inventory dataset. (Source: *self-work*, created on draw.io)

The components involved in the DAP begin from the City of Vancouver data portal. The data team of the City of Vancouver works on the raw data ingested into the raw data S3 bucket. This raw data is then sent to AWS Glue DataBrew for data profiling and cleaning. The cleaned and transformed are stored in the transformed S3 bucket. A crawler is created for data cataloging and summarisation using AWS Glue services. The curated data is stored in the curated S3 bucket.

Step 1: Data Ingestion

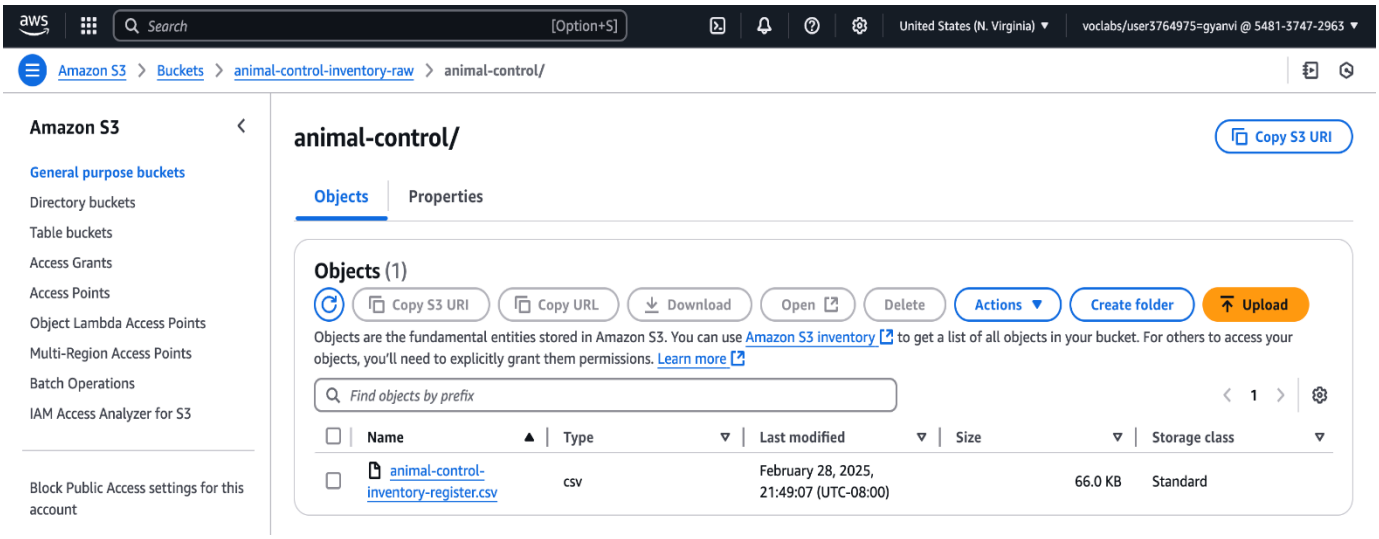
The data ingestion step includes extracting data from its source and loading it into an AWS S3 bucket. In my case, I have exported a CSV file named ‘Animal-control-inventory-register’, filtered for 2024 with a total of 510 rows and 18 columns from the data portal of the City of Vancouver.

Using an AWS storage service named S3, I have created a raw bucket called ‘animal-control-inventory-raw’ to ingest the raw data into a folder called ‘animal-control’. Our ingestion

rate is daily and since the dataset is already filtered for 2024, I have not created any other sub-folder to keep visualization easier. I have successfully loaded my CSV dataset into the animal-control folder.

Figure 2

AWS S3 bucket with raw ingested data



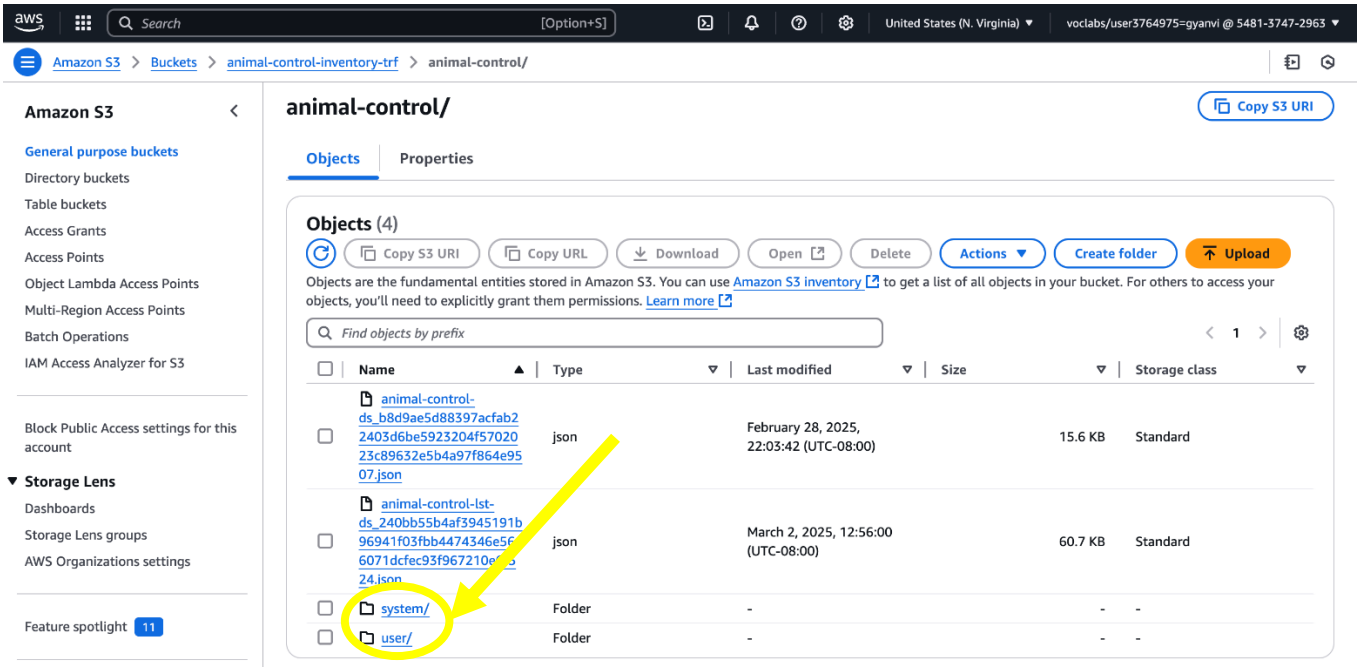
Note. The data ingestion step is performed by ingesting data from the source- City of Vancouver data portal and loaded into our raw S3 bucket. Source: AWS S3, *self-work*

Step 2: Data Profiling

Data profiling is a crucial step as it helps us to understand the structure and identify various possible issues with the dataset. I have created another S3 bucket named ‘animal-control-inventory-trf’ to store transformed data after performing data profiling and data cleaning. Inside this bucket, we have created another sub-folder called ‘animal-control’, which consists of two more sub-folders named ‘user’ and ‘system’. The user folder will have data in a user-friendly readable format, which is .csv with a semi-colon as the delimiter and will generate a single output file. The system folder will contain multiple files of snappy type which is easy for the system to read.

Figure 3

AWS S3 bucket with transformed data and two sub-folders

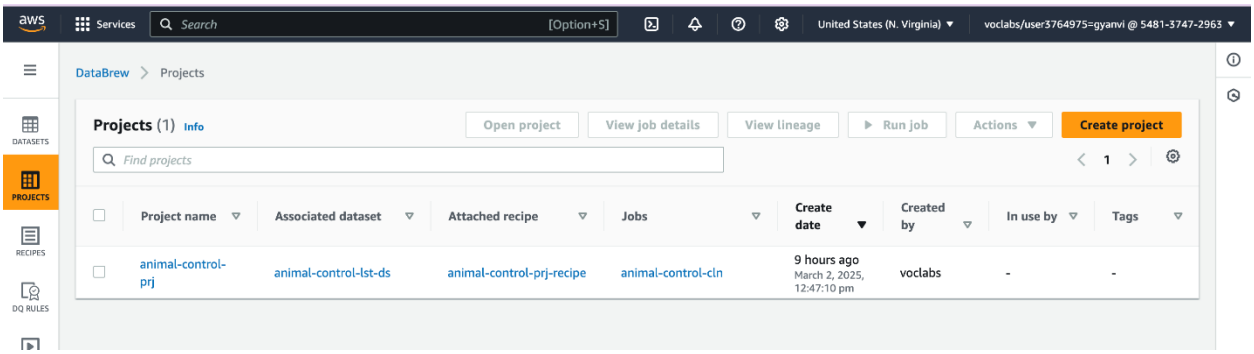


Note. The data profiling step is initiated by creating a new S3 bucket with two folders-user and system to store the transformed data. Source: AWS S3, *self-work*

Next, I utilized AWS Glue DataBrew to visualize data for preparation without writing any code. The first step includes creating a project. I have created a project called 'animal-control-prj' and its associated dataset called 'animal-control-lst-ds' is derived from the raw bucket.

Figure 4

Project created for data profiling



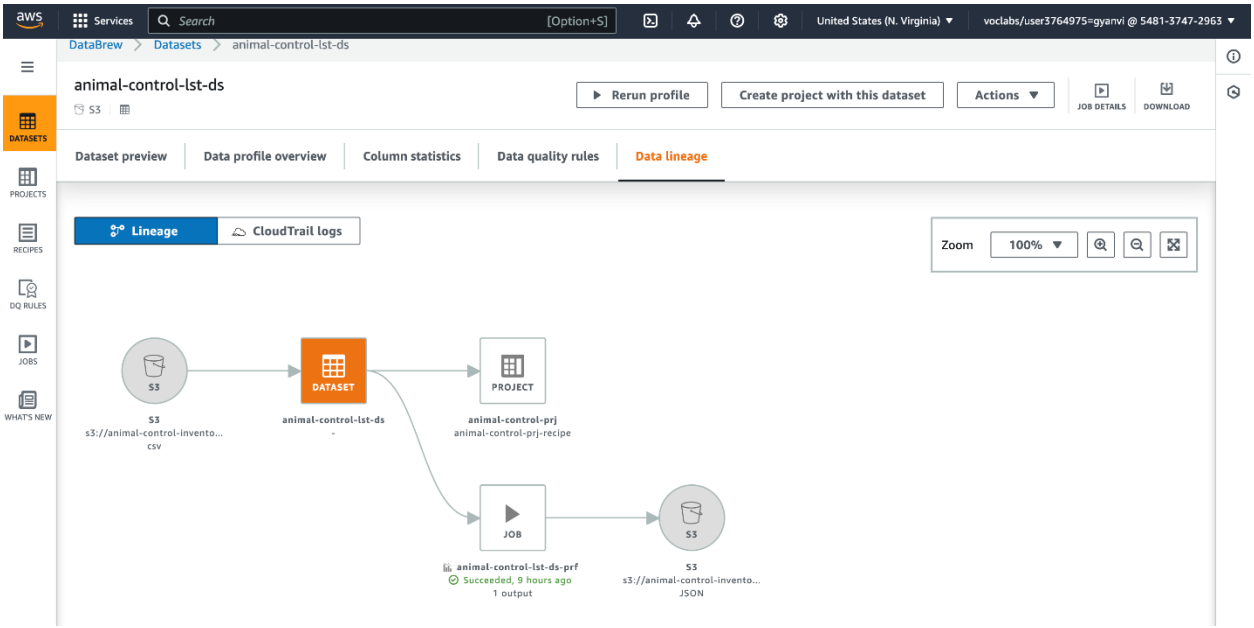
Note. A project to visualize data for data profiling is created using AWS Glue DataBrew. Source: AWS Glue DataBrew, self-work

Step 3: Data Cleaning

Data cleaning is very important to clean and prepare the raw data for analysis. It handles all errors in the dataset such as missing values, duplicate values, outdated values, etc. All 17 issues are taken care of using AWS DataBrew cleaning techniques by creating a cleaning job named ‘animal-control-cln’. The recipe created handles all the cleaning steps that I have used to clean my dataset. There is a total of 24 recipe steps implemented to clean and store our data in the transformed S3 bucket.

Figure 5

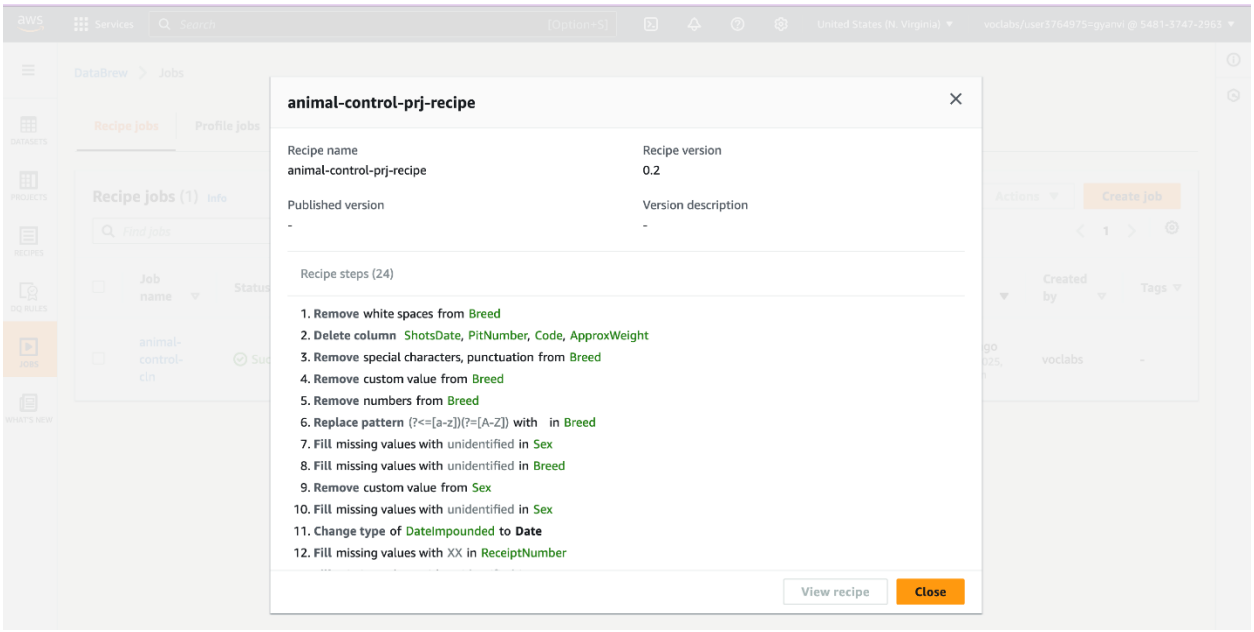
Data lineage visualization for the job created



Note. Data lineage visualization in AWS Glue DataBrew shows how the dataset is taken for the project from the raw bucket and the job is run to clean the raw data. Source: AWS Glue DataBrew, *self-work*

Figure 6

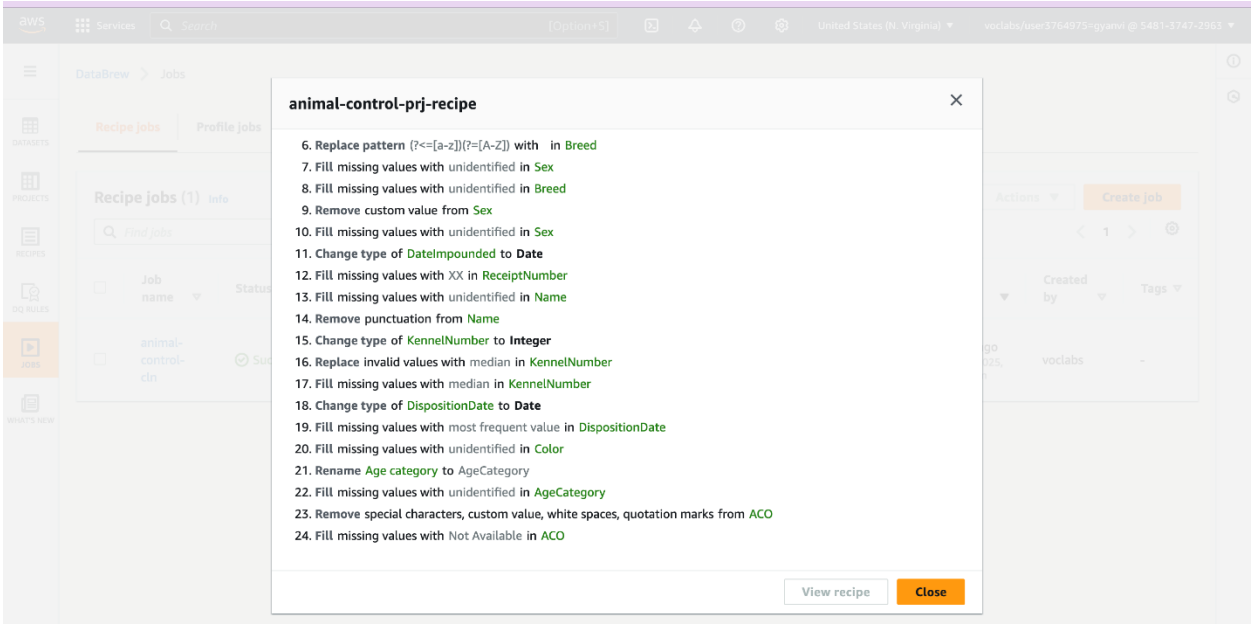
Recipe steps for data cleaning



Note. The overview of the recipe steps used to clean our animal-control-inventory dataset is listed using AWS Glue DataBrew. Source: AWS Glue DataBrew, *self-work*

Figure 7

Recipe steps for data cleaning



Note. The overview of the recipe steps used to clean our animal-control-inventory dataset is listed using AWS Glue DataBrew *(contd.)*. Source: AWS Glue DataBrew, *self-work*

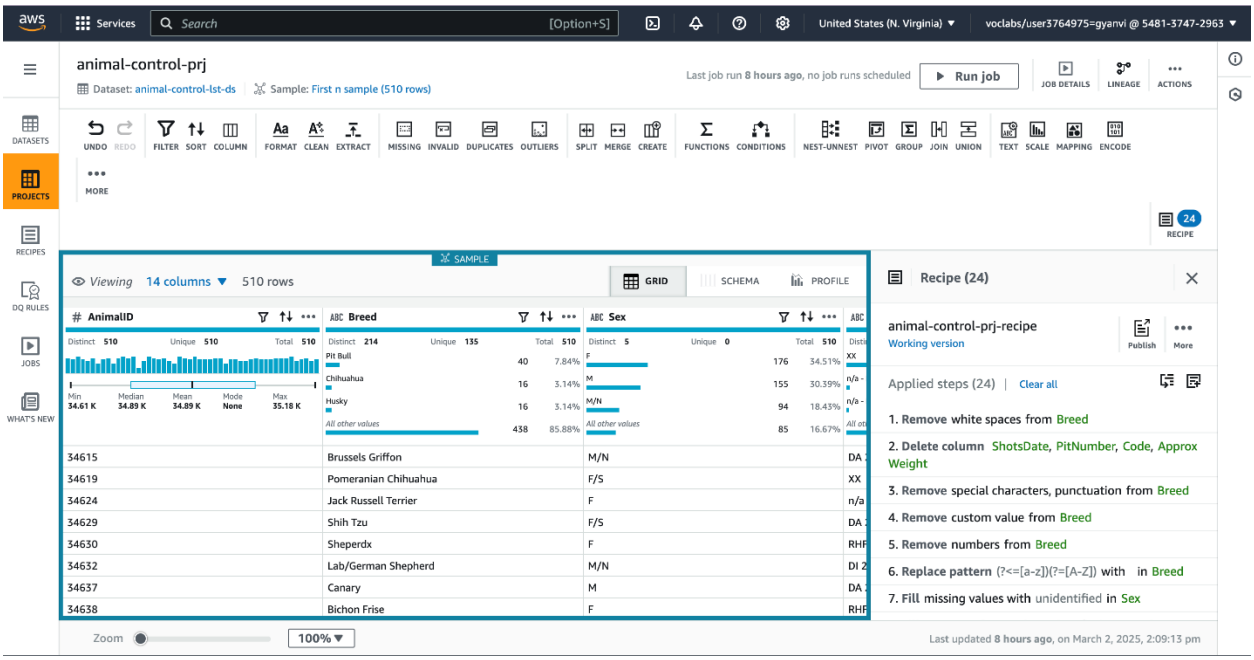
Data cleaning steps used:

1. Removing all leading and trailing white spaces
2. Delete four columns that had no data.
3. Removed special characters and unwanted punctuation.
4. Removed numbers from the string datatype columns.
5. Filled custom value using Regex code for column Breed to put space between sentence cases.
6. Changed format from timestamp to date.
7. Filled missing values with ‘unidentified’ for better understanding.

8. Changed the datatype of column Kennel number from string to integer.
9. Changed the structure of the column name to sentence case.
10. Filled custom value using most frequent data for disposition date.

Figure 8

Cleaned dataset ready for analysis



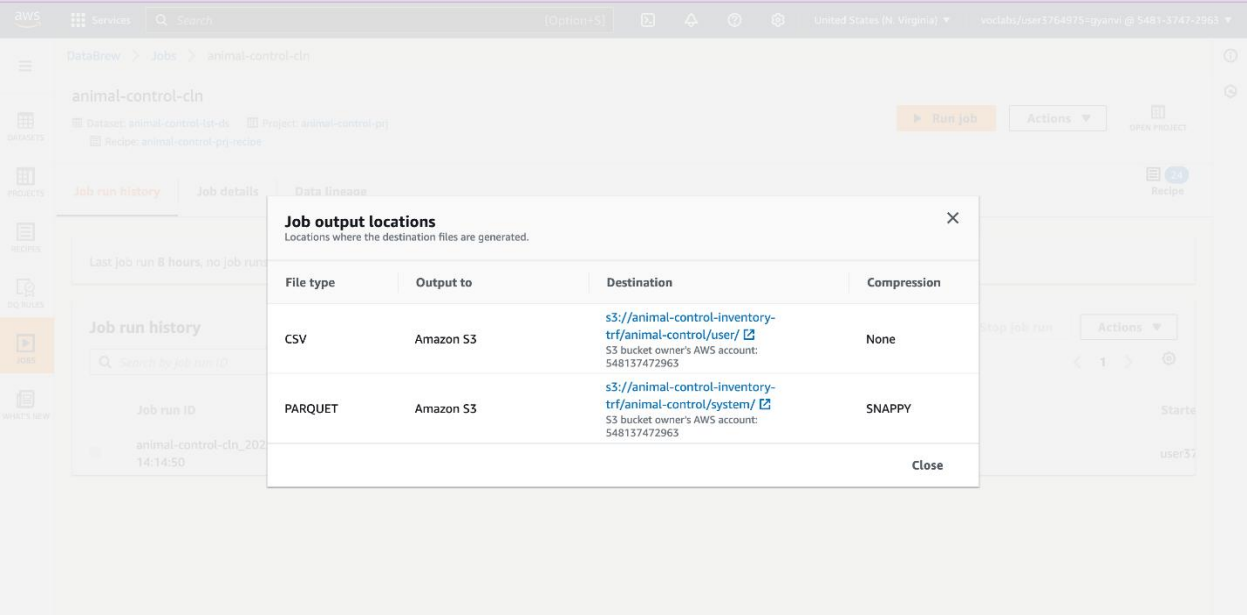
Note. The cleaned dataset consists of 14 columns and 510 rows and has no dataset issues.

Source: AWS Glue DataBrew, *self-work*

The transformed data after the cleaning procedure is stored in the bucket 'animal-control-inventory-trf'. After the cleaning job is run, the output is stored in two files- CSV and PARQUET. The former is user-friendly, and the latter is system friendly. The partition key for the system is the *Age Category* of the animals. It is the most suitable categorical column available in the dataset.

Figure 9

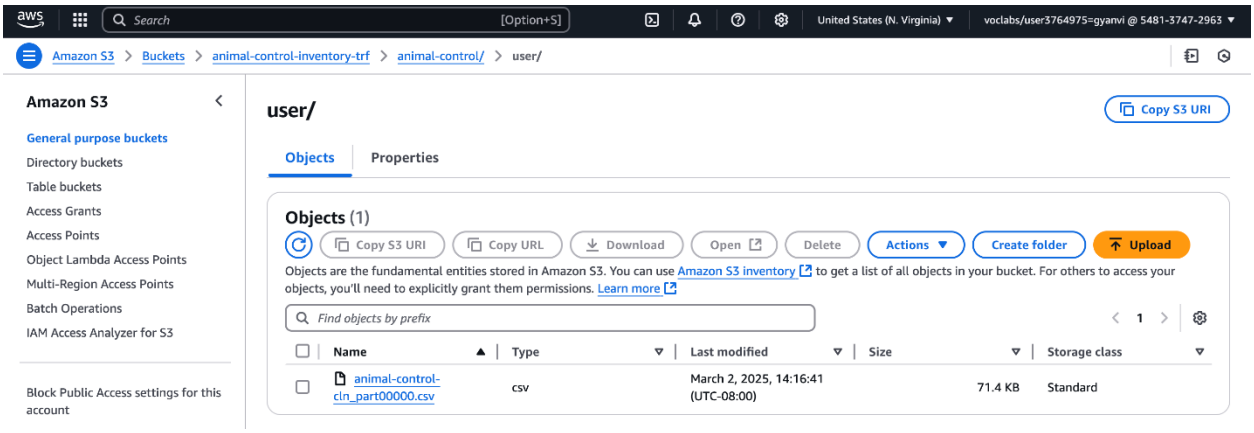
Output of the job run into two folders of the transformed bucket



Note. The image shows a summary of the output locations of the job that was created and run, consisting of two file types stored in a transformed S3 bucket. Source: AWS S3, *self-work*

Figure 10

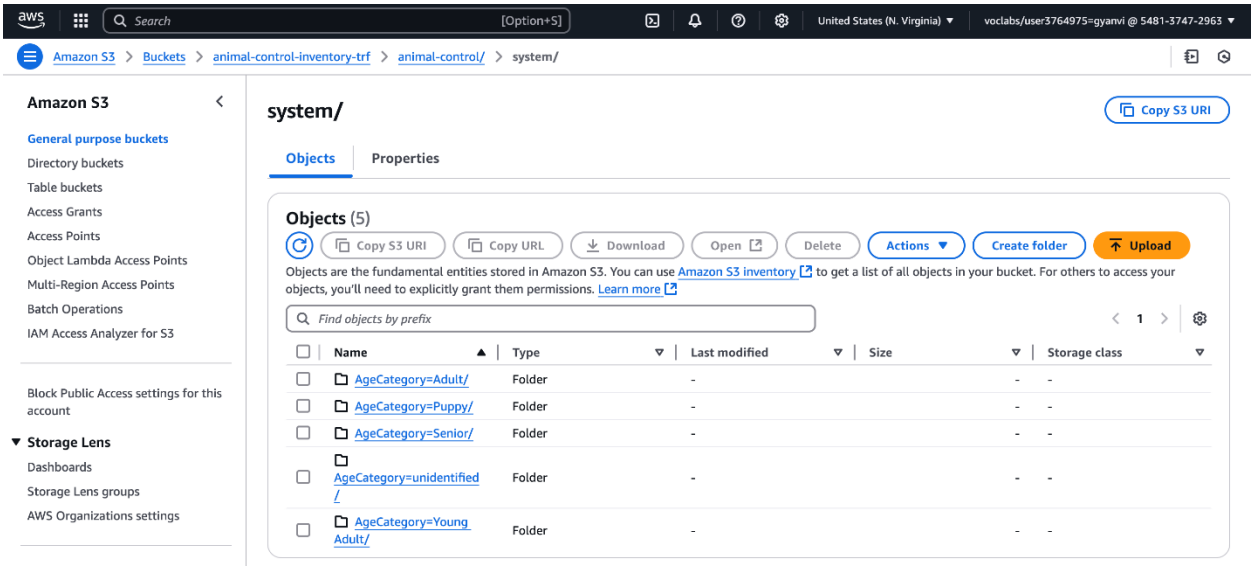
User folder of animal-control-inventory-trf bucket



Note. The user folder of the transformed bucket contains the cleaned dataset file without any partitions. Source: AWS S3, *self-work*

Figure 11

System folder of animal-control-inventory-trf bucket



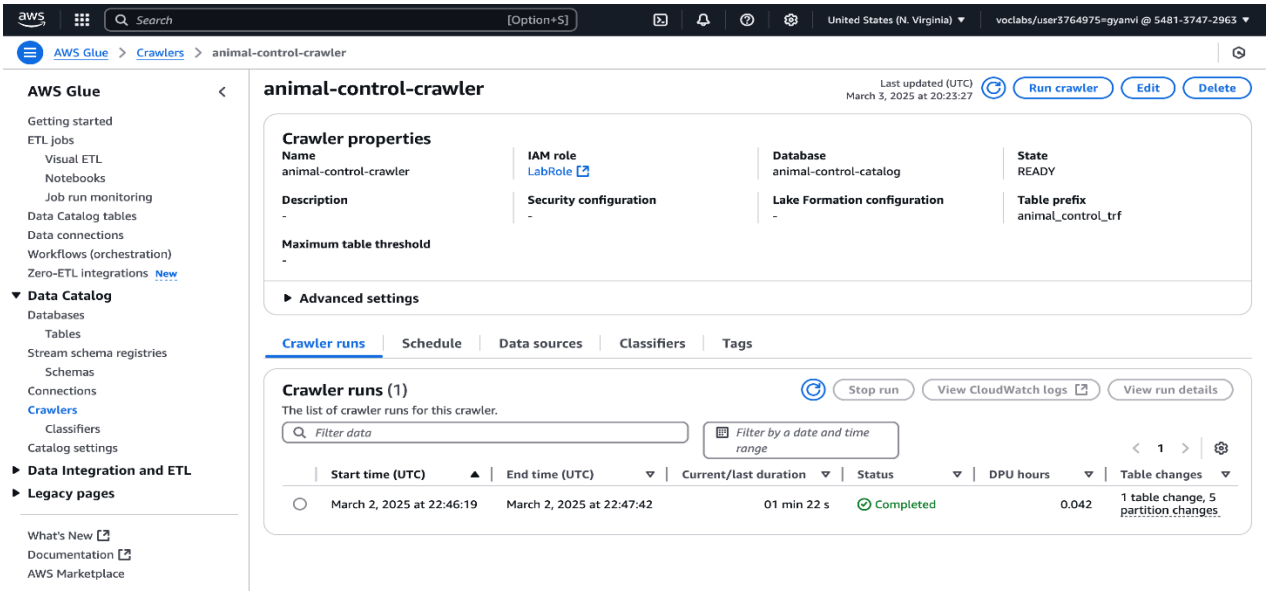
Note. The system folder of the transformed bucket contains the folders partitioned according to the age category of the animals. Source: AWS S3, *self-work*

Step 4: Data Cataloging

The data catalog stands as a single source of truth for my metadata. I started by creating a database on AWS Glue named ‘animal-control-catalog’. A crawler is created and named ‘animal-control-crawler’ to create the meta table in my data catalog.

