

ETL Pipeline for Data Processing & Visualization on AWS QuickSight

Author: - Sarav Shah
Professor: Dr Joann Ordille
Rutgers Business School, NJ

Contents

| | | |
|----------|-------------------------|--|
| 1 | Abstract | |
| 2 | Introduction | |
| 3 | ETL Architecture | |
| 4 | EDA | |
| 5 | Methodology | |
| 5.1 | Amazon S3 | |
| 5.2 | AWS Glue & Data Catalog | |
| 5.3 | AWS Lambda | |
| 5.4 | AWS Athena | |
| 5.5 | AWS QuickSight | |
| 6 | Results | |
| 7 | Conclusion | |

1. ABSTRACT

This document outlines a streamlined ETL (Extract, Transform, Load) pipeline tailored for efficient data processing and visualization on AWS QuickSight. Utilizing AWS services such as S3 storage, AWS Glue, Lambda, Athena, Glue ETL jobs, and QuickSight, this pipeline seamlessly extracts data, preprocesses it with Lambda functions, and transforms it using Glue ETL jobs. The structured data is stored in S3 or loaded into a data warehouse for further analysis.

Key components include AWS Glue for metadata management, Lambda for data cleaning, Athena for SQL queries, and QuickSight for cloud-native visualizations. The serverless architecture ensures scalability, and QuickSight provides a user-friendly interface for creating interactive dashboards and reports. This ETL pipeline exemplifies the integration and efficiency of AWS services, offering organizations a robust solution for data-driven decision-making.

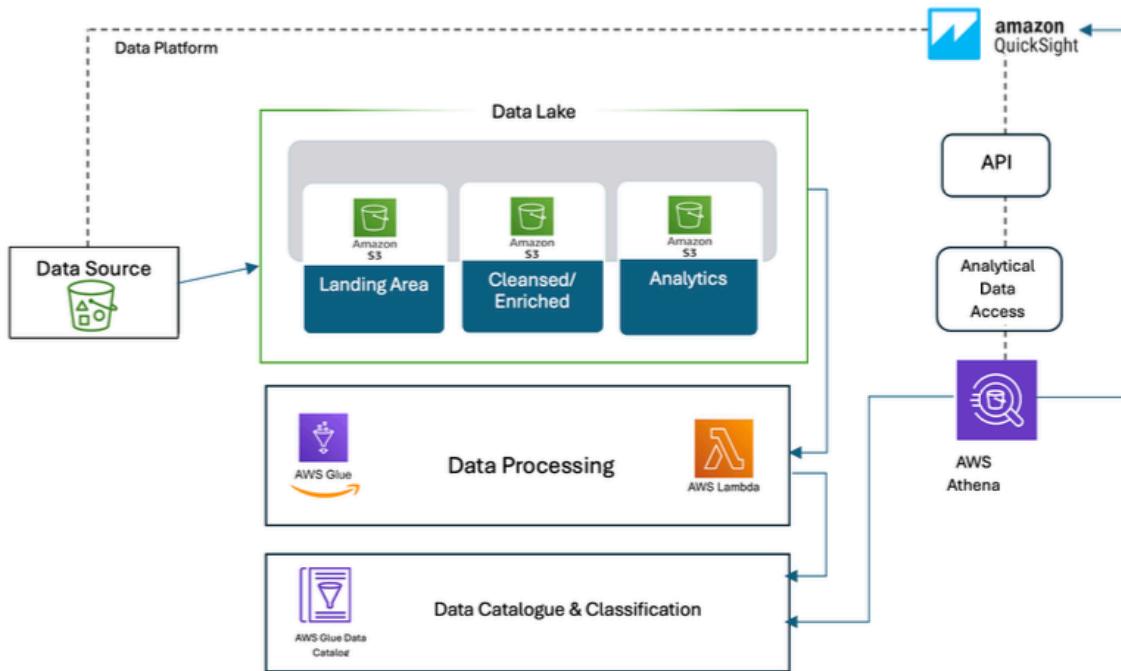
2. INTRODUCTION

In the realm of data-driven decision-making, constructing a resilient ETL (Extract, Transform, Load) pipeline is pivotal for organizations seeking efficient data processing and visualization. This framework serves as the backbone for seamless extraction, transformation, and loading of data, facilitating a streamlined journey from raw datasets to insightful visualizations.

At the core of this pipeline is the orchestration of AWS services, with each contributing to specific stages of the data journey. Beginning with data extraction and storage in Amazon S3, the pipeline traverses through AWS Glue and Lambda functions for data preprocessing. Subsequently, Amazon Athena facilitates SQL queries for querying processed data, and AWS Glue ETL jobs automate the transformation process. The culminating visualization step unfolds on Amazon QuickSight, renowned for its cloud-native business intelligence service, offering interactive dashboards and user-friendly reports.

This introduction sets the stage for a comprehensive exploration of the ETL pipeline, accentuating the synergy of AWS services, and highlighting the unique advantages offered by AWS QuickSight. Furthermore, a comparative analysis with Tableau, another leading visualization tool, will be undertaken to provide insights into the strengths and distinctions of each platform. As organizations strive for agility, scalability, and actionable insights, the integration of AWS services and the ensuing comparison with Tableau emerge as strategic approaches to harness the full potential of big data analytics.

3. ETL ARCHITECTURE



S3 serves as the primary data storage solution. Raw, unprocessed data is commonly stored in an S3 landing zone. AWS Glue automates the process of cleaning, transforming, and preparing data for analysis. AWS Lambda provides serverless compute service for running code without provisioning servers, and the Lambda functions can be triggered by events. After transformation, the cleaned and processed data is stored back in a different S3 bucket. Amazon Athena query service allows SQL-based queries directly on data stored in S3. A fully managed ETL job is executed to further process data if needed before analysis. Amazon QuickSight provides Business analytics service for building visualizations, dashboards, and reports.

4. EXPLORATORY DATA ANALYSIS

Each column in the given list corresponds to specific information in the dataset. Here a description of some of the relevant columns:

'country': Represents the name of the country.

'rank': Indicates the rank of the country based on a certain criterion (e.g., population, area, etc.).

'area': Denotes the total area of the country in square kilometers.

'landAreaKm': Represents the land area of the country in square kilometers.

'cca2' & 'cca3' : Represents country code assigned by ISO.

'netChange': Indicates the net change in population for a specific time period (e.g., yearly net change).

'growthRate': Represents the growth rate of the population, typically expressed as a percentage.

'worldPercentage': Denotes the percentage of the world's population that the country represents.

'pop1980': Represents the population estimate for the year 1980.

'pop2023': Represents the population estimate for the year 2023.

'pop2030': Represents the projected population estimate for the year 2030.

'pop2050': Represents the projected population estimate for the year 2050.

```
import pandas as pd
import numpy as np

data = pd.read_csv("/Users/sarav/Desktop/Study Materials/Fall' 23/Advanced DB/Project/Final data set/world population/countries-table.csv")
data

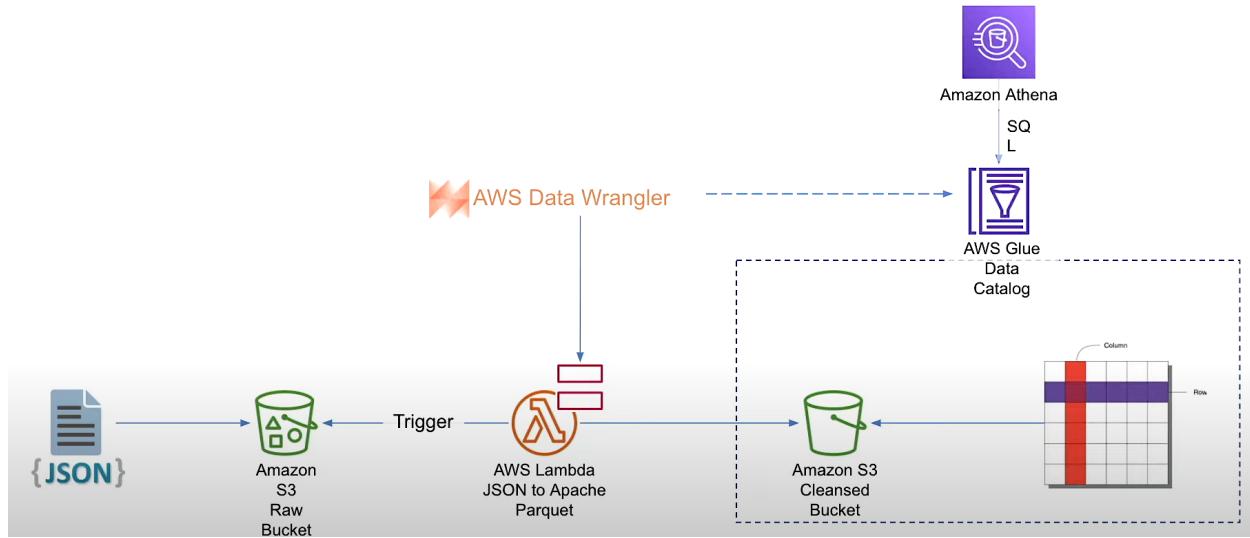
data = data.dropna()
data.head()

data2 = data.describe().T
data2
```