Figure 12

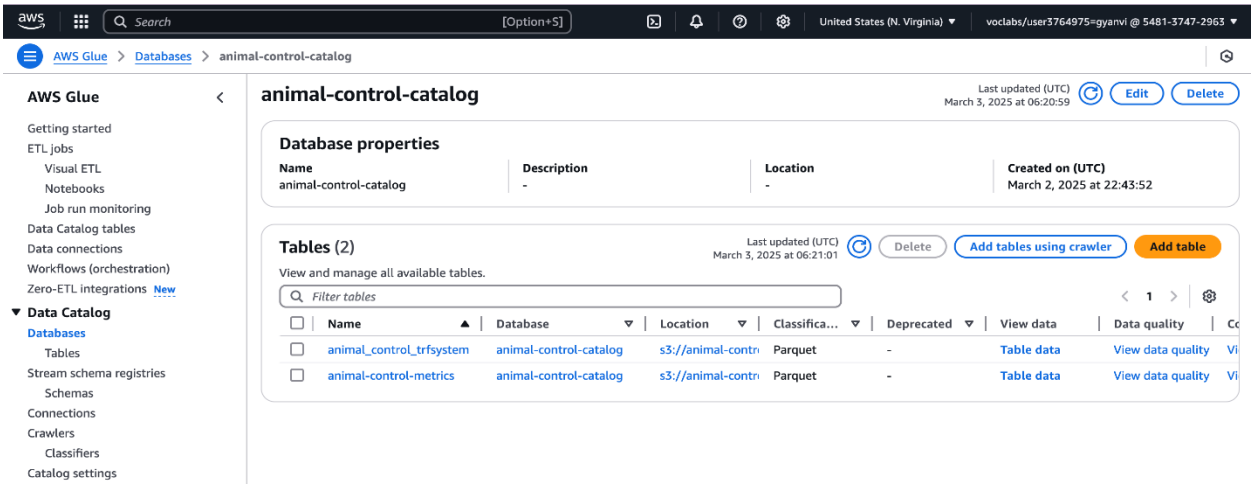
Crawler creation



Note. A crawler is created and named ‘animal-control-crawler’ to create the meta table in my data catalog. Source: AWS Glue, *self-work*

Figure 13

Data catalog with meta tables created



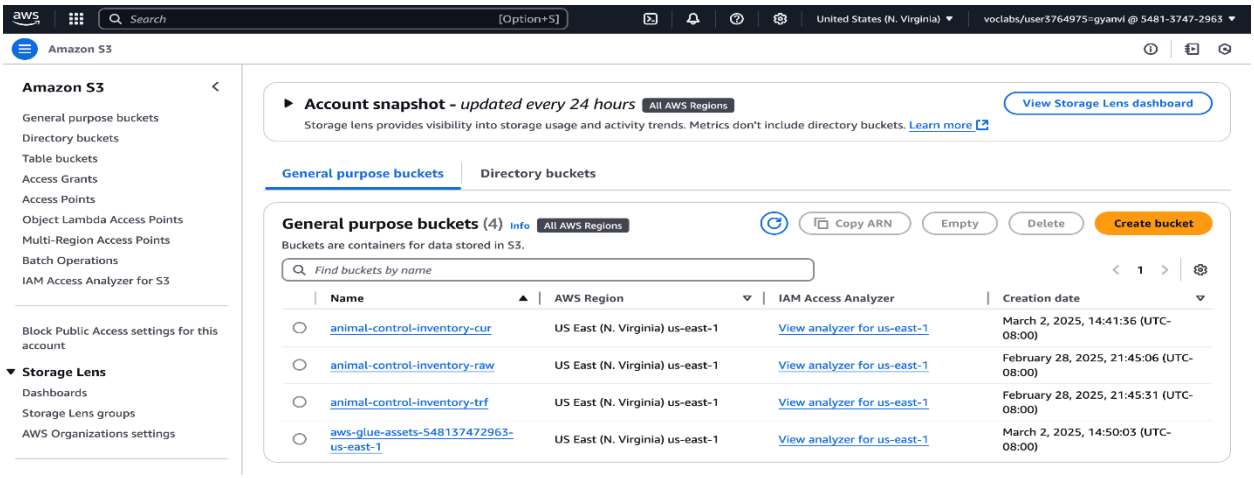
Note. Two meta tables, one with the desired metrics were created in my data catalog. Source: AWS Glue, *self-work*

Moving back to AWS S3, another bucket to store data in the curated zone is created with the name ‘animal-control-inventory-cur’. It has a folder named ‘animal-control’ and then another

folder called ‘metrics’ and two sub-folders named user and system to store user and system-friendly output. Now we have S3 buckets.

Figure 14

AWS S3 buckets- raw, transformed and curated buckets



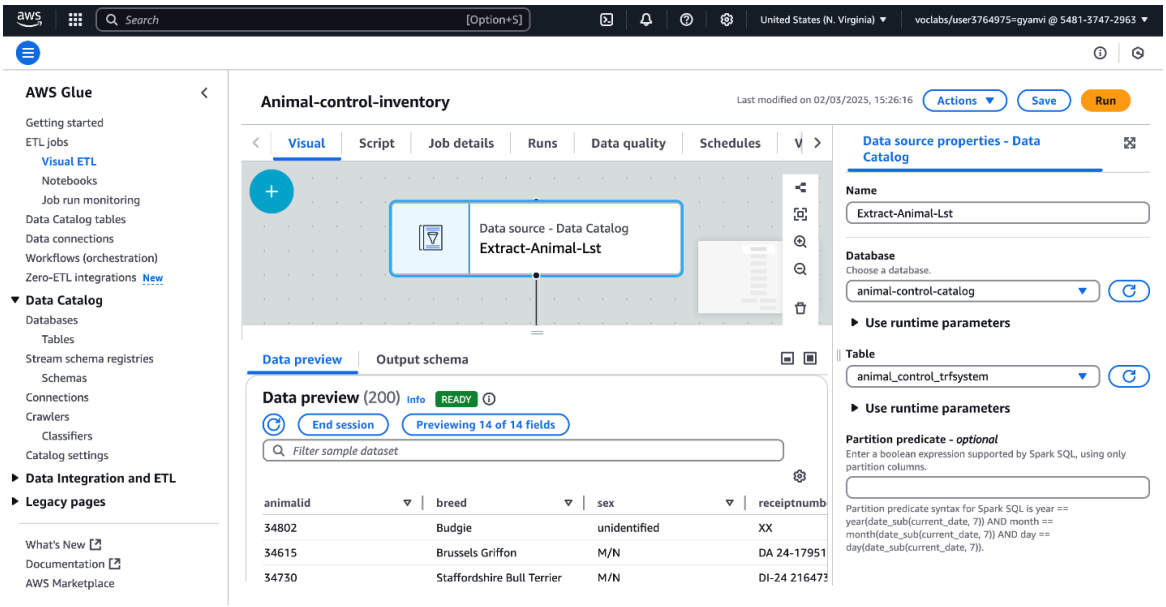
Note. A list of all the buckets created in the AWS S3 environment, including raw, transformed and curated buckets. Source: AWS S3, *self-work*

Step 5: Data Summarization

For data summarization, I implemented an ETL job named ‘Animal-control-inventory’ to create an ETL pipeline. The first step is Extraction which includes inputting a data catalog. The database chosen is ‘animal-control-catalog’ and the table is ‘animal_control_trfsystem’.

Figure 15

ETL Step 1

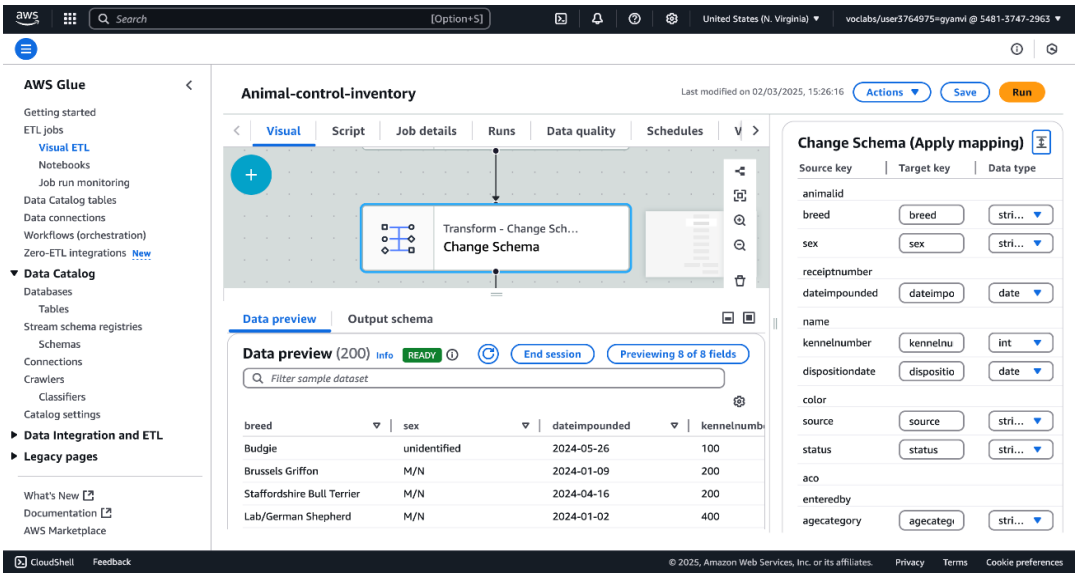


Note. The first step in the visual ETL called extraction is carried out using a data catalog Source: AWS Glue, *self-work*

Next, under transformation, I applied the change schema technique by dropping a few columns called Animal Id, receipt number, name, colour, and ACO, entered by as they do not contribute to summarization.

Figure 16

ETL Step 2

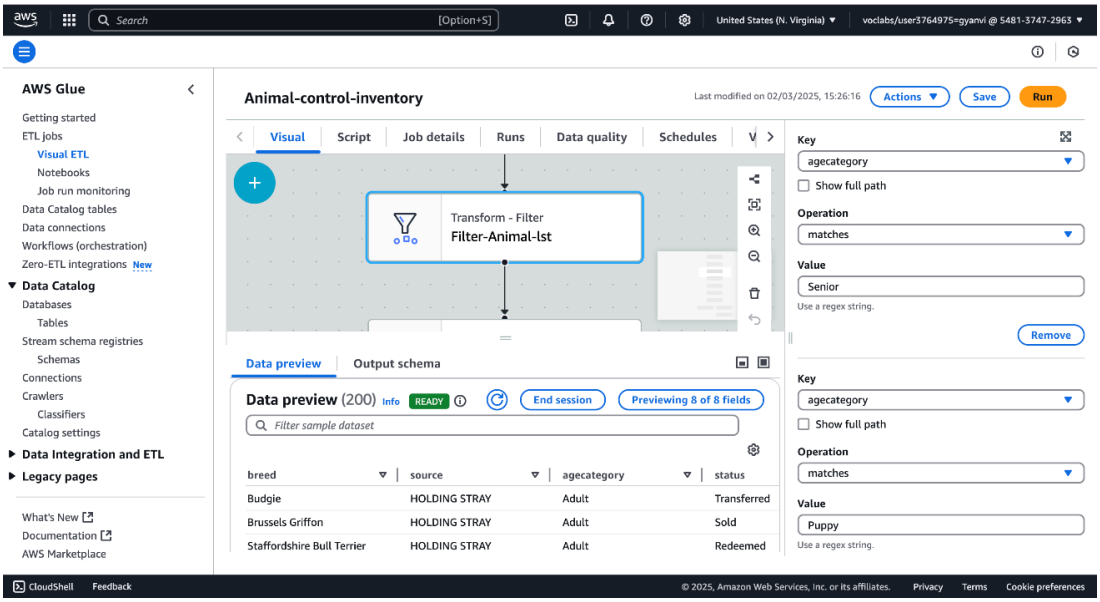


Note. Transformation step called change schema to apply mapping techniques. Source: AWS Glue, *self-work*

Another step under transformation is filter, where I filtered using Global OR and applied four filter conditions. The key is the age category and the operation is ‘matches’. The four values are young, senior, puppy and adult.

Figure 17

ETL Step 3

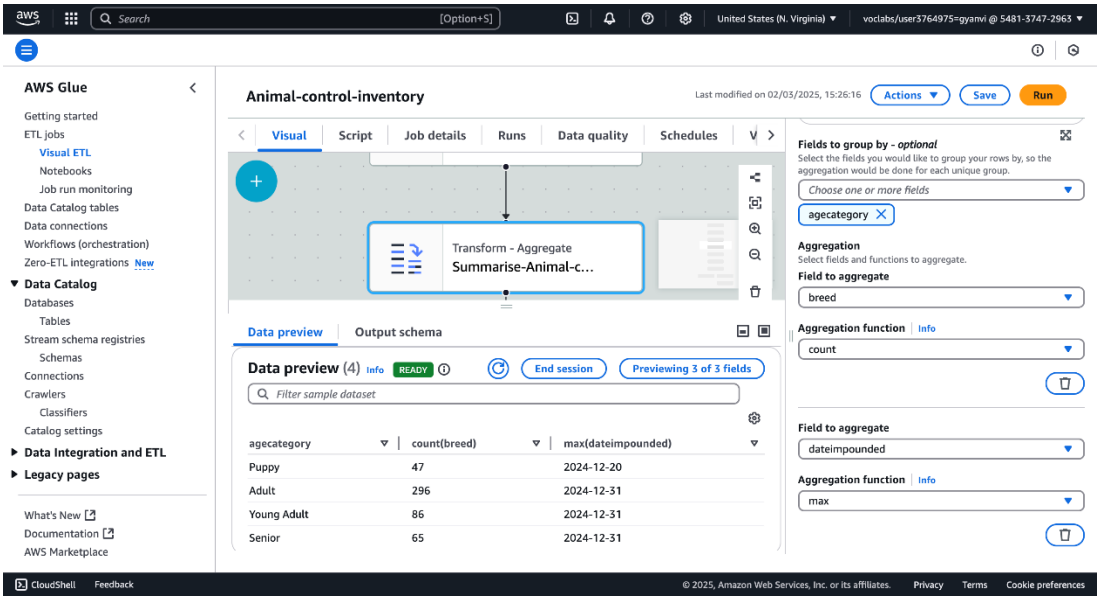


Note. The transformation step called filtering to apply conditions to dataset Source: AWS Glue, *self-work*

Another transformation step is aggregate. I chose the age category as the field to group by as it is the most suitable categorical data. The fields to aggregate are breed and date impounded with count and max as their respective aggregate functions.

Figure 18

ETL Step 4

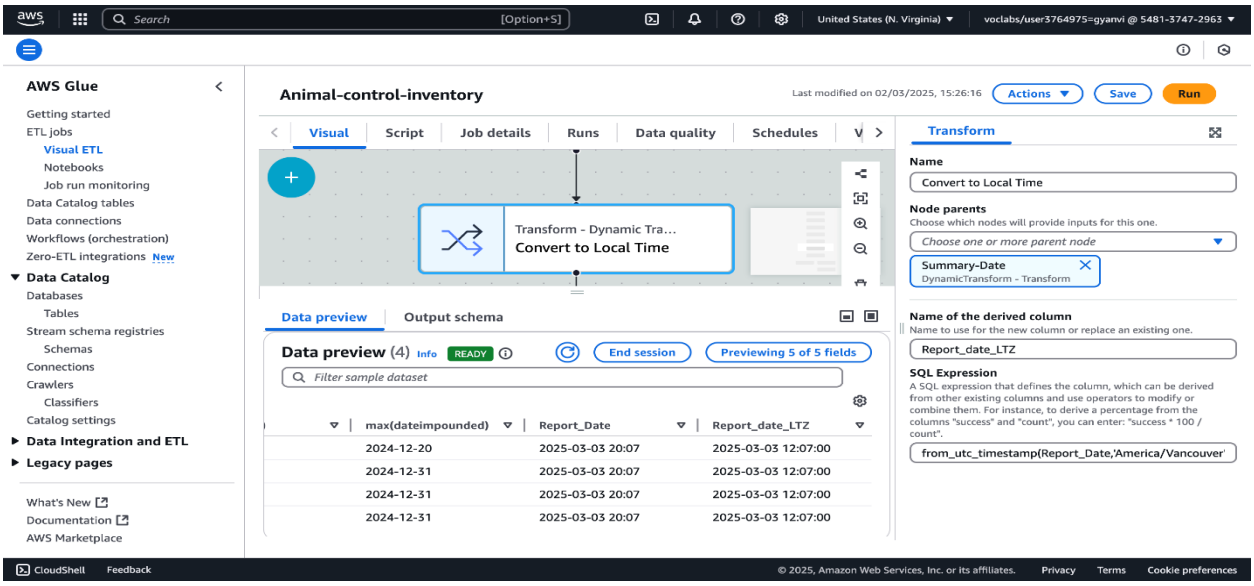


Note. Transformation step called aggregate to perform count and max operations Source: AWS Glue, *self-work*

Next, I added a date and timestamp column under the transformation technique and then used the derived column to change the time zone from American UTC to Vancouver local time. The SQL expression used to do this conversion is from_utc_timestamp(Report_Date,'America/Vancouver').

Figure 19

ETL Step 5 and 6



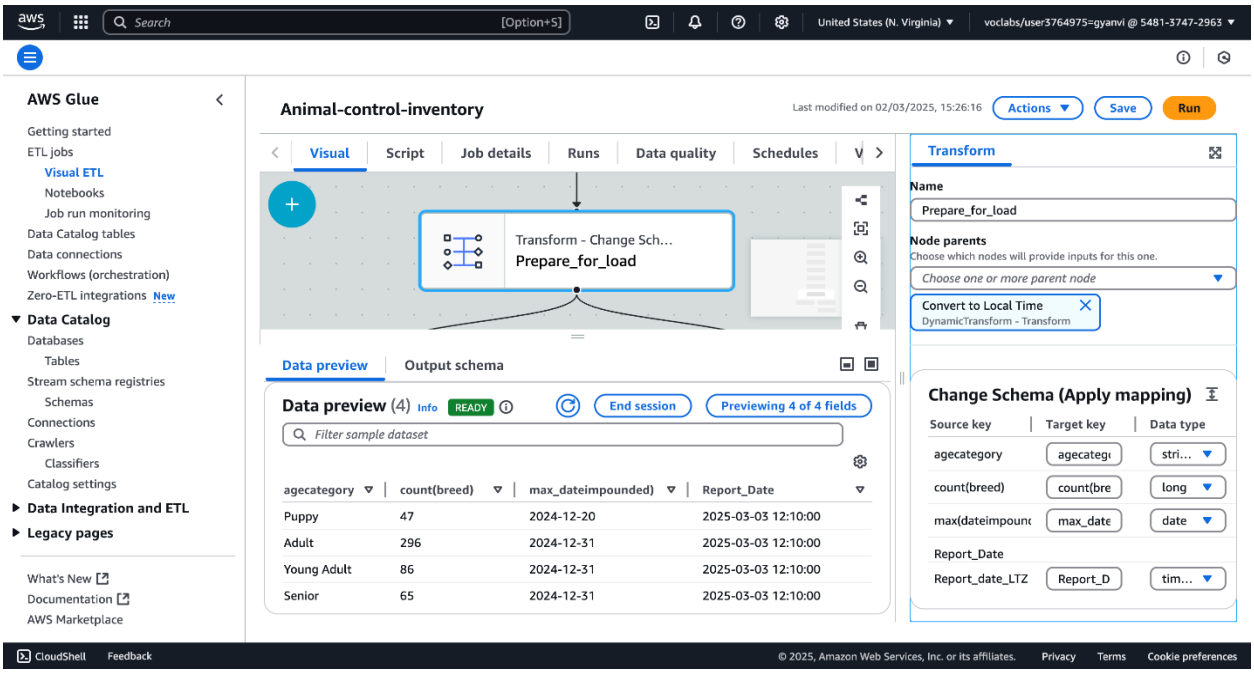
Note. Transformation technique to insert current timestamp and convert it into local time zone

Source: AWS Glue, *self-work*

We will now use the change schema technique to drop the report date column and keep the updated report date to prepare for the load.

Figure 20

ETL Step 7

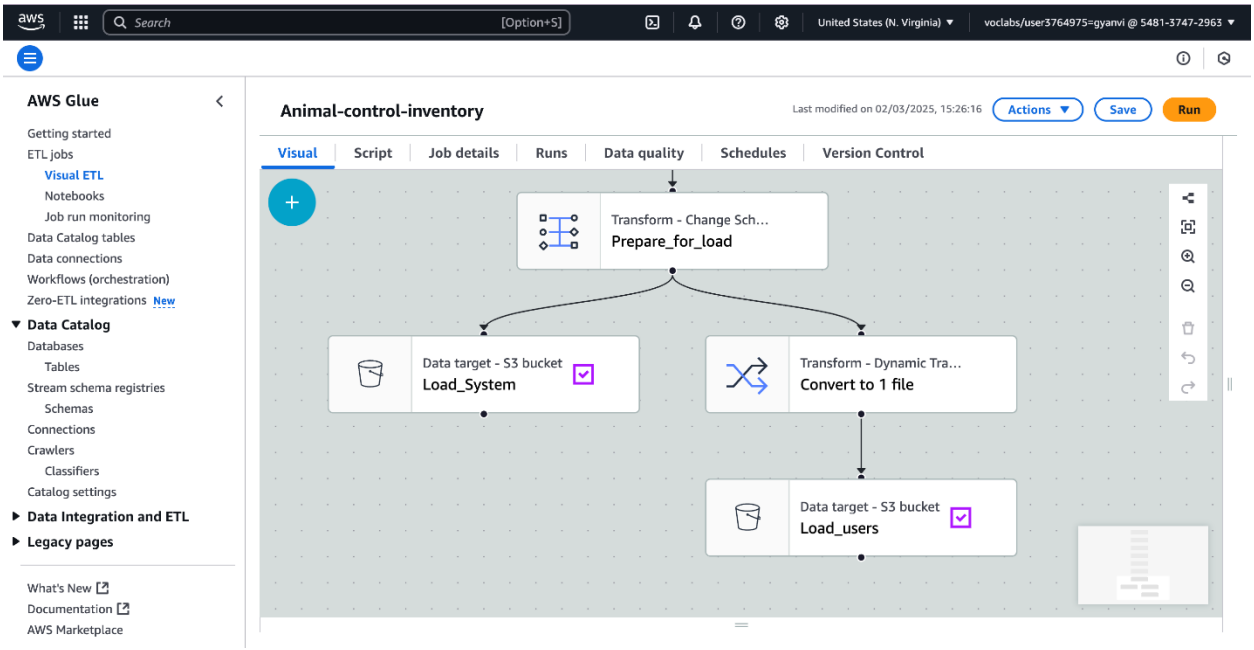


Note. Transformation step to change schema and prepare for loading by dropping American time zone column for report date. Source: AWS Glue, *self-work*

The last step in summarization concludes with two S3 buckets, one for the system and one for users as mentioned in the curated bucket. Before loading the user's S3 bucket, I implemented auto-balancing to convert the output into one file, so it is easier for the user to understand. The table name for the outputs is ‘animal-control-metrics’. The first partition category for the system file is report date and the second partition is using age category.

Figure 21

ETL Step 8, 9, 10

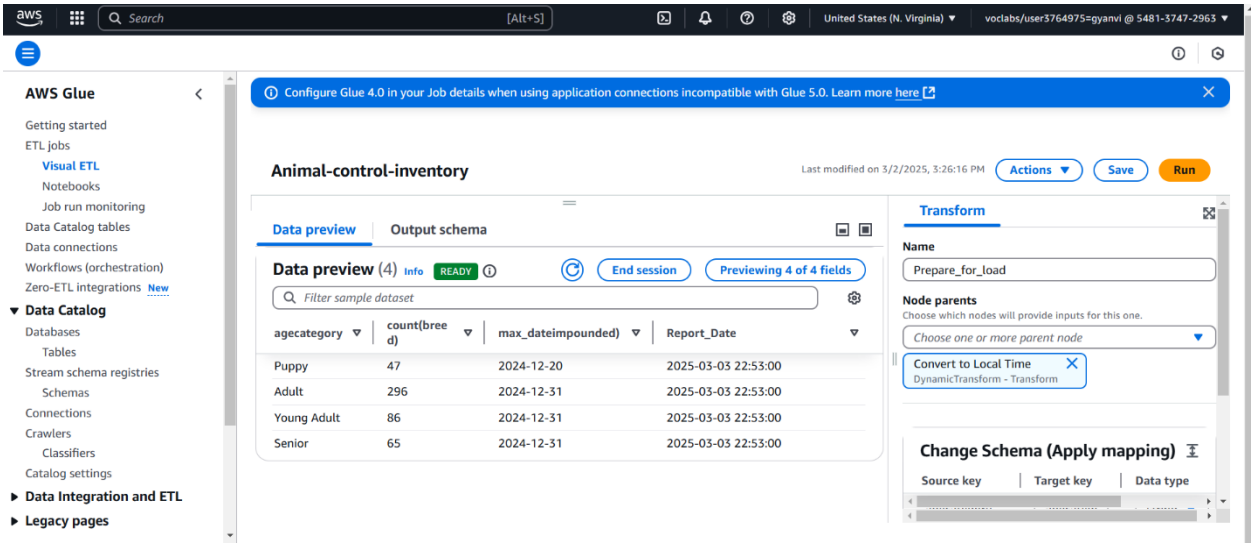


Note. Created two S3 buckets to load system and user data. Source: AWS Glue, *self-work*

The only partition for user output is the report date. Both the metrics- count of breed and maximum impound date are based on the age category of the animal.

Figure 22

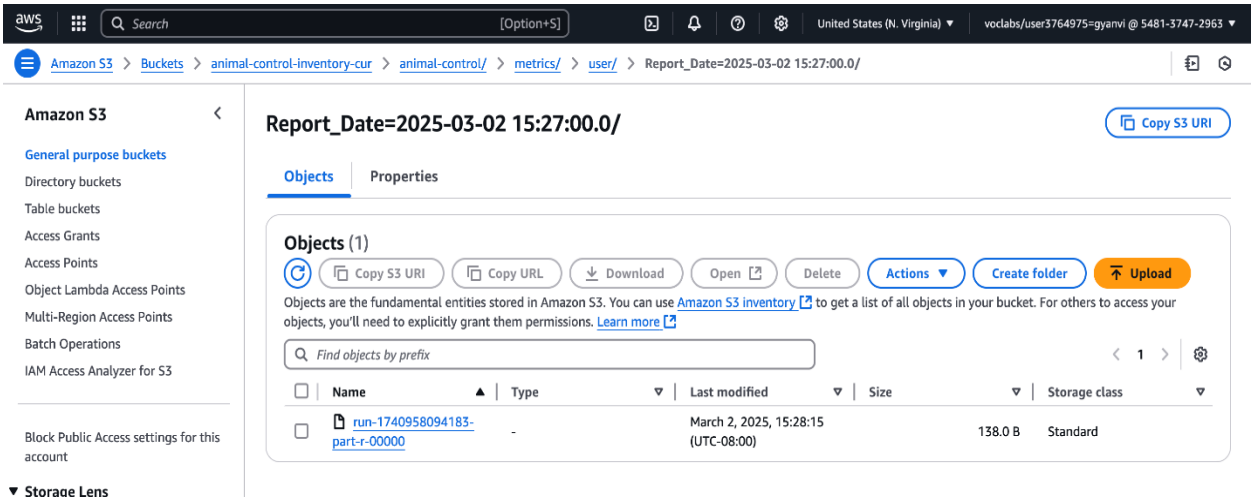
Output of summarization



Note. The two metrics created are counting the number of breeds and the latest (maximum) date of impoundment according to the age category Source: AWS Glue, *self-work*

Figure 23

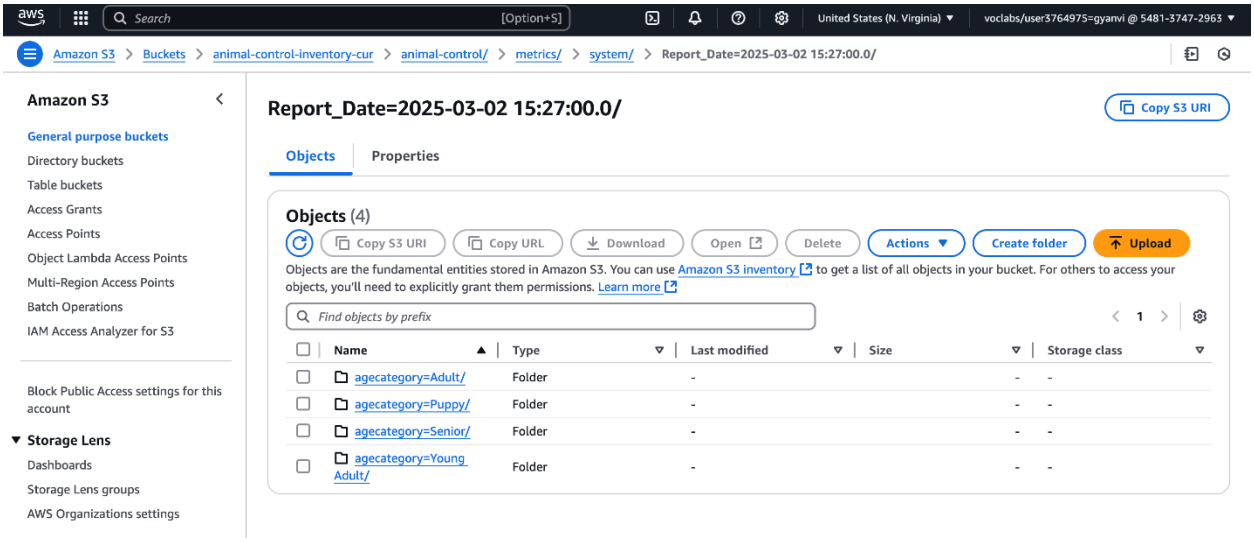
User output in the curated bucket partitioned by the report date



Note. The user folder contains the report date folder which contains the user output file Source: AWS S3, *self-work*

Figure 24

System output based on the age category and partitioned by the report date



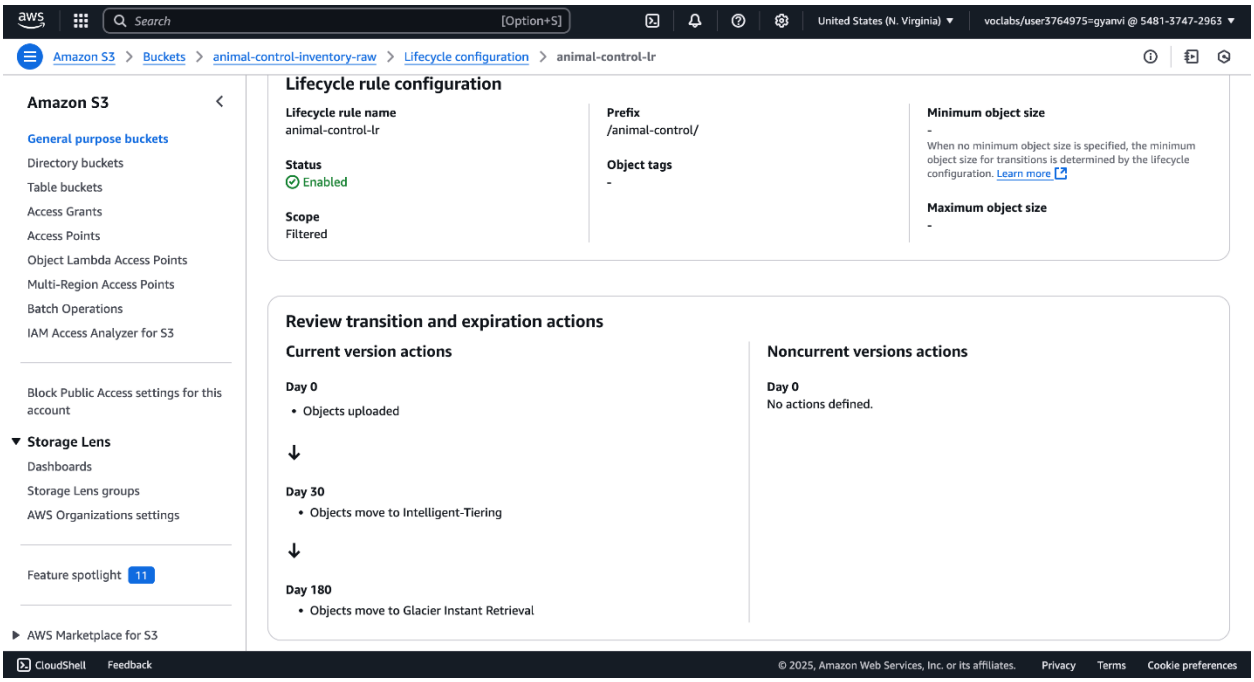
Note. The system folder is partitioned by the report date and this folder consists of other folders partitioned by the age category. Source: AWS S3, *self-work*

Cost Estimation

To further optimize my cost, I have created a lifecycle rule for cost management. The present storage class for my S3 object is standard as the animal control inventory data needs to be accessed frequently and retrieved within milliseconds. However, after 30 days, our storage class can shift to Intelligent tiering as the data trends can vary- there might be fluctuations in accessing data depending on the animals being taken into the custody of the City of Vancouver. Finally, after 180 days, the storage class can shift to glacier instant retrieval for data that sits for a long time and is hardly accessed but when needed, it can be retrieved within milliseconds.

Figure 25

Lifecycle Configuration



Note. Lifecycle rule configuration shows the transition from the standard storage class to Intelligent tiering and to Glacier Instant Retrieval. Source: AWS S3, *self-work*

DAP Design and Implementation (Individual work – Peng Wang)

Descriptive Analysis

I selected 3-1-1 contact metrics, including the data in 2025 and 2024, which has a total of around 400 rows, and the components included date, calls offered, calls handled, calls abandoned, average speed of answer, service level, and BI ID number. Additionally, I implemented this dataset into the AWS platform and gave advice on questions. The first question is about whether the average speed of answer is standard. The second question is about whether call abandoned time is also standard.