| Index | country | rank | area | landAreaKm | cca2 | cca3 | netChange | growthRate | worldPercentage | density | densityMi | place | pop1980 | pop2000 | pop2010 | pop2022 | pop2023 |
|-------|----------------|------|-------------|-------------|------|------|-----------|------------|-----------------|---------|-----------|-------|-----------|------------|------------|------------|------------|
| 0 | India | 1 | 3.28759e+06 | 2.97319e+06 | IN | IND | 8,4184 | 8.0001 | 0.1785 | 408,583 | 1244,5 | 356 | 695828385 | 1859633675 | 1248613628 | 1417173173 | 1428627663 |
| 1 | China | 2 | 9.70896e+06 | 9.4247e+06 | CN | CN | -0.0113 | -0.0002 | 0.1781 | 151,27 | 391,788 | 156 | 982372466 | 126499869 | 1348191368 | 1425887377 | 1425671152 |
| 2 | United States | 3 | 9.37261e+06 | 9.14742e+06 | US | USA | 8,6581 | 0.0005 | 0.0425 | 37,1586 | 96,2666 | 848 | 231448018 | 282398554 | 311182845 | 338289857 | 339986563 |
| 3 | Indonesia | 4 | 1.98457e+06 | 1.87752e+06 | ID | IDN | 8,0727 | 0.0074 | 0.0347 | 147,82 | 382,853 | 368 | 148177095 | 214872421 | 244816173 | 277581339 | 277534122 |
| 4 | Pakistan | 5 | 881912 | 778888 | PK | PAK | 0,1495 | 0.0198 | 0.83 | 311,962 | 887,983 | 586 | 88624857 | 154369924 | 194454498 | 235824862 | 248485568 |
| 5 | Nigeria | 6 | 923768 | 918978 | NG | NGA | 0,1658 | 0,0241 | 0.828 | 245,731 | 636,444 | 566 | 72951439 | 122851984 | 168952853 | 218541212 | 22384632 |
| 6 | Brazil | 7 | 8.51577e+06 | 8.35814e+06 | BR | BRA | 8,0393 | 0.0052 | 0.027 | 25,0936 | 67,8645 | 76 | 12228838 | 175973728 | 196353492 | 215313498 | 216422446 |
| 7 | Bangladesh | 8 | 147578 | 139178 | BD | BGD | 8,0557 | 0.0183 | 0.0216 | 1328,68 | 3441,28 | 58 | 83929765 | 129193327 | 148391139 | 171186372 | 172954319 |
| 8 | Russia | 9 | 1.76982e+07 | 1.63769e+07 | RU | RUS | -0,0158 | -0.0019 | 0.018 | 8,82 | 22,8439 | 643 | 136257420 | 146944839 | 143242599 | 144713134 | 144443459 |
| 9 | Mexico | 10 | 1.96438e+06 | 1.94395e+06 | MX | MEX | 8,0299 | 0.0075 | 0.016 | 66,8797 | 171,146 | 484 | 67785186 | 97873442 | 112532481 | 127584125 | 128455567 |
| 10 | Ethiopia | 11 | 1.10453e+06 | 1.12857e+06 | ET | ETH | 8,1888 | 0.0255 | 0.0158 | 112,113 | 298,372 | 231 | 34945469 | 67831867 | 89237791 | 123379924 | 126527868 |
| 11 | Japan | 12 | 377938 | 364588 | JP | JPN | -0,0249 | -0.0053 | 0.0154 | 338,257 | 876,885 | 392 | 117624196 | 126883861 | 128185432 | 129515692 | 132394513 |
| 12 | Philippines | 13 | 342553 | 299178 | PH | PHL | 8,0561 | 0.0154 | 0.0147 | 393,525 | 1019,23 | 688 | 48419546 | 77958223 | 94636708 | 115559899 | 117373768 |
| 13 | Egypt | 14 | 1.00245e+06 | 994548 | EG | EGY | 8,0551 | 0.0156 | 0.0141 | 113,232 | 293,27 | 818 | 43748556 | 71371371 | 87252413 | 118998183 | 112716598 |
| 14 | DR Congo | 15 | 2.34486e+06 | 2.26785e+06 | CD | COD | 8,185 | 0.0329 | 0.0128 | 45,1883 | 116,831 | 188 | 26788686 | 48616317 | 66391257 | 99818212 | 102628888 |
| 15 | Vietnam | 16 | 331212 | 313429 | VN | VNM | 8,0288 | 0.0068 | 0.0123 | 315,411 | 816,914 | 784 | 52968278 | 79801142 | 87411012 | 98186656 | 98854956 |
| 16 | Iran | 17 | 1.6482e+06 | 1.6225e+06 | IR | IRN | 8,0288 | 0.0087 | 0.0111 | 54,6801 | 142,347 | 364 | 38526664 | 65543483 | 75737855 | 88558578 | 89172767 |
| 17 | Turkey | 18 | 703562 | 703562 | TR | TUR | 8,0143 | 0.0056 | 0.0187 | 189,521 | 283,658 | 792 | 44889869 | 64113547 | 73195345 | 85341241 | 85816199 |
| 18 | Germany | 19 | 357114 | 349398 | DE | DEU | -0,0012 | -0.0049 | 0.0184 | 238,4 | 617,456 | 276 | 77786783 | 81551677 | 81325498 | 83369843 | 83294633 |
| 19 | Thailand | 20 | 513128 | 510898 | TH | THA | 8,0883 | 0.0015 | 0.009 | 148,542 | 364,083 | 764 | 45737753 | 63066683 | 68278489 | 71697838 | 71881279 |
| 20 | United Kingdom | 21 | 242988 | 241938 | GB | GBR | 8,0872 | 0.0034 | 0.0085 | 279,985 | 725,162 | 826 | 56326328 | 58808843 | 62768439 | 67589936 | 67736882 |
| 21 | Tanzania | 22 | 945887 | 885888 | TZ | TZA | 8,0652 | 0.0296 | 0.0084 | 76,1324 | 197,183 | 834 | 19297659 | 34463784 | 45118527 | 65497748 | 67438186 |
| 22 | France | 23 | 551695 | 547557 | FR | FRA | 8,0864 | 0.0081 | 0.0081 | 118,265 | 386,385 | 258 | 53713838 | 58665453 | 62444567 | 64626628 | 64755684 |
| 23 | South Africa | 24 | 1.22184e+06 | 1.21399e+06 | ZA | ZAF | 8,0173 | 0.0087 | 0.0075 | 49,8822 | 126,988 | 718 | 29463549 | 46813266 | 51784921 | 59893885 | 60414495 |
| 24 | Italy | 25 | 381336 | 295717 | IT | ITA | -0,0054 | -0.0028 | 0.0074 | 199,478 | 515,612 | 388 | 56329482 | 56966397 | 59822458 | 59837474 | 58878762 |
| 25 | Kenya | 26 | 580367 | 569148 | KE | KEN | 8,0346 | 0.0159 | 0.0069 | 96,8138 | 256,748 | 484 | 16187124 | 38851686 | 41517895 | 54827487 | 55108586 |
| 26 | Myanmar | 27 | 676578 | 652678 | MM | MMR | 8,0125 | 0.0074 | 0.0066 | 83,6227 | 216,583 | 184 | 33465781 | 45538332 | 49396988 | 54179396 | 54579977 |
| 27 | Colombia | 28 | 1.14175e+06 | 1.18954e+06 | CO | COL | 8,0888 | 0.0041 | 0.0065 | 46,9447 | 121,587 | 170 | 26176195 | 39215135 | 44816188 | 51874824 | 52885168 |
| 28 | South Korea | 29 | 100218 | 97688 | KR | KOR | -0,0012 | -0.0086 | 0.0065 | 538,574 | 1374,19 | 418 | 38178581 | 46788591 | 48813842 | 51815818 | 51784859 |
| 29 | Uganda | 30 | 241558 | 208528 | UG | UGA | 8,0423 | 0.0282 | 0.0061 | 242,282 | 627,51 | 800 | 13284262 | 24828697 | 32341728 | 47249585 | 48822334 |
| 30 | Sudan | 31 | 1.88687e+06 | 1.86864e+06 | SD | SDN | 8,0394 | 0.0263 | 0.0086 | 25,7543 | 66,7836 | 729 | 16673586 | 26208773 | 33739933 | 46874204 | 48189886 |
| 31 | Spain | 32 | 585992 | 499557 | ES | ESP | -0,0014 | -0.0088 | 0.0059 | 95,1236 | 246,37 | 724 | 37491666 | 48741651 | 46572772 | 47558638 | 4751628 |