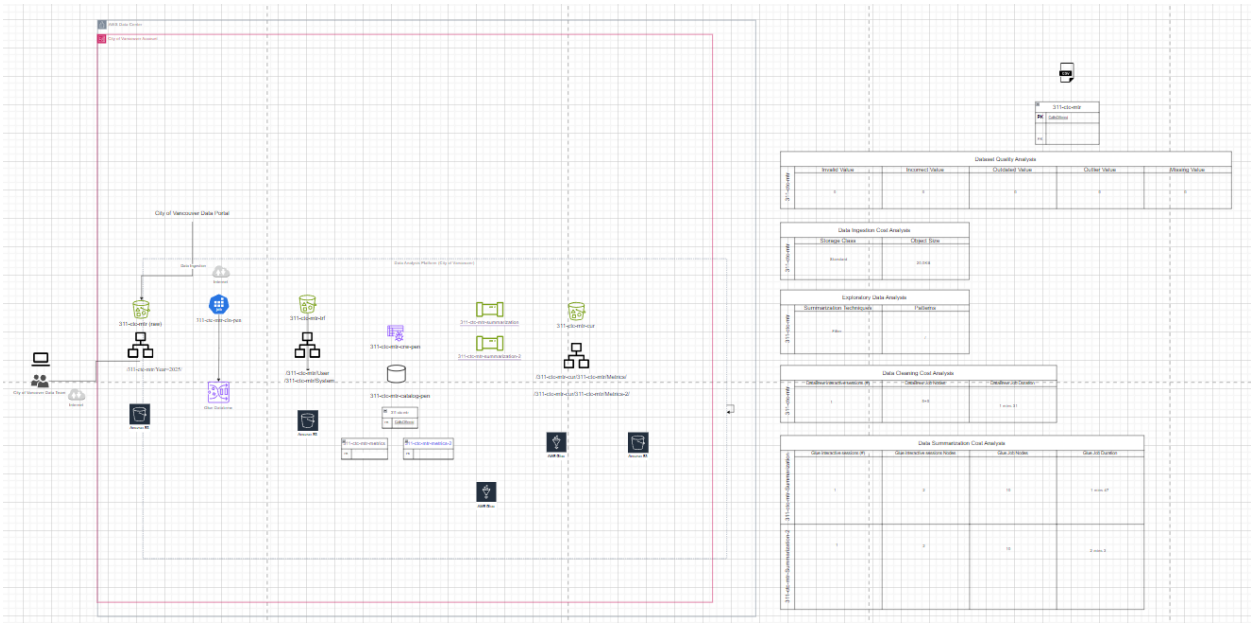
On the left side of the draw.io file, I downloaded the datasets from the City of Vancouver Data Portal and then implemented the data ingestion by creating a raw bucket and sub-folders in Amazon S3 and then uploaded this dataset into this raw bucket. Next, I ran a job in AWS Glue

Databrew and created transfer buckets in AWS S3 and sub-folders including the system and the user to receive the cleaning data. Next, I achieved the Crawlers to prepare for the data catalog, which was classified by two metrics. Finally, I created a new bucket in AWS S3 to receive the summarization data, which is in AWS Glue, into those two metrics.

Additionally, on the right side of the draw.io file, it shows the dataset information. The dataset was named 311-ctc-mtr, and its primary key is Call-Offered. Next, the data quality is all good because there are not any invalid values, incorrect values, and so on. When proceeding with data ingestion, the storage class is standard, and the object size is 20 KB. I only use filter to do the EDA because this dataset only shows numerical values. Next, when proceeding with data cleaning cost analysis, the databrew interactive session is 1, and databrew job notes are 5+5, and databrew job duration is 1 minute 31 seconds. When proceeding with data summarization cost analysis, the glue interactive session, the glue interactive session node, and the glue job node are the same for those two summarizations, which are 1, 2, and 10, respectively, but the difference is the glue job duration. The first summarization ran for 1 minute 47 seconds, and the second one ran for 2 minutes 3 seconds.

Figure 26

All Steps Using draw.io

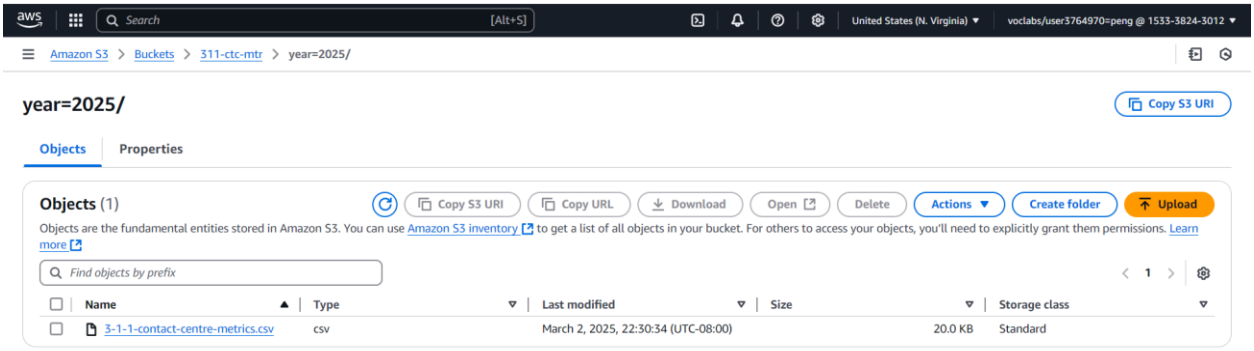


Note. The screenshot shows all the steps using the draw.io file.

Step 1: Data Ingestion

Figure 27

S3 Bucket Building



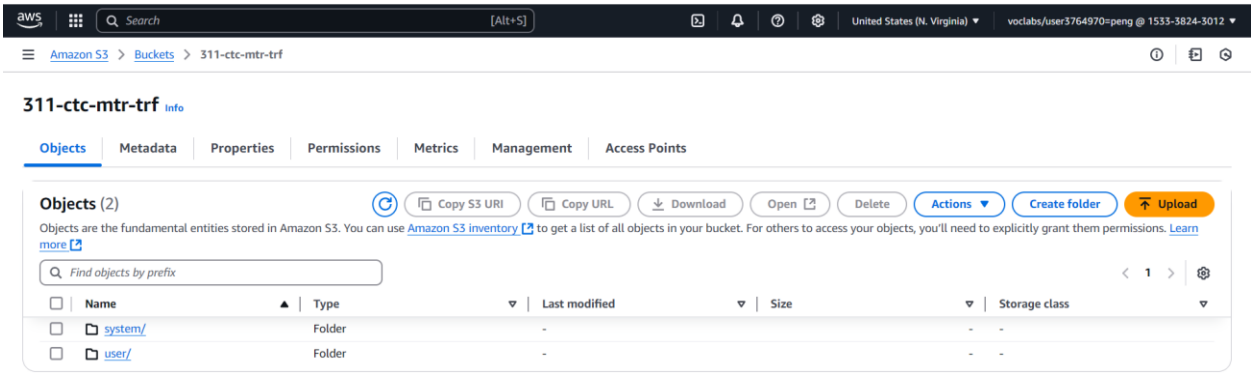
Note. The screenshot shows the S3 bucket. Source from AWS.

Explanation: I downloaded the datasets from the City of Vancouver data portal and created buckets called 311-ctc-mtr as my raw bucket, and then I also created sub-folders called year=2025 which received my dataset when uploading.

Step 2: Data Profiling

Figure 28

Transfer Buckets Building

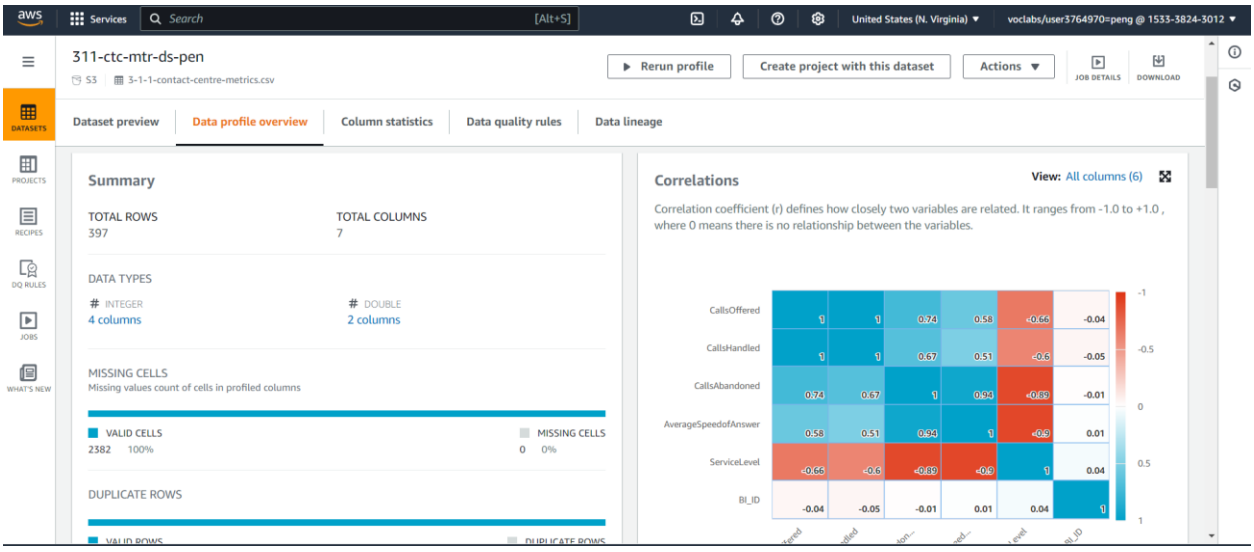


Note. The screenshot shows the transfer bucket has already been built. Source from AWS.

Explanation: I created another bucket called 311-ctc-mtr-trf, which is for data transfer. After that, I created the sub-folders, including system and user, for different uses.

Figure 29

Data Profile Overview



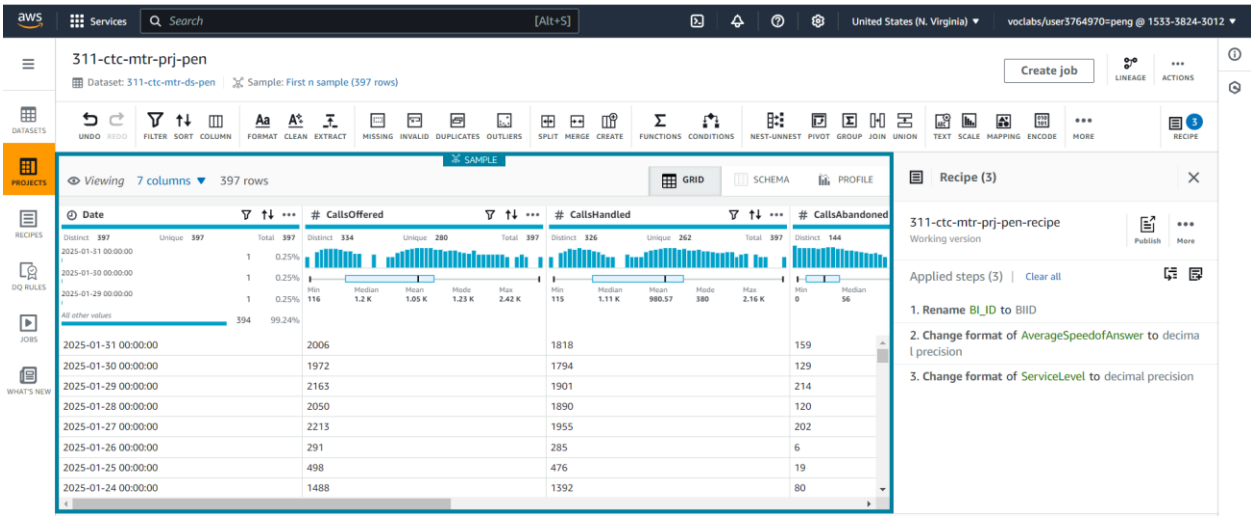
Note. The screenshot shows the data profile overview. Source from AWS.

Explanation: This is the data profile screenshot after running the job.

Step 3: Data Cleaning

Figure 30

Data Cleaning

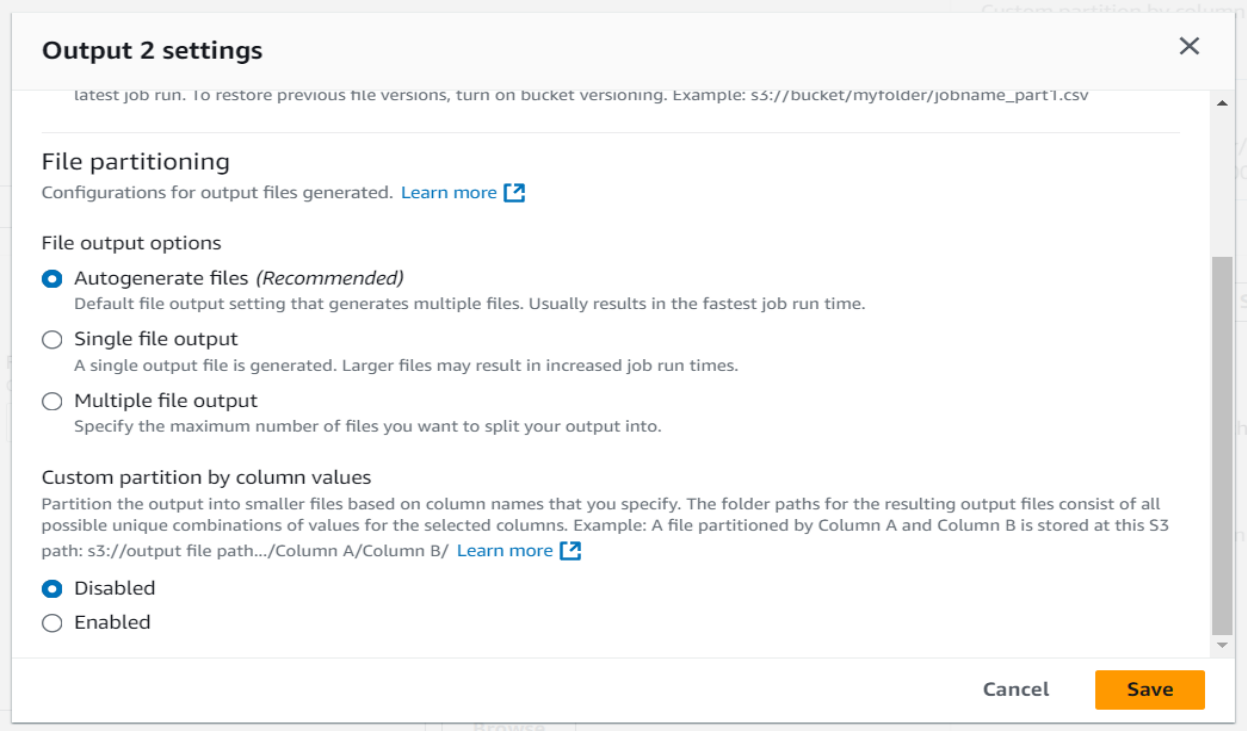


Note. The screenshot shows the data cleaning. Source from AWS.

Explanation: I cleaned the datasets, including renaming, and changing the format to decimal precision of 2 values. That is because it is more directly visual to analyze and keep the data consistently visually.

Figure 31

Output 2 settings

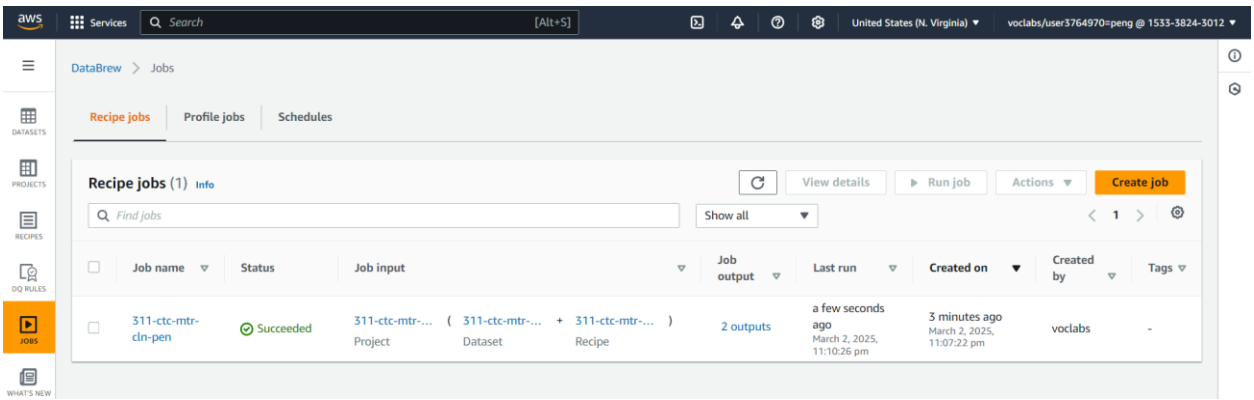


Note. The screenshot shows output 2 settings and no selection of "Enabled" for the custom partition. Source from AWS.

Explanation: When proceeding with data cleaning, I didn't choose custom partition as enabled because there is no column data in my datasets, so it is not effective when classifying the numerical values.

Figure 32

Recipe Jobs

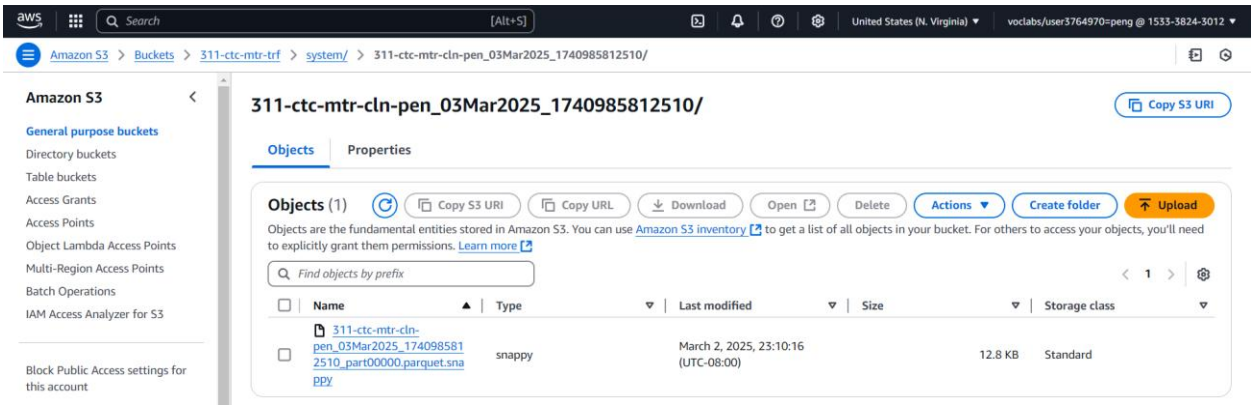


Note. The screenshot shows recipe jobs. Source from AWS.

Explanation: when the job is run successfully, the cleaning data can be found in system and user buckets.

Figure 33

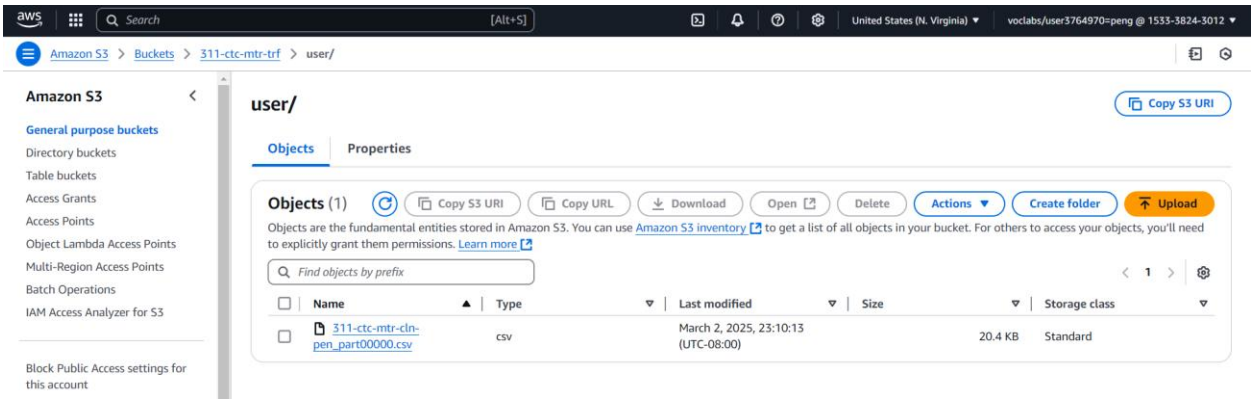
System Bucket for Cleaning Data



Note. The screenshot shows recipe jobs. Source from AWS.

Figure 34

User Bucket for Cleaning Data



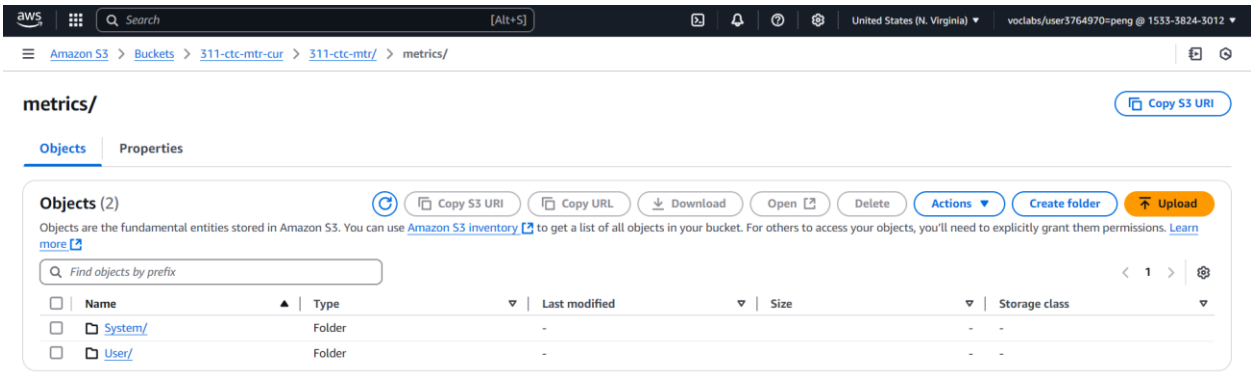
Explanation: Now, the cleaning data has already been in the transfer bucket, which will be used for data cataloging.

Step 4

Data Cataloging

Figure 35

Metrics Folders Building

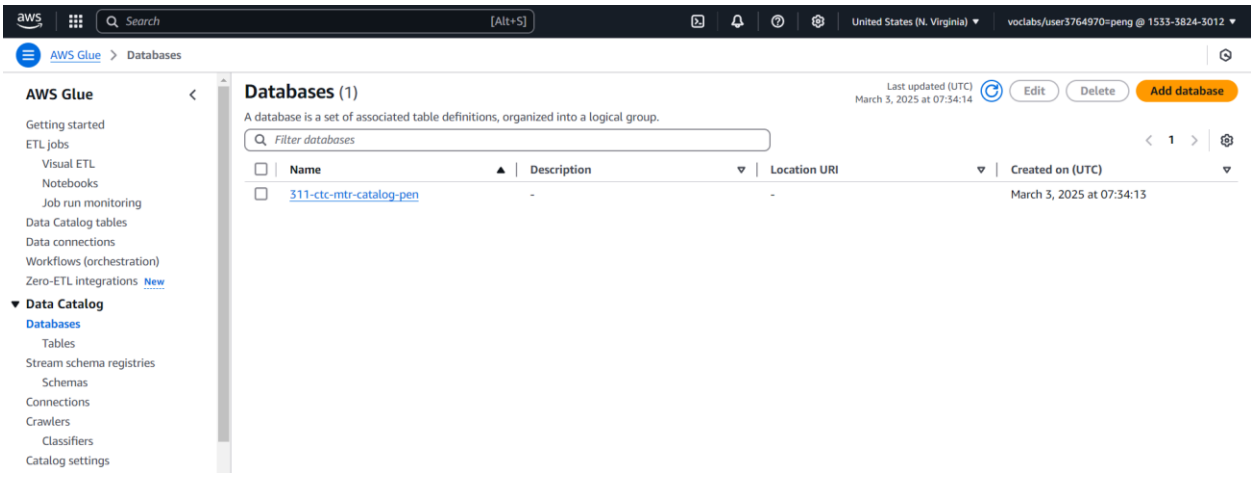


Note. The screenshot shows the metrics folder building. Source from AWS.

Explanation: I created a new bucket called 311-ctc-mtr-cur and sub-folders for metrics including system and user folders, which will be stored for summarization.

Figure 36

Databases Building

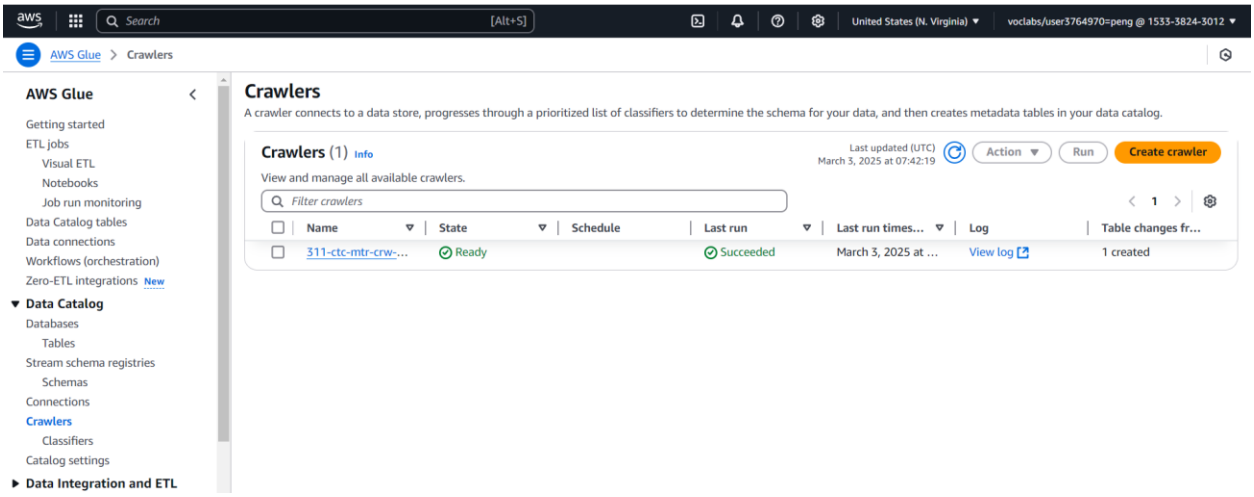


Note. The screenshot shows the database building. Source from AWS.

Explanation: I created a database called 311-ctc-mtr-catalog-pen to receive the cleaning data.

Figure 37

Crawlers Building

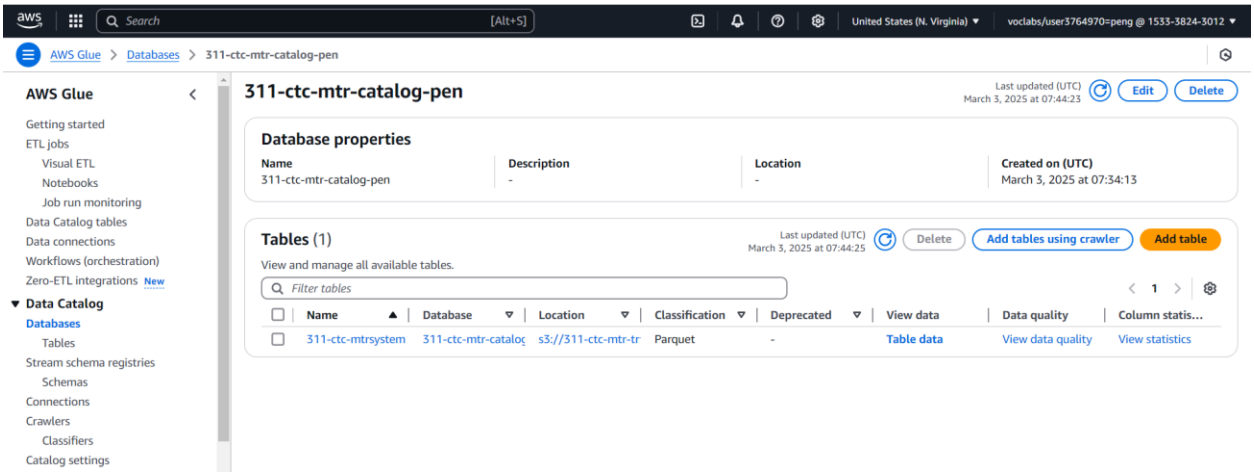


Note. The screenshot shows the crawler building. Source from AWS.

Explanation: I created a crawler to prepare for the data catalog.

Figure 38

Transfer Cleaning Data to Databases for catalog



Note. The screenshot shows the cleaning data has already been transferred to datasets for catalog.

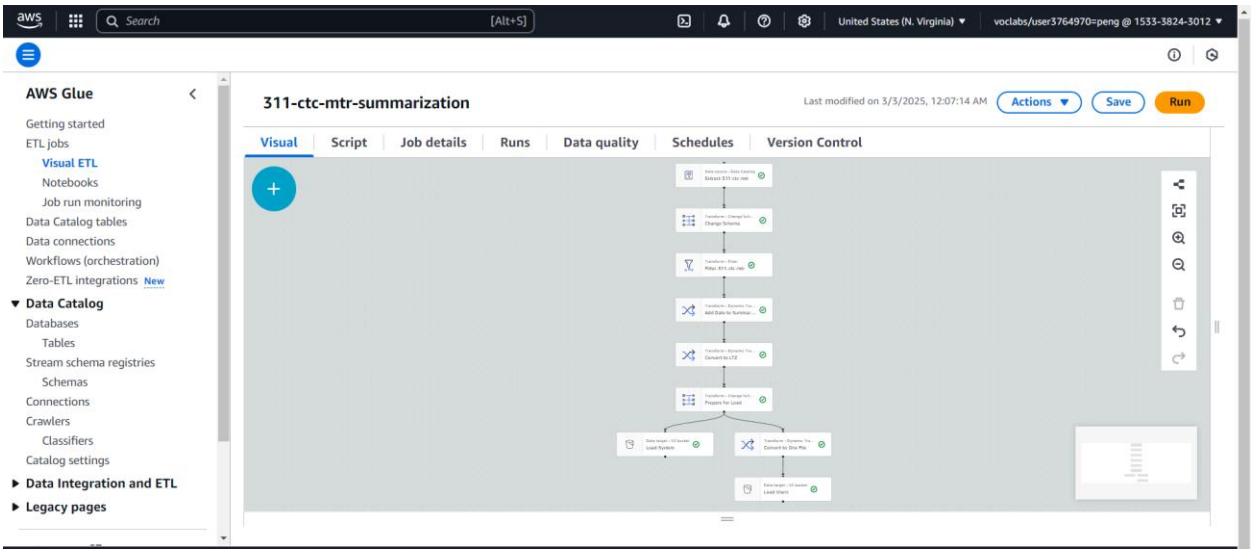
Source from AWS.

Explanation: When the Crawler was ready, the cleaning data was transferred to databases for cataloging and prepared for summarization.

Step 5: Data Summarization

Figure 39

ETL pipeline 1

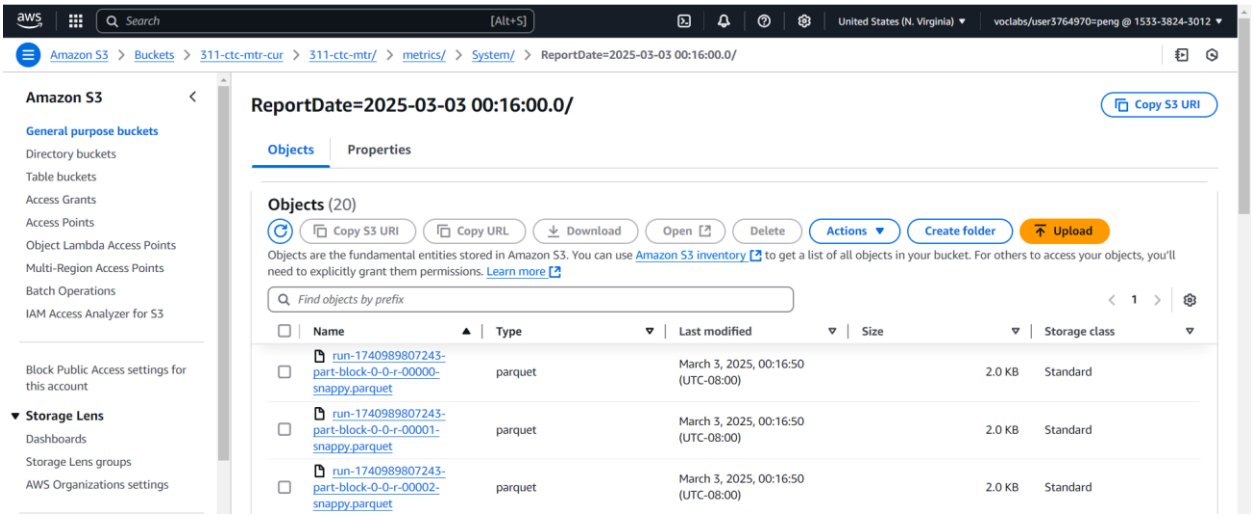


Note. The screenshot shows dataset summarization. Source from AWS.

Explanation: I extracted the data from the system, and then I didn't change the schema because all the data was good. Next, I did a filter for the key to the average speed of the answer less than 100. Next, I added the date to the summary report and changed the date to local time in Vancouver, which dropped the date and the original report date. Next, I loaded the data into the system. Meanwhile, I loaded the data into users by CSV file by converting one file.

Figure 40

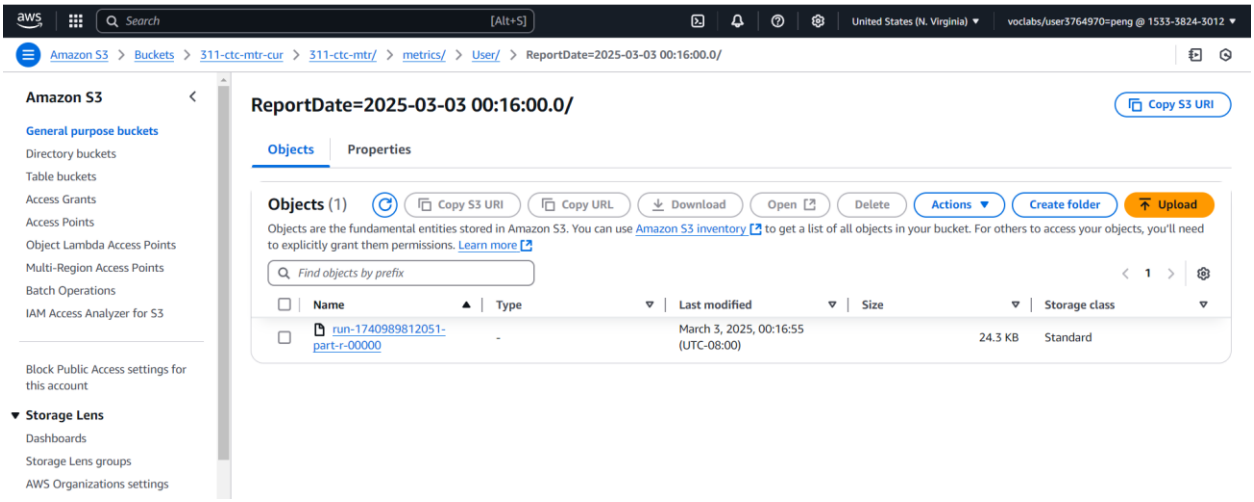
Metrics Report for System



Note. The screenshot shows the metrics report for the system. Source from AWS.

Figure 41

Metrics Report for User

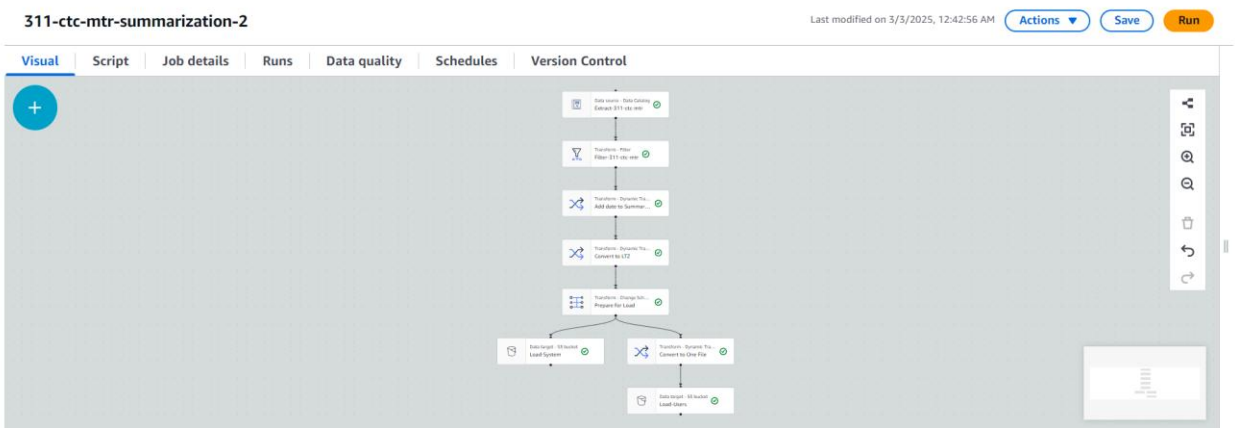


Note. The screenshot shows the metrics report for users. Source from AWS.

Explanation: When I ran my pipeline successfully, the report data was shown in the system and users, respectively, in order for summarization. For example, users can download the final dataset after cleaning.

Figure 42

ETL pipeline 2

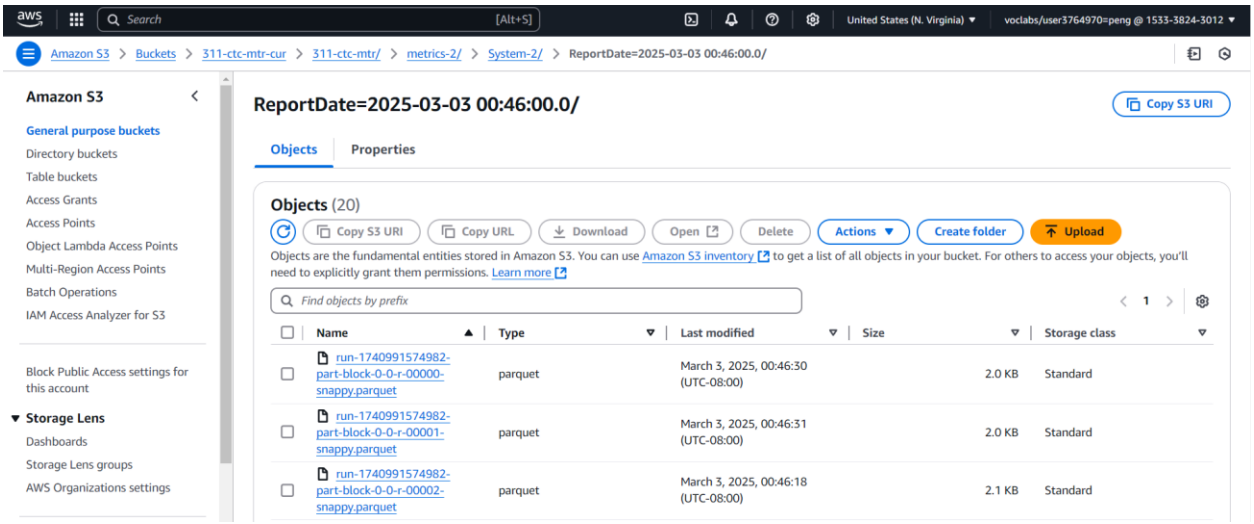


Note. The screenshot shows dataset summarization-2. Source from AWS.

Explanation: I extracted the data from the system, and then I did a filter for the key to the call abandoned less than 100. Next, I added the date to the summary report and changed the date to local time in Vancouver, which dropped the date and the original report date. Next, I loaded the data into the system. Meanwhile, I loaded the data into users by CSV file by converting one file.

Figure 43

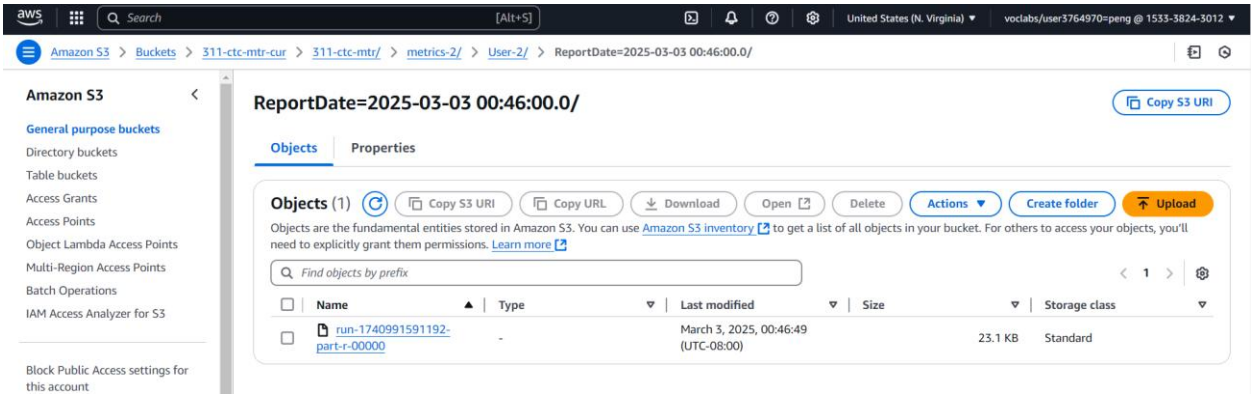
Metrics Report for System - 2



Note. The screenshot shows the metrics report for the system - 2. Source from AWS.

Figure 44

Metrics Report for User – 2



Note. The screenshot shows the metrics report for users – 2. Source from AWS.

Explanation: When I ran my pipeline successfully again, the report data was shown in the system and users, respectively, in order for summarization. For example, users can download the final dataset after cleaning.

DAP Design and Implementation (Individual work – Muhammad Zulqarnain Shahzad)

Descriptive Analysis

Dataset Chosen

- Business Licences – Consulting and Management Services dataset from the City of Vancouver Open Data Portal.
- Link: Consulting and Management Services 2024

Why This Dataset

- It meets the required volume (approximately 500 rows) for the assignment.

- It contains key information on employee counts and license fees, which is crucial for understanding the performance of different management consulting subdomains.

Goal

To analyze the performance of management consulting subdomains by determining which ones have the highest employee counts and evaluating key cost metrics (such as license fee paid).

Analytical Questions

- Which management consulting subdomain has the highest number of employees?
- What is the maximum, minimum, and average employee counts across these subdomains?
- What is the average license fee paid by organizations in each subdomain?

Metrics

Employee Count Metrics

- Maximum employee count per subdomain
- Minimum employee count per subdomain
- Average employee count per subdomain

License Fee Metric

- Average license fee paid

Overview: What I Did and Why I Did It

In this project, I downloaded the dataset from the City of Vancouver data portal website and selected the December 2024 data, as it contained approximately 500 rows the required size for this project. I then uploaded the original CSV file (which uses a semicolon as its delimiter) to an S3 bucket named “bl-cms-dec-2024-zul” to keep the raw data intact. I did this to ensure that the dataset would be properly analysed later on. Next, I used AWS Glue DataBrew to profile and clean the data, where I corrected the delimiter, standardized column names, reformatted date fields, and dropped unnecessary columns. This cleaning step made the dataset consistent and easier to work with. After that, I ran an AWS Glue Crawler to automatically create a database schema, making it simpler to query the cleaned data with Athena. Finally, I built a single ETL pipeline in AWS Glue Studio to transform, filter, and aggregate the data (for example, to calculate how many licenses were issued or such as the number of employees > 50) and stored the final results in the transform, system, and user folders. This streamlined process ensures that the data is quickly accessible, neatly organized, and ready for further analysis. For a more detailed step-by-step breakdown, please refer to the subsequent sections of this document.

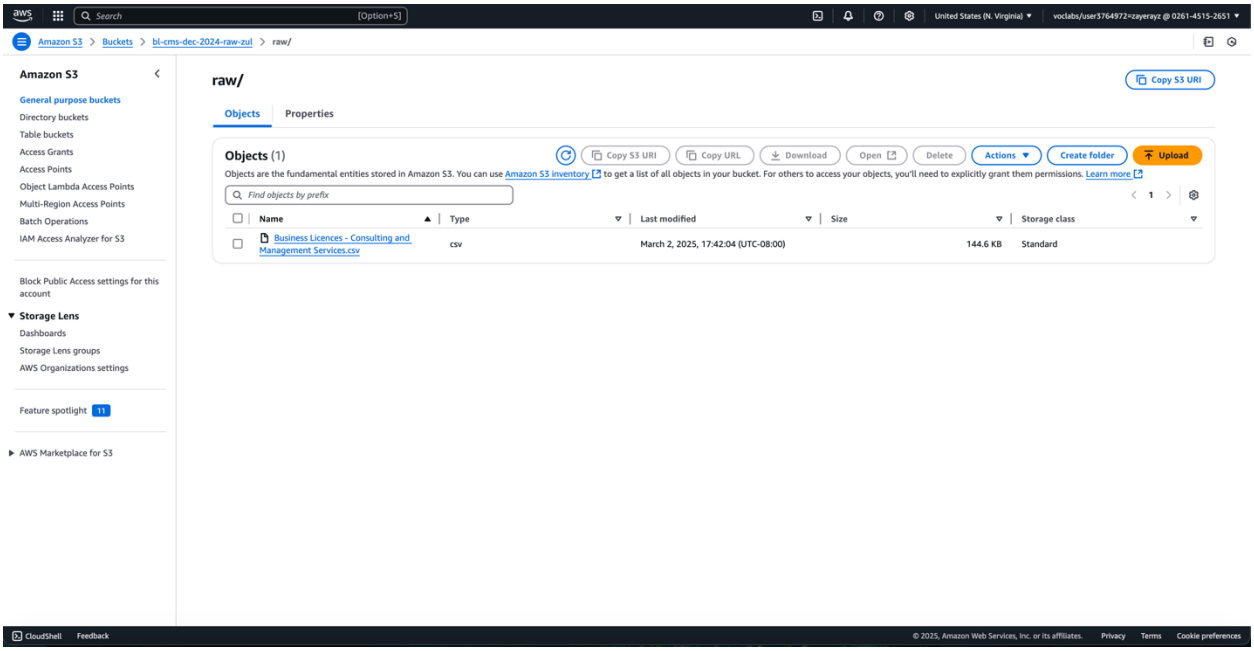
Step 1: Data Ingestion

I started by creating an S3 bucket named “bl-cms-dec-2024-zul” to serve as the centralized repository for my project data. Within this bucket, I created a folder named “raw” where I uploaded the original CSV file. The file contains Business Licenses – Consulting and Management Services data and uses a semicolon as its delimiter. It was essential to preserve this detail during ingestion so that the data would be parsed correctly in later steps. I verified the file

properties (such as file size, upload date, etc.) in the S3 console to ensure that the raw data was intact and accessible.

Figure 45

S3 Bucket – Raw Folder and Uploaded CSV



Note. Screenshot from AWS S3 console showing bucket “bl-cms-dec-2024-zul” with the “raw” folder containing the original Business Licences – Consulting and Management Services CSV file. Source: AWS, *self-work*

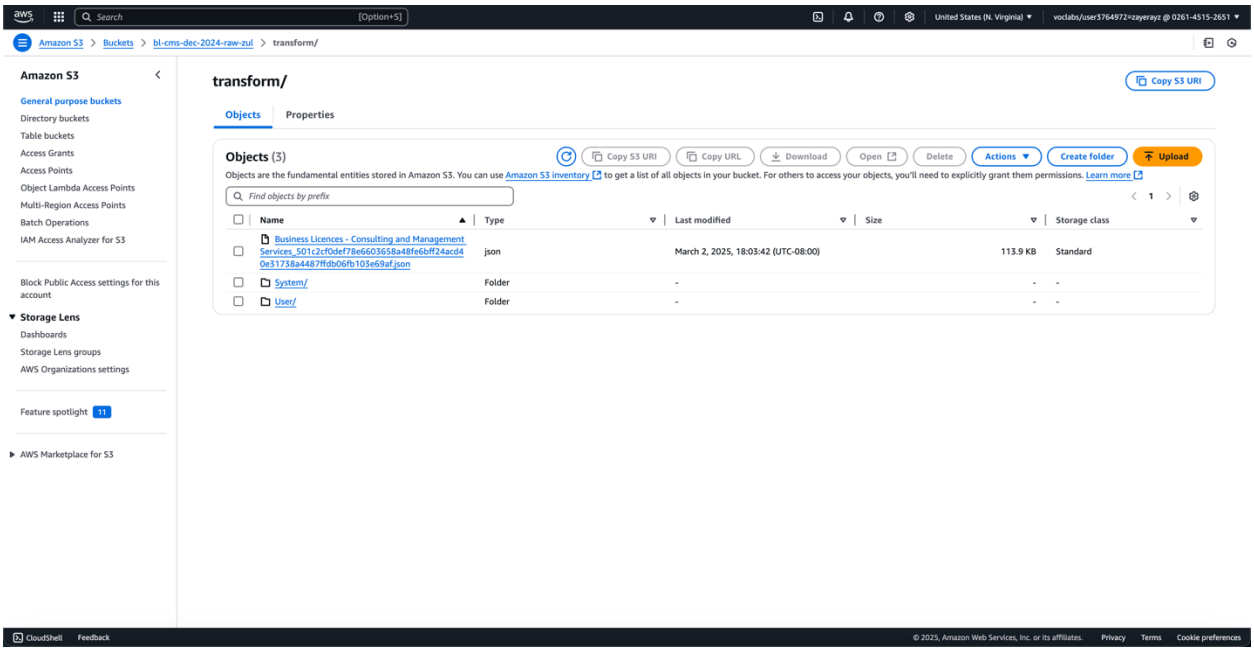
I show my S3 bucket named “bl-raw-zul” with the “raw” folder clearly visible. Here, I uploaded the original CSV file containing Business Licenses – Consulting and Management Services data. Since the CSV uses a semicolon as its delimiter, I made sure to preserve that detail during ingestion. I also verified the file’s properties such as size and upload date to confirm that the data was intact before moving on.

Step 2: Data Profiling

Next, I set up an AWS Glue DataBrew project to profile the dataset. Initially, I encountered an issue where the data was not splitting into columns correctly because the default CSV delimiter was a comma. I corrected this by configuring the project to use a semicolon as the delimiter exactly as specified on the data portal. With the correct configuration, the data preview in DataBrew displayed the columns properly, and summary statistics (data types, null counts, etc.) were generated. This profiling step was crucial to identify any data quality issues and to plan the cleaning steps accordingly.

Figure 46

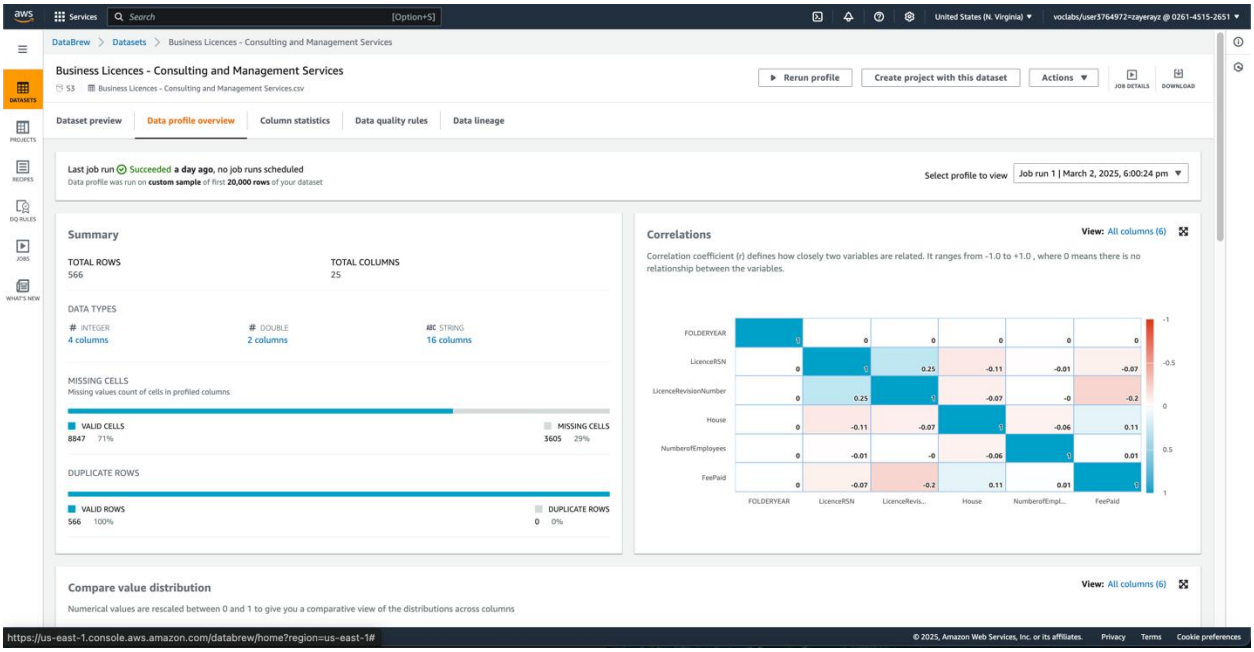
Transform



Note. Screenshot from AWS S3 console showing Transform bucket + System and User folders. Source: AWS, *self-work*

Figure 47

DataBrew Project – Data Preview



Note. Screenshot showing the data preview in DataBrew with summary statistics (data types, null counts) confirming proper column splitting. Source: AWS, *self-work*

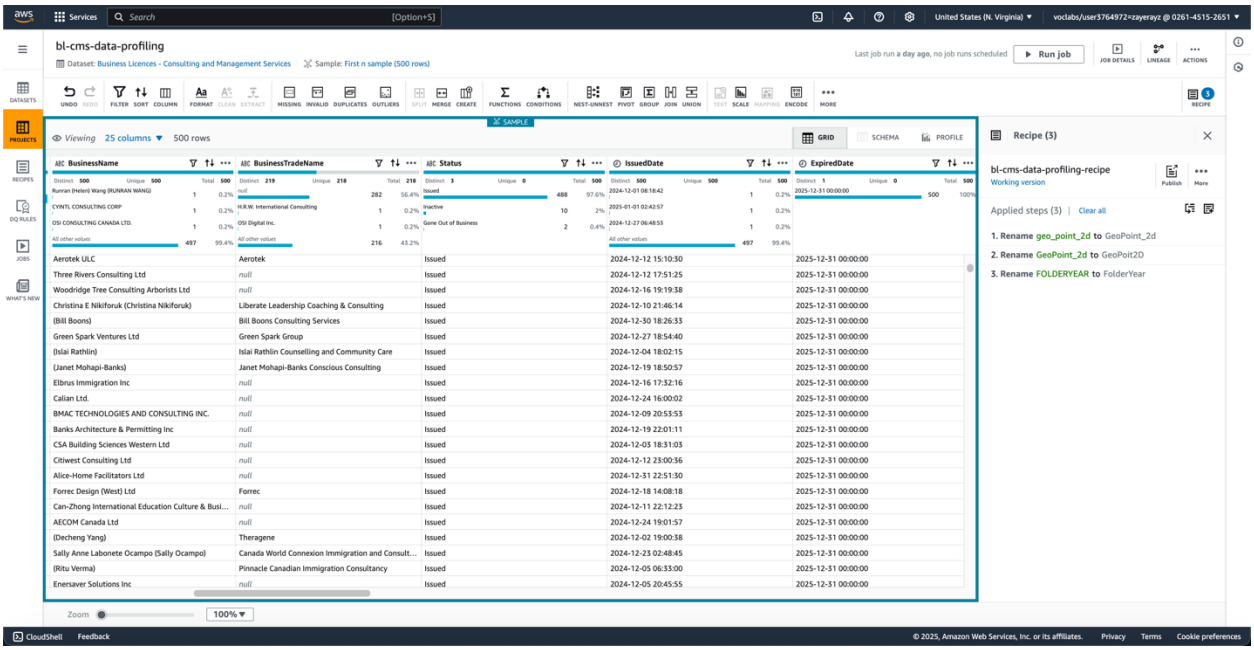
Here, I set up an AWS Glue DataBrew project to profile my dataset. Initially, the data did not split into columns because the default delimiter was a comma. After configuring the project to use a semicolon, the preview now shows the columns correctly. This step was key to identifying data quality issues early on. This shows the summary statistics generated by DataBrew, including data types and null counts. The view reassured me that all columns are correctly interpreted, and I'm set to proceed to cleaning.

Step 3: Data Cleaning

Once the data profile was verified, I created a cleaning recipe in DataBrew. In the recipe editor, I standardized the column names to be more descriptive and reformatted the date fields into the “yyyy-mm-dd” format. I also reviewed all columns and dropped those that were not needed for the analysis (for example, columns like “geom” and “geopoint2d”). This process streamlined the dataset, reducing unnecessary clutter and improving processing efficiency. The cleaned data was then output to two separate locations: a “system” folder for technical verification and a “user” folder for the final analysis-ready data.

Figure 48

DataBrew Recipe – Cleaning Steps Overview

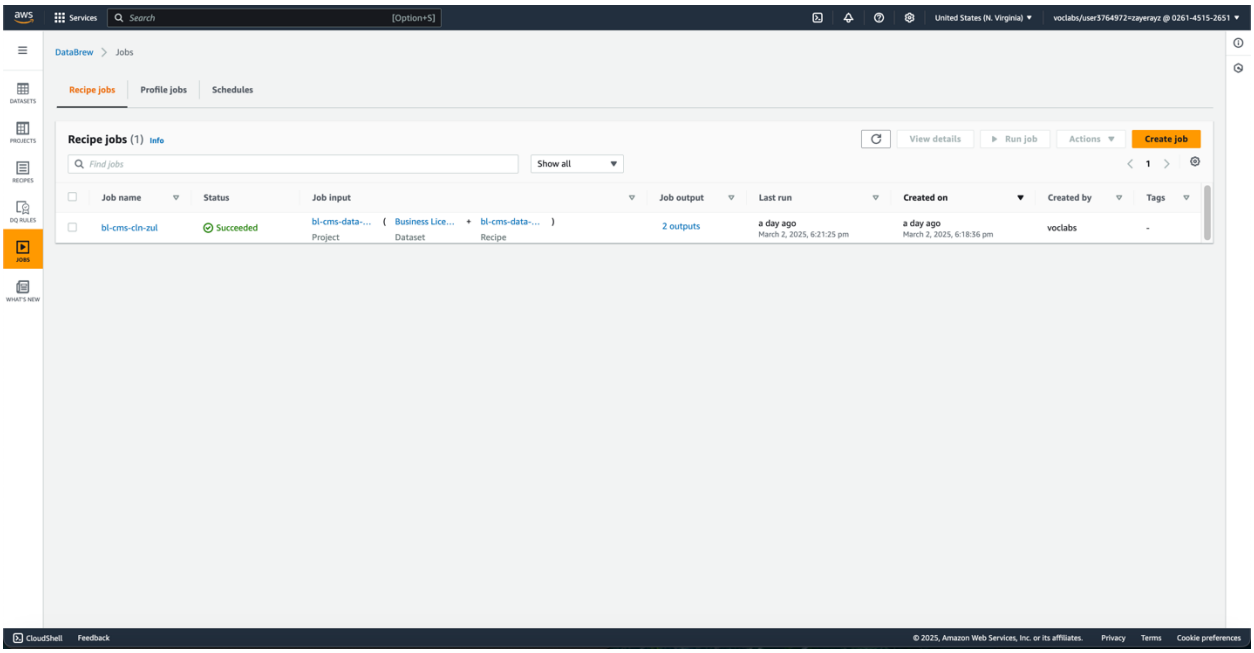


Note. Screenshot of the cleaning recipe in DataBrew where column names are standardized and date fields reformatted to “yyyy-mm-dd,” with unnecessary columns dropped. Source: AWS, *self-work*

In this image, you can see the cleaning recipe in DataBrew. I standardized the column names to be more descriptive and reformatted date fields into the “yyyy-mm-dd” format. Additionally, I dropped unnecessary columns (e.g., geographic coordinate fields) to simplify the dataset and improve processing efficiency.

Figure 49

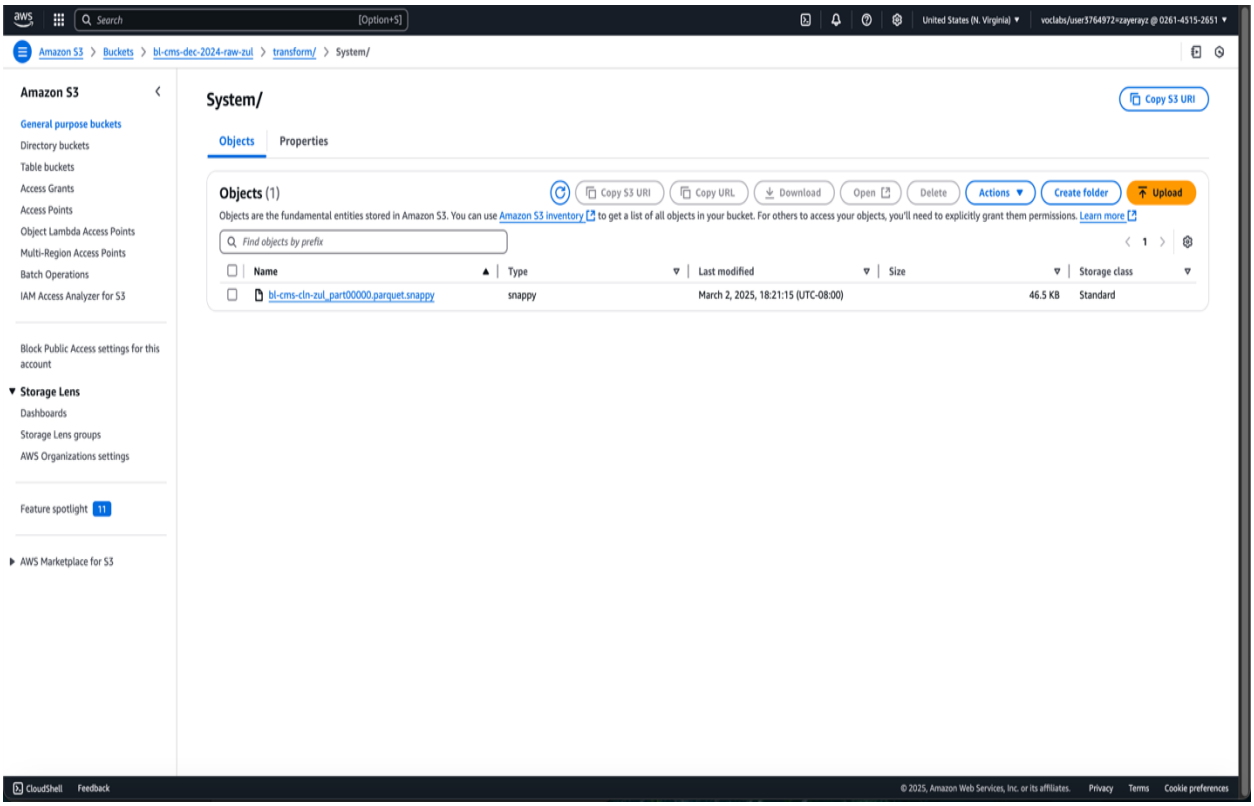
Recipe Job Successful



Note. Screenshot indicating the successful execution of the cleaning recipe job in DataBrew. Source: AWS, *self-work*

Figure 50

Cleaned Data Output - System Folder

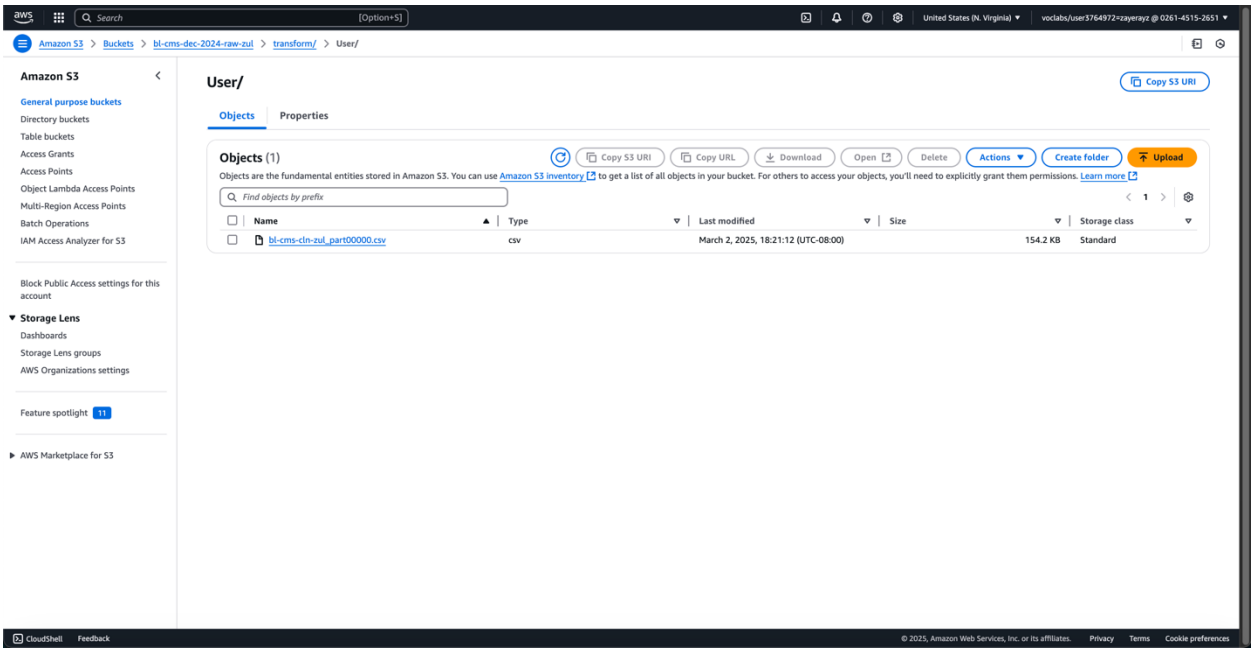


Note. Screenshot from AWS S3 showing the cleaned dataset stored in the system folder for technical verification. Source: AWS, *self-work*

This screenshot displays the cleaned dataset as it appears in the system folder. This output is used for technical verification; you can see that only the essential columns remain and that the formatting is consistent.