| Index | count | mean | std | min | 25% | 50% | 75% | max |
|-----------------|-------|-------------|-------------|---------|----------|-------------|-------------|-------------|
| rank | 224 | 112.915 | 65.3049 | 1 | 56.75 | 112.5 | 169.25 | 227 |
| area | 224 | 603662 | 1.79693e+06 | 2.02 | 6106.25 | 87480.5 | 446762 | 1.70982e+07 |
| landAreaKm | 224 | 578248 | 1.72362e+06 | 2 | 5832.5 | 85497 | 416370 | 1.63769e+07 |
| netChange | 224 | 0.0103929 | 0.0349172 | -0.0286 | 0 | 0.0009 | 0.008075 | 0.4184 |
| growthRate | 224 | 0.009979991 | 0.0124894 | -0.0745 | 0.002575 | 0.00835 | 0.017775 | 0.0498 |
| worldPercentage | 224 | 0.00448482 | 0.0175205 | 0 | 0.0001 | 0.0008 | 0.003075 | 0.1785 |
| density | 224 | 460.344 | 2021.32 | 0.138 | 42.8044 | 99.8273 | 242.497 | 21402.7 |
| densityMi | 224 | 1192.29 | 5235.22 | 0.3574 | 110.863 | 258.553 | 628.068 | 55433 |
| place | 224 | 434.094 | 255.71 | 4 | 213.5 | 429 | 659.25 | 894 |
| pop1980 | 224 | 1.98275e+07 | 8.34989e+07 | 6568 | 322399 | 3.37943e+06 | 1.03772e+07 | 9.82372e+08 |
| pop2000 | 224 | 2.74338e+07 | 1.14036e+08 | 9638 | 434857 | 4.75962e+06 | 1.63293e+07 | 1.2641e+09 |
| pop2010 | 224 | 3.11679e+07 | 1.26811e+08 | 10241 | 535598 | 5.4401e+06 | 2.07814e+07 | 1.34819e+09 |
| pop2022 | 224 | 3.55838e+07 | 1.39608e+08 | 11312 | 624822 | 6.15604e+06 | 2.44374e+07 | 1.42589e+09 |
| pop2023 | 224 | 3.5897e+07 | 1.40239e+08 | 11396 | 625673 | 6.23591e+06 | 2.44827e+07 | 1.42863e+09 |
| pop2030 | 224 | 3.81314e+07 | 1.44715e+08 | 11229 | 661922 | 6.69341e+06 | 2.83299e+07 | 1.51499e+09 |
| pop2050 | 224 | 4.33211e+07 | 1.51192e+08 | 11349 | 768297 | 8.33673e+06 | 3.66369e+07 | 1.67049e+09 |

5. METHODOLOGY



This section explains the steps and approach carried out to build the ETL pipeline. ETL is a common data wrangling process that involves extracting data from a source system, transforming it into a usable format, and then loading it into a target system. We will now review each section of the pipeline:

5.1 Amazon S3

S3 serves as the primary data storage solution. Raw, unprocessed data is commonly stored in an S3 landing zone. After transformation, the cleaned and processed data is stored back in S3 or moved to a different S3 bucket. Here is a detailed view of all the S3 buckets that were created and used throughout the pipeline formation and execution.

| General purpose buckets (7) Info | | | | |
|--|---------------------------------|-------------------------------|---|---|
| Buckets are containers for data stored in S3. Learn more | | | | |
| <input type="text"/> Find buckets by name | | | | |
| Name | AWS Region | Access | Creation date | |
| aws-athena-query-results-us-east-1-786302495125 | US East (N. Virginia) us-east-1 | Bucket and objects not public | December 20, 2023, 05:07:31 (UTC-05:00) | C Copy ARN Empty Delete Create bucket |
| aws-glue-assets-786302495125-us-east-1 | US East (N. Virginia) us-east-1 | Bucket and objects not public | December 20, 2023, 04:34:38 (UTC-05:00) | |
| de-on-population-analytics-dev | US East (N. Virginia) us-east-1 | Bucket and objects not public | December 20, 2023, 04:30:38 (UTC-05:00) | |
| de-on-population-cleaned-useast1-dev | US East (N. Virginia) us-east-1 | Bucket and objects not public | December 19, 2023, 16:14:46 (UTC-05:00) | |
| de-on-population-raw-useast1-dev | US East (N. Virginia) us-east-1 | Bucket and objects not public | December 19, 2023, 02:55:12 (UTC-05:00) | |
| de-on-population-semistructured-useast1-dev | US East (N. Virginia) us-east-1 | Bucket and objects not public | December 19, 2023, 03:02:50 (UTC-05:00) | |
| de-population-raw-useast1-athena-job | US East (N. Virginia) us-east-1 | Bucket and objects not public | December 19, 2023, 04:57:36 (UTC-05:00) | |

5.2 AWS Glue & Data Catalog

AWS Glue is a fully managed ETL service. It automates the process of cleaning, transforming, and preparing data for analysis. Glue supports visual programming and code-based scripting for data transformation tasks.