Figure 51

Cleaned Data Output - User Folder



Note. Screenshot from AWS S3 showing the final cleaned CSV stored in the user folder, ready for analysis. Source: AWS, *self-work*

Here, the final cleaned CSV is stored in the user folder. This version is what I will use for analysis, ensuring that all extraneous data has been removed, and the dataset is ready for the next phase.

Step 4

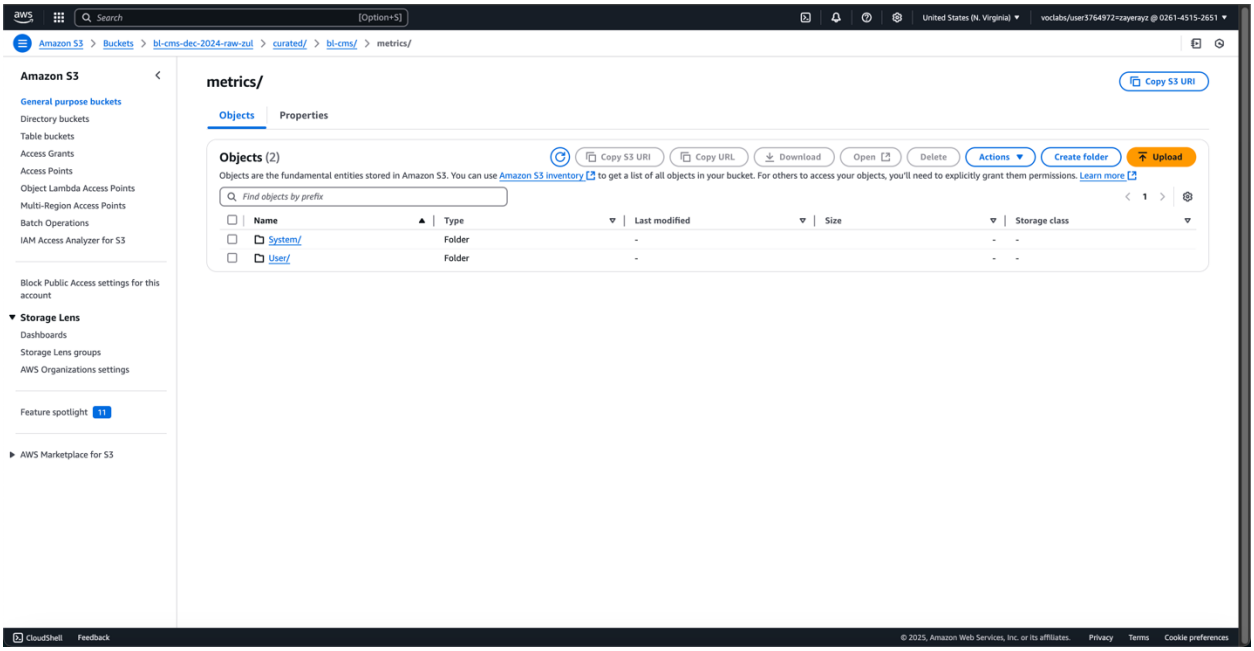
Data Cataloging

After cleaning, I used AWS Glue Crawler to catalog the dataset. I configured the crawler to scan the cleaned data stored in the designated folder (it could be the “user” folder or a dedicated cataloging folder, depending on your setup). The crawler automatically created a database in AWS Glue, which organizes the data with a well-defined schema. This step is

essential for enabling efficient querying in AWS Athena and ensuring that the dataset is accessible for further ETL processing.

Figure 52

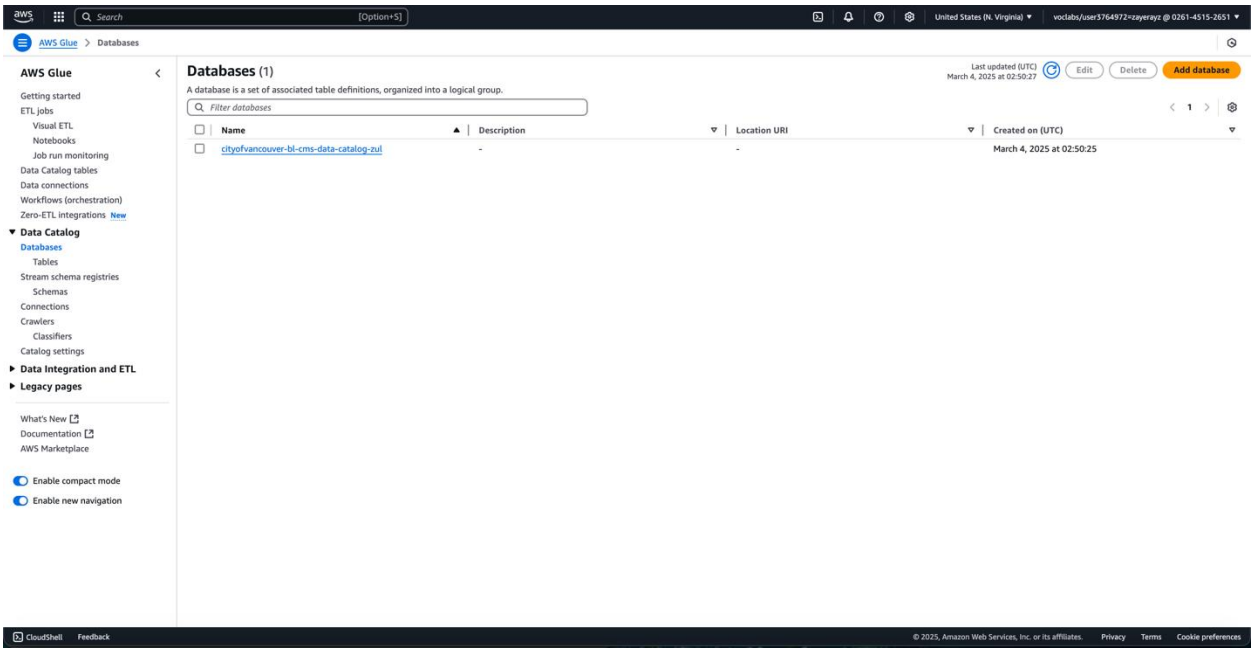
Metrics Folder



Note. Screenshot displaying the metrics folder created for further processing and cataloging. Source: AWS, *self-work*

Figure 53

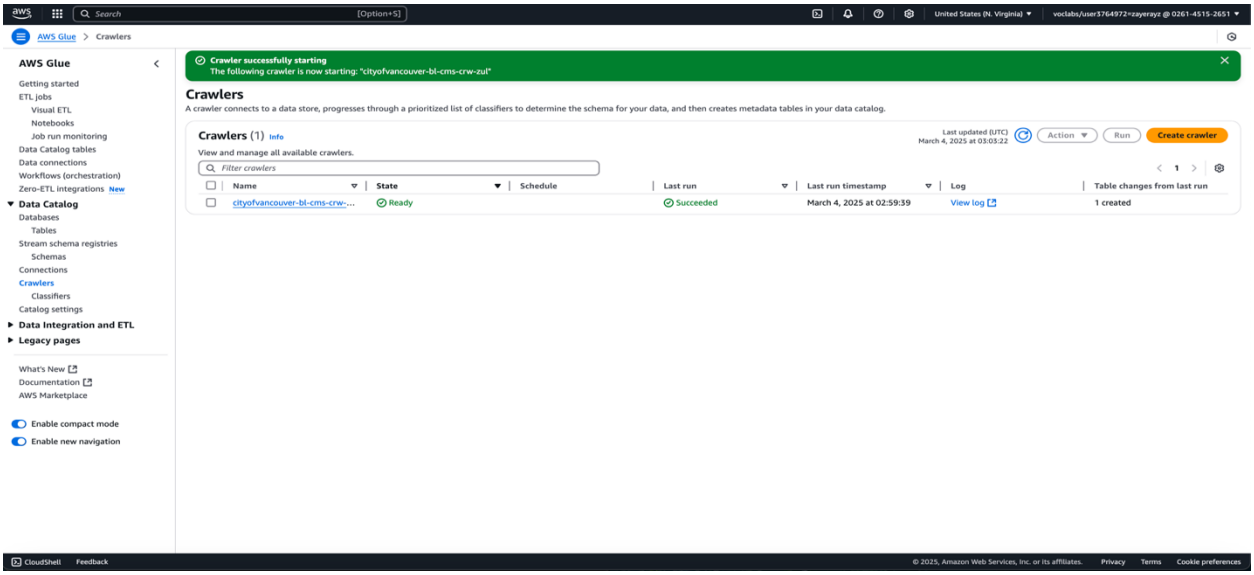
AWS Glue Database – Cataloging the Cleaned Data



Note. Screenshot showing the database created by AWS Glue Crawler that catalogs the cleaned data from the user folder. Source: AWS, *self-work*

Figure 54

AWS Glue Crawler

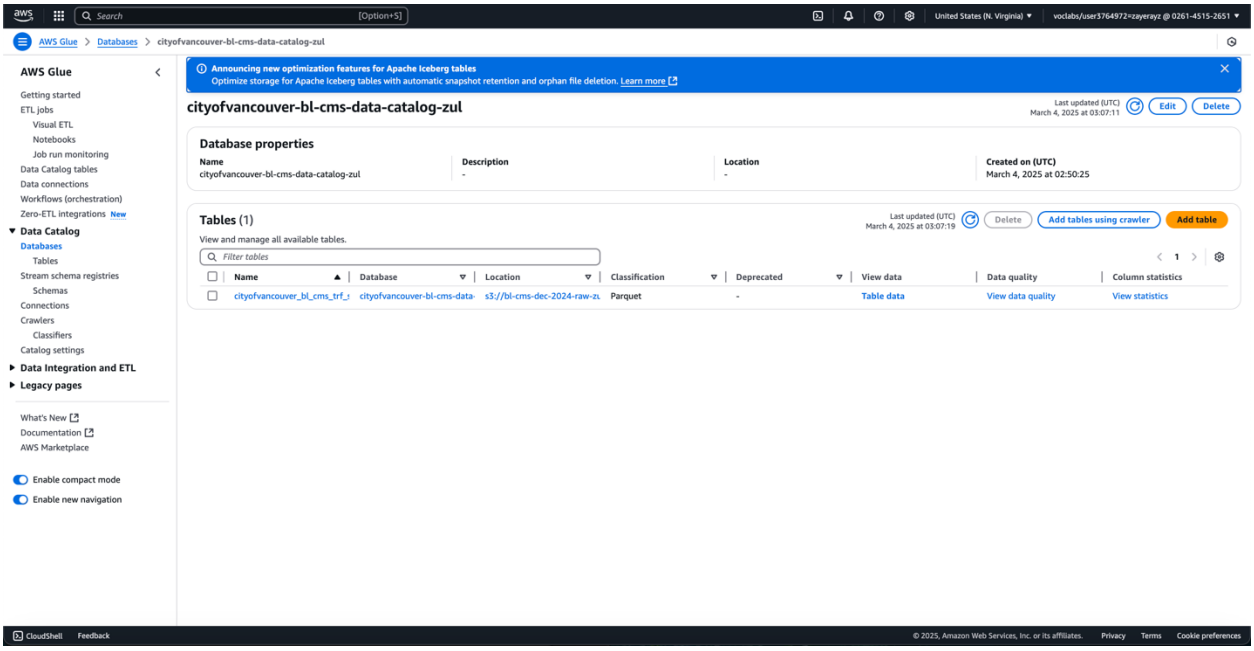


Note. Screenshot capturing the Glue Crawler scanning the cleaned data to build the database schema. Source: AWS, *self-work*

After cleaning, I ran an AWS Glue Crawler on the cleaned dataset stored in the user folder. This screenshot shows the crawler in action as it scans the data and automatically creates a database with a well-defined schema.

Figure 55

Cataloged Database and Table Schema in AWS Glue



Note. Screenshot of the final database and table schema generated by the crawler, confirming organized data for querying. Source: AWS, *self-work*

This image displays the resulting database and table schema from the crawler. It confirms that the data is organized and accessible for querying.

Step 5

ETL Pipeline and Data Summarization

For the transformation and summarization phase, I built a single, continuous ETL pipeline in AWS Glue Studio. Unlike some projects that employ two separate pipelines for multiple metrics, my analysis was accomplished with one streamlined flow. The pipeline starts with the “Prepare for Load” node, which connects directly to an output node configured to write the transformed data into the “transform” folder of my S3 bucket.

Within this pipeline, I applied several key transformation steps:

• **Filtering:** I filtered out rows that did not meet the specific criteria relevant to my analysis.

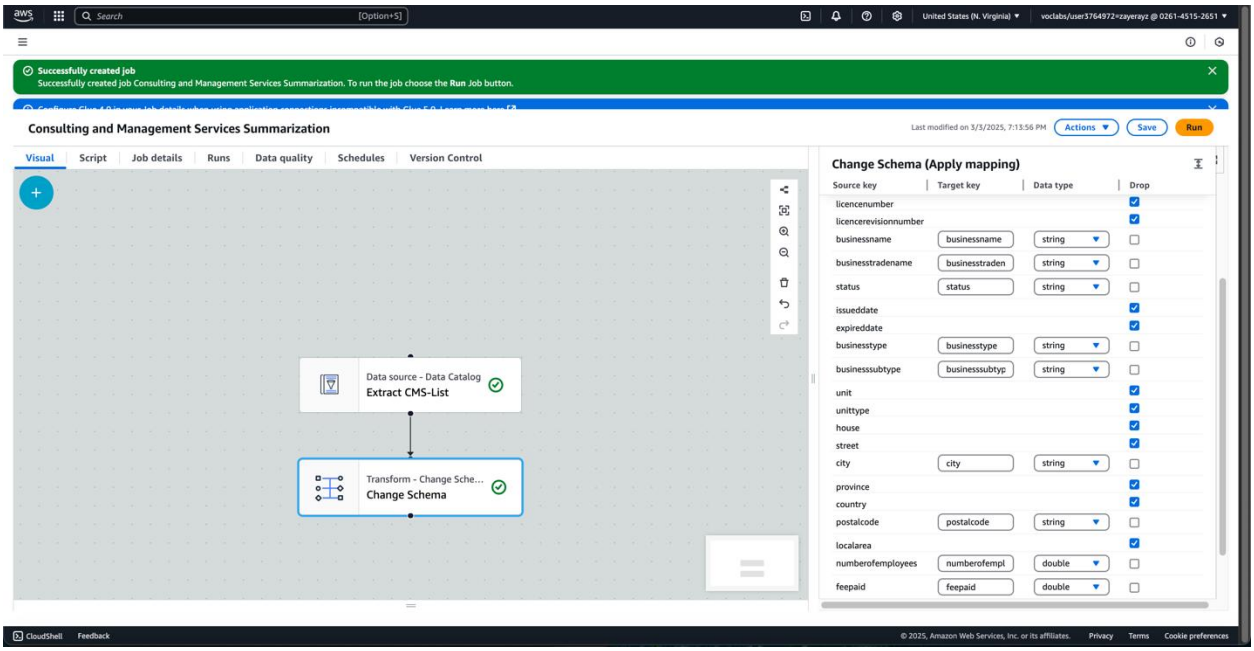
• **Aggregation:** I performed aggregation operations to compute my primary metric (for instance, the total number of licenses issued or the average processing time).

• **Schema Changes:** I re-mapped column names and dropped fields that were unnecessary, ensuring that the final output contained only the relevant information.

The final output of the ETL pipeline is then written to designated folders. In my case, the output is available in both the “system” folder (for technical verification) and the “user” folder (as the final, summarized dataset). Additionally, I ensured that the final data appears in the “transform” folder if that’s where intermediate or final transformation files are meant to be stored according to the project structure.

Figure 56

Schema Changes and Final Mapping

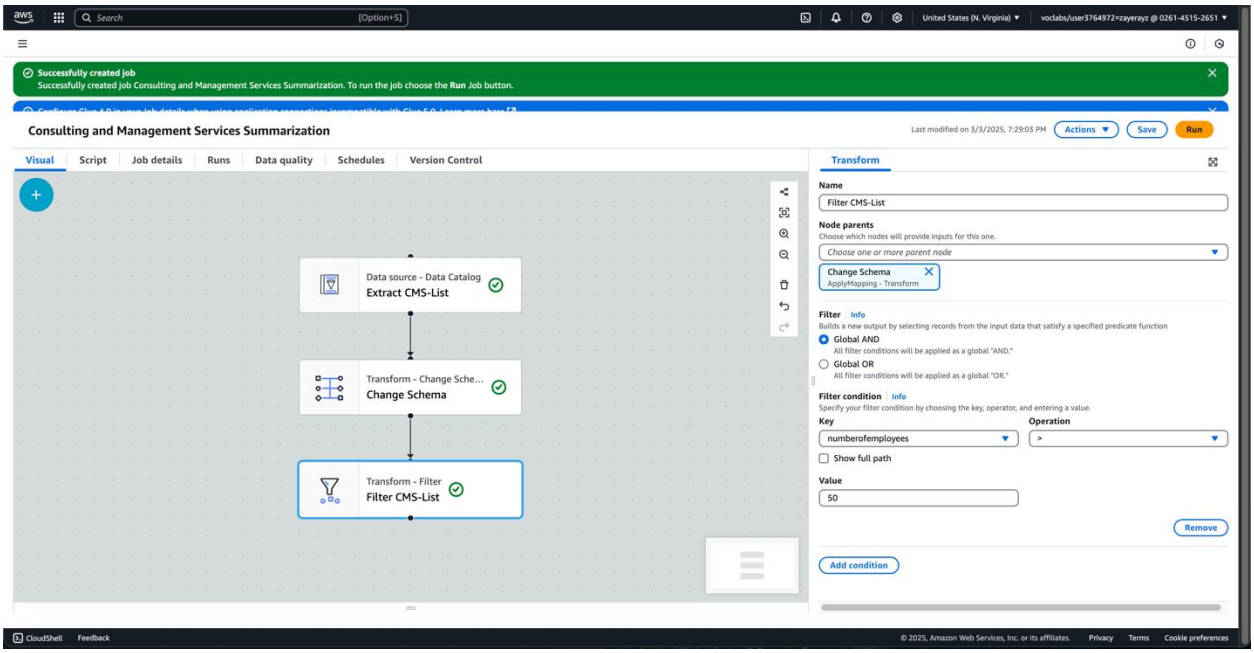


Note. Screenshot of the final database and table schema generated by the crawler, confirming organized data for querying. Source: AWS, *self-work*

This screenshot focuses on the schema change step, where I re-mapped column names and dropped unnecessary fields. These adjustments ensure that the final output includes only relevant information and is easy to interpret.

Figure 57

Detailed Transformation - Filtering and Aggregation

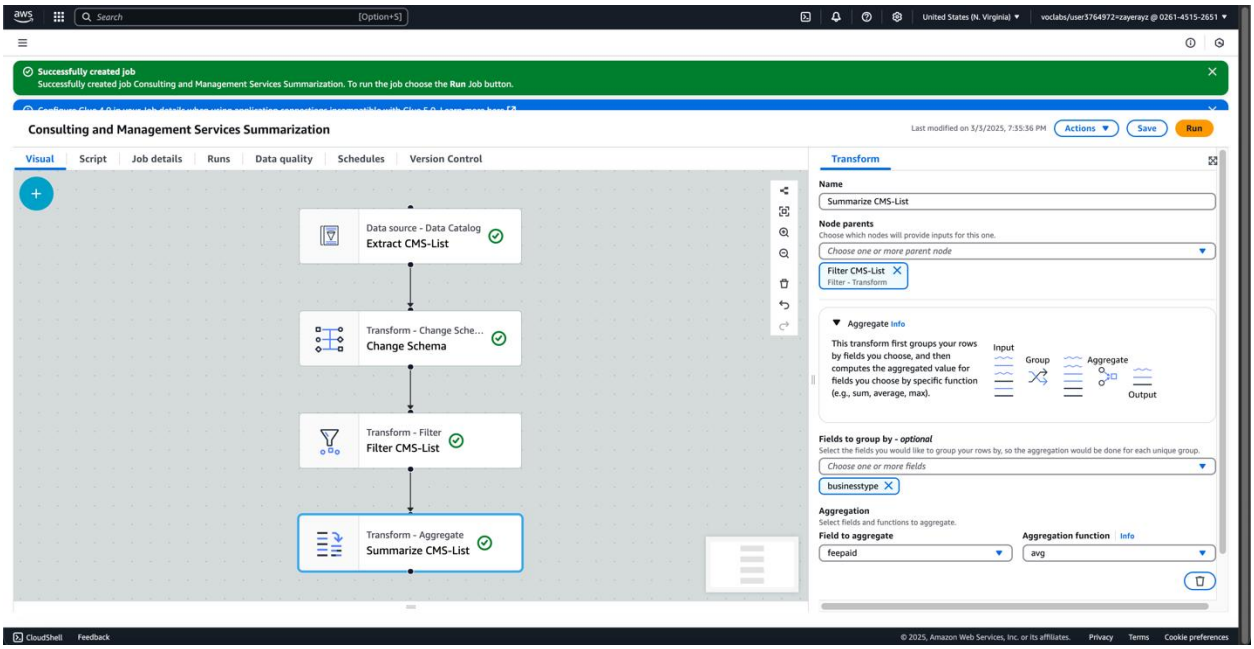


Note. Screenshot highlighting the filtering and aggregation steps used to compute the primary metrics in the ETL process. Source: AWS, *self-work*

In this image, I highlight the transformation steps within the pipeline. I applied filtering to remove rows that do not meet the required criteria and used aggregation functions to calculate my primary metric (such as the number of employees > 50).

Figure 58

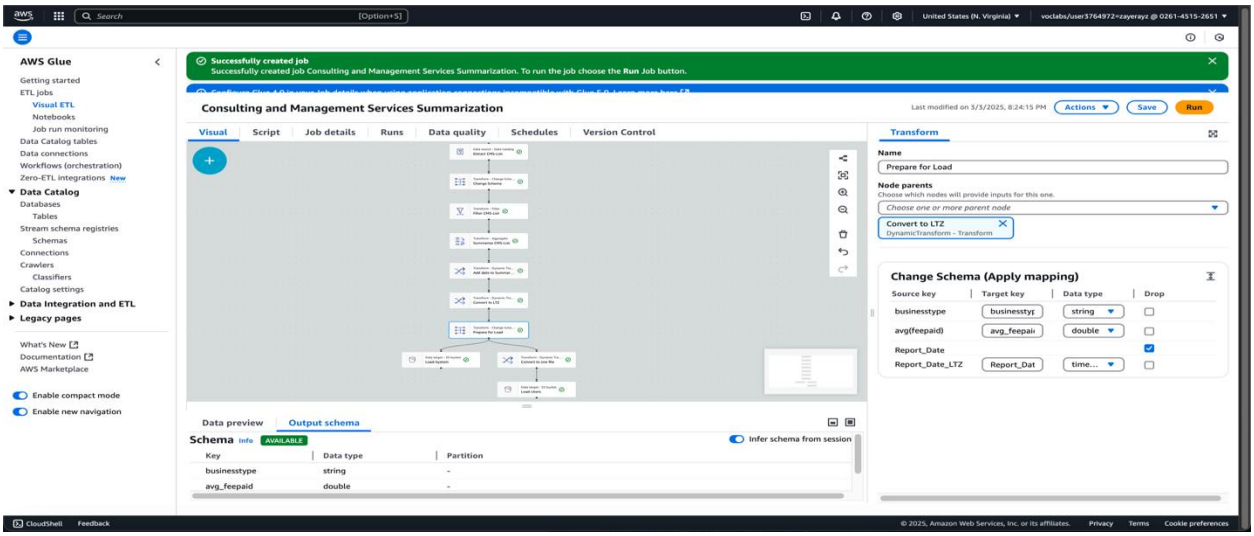
Aggregate



Note. Screenshot showing the aggregated data output after the transformation step in the ETL pipeline. Source: AWS, *self-work*

Figure 59

Final Summarized Data – System Folder and User Folder

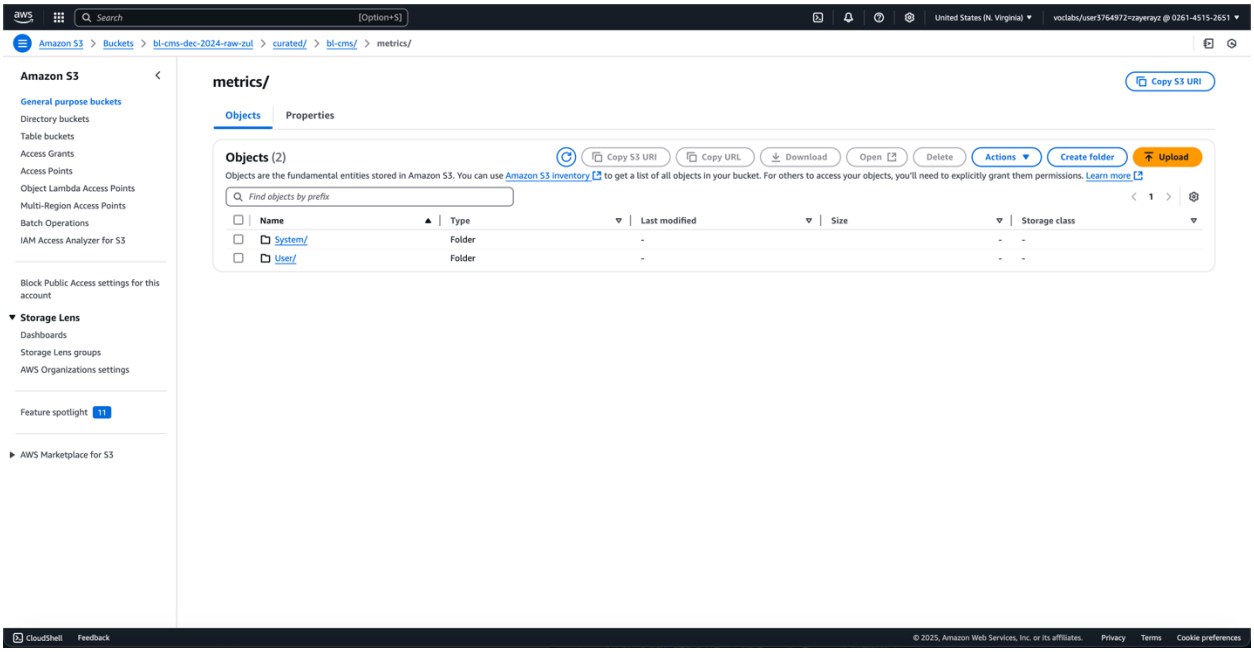


Note. Screenshot displaying the final summarized dataset stored in both the system and user folders, confirming successful ETL processing. Source: AWS, *self-work*

This screenshot displays the final output stored in the system folder and user folder. It confirms that the pipeline has successfully produced the transformed and aggregated dataset, which is now ready for analysis.

Figure 60

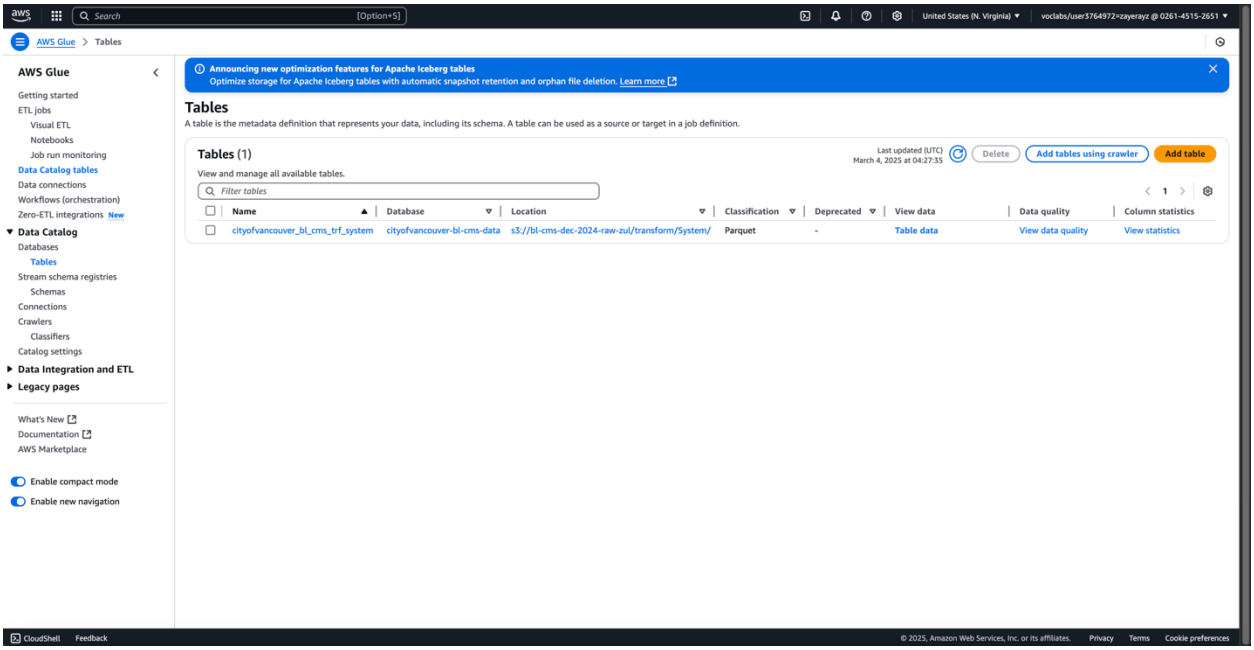
Curated System and Users



Note. Screenshot illustrating the organized folder structure in the curated S3 bucket where the final outputs are stored. Source: AWS, *self-work*

Figure 61

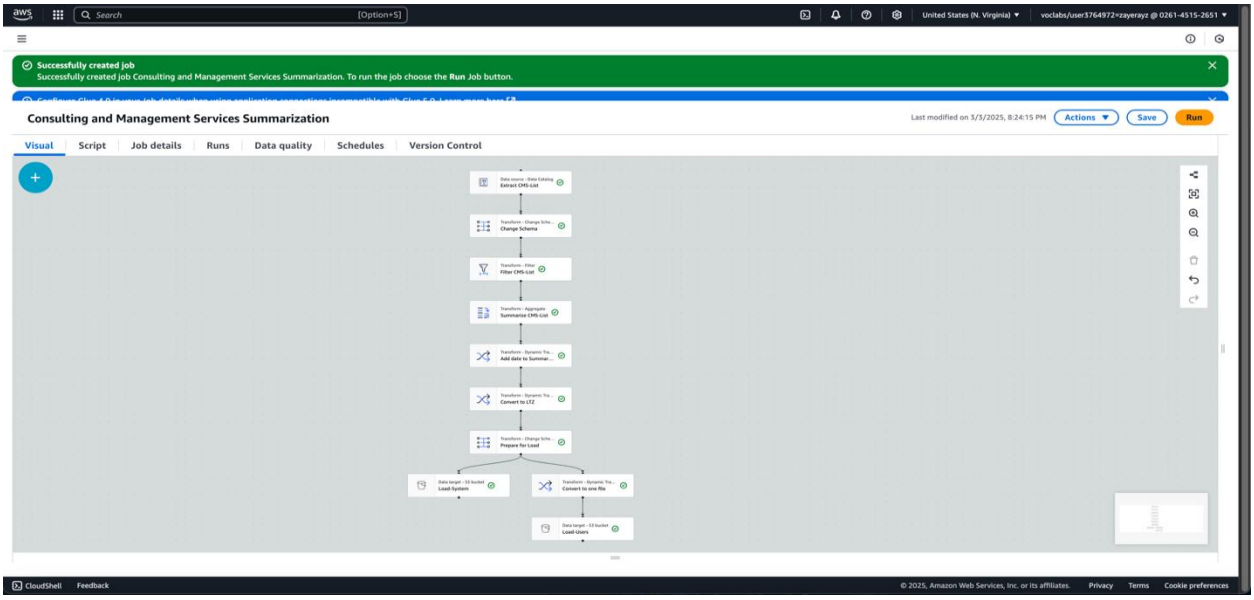
Tables



Note. Screenshot showing the final data tables generated from the ETL process, ready for analysis. Source: AWS, *self-work*

Figure 62

Overview of the ETL Pipeline in AWS Glue

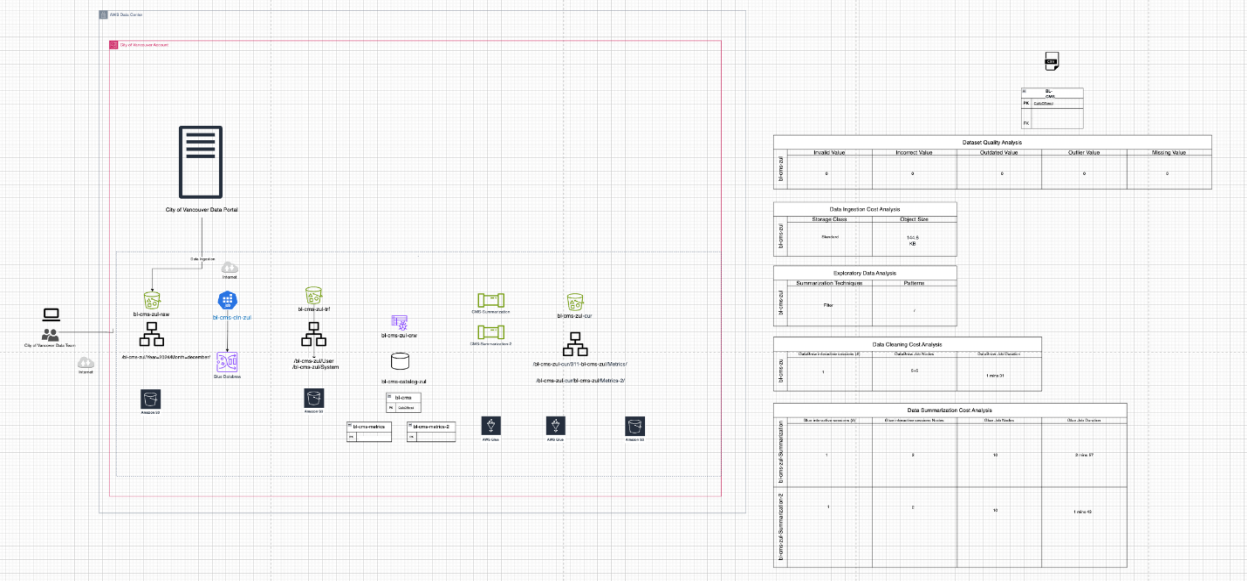


Note. Screenshot depicting the overall ETL pipeline flow from raw data ingestion to final output. Source: AWS, *self-work*

This screenshot shows my single, continuous ETL pipeline in Glue. The pipeline starts at the “Extract CMS-List” node and flows into an output node configured. This visual confirms that the pipeline is set up as a continuous flow.

Figure 63

Data Pipeline Architecture Diagram



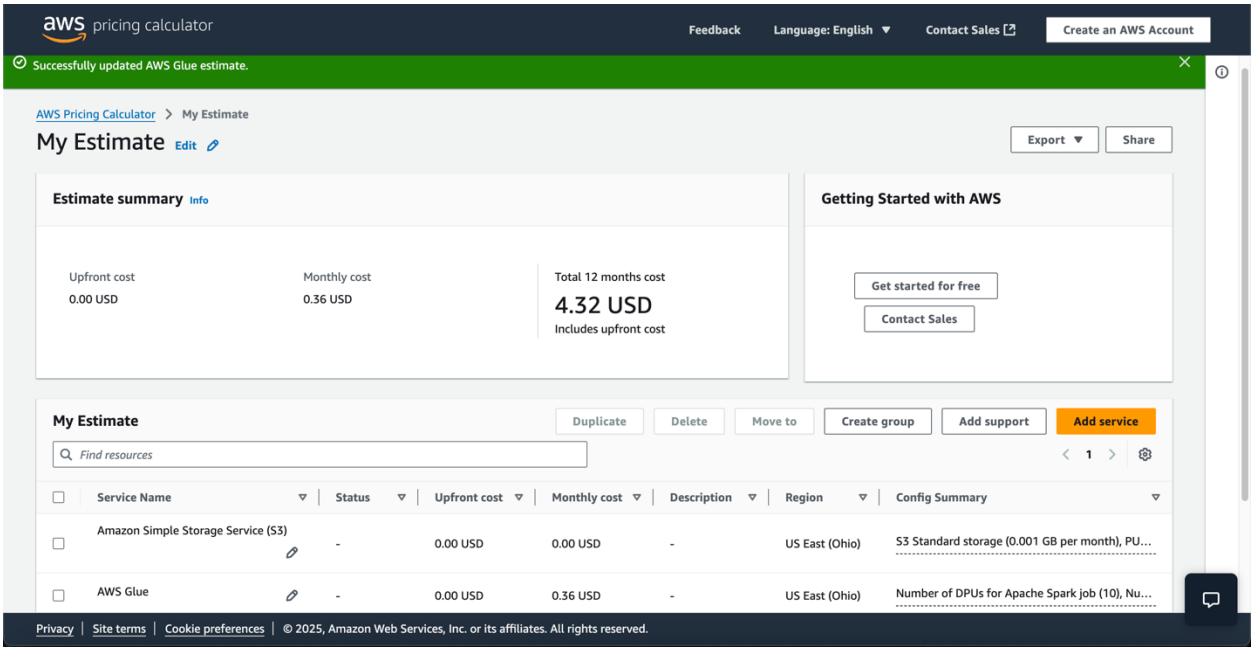
Note. Diagram created using draw.io that visually summarizes the entire data analytic process, from S3 ingestion and DataBrew cleaning to Glue cataloging and ETL transformation. (Source: *self-work*, created on draw.io)

This diagram provides a comprehensive view of the entire process I implemented. On the left, you can see how the raw CSV file is stored in Amazon S3 (bucket “bl-cms-dec-2024-zul”). Moving right, the dataset is profiled and cleaned with AWS Glue DataBrew, where I corrected the delimiter to semicolon, renamed columns, and removed unnecessary fields. Next, the AWS

Glue Crawler automatically creates a database and table schema. Finally, a single ETL pipeline in AWS Glue Studio applies transformations (filtering, aggregation, schema changes) and writes the results to the “transform,” “system,” and “user” folders in S3. This end-to-end flow ensures that the City of Vancouver’s Business Licenses data is properly ingested, cleansed, cataloged, and summarized for further analysis.

Figure 65

AWS Pricing Calculator - Final Cost Estimate



Note. Screenshot of the AWS Pricing Calculator showing an estimated annual cost of $4.32, which includes S3 storage and AWS Glue usage. Source: AWS Cost calculator, *self-work*

In this screenshot, I ran my project on AWS Pricing Calculator for the Business Licenses, Consulting and Management Services dataset on AWS, and I ended up with a total of about $4.32 for 12 months. That cost includes my S3 storage (which is tiny, since the dataset is only a few hundred KB) and my AWS Glue jobs (for data cleaning, cataloging, and the ETL pipeline).

Because my data is so small and my ETL job runs for just 2 minutes, the monthly bill is super low. I didn't add any cost for Athena, because I either haven't run queries there or only ran a few small ones, which would be negligible anyway. So, if I only used S3 and Glue, $4.32 a year totally makes sense.

Basically, I'm paying a few cents a month to store the raw and cleaned data in S3, and the rest of the cost comes from Glue usage during my short ETL runs. If I ever scale up the data or run more frequent jobs, this cost will increase but for now, it's perfect for a small project. This also shows that AWS can be pretty cheap when your dataset and usage are minimal.

DAP Design and Implementation (Individual work – Haoran Xu)

Descriptive Analysis

I selected the dataset called "Public Trees", also to make sure my data volume is neither too large nor small. I filter the dataset on the data portal by "Species Name", and chose "Acutissima" as my target, it contains around 500 rows.

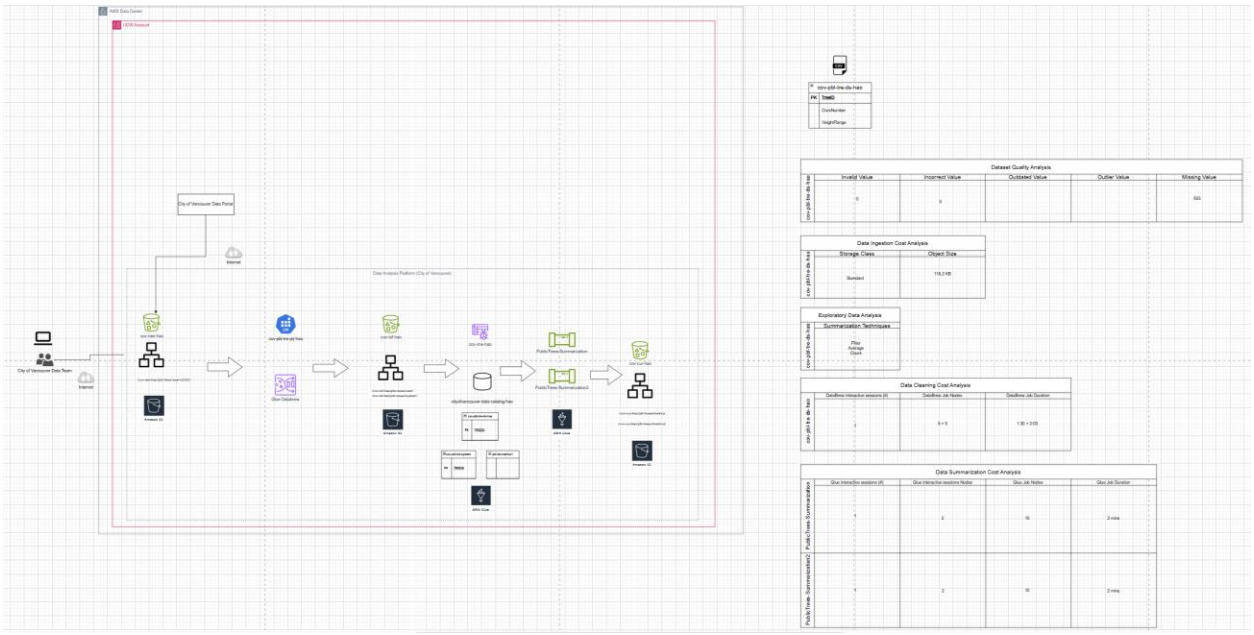
The dataset contains columns like TreeID, CivicNumber, STD Street, etc. My descriptive analysis questions are in the filter with Diameter greater than or equal to 10, is there a direct relationship between diameter and height range? And under the filter with Diameter greater than or equal to 10, and in a certain number of samples, what is the proportion of each different height range?

To answer this descriptive analysis question, I filtered and removed some irrelevant columns (which would be reflected in my ETL pipeline). The remaining columns are Diameters (will be AvgDiameter later), HeightrangeID and Heightrange. The screenshot below shows the entire process of my plan on drowio before implementation.

The first step is data ingestion, which means I download the corresponding dataset from the data portal, create a raw bucket in the S3 function and upload the dataset. The second step is data profiling. In this step, we use AWS Glue Databrew. I created a profile for my dataset. This allows me to easily browse if there are any data quality issues in my table, and I can also correct the corresponding problems. And we also need to create a bucket in S3 for data transformation. To store the clean data after cleaning, we put it in the system and user folders respectively. The third step is data cleaning. We also perform this step in Glue Databrew. We can browse the schema of our dataset to check if they have format problems or change the column name to make them clearer. Cleaning is a very important step because without clean data we cannot move on to the catalog. The fourth step is data cataloging. I first created a curated data bucket using the S3 function. Then I will create our database in AWS Glue and use crawler to get our data and store it in it ready for summarization. The last step is data summarization, which is also performed in AWS Glue. This is a crucial step to get the "descriptive analysis" we use to answer the descriptive analysis question. We will use ETL to create a series of pipelines, including uploading data, changing the schema format, filtering our data if necessary, adding our premises and assumptions, and attaching the time in the local timezone, and finally uploading the results to the curated data bucket we created in advance.

Figure 64

Drowio

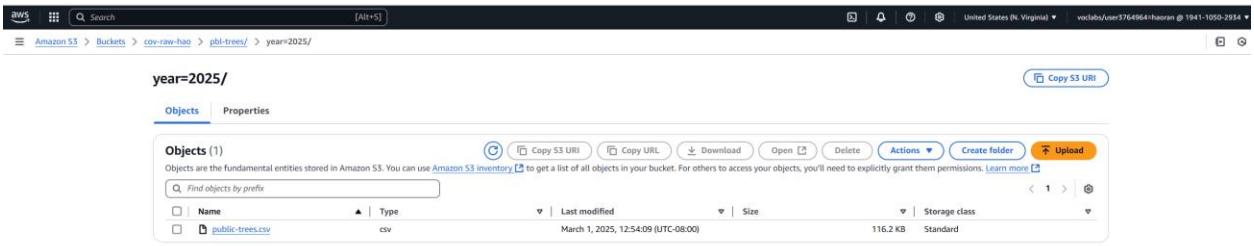


Note. This screenshot shows the drowio design part including the cost estimation at the right-side. Source: Made by author using Drow.io

Step 1: Data Ingestion

Figure 65

Raw Bucket



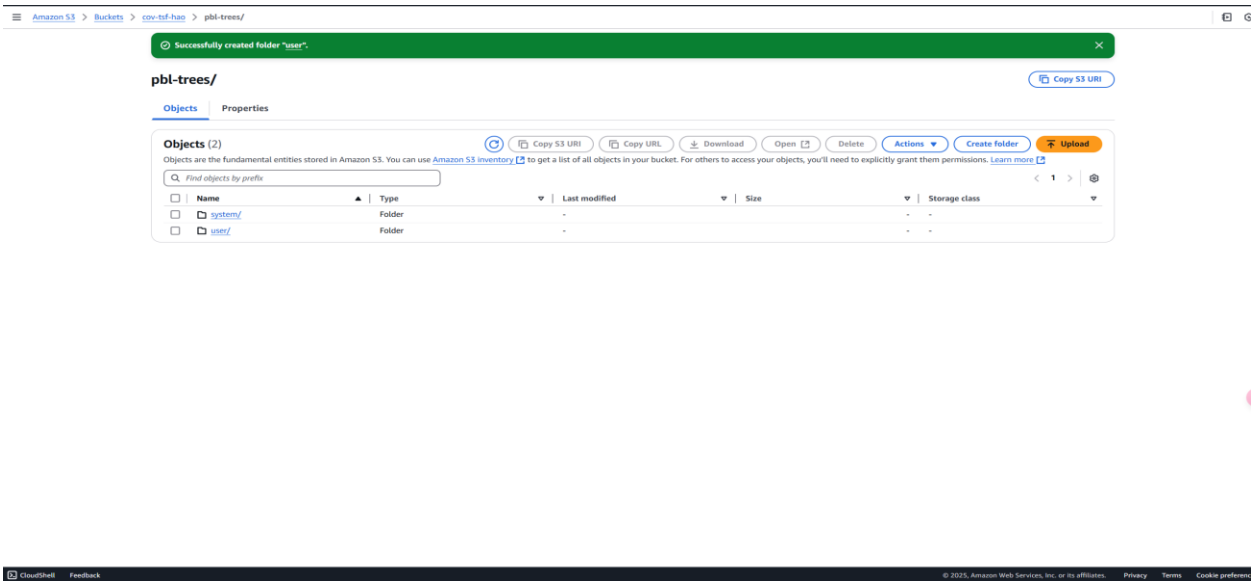
Note. This screenshot shows the raw bucket in AWS S3. Source: AWS S3

After downloading the datasets from City of Vancouver data portal, I created a bucket called cov-raw-hao as my raw data bucket. It is for storing my raw dataset. I create sub-folder inside it until year=2025 and then upload my downloaded dataset there as my raw dataset.

Step 2: Data Profiling

Figure 66

Transfer Bucket

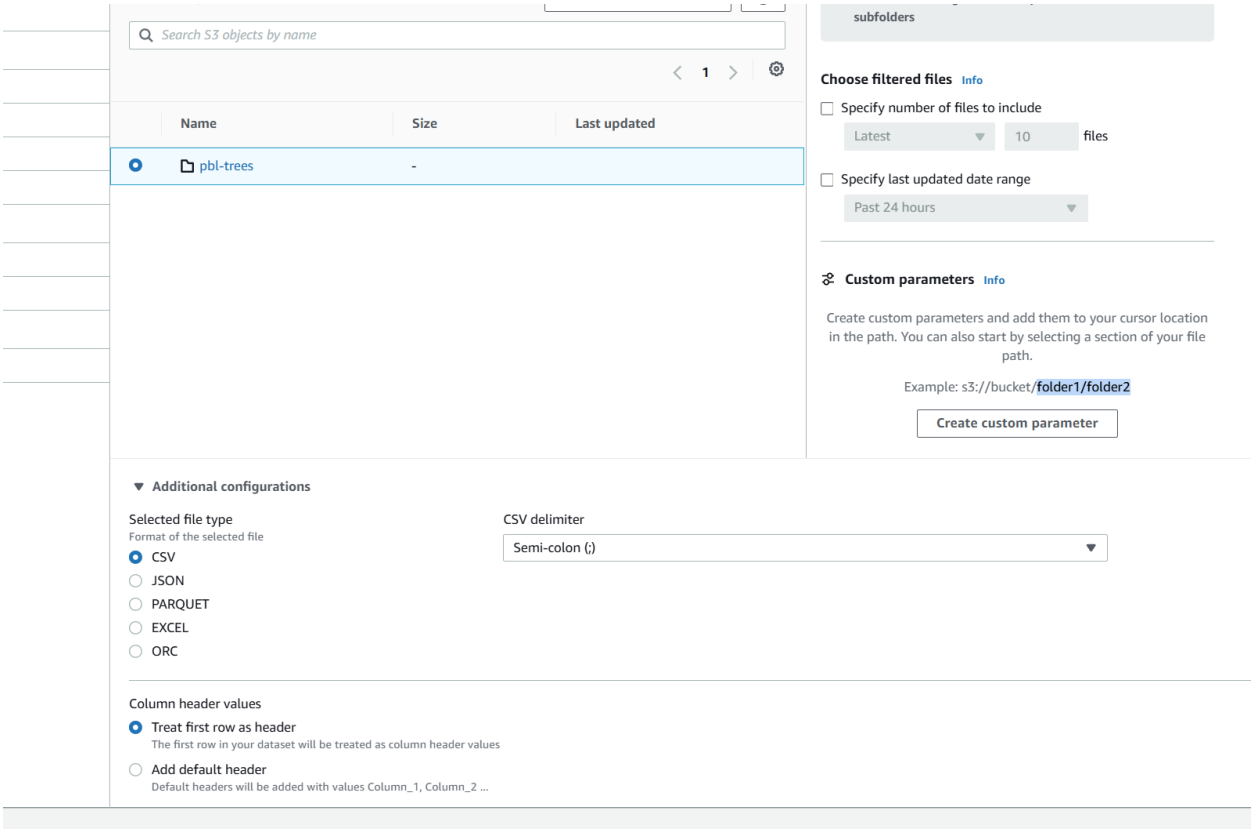


Note. This screenshot shows the transfer bucket in S3. Source: AWS S3

After Data ingestion, it's time for data profiling. Firstly, I choose to create another bucket called cov-tsf-hao which is for transfer bucket. And I create the sub-folder inside it until system and user folder. Currently, we have two different sub-folders for different uses.

Figure 67

Process of Profiling

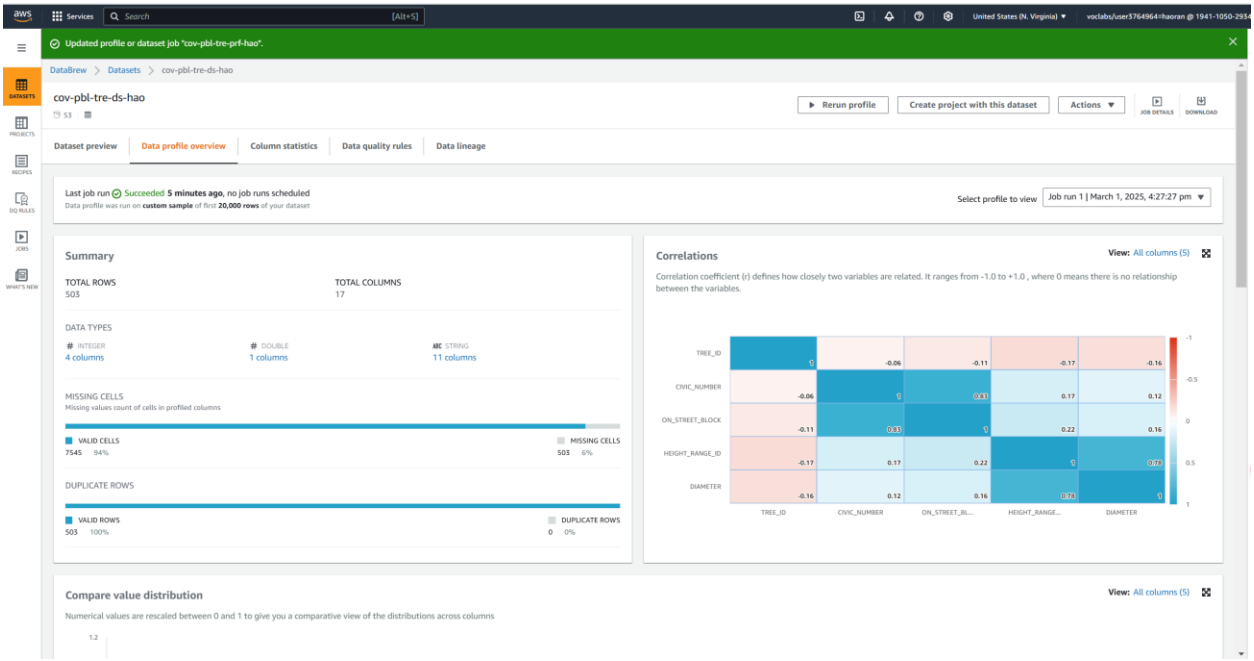


Note. This screenshot shows the process of creating data profiling. Source: AWS Glue Databrew

Here's when I'm creating the project, I failed last time because I don't change the CSV delimiter to Semi-colon (it's described on the data portal that the delimiter for csv file is semi-colon), I change the correct delimiter this time.

Figure 68

Data Profiling Completed



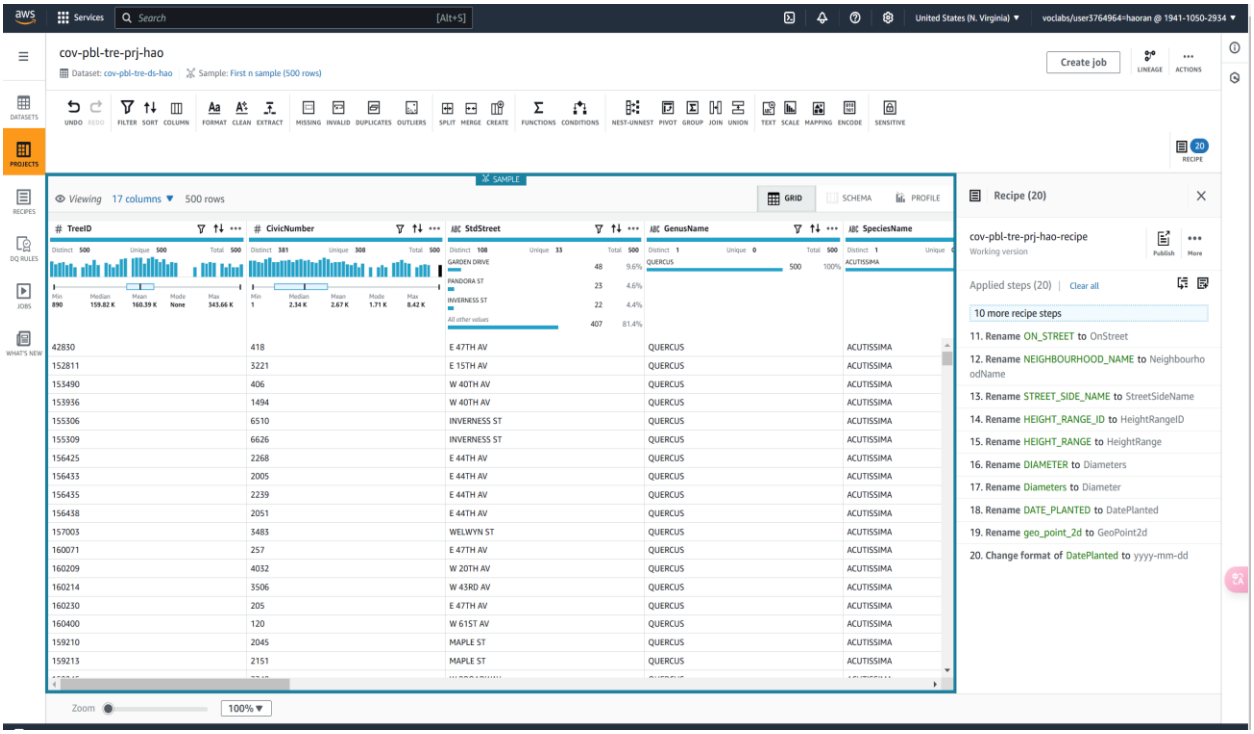
Note. This screenshot shows the profiling result in AWS Glue Databrew. Source: AWS Glue Databrew

Here's the profile screenshot after I create and run the job. And now it's time for data cleaning

Step 3: Data Cleaning

Figure 69

Data Cleaning Process



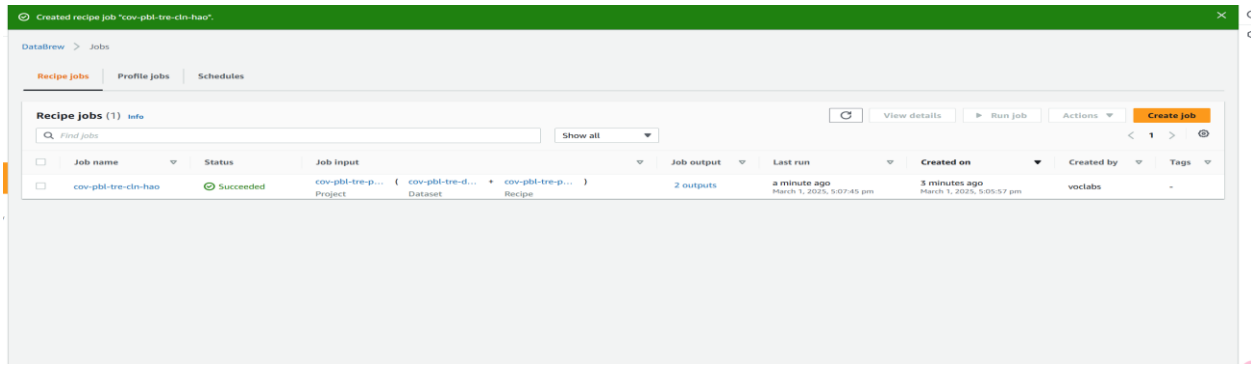
Note. This screenshot shows the Data cleaning process in AWS Glue Databrew. Source: AWS Glue Databrew

For the data cleaning, step 1 I changed all the column title to a clearer format to watch. I also changed the format of time-date format to yyyy-mm-dd.

All the formats are clean and structured here and I'm planning to create and run the cleaning job.

Figure 70

Cleaning Job Completed

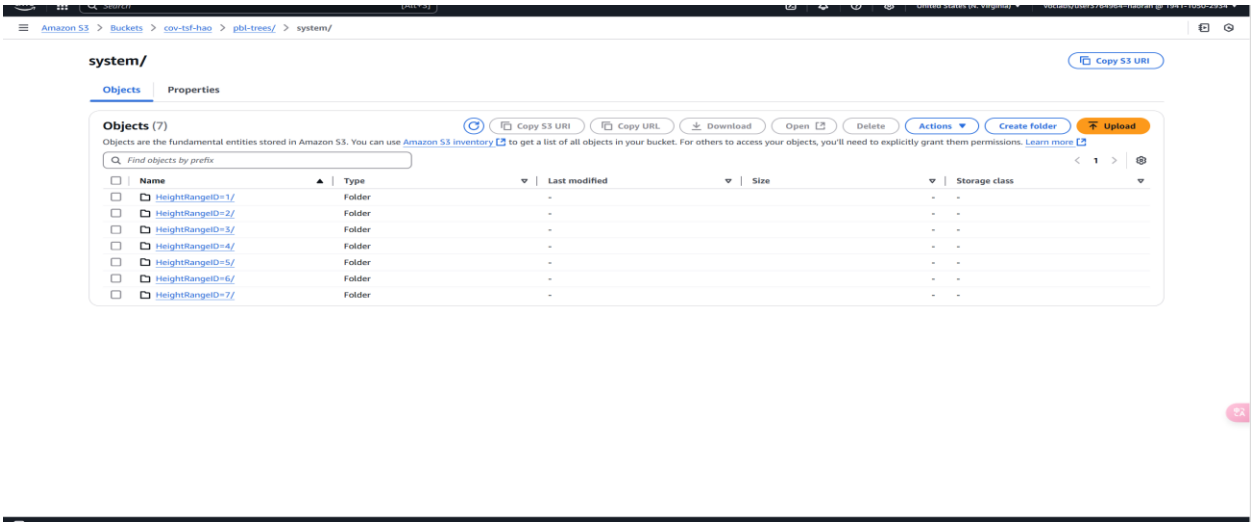


Note. This screenshot shows completion of cleaning job in AWS Glue Databrew. Source: AWS Glue Databrew

Here's the screenshot after I create and run the cleaning job.

Figure 71

System Output of Cleaned Dataset

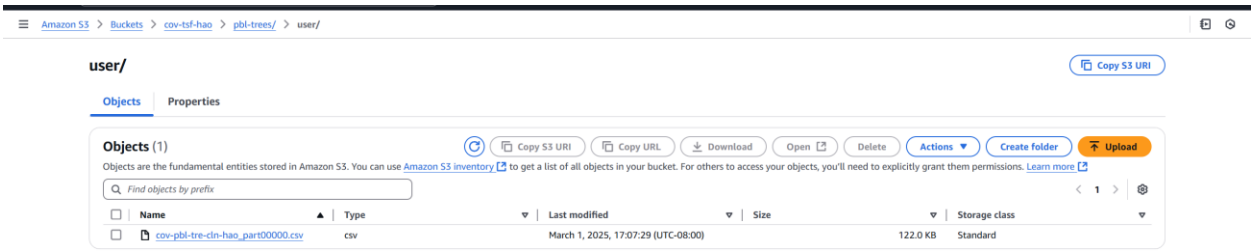


Note. This screenshot shows the cleaned dataset in system folder in AWS S3. Source: AWS S3

Here's the system folder of parq format and I used "Columns to partition by" and choose HeightRangeID as the output.

Figure 72

User output of Cleaned Dataset



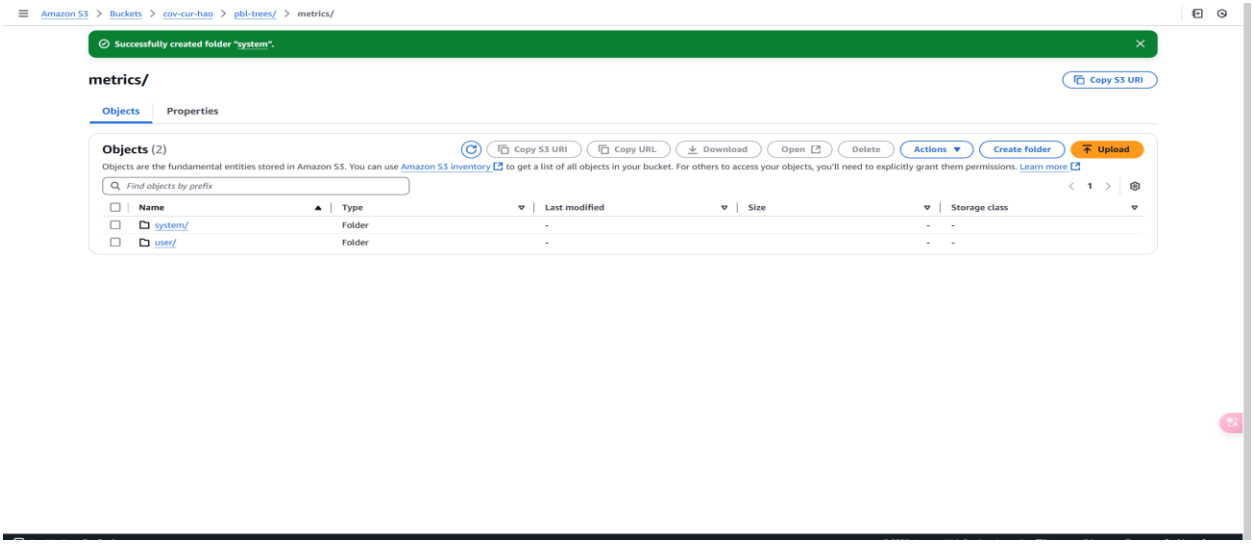
Note. This screenshot shows the cleaned data set-in user-friendly format in AWS S3. Source: AWS S3

Here's the user folder, cleaned csv file is stored here. Now we are going to Data Cataloging for next part.