We first created a crawler in AWS glue for building the raw data catalog and semi structured data catalog.

Upon running successfully, the crawler adds tables in the database.

The screenshot shows the AWS Glue Crawler list page. At the top, there is a breadcrumb navigation: AWS Glue > Crawlers. Below the breadcrumb, the title "Crawlers" is displayed. A subtitle explains: "A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog." There is a "Create crawler" button in the top right corner. The main area is a table titled "Crawlers (2) info". The table has columns: Name, State, Schedule, Last run, Last run timestamp, Log, and Table changes from last The first row shows a crawler named "de-on-population-semistructured-useast1-cra..." with a "Ready" state, last run at "Succeeded" on December 20, 2023, and 1 created table. The second row shows a crawler named "de-population-raw-glue-catalog-1" with a "Ready" state, last run at "Succeeded" on December 19, 2023, and 1 created table. The table includes a "Filter crawlers" input field and navigation controls (previous, next, last, first).

| Name | State | Schedule | Last run | Last run timestamp | Log | Table changes from last ... |
|--|-------|----------|-----------|-----------------------|--------------------------|-----------------------------|
| de-on-population-semistructured-useast1-cra... | Ready | | Succeeded | December 20, 2023 ... | View log | 1 created |
| de-population-raw-glue-catalog-1 | Ready | | Succeeded | December 19, 2023 ... | View log | 1 created |

To Query on the data table, we will be redirected to AWS Athena

Upon running the select we get error. It's not able to understand the json data. (array data type).

The reason why we are facing this issue because of an issue called SerDe.

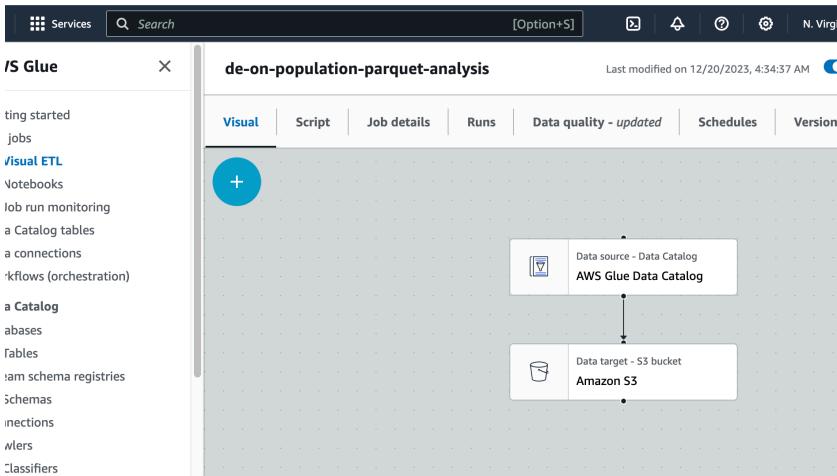
The SerDe expects each JSON document to be on a single line of text with no line termination characters separating the fields in the record. If the JSON text is in pretty print format, you may receive an error message.

The screenshot shows the AWS Glue Data Catalog interface. On the left, there's a sidebar for 'Data' with dropdowns for 'Data source' (AwsDataCatalog) and 'Database' (de-population-raw-db). Below that is a 'Tables and views' section with a 'Create' button and a search bar. Under 'Tables', there is one entry: 'de_on_population_raw_useast1_dev'. The main area has two tabs: 'Query 1' and 'Query 2'. 'Query 1' contains the SQL command: 'SELECT * FROM "AwsDataCatalog"."de-population-raw-db"."de_on_population_raw_useast1_dev" limit 10;'. The status bar at the bottom indicates the query failed: 'Failed' with a red icon, 'Time in queue: 106 ms', 'Run time: 575 ms', and 'Data scanned: -'. A red box highlights the error message: 'HIVE_CURSOR_ERROR: Row is not a valid JSON Object - JSONException: Missing value at 1 [character 2 line 1] This query ran against the "de-population-raw-db" database, unless qualified by the query. Please post the error message on our forum or contact customer support with Query id: 13a494d5-4d3d-419e-8090-0ec2b4cdf53d'.

AWS Glue ETL job

Glue ETL jobs work with a data structure called a DynamicFrame, which is an extension of Apache Spark DataFrames. It provides a flexible and schema-less representation of your data. Glue offers Fully managed ETL service for processing data at scale.

It writes the transformed data to the destination specified.