Step 4: Data Cataloging

Figure 73

Curated Bucket Creation

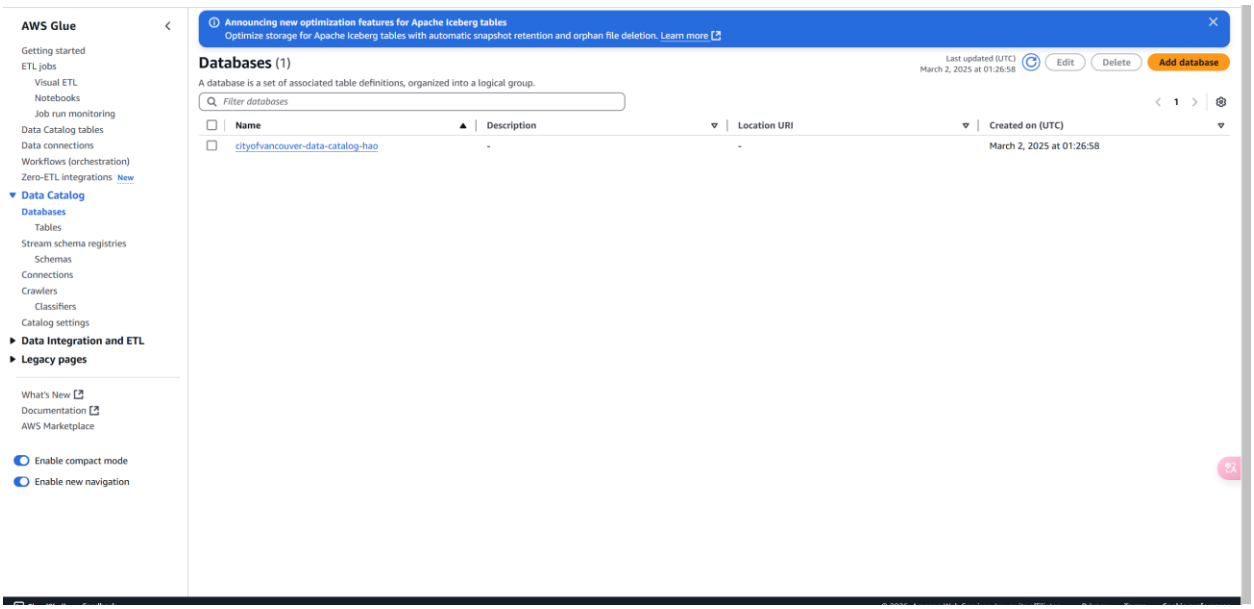


Note. This screenshot shows the curated bucket in AWS S3. Source: AWS S3

Firstly, I create a new bucket called cov-cur-hao and sub-folders for metrics until system and user folders.

Figure 74

Databased Creation

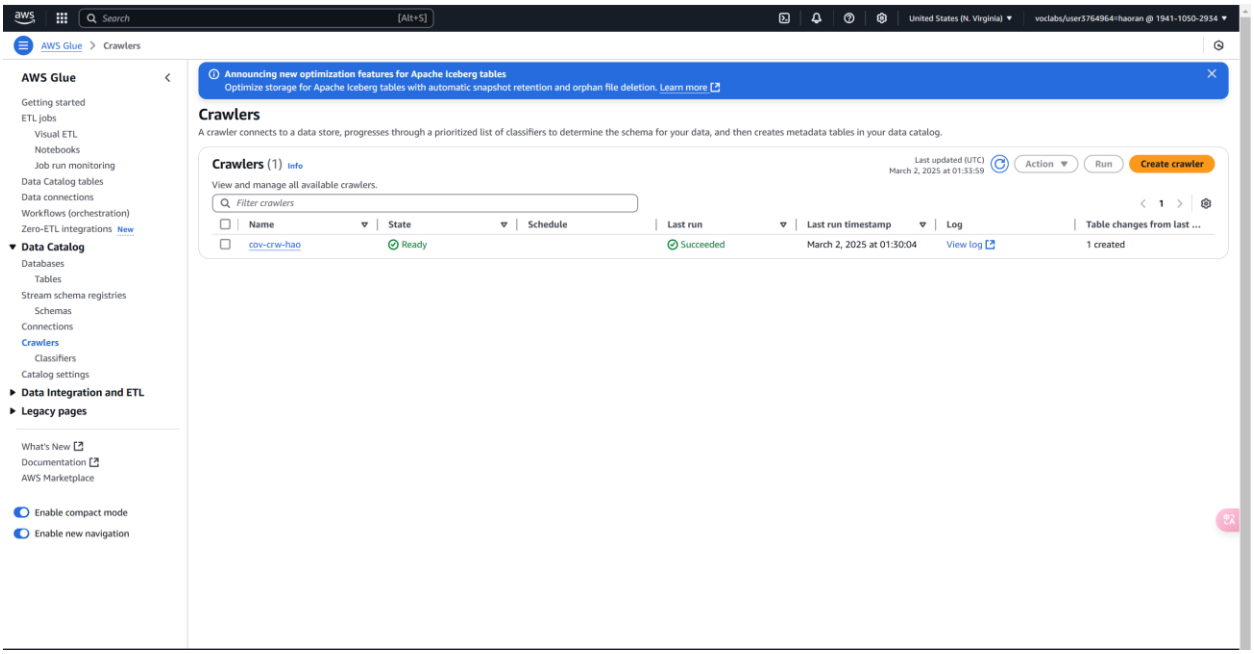


Note. This screenshot shows the database creation in AWS Glue. Source: AWS Glue

I created the database here.

Figure 75

Crawler Creation

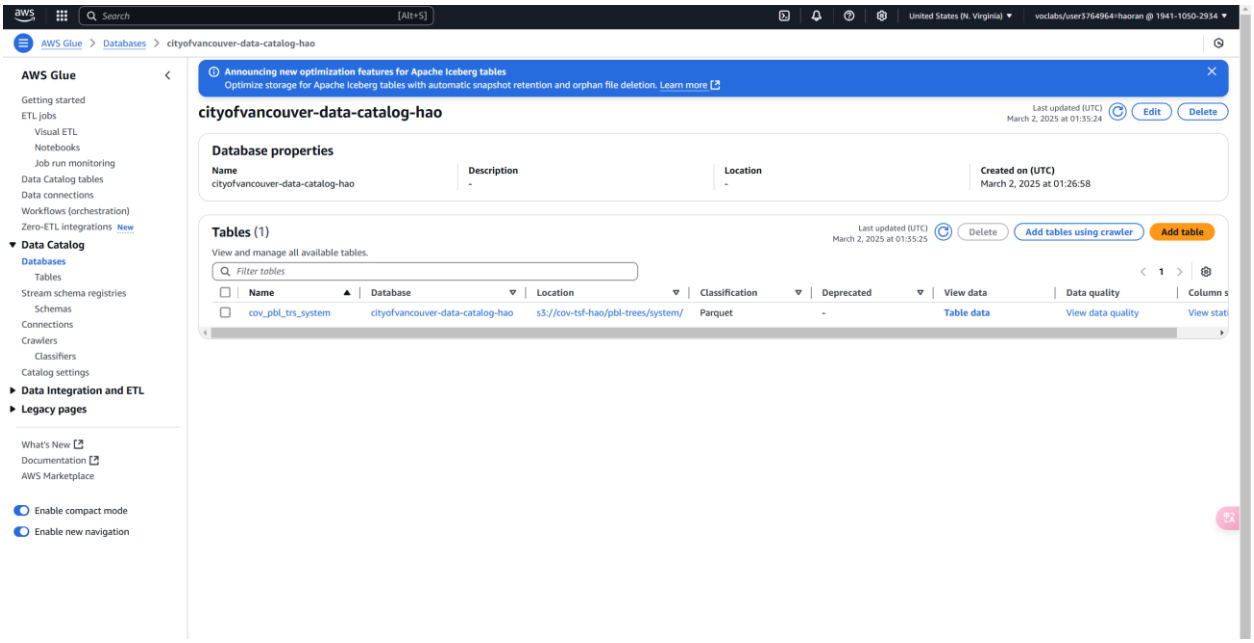


Note. This screenshot shows the successful run of Crawler in AWS Glue. Source: AWS Glue

Here I created a crawler to get the data for me automatically and store them into my database.

Figure 76

Result of Running Crawler



Note. This screenshot shows the output after running crawler in AWS Glue. Source: AWS Glue

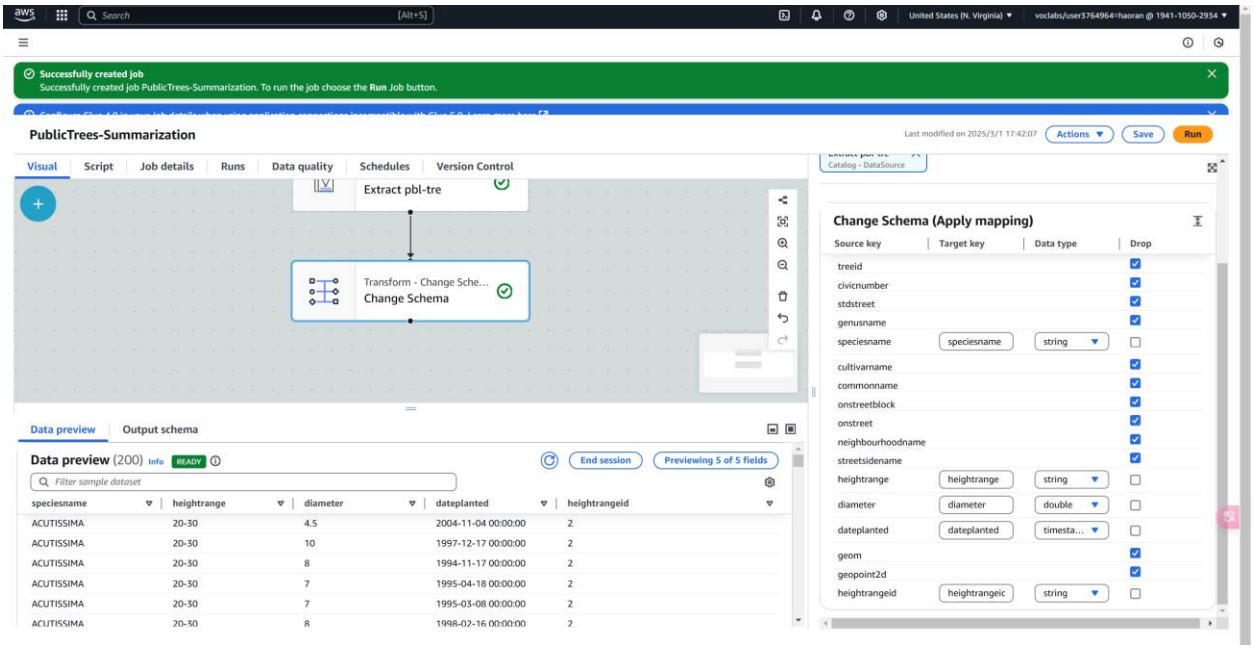
Here's the result of running my crawler. And this is the end of cataloging. Next, we are going to summarization part.

Step 5: Data Summarization

Summarization Metric 1 (ETL pipeline 1)

Figure 77

Change Schema of Summarization 1

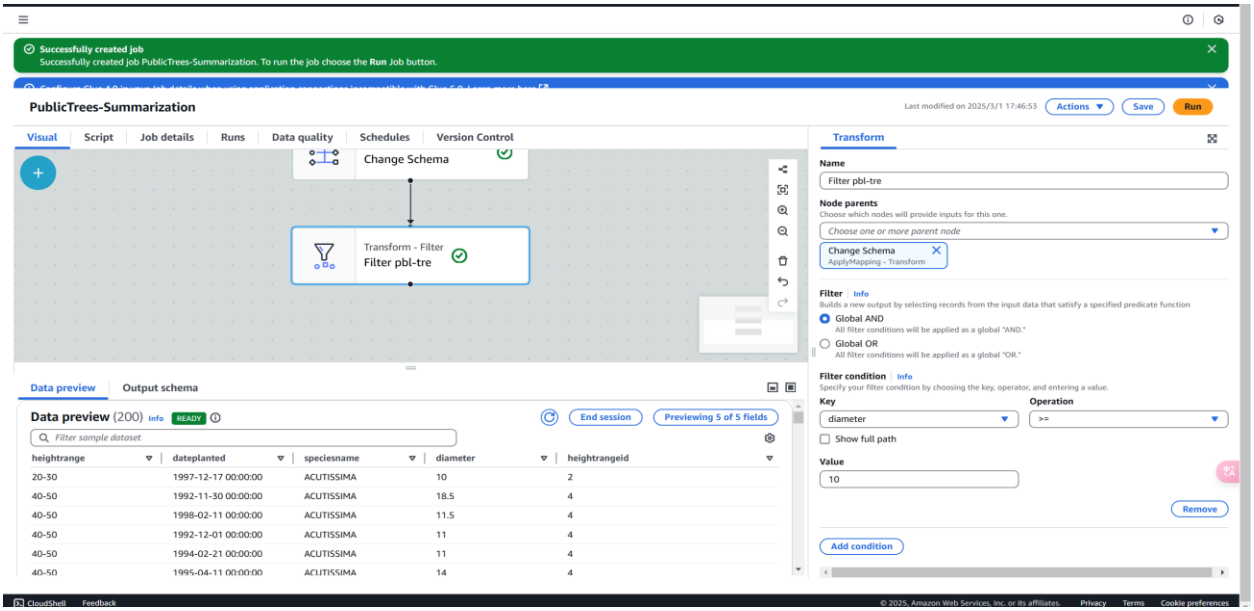


Note. This screenshot shows the change schema step in AWS Glue Visual ETL. Source: AWS Glue

I use “Change Schema” to drop some useless columns which I don’t want it to appear in my summarization. And I change the data type of dateplanted to timetable format. Finally, I only keep 5 columns here.

Figure 78

Filter of Summarization 1

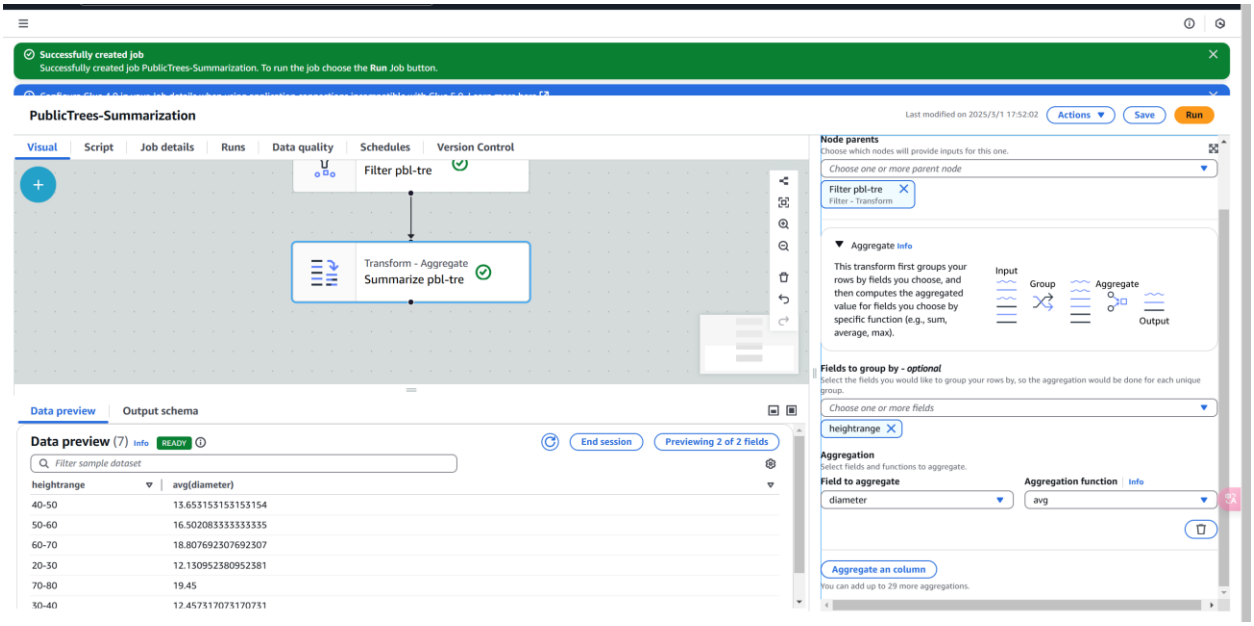


Note. This screenshot shows the filter step in AWS Glue Visual ETL. Source: AWS Glue

I use filter to filter my rows and keep those the diameter is equal or greater than 10. Now I only have 200 rows cut from 500 rows.

Figure 79

Aggregate of Summarization 1

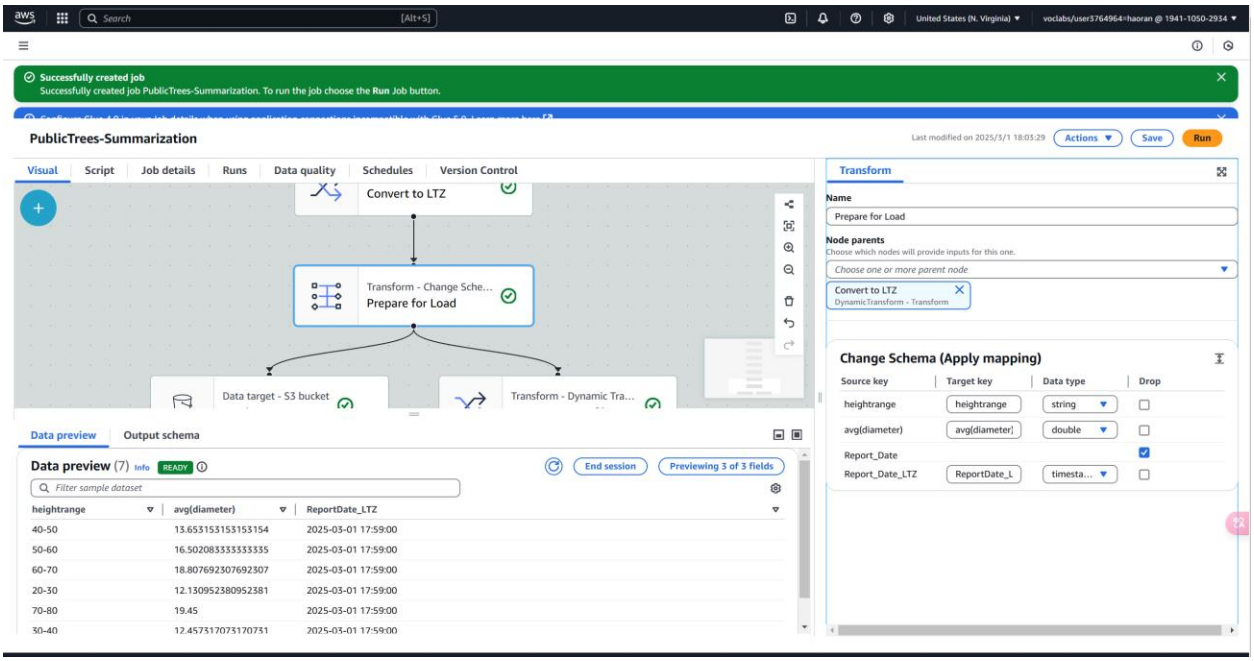


Note. This screenshot shows the aggregate step of summarization 1 in AWS Glue Visual ETL. Source: AWS Glue

Now I use aggregate to do summarization for my first metric. I want to know the average diameter of different heightrange of trees. (And I only count those diameters are equal or greater than 10 from last step).

Figure 80

Prepare for Load of Summarization 1

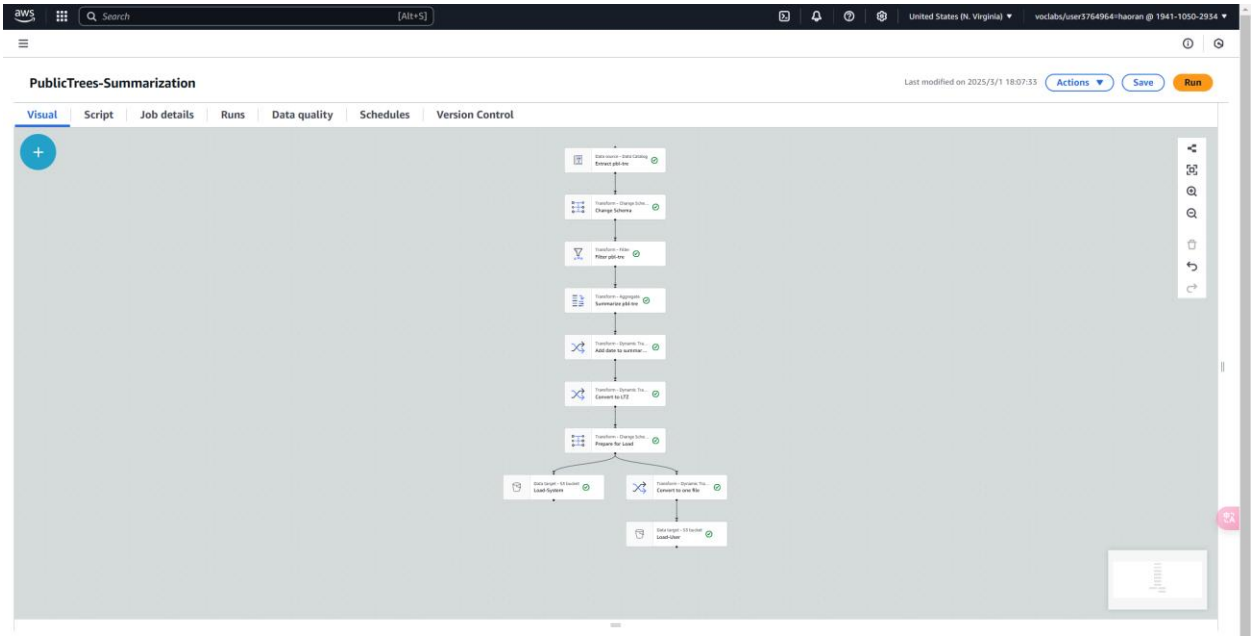


Note. This screenshot shows the prepare for load step in AWS Glue Visual ETL. Source: AWS Glue

I create the add current-timesstamp, convert the timezone to local time zone, and change the schema again here, to drop the old time data column and edit the column name for LTZ to clearer one.

Figure 81

Overall Look of Summarization 1



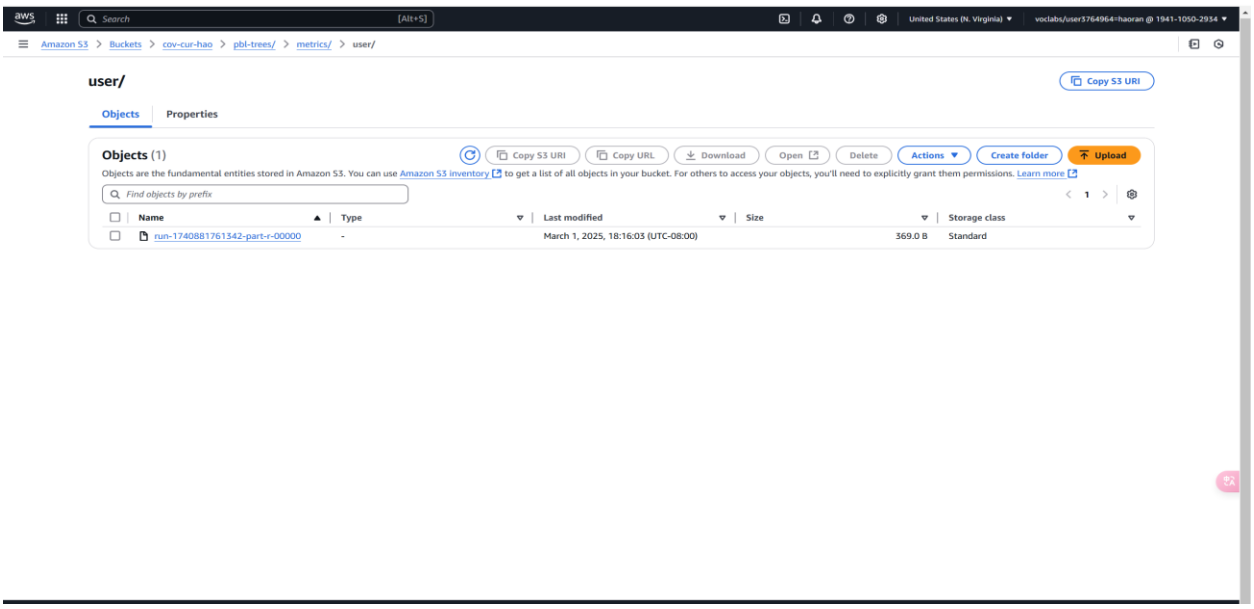
Note. This screenshot shows the overall look of summarization 1 in AWS Glue Visual ETL.

Source: AWS Glue

Here's the final look of the ETL pipeline of metric1.

Figure 82

User Output of Summarization 1

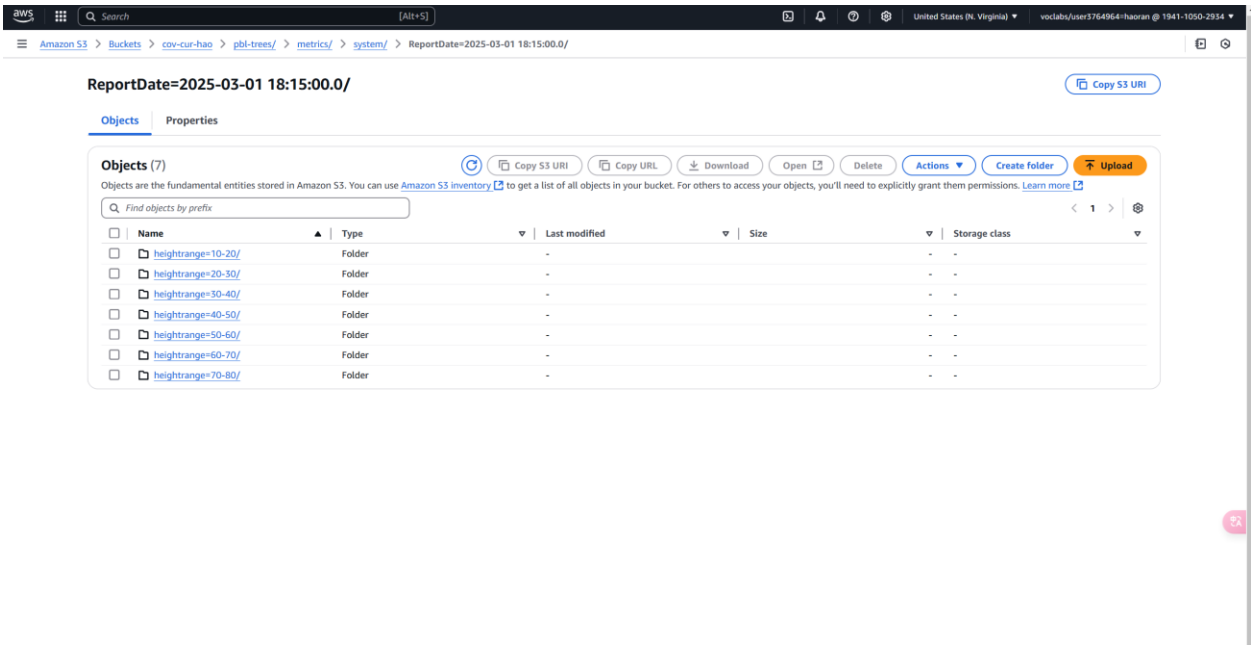


Note. This screenshot shows the user output of summarization 1 in AWS S3. Source: AWS S3.

Here’s the output after summarization. It is stored as user-friendly format in the user-folder .

Figure 83

System Output of Summarization 1

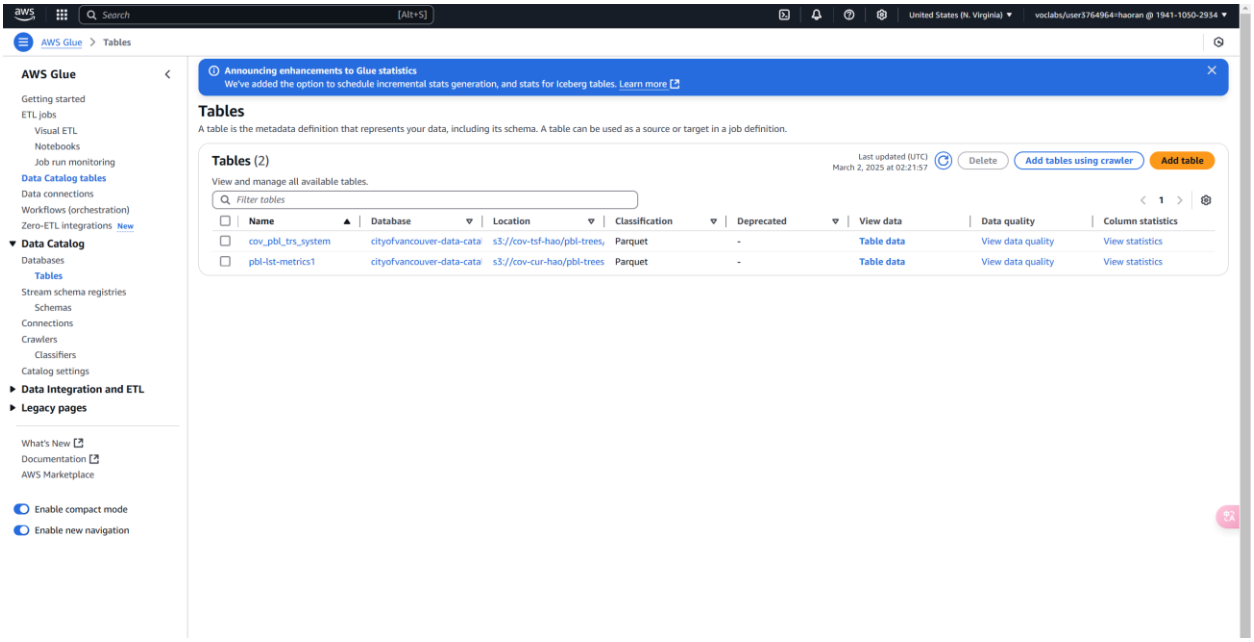


Note. This screenshot shows the system output of summarization 1 in AWS S3. Source: AWS S3

Here’s the output of summarization 1. It is stored as system-friendly format in the system folder.

Figure 84

Tables Screenshot

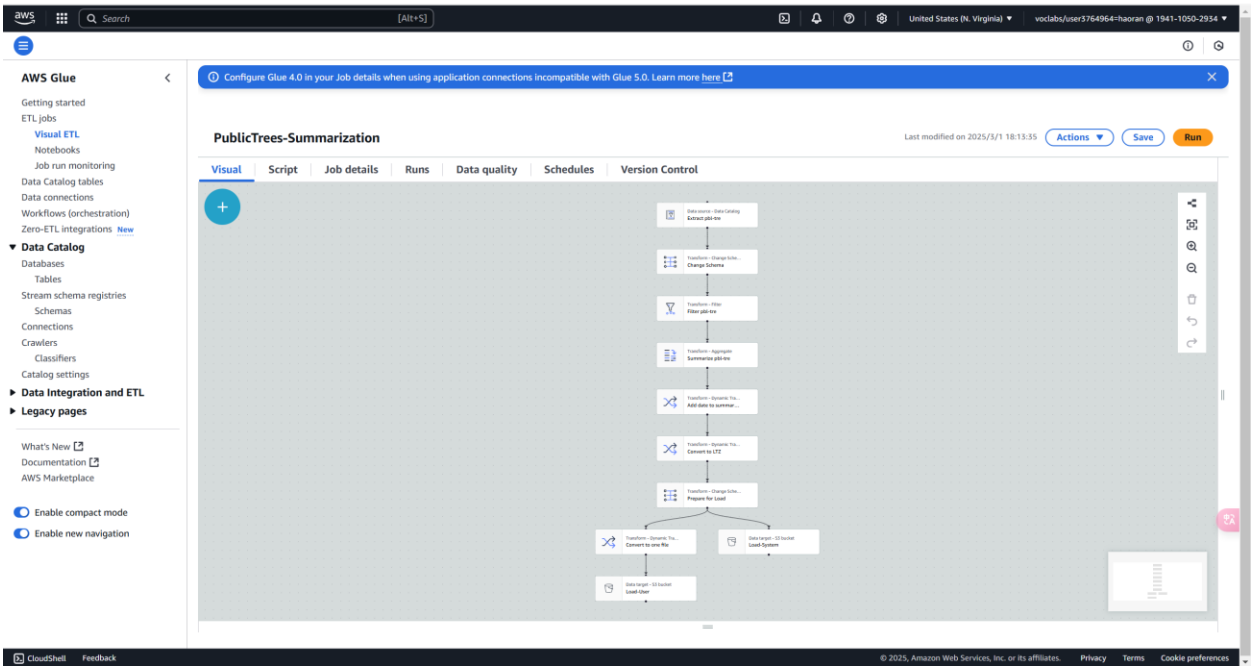


Note. This screenshot shows the tables output after summarization 1 in AWS Glue. Source: AWS Glue

Here's Tables screenshot in AWS Glue after running the summarization 1.

Figure 85

Overall look of Metric 1.



Note. This screenshot shows the overall look of summarization 1 in AWS Glue Visual ETL.

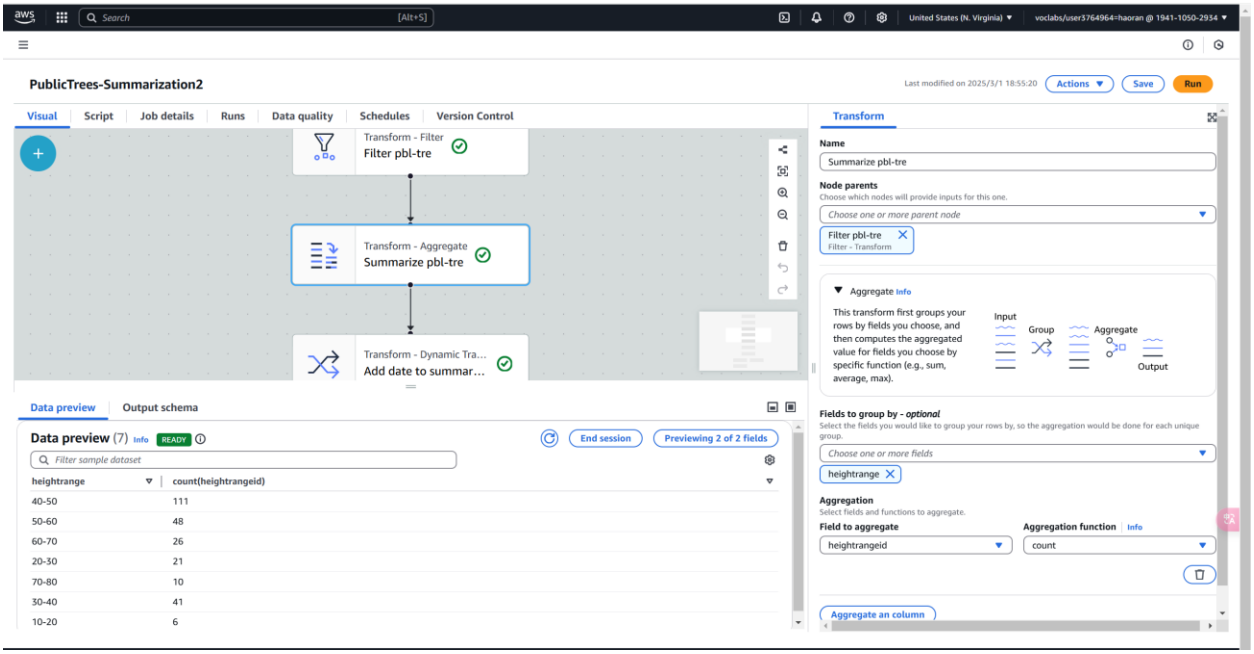
Source: AWS Glue

Here's the overall look of Metric 1 before running it.

Summarization Metric 2 (ETL pipeline 2)

Figure 86

Aggregate of Summarization 2



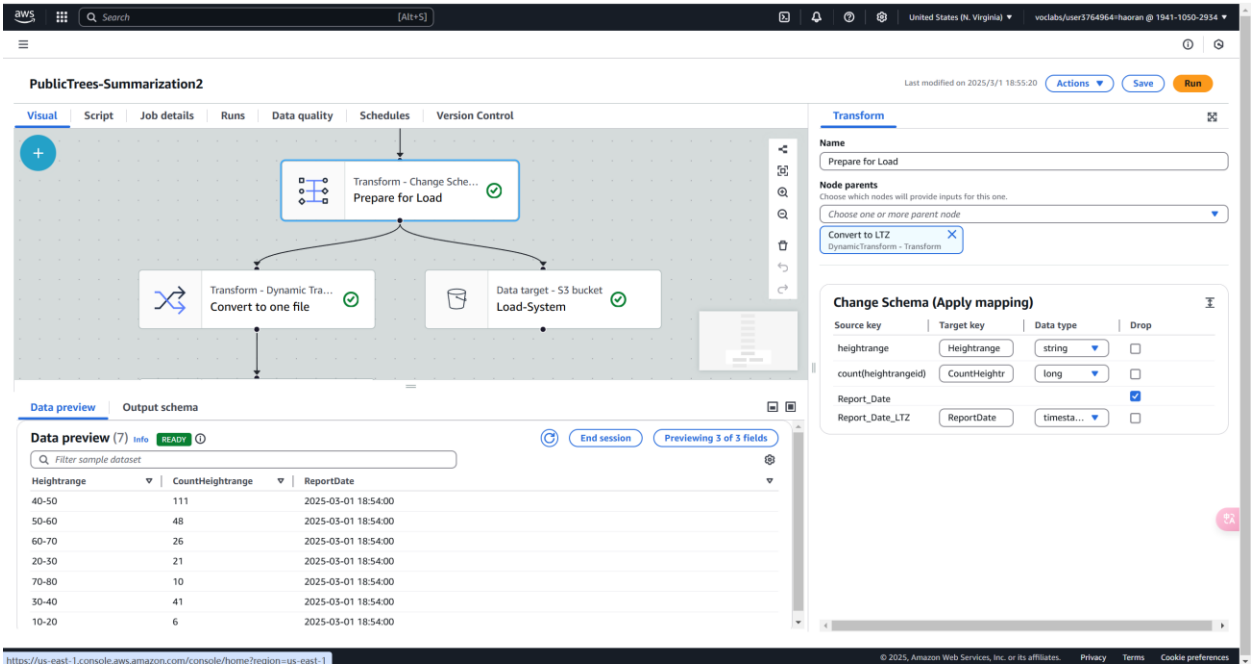
Note. This screenshot shows the aggregate step of summarization 2 in AWS Glue Visual ETL.

Source: AWS Glue

I created a new ETL pipeline2 by downloading and uploading pipeline1, because most of the parts will be the same. And I change the Summarization part to count how many are they for each heightrange.

Figure 87

Prepare for Load of Summarization 2

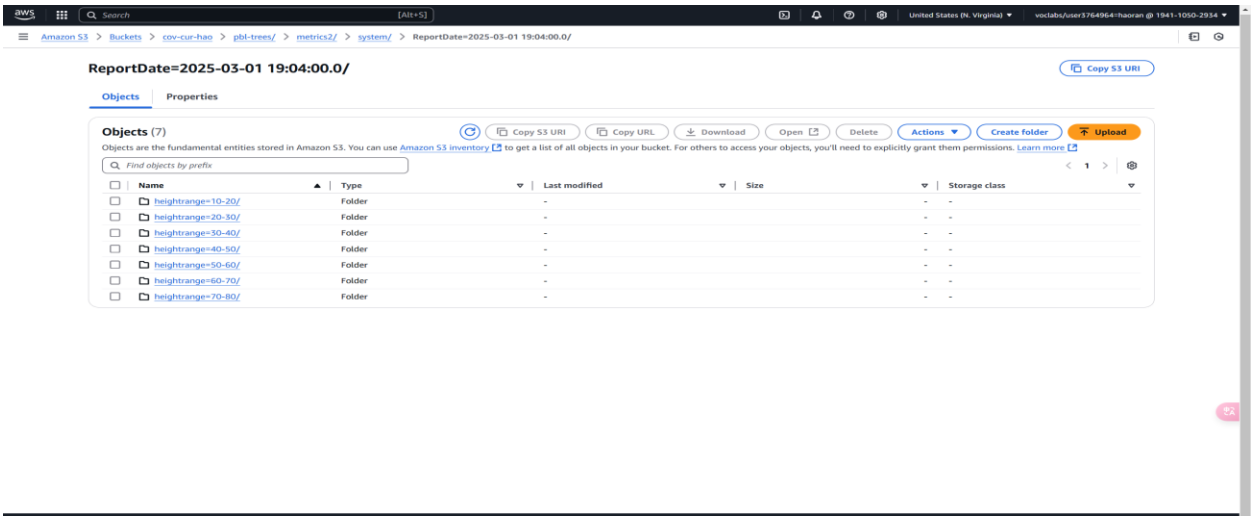


Note. This screenshot shows the prepare for load step in AWS Glue Visual ETL. Source: AWS Glue

I also change the column title of countheightrangeid to a clearer name “CountHeight” by using change schema before loading them to system and user folders.

Figure 88

System Output of Summarization 2

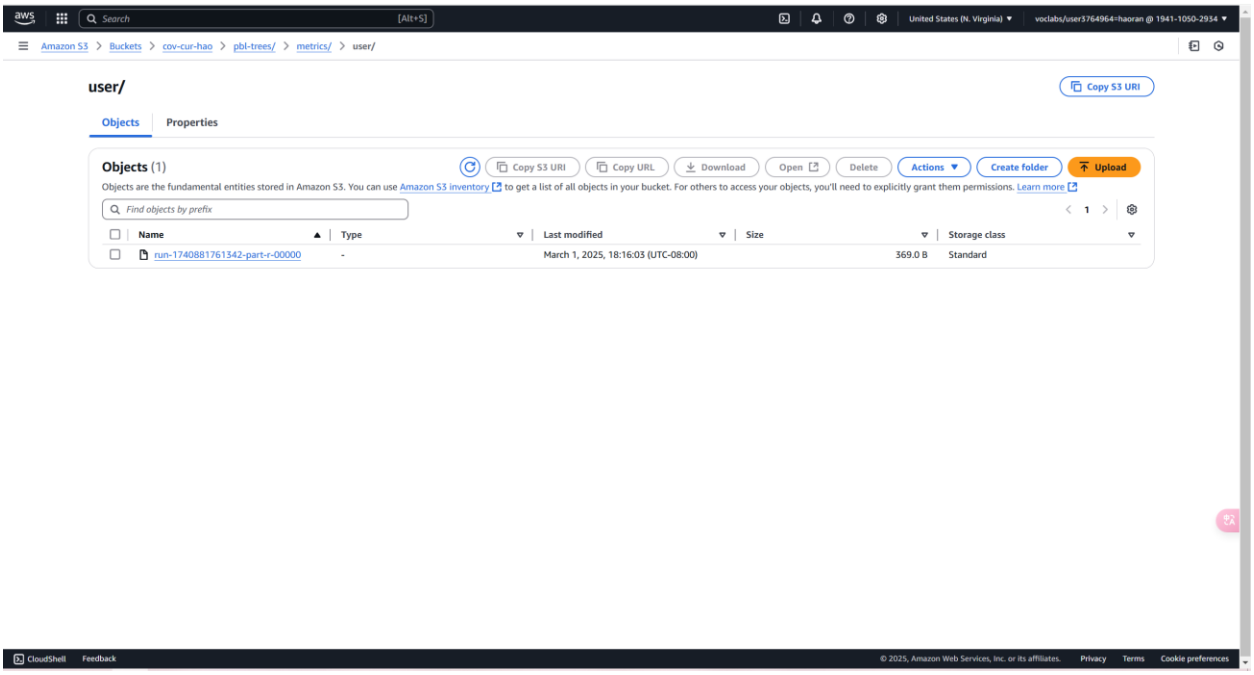


Note. This screenshot shows the system output of summarization 2 in AWS S3 Source: AWS S3

Here's the System-folder screenshot, storing the output dataset of summarization 2 in system-friendly format.

Figure 89

User Output of Summarization 2

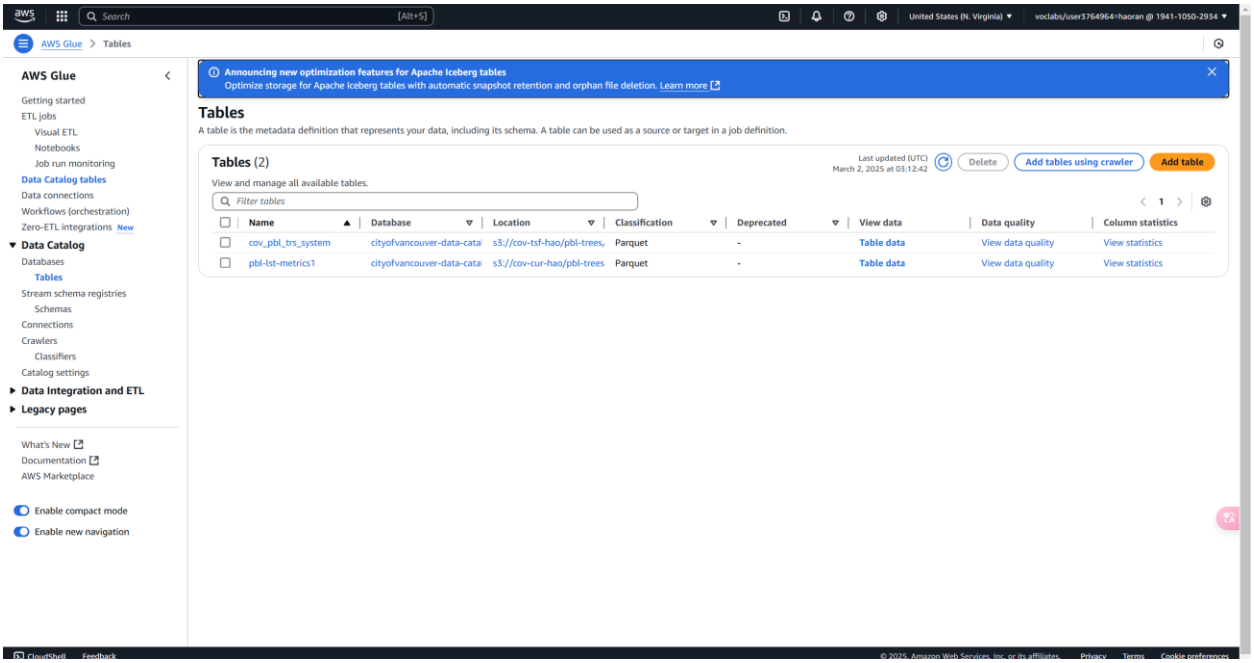


Note. This screenshot shows the user output of summarization 2 in AWS S3 Source: AWS S3

Here's the user-folder screenshot, storing the output dataset of summarization 2 in user-friendly format.

Figure 90

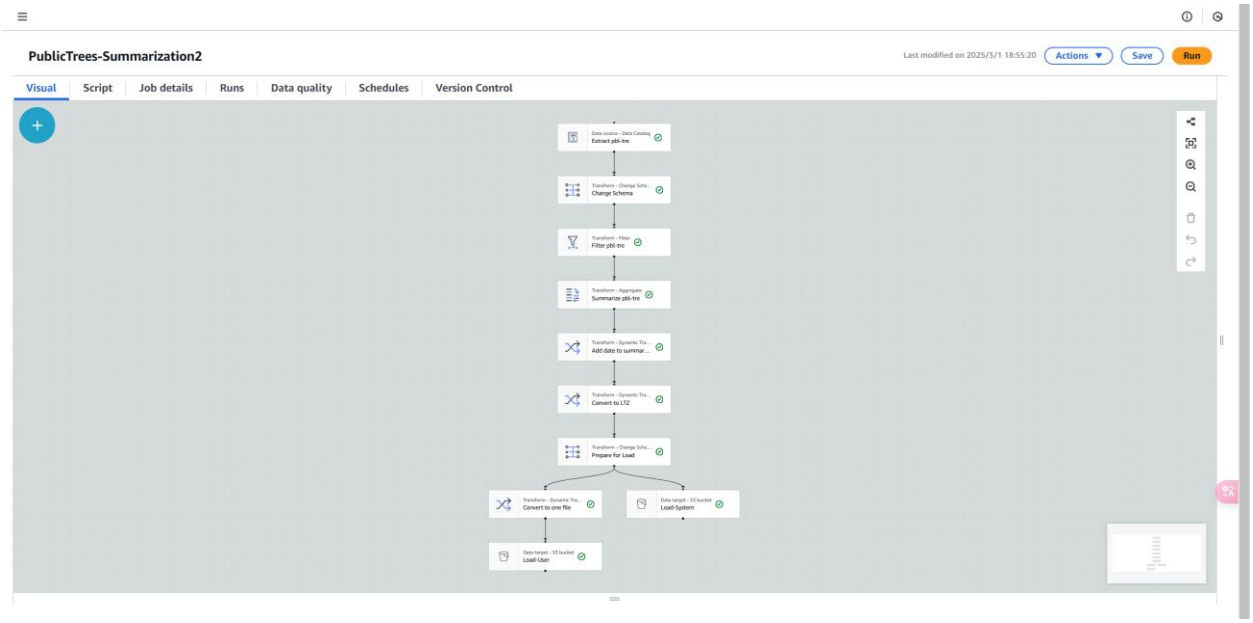
Tables Output of Summarization 2



Note. This screenshot shows the tables output of summarization 2 in AWS Glue. Source: AWS Glue

Figure 91

Overall Look of Metric 2



Note. This screenshot shows the final overall look of summarization 2 in AWS Glue. Source: AWS Glue

Here's the final overall look of summarization 2 for metric2.

DAP Estimated Cost (Teamwork)

Total Estimated Cost of The Team

The estimated total cost for our team would be USD 68.04 for 12 months, and the specific explanation for each member is below.

DAP Estimated Cost (Gyanvi Bhardwaj)

To estimate the monthly and yearly costs, I have used the AWS cost calculator. The first service added is AWS S3. Under this, I have added S3 Standard, S3 Intelligent Tiering, S3 Glacier Instant retrieval and data transfer. My object storage size is 0.0000629425 GB. The

lifecycle request per month is 1 as the only transition needed after 30 days would be from standard to intelligent tiering. Data returned and scanned by S3 Select is 1 GB per month.

The next service added is AWS Glue. Under Glue, the following services are added: AWS Glue ETL jobs and interactive sessions, AWS Glue Data Catalog storage requests, AWS Glue DataBrew interactive sessions and AWS Crawlers. There are 10 DPUs for the Apache Spark job, 0.0625 DPUs for the Python Shell job and the Apache Spark ETL job run duration is 2 minutes. The DataBrew jobs created is 1 and the number of nodes consumed for the DataBrew job is 5. The duration of the DataBrew job is approximately 2 minutes (1 minute 53 seconds). There is 1 crawler created with the duration for each crawler is 82 seconds.

Therefore, in total the two services with their sub-services cost 28.20 USD yearly and 2.35 USD monthly with 0 USD as the upfront cost.

Figure 92

Cost Estimation of Gyanvi Bhardwaj

aws

Contact your AWS representative: Contact Sales

Export Date: 03/04/2025 **Language:** English

Estimate url

Estimate summary

Upfront cost	Monthly cost	Total 12 months cost
0.00 USD	2.35 USD	28.20 USD Includes upfront cost

Detailed Estimate

Name	Group	Region	Upfront cost	Monthly cost
Amazon Simple Storage Service (S3)	-	US East (Ohio)	0.00 USD	0.00 USD
Status	-			
Description:	-			
Config summary	PUT, COPY, POST, LIST requests to S3 Standard (2), GET, SELECT, and all other requests from S3 Standard (1), S3 Standard storage (0.0000629425 GB per month) S3 INT Average Object Size (16 MB), Percentage of Storage in INT-Frequent Access Tier (1), Lifecycle Transition requests (1)			

Name	Group	Region	Upfront cost	Monthly cost
AWS Glue	-	US East (Ohio)	0.00 USD	2.35 USD
Status	-			
Description:	-			
Config summary	Number of DPUs for Apache Spark job (10), Number of DPUs for Python Shell job (0.0625) Number of interactive sessions for DataBrew (2), Number of nodes consumed for DataBrew job (5) Number of crawlers (1)			

Acknowledgement

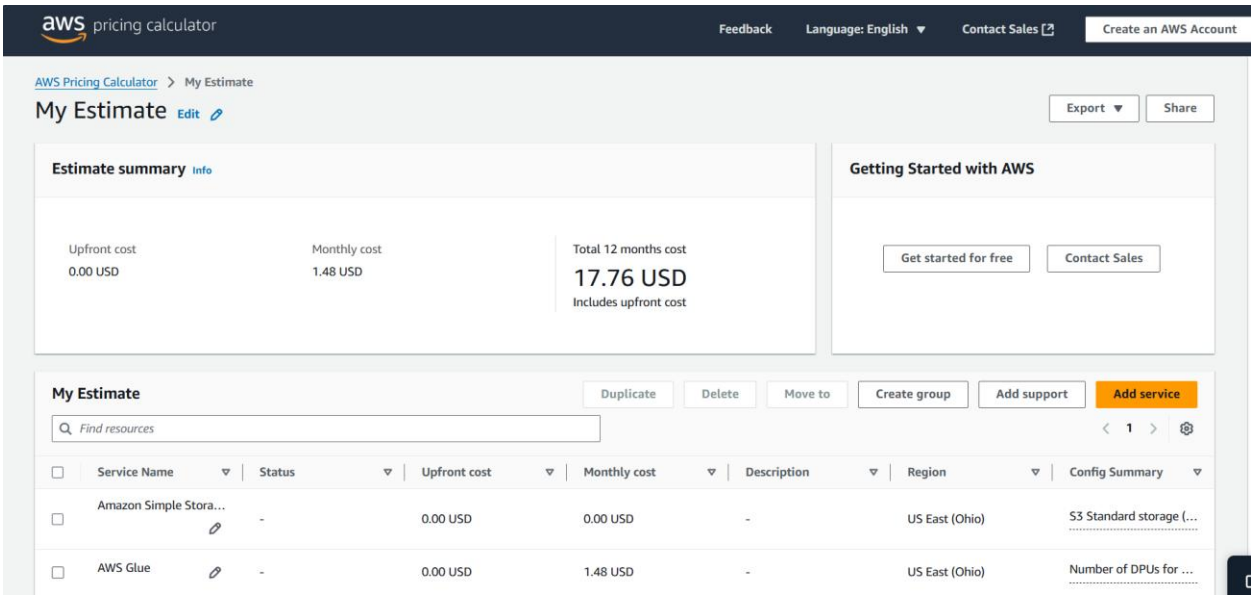
AWS Pricing Calculator provides only an estimate of your AWS fees and doesn't include any taxes that might apply. Your actual fees depend on a variety of factors, including your actual usage of AWS services. Learn more

Note. The total yearly cost of availing the mentioned AWS services is 28.20 USD which also includes the upfront cost. Source: AWS Cost calculator, *self-work*

DAP Estimated Cost (Peng Wang)

Figure 93

Cost Estimation of Peng Wang



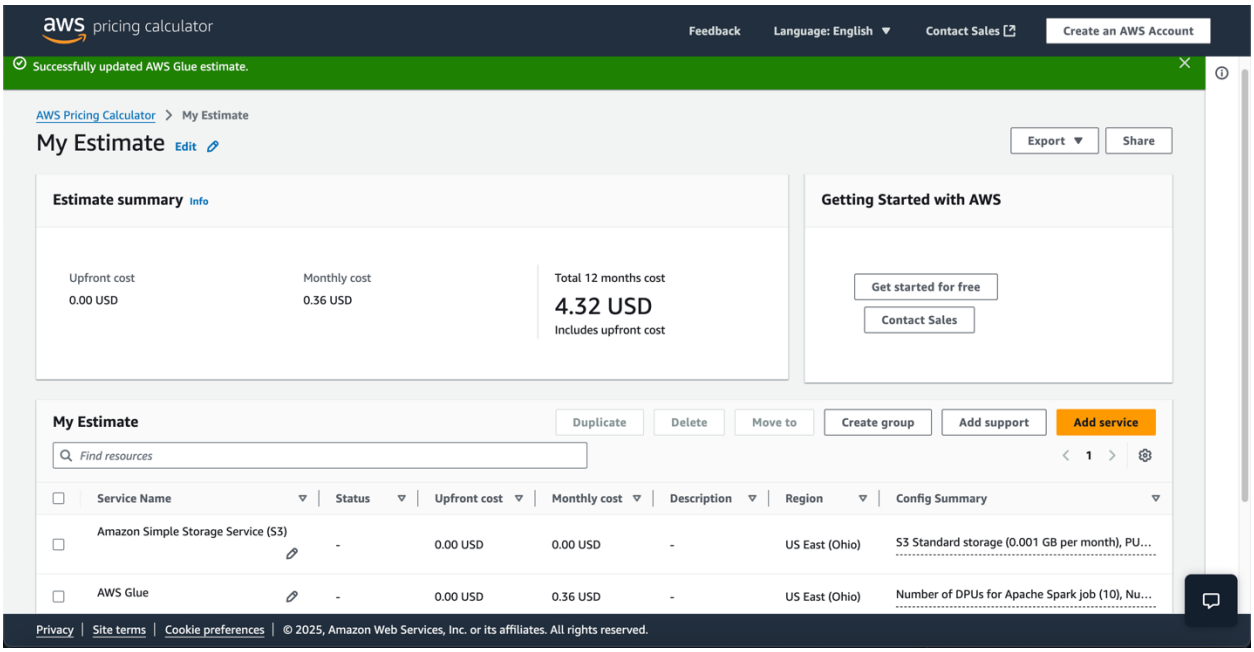
Note. The screenshot shows the cost estimation. Source: AWS cost calculator.

Explanation: My individual cost is 17.76 USD for total 12 months. Specifically, I used AWS S3 and AWS Glue. Regarding AWS S3, I used 0.000928 GB per month as the standard storage, and others are not necessary. Regarding AWS Glue, I used 10 DPUs for the Apache Spark job, 0.0625 DPUs for the Python Shell job, 0.000003 million per month for an object stored, and 0.000003 million per month for access requests, one interactive session for DataBrew, five nodes consumed for DataBrew job, and one crawler.

DAP Estimated Cost (Muhammad Zulqarnain Shahzad)

Figure 94

Cost Estimation of Muhammad Zulqarnain Shahzad



In this screenshot, I ran my project on AWS Pricing Calculator for the Business Licenses, Consulting and Management Services dataset on AWS, and I ended up with a total of about $4.32 for 12 months. That cost includes my S3 storage (which is tiny, since the dataset is only a few hundred KB) and my AWS Glue jobs (for data cleaning, cataloging, and the ETL pipeline). Because my data is so small and my ETL job runs for just 2 minutes, the monthly bill is super low. I didn't add any cost for Athena, because I either haven't run queries there or only ran a few small ones, which would be negligible anyway. So, if I only used S3 and Glue, $4.32 a year totally makes sense.

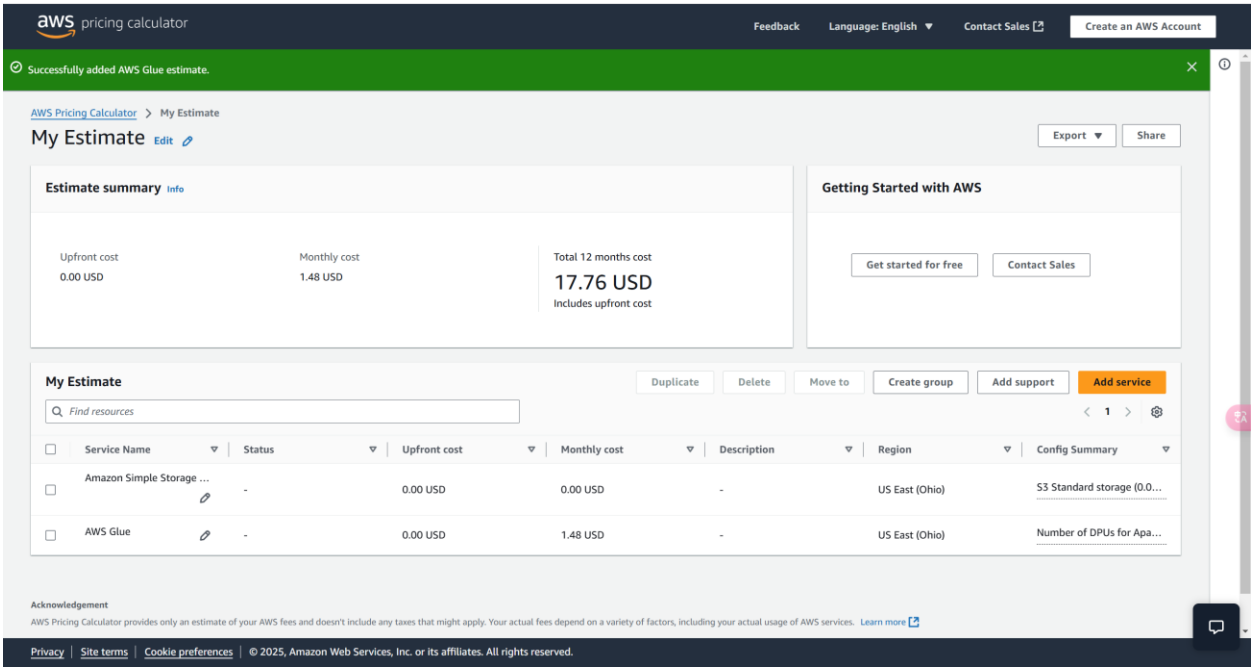
Basically, I'm paying a few cents a month to store the raw and cleaned data in S3, and the rest of the cost comes from Glue usage during my short ETL runs. If I ever scale up the data or

run more frequent jobs, this cost will increase but for now, it's perfect for a small project. This also shows that AWS can be pretty cheap when your dataset and usage are minimal.

DAP Estimated Cost (Haoran Xu)

Figure 95

Cost Estimation of Haoran Xu



Note. This screenshot is showing the estimate cost on AWS pricing calculator. Source: AWS pricing calculator.

My total cost estimation contains two main parts. Firstly, I used 0.0133GB monthly S3 standard storage. And I don't use other services like PUT, COPY, POST, GET, etc.

Secondly, I use AWS Glue for these uses:

1. Profiling and Cleaning in AWS Data Gluebrew. It contains 1 interactive session for Data Gluebrew.

2. Cataloging, Crawler and Summarization in AWS Glue. It contains both 0.000002 million objects stored and accesses per month. Using 10 DPUs for Apache Spark job and 0.0625 DPUs for Python shell job.

Total will be 1.48 USD cost per month and 17.76 USD cost annually.

Conclusion

Our project demonstrates a comprehensive, end-to-end data analytic platform built on AWS for the City of Vancouver, showcasing the power of cloud-based analytics across multiple datasets. As a team, we collectively tackled every required step, from data ingestion to cost estimation, while each member focused on a distinct dataset and set of analytical questions.

We began by extracting our datasets from the City of Vancouver Open Data Portal, each dataset was carefully selected to meet our volume requirements and provide valuable insights. Whether it was the Public Trees, 3-1-1 Contact Metrics, Business Licences for Consulting and Management Services, or the Animal Control Inventory Register, every dataset was first uploaded to dedicated raw S3 buckets. This ensured that the source data, including critical details like delimiters (e.g., the semicolon in our CSV files), was preserved without alteration.

Next, using AWS Glue DataBrew, each team member profiled and cleaned their dataset. We addressed data quality issues by correcting delimiter settings, standardizing column names, reformatting date fields, and removing unnecessary or redundant columns. This uniform cleaning process not only enhanced data consistency but also laid a strong foundation for accurate analysis.

To organize our cleaned data for subsequent processing, we used AWS Glue Crawlers to automatically catalog the datasets, generating well-defined database schemas. This step was

crucial for enabling efficient querying and integration of our data through AWS Athena and subsequent ETL processes.

For the transformation and summarization phases, each member implemented their own ETL pipelines in AWS Glue applying targeted filtering, aggregation, and schema mapping to derive the key metrics required by our descriptive analysis questions. While the specific transformation details varied depending on the dataset (e.g., average tree diameter versus height range, call handling metrics, Number of licenses and average number of employees, or animal impoundment statistics), the underlying methodology was consistent. We ensured that our final outputs were stored in clearly designated folders (transform, system, and user) for both technical verification and user-friendly access.

Finally, we conducted a detailed cost estimation using the AWS Pricing Calculator. By carefully accounting for our minimal storage needs, short duration ETL jobs, and light query loads, our estimated annual cost remained very low. This not only highlights the cost-efficiency of AWS for small-scale projects but also demonstrates the scalability of our platform should the data volume or processing frequency increase, our approach could easily adapt.

In summary, our group project fulfills all the requirements: we designed and implemented a data analytic platform that supports descriptive analysis and provided a thorough report that breaks down each step of the process. By integrating individual efforts into a cohesive, cloud-based solution, we've shown that the City of Vancouver can leverage AWS to drive data-based decision-making with minimal cost and high efficiency. Our project stands as a robust, scalable foundation for future data analytic endeavors, and we're proud of the collaborative effort that made it possible.

References

Animal control inventory - register. (2019). Vancouver.ca. https://opendata.vancouver.ca/explore/dataset/animal-control-inventory-register/information/?sort=dateimpounded&refine.dateimpounded=2024

City of Vancouver. (2025). *Business licences – consulting and management services*. City of Vancouver Open Data Portal. Retrieved February 25, 2025, from https://opendata.vancouver.ca/