[Visual](#) | [Script](#) | [Job details](#) | [Runs](#) | [Data quality - updated](#) | [Schedules](#) | [Version Control](#)**Job runs (1/2)** [Info](#)Last updated (UTC)
December 20, 2023 at 18:27:26[View details](#)[Stop job run](#)[Table View](#)[Card View](#) Filter job runs by property

< 1 >

| Run status | Retries | Start time (UTC) | End time (UTC) | Duration | Capacity ... | Worker type |
|------------|---------|---------------------|---------------------|----------|--------------|-------------|
| Succeeded | 0 | 2023/12/20 18:24:18 | 2023/12/20 18:25:40 | 1 m 12 s | 10 DPUs | G.1X |
| Succeeded | 0 | 2023/12/20 09:34:45 | 2023/12/20 09:36:03 | 1 m 8 s | 10 DPUs | G.1X |

=

[Run details](#)[Input arguments \(10\)](#)[Continuous logs](#)[Run insights](#)[Metrics](#)[Spark UI](#)

| | | | |
|-----------------------------------|------------------------------|--------------|------------------------------|
| Job name | Start time (UTC) | Glue version | Last modified on (UTC) |
| de-on-population-parquet-analysis | December 20, 2023 6:24:18 PM | 4.0 | December 20, 2023 6:25:40 PM |

| | | | |
|---|------------------------------|-------------|----------------|
| Id | End time (UTC) | Worker type | Log group name |
| jr_7f674a6e1eafac6579b1dd6d339dc5c59273b29d50b5dfa437c405272df7e512 | December 20, 2023 6:25:40 PM | G.1X | /aws-glue/jobs |



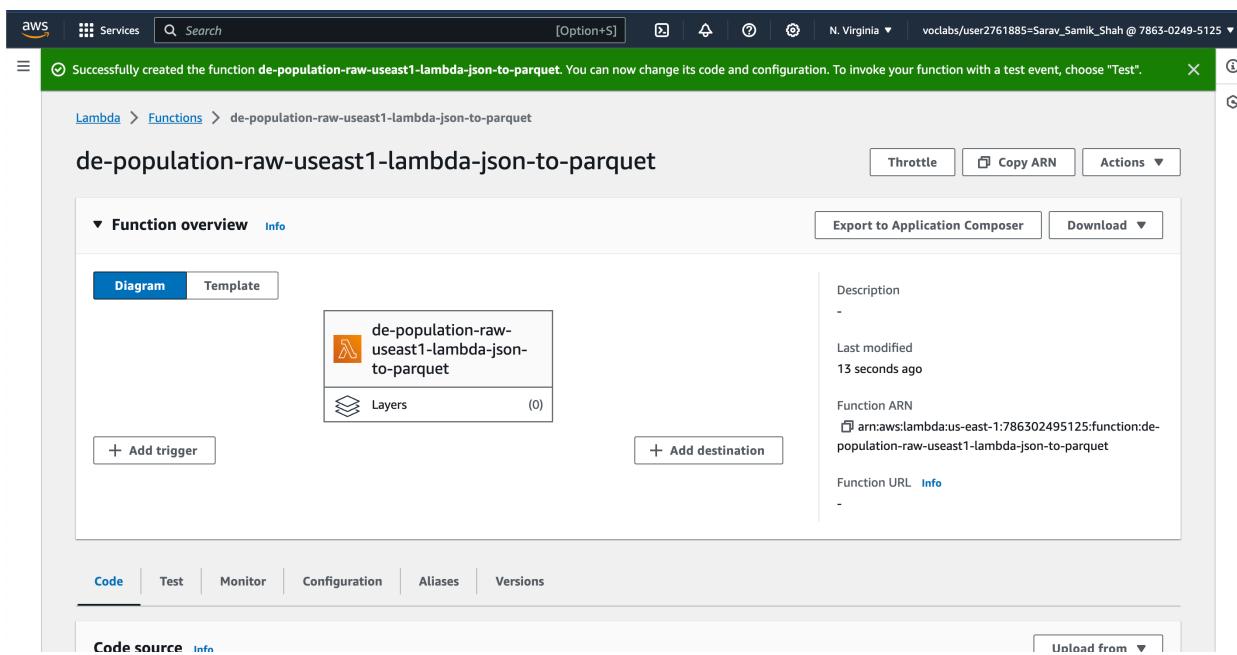
| | | | |
|------------|---------------|--------------|-------------------|
| Run status | Start-up time | Max capacity | Number of workers |
|------------|---------------|--------------|-------------------|

5.3 AWS Lambda

Parquet is a columnar storage format, meaning that it stores data column-wise rather than row-wise.

Parquet files store schema information, which allows for schema evolution. This means that as the schema of your data evolves over time (e.g., new columns are added), Parquet files can handle these changes without requiring modification of the entire dataset. Example: once if you convert the datatype of a column, this format will be saved and all the new files that will be processed be stored with the updated data format.

Hence we convert the JSON file to Parquet file format.



Further we do some data transformation in AWS lambda and stored the cleaned data again in S3 buckets.

Layers enable you to isolate common code and dependencies, making it easier to test and update shared components independently of the function logic.

The code used for data cleaning is given below and we have declared some global variables for better security.

| Code | Test | Monitor | Configuration | Aliases | Versions |
|-----------------------|------|---------|---------------|---------|----------|
| General configuration | | | | | |
| Triggers | | | | | |
| Permissions | | | | | |
| Destinations | | | | | |
| Function URL | | | | | |
| Environment variables | | | | | |
| Tags | | | | | |

The screenshot shows the AWS Lambda Test & Deploy interface. The left sidebar displays the environment variables and the Lambda function code. The main area shows the function code:

```
import awswrangler as wr
import pandas as pd
import urllib.parse
import os

os_input_s3_cleansed_layer = os.environ['s3_cleansed_layer']
os_input_glue_catalog_db_name = os.environ['glue_catalog_db_name']
os_input_glue_catalog_table_name = os.environ['glue_catalog_table_name']
os_input_write_data_operation = os.environ['write_data_operation']

def lambda_handler(event, context):
    # Get the object from the event and show its content type
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')
    try:
        # Creating DF from content
        df_raw = wr.s3.read_json(f's3:///{key}')
    except Exception as e:
        print(e)
        print(f'Error getting object {key} from bucket {bucket}. Make sure they exist and your bucket is in the same region as this function')
        raise e

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

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

The test function that we used is S3-put:

This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional

s3-put

Event JSON

Format JSON

```
1< "requestParameters": {
  2   "sourceIPAddress": "127.0.0.1"
  3 },
  4 "responseElements": {
  5   "x-amz-request-id": "EXAMPLE123456789",
  6   "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEF"
  7 },
  8 "s3": {
  9   "S3SchemaVersion": "1.0",
 10   "configurationId": "testConfigRule",
 11   "bucket": {
 12     "name": "de-on-population-raw-useast1-dev",
 13     "ownerIdentity": {
 14       "principalId": "EXAMPLE"
 15     },
 16     "arn": "arn:aws:s3:::de-on-population-raw-useast1-dev"
 17   },
 18   "object": {
 19     "key": "countries-table.json",
 20     "size": 1024,
 21     "eTag": "0123456789abcdef0123456789abcdef",
 22     "sequencer": "0A1B2C3D4E5F678901"
 23   }
 24 }
 25 }
```

Cancel

Invoke

Save

Layers enable you to isolate common code and dependencies, making it easier to test and update shared components independently of the function logic.

Adding the AWS Lambda layer:

The screenshot shows the 'Layers' section of the AWS Lambda console. A single layer is listed:

| Merge order | Name | Layer version | Compatible runtimes | Compatible architectures | Version ARN |
|-------------|--------------------------|---------------|---------------------|--------------------------|--|
| 1 | AWSDataWrangler-Python38 | 3 | python3.8 | x86_64 | arn:aws:lambda:us-east-1:336392948345:layer:AWSDataWrangler-Python38:3 |

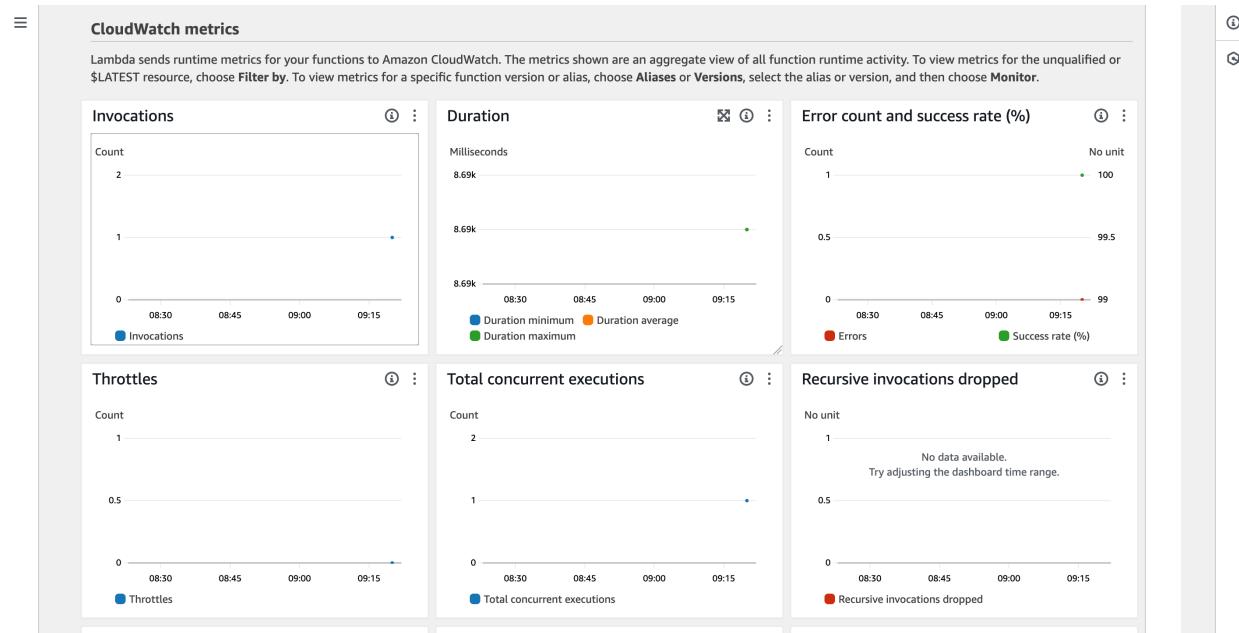
Buttons for 'Edit' and 'Add a layer' are visible at the top right.

Creating a trigger to automate the Lambda function to clean the data each time a new dataset is uploaded to the S3 bucket:

The screenshot shows the 'Add trigger' configuration page for an S3 bucket. The selected bucket is 's3/de-on-population-raw-useast1-dev'. The 'Event types' section is set to 'All object create events'. The 'Prefix - optional' field contains 'e.g. images/' and the 'Suffix - optional' field contains '.json'.

The screenshot shows the Lambda function overview for 'de-population-raw-useast1-lambda-json-to-parquet'. A success message indicates the trigger was added successfully. The 'Function overview' section shows the function's layers and triggers. One trigger is listed: 'S3' with the prefix 'de-on-population-raw-useast1-dev'. The 'Configuration' tab is selected at the bottom.

Whenever a new file / dataset is uploaded in the targeted S3 bucket, the Lambda function gets triggered.



The clean version of the data is also now available in AWS Glue -> Data Catalog -> The database that we specified -> Tables.

The AWS Glue Data Catalog interface shows the following details:

- Database Properties**:
 - Name: de-population-cleaned-db
 - Description: -
 - Location: -
 - Created on (UTC): December 20, 2023 at 02:46:43
- Tables (1)**:
 - Table name: cleaned_population_
 - Database: de-population-cleaned-db
 - Type: Parquet
 - Last updated (UTC): December 20, 2023 at 02:48:56

We can see here the structured version of our data is now created:

The screenshot shows the AWS Glue Schema view. On the left, there's a sidebar with navigation links for Getting started, ETL jobs, Visual ETL, Notebooks, Job run monitoring, Data Catalog tables, Data connections, Workflows (orchestration), Data Catalog (Tables, Stream schema registries, Schemas, Connections, Crawlers, Classifiers, Catalog settings), Data Integration and ETL, Legacy pages, and links for What's New, Documentation, and AWS Marketplace. There are also checkboxes for Enable compact mode and Enable new navigation.

The main area is titled "Schema (19)" and contains a table with 19 rows. The columns are: #, Column name, Data type, Partition key, and Comment. The data is as follows:

| # | Column name | Data type | Partition key | Comment |
|----|-----------------|-----------|---------------|---------|
| 1 | place | bigint | - | - |
| 2 | pop1980 | double | - | - |
| 3 | pop2000 | double | - | - |
| 4 | pop2010 | double | - | - |
| 5 | pop2022 | double | - | - |
| 6 | pop2023 | double | - | - |
| 7 | pop2050 | double | - | - |
| 8 | pop2050 | double | - | - |
| 9 | country | string | - | - |
| 10 | area | double | - | - |
| 11 | landareakm | double | - | - |
| 12 | cca2 | string | - | - |
| 13 | cca3 | string | - | - |
| 14 | netchange | double | - | - |
| 15 | growthrate | double | - | - |
| 16 | worldpercentage | double | - | - |
| 17 | density | double | - | - |
| 18 | densitymi | double | - | - |
| 19 | rank | bigint | - | - |

5.4 AWS Athena

Amazon Athena is an interactive query service that enables querying data directly from Amazon S3 using SQL. It is a serverless and fully managed service, allowing users to analyze and derive insights from data stored in S3 without the need to set up and manage complex infrastructure.

Here's a list of the databases and tables available in AWS glue:

AWS Glue > Databases

Databases (4)

Last updated (UTC)
December 20, 2023 at 15:32:36

| <input type="checkbox"/> | Name | Description | Location URI | Created on (UTC) |
|--------------------------|--|-------------|--------------|-------------------------------|
| <input type="checkbox"/> | db_population_analytics | - | - | December 20, 2023 at 09:33:35 |
| <input type="checkbox"/> | de-population-cleaned-db | - | - | December 20, 2023 at 02:46:43 |
| <input type="checkbox"/> | de-population-raw-db | - | - | December 19, 2023 at 09:18:29 |
| <input type="checkbox"/> | default | - | - | December 20, 2023 at 09:33:35 |

aws Services Search [Option+S] N. Virginia v vocabs/user2761885=Sarav_Samik_Shah @ 7865-0249-5125 ▾

AWS Glue > Databases > de-population-raw-db

de-population-raw-db

Database properties

| Name | Description | Location | Created on (UTC) |
|----------------------|-------------|----------|-------------------------------|
| de-population-raw-db | - | - | December 19, 2023 at 09:18:29 |

Tables (3)

Last updated (UTC)
December 20, 2023 at 08:45:38

| <input type="checkbox"/> | Name | Database | Location | Classification | Deprecated | View data | Data quality |
|--------------------------|---|----------------------|-------------------------|----------------|------------|------------|-------------------|
| <input type="checkbox"/> | cleaned_population_reference_data | de-population-raw-db | s3://de-on-population-i | Parquet | - | Table data | View data quality |
| <input type="checkbox"/> | de_on_population_raw_useast1_dev | de-population-raw-db | s3://de-on-population-i | JSON | - | Table data | View data quality |
| <input type="checkbox"/> | de_on_population_semistructured_useast1_dev | de-population-raw-db | s3://de-on-population-i | CSV | - | Table data | View data quality |

We can now query on the data using Amazon Athena.

Query:

Amazon Athena > Query editor

Editor | Recent queries | Saved queries | Settings

Workgroup primary

Data | **Query 1** | **Query 4** | **Query 5** | **Query 6**

1 `SELECT * FROM "AwsDataCatalog"."de-population-cleaned-db"."cleaned_population_reference_data" limit 10;`

Data source: AwsDataCatalog

Result:

Query results | Query stats

Completed | Time in queue: 100 ms | Run time: 635 ms | Data scanned: 30.59 KB

Results (10)

Copy | Download results

| # | place | pop1980 | pop2000 | pop2010 | pop2022 | pop2023 | pop2030 | pop2050 | country | area | landareakm |
|---|-------|--------------|---------------|---------------|---------------|---------------|----------------------|---------------|---------------|-------------|------------|
| 1 | 356 | 6.96828385E8 | 1.059633675E9 | 1.24061362E9 | 1.417173173E9 | 1.428627663E9 | 1.51499408E9 | 1.670490596E9 | India | 3287590.0 | 2973190.0 |
| 2 | 156 | 9.82372466E8 | 1.26409069E9 | 1.348191368E9 | 1.425887337E9 | 1.425671352E9 | 1.415605906E9 | 1.312636325E9 | China | 9706961.0 | 9424702.9 |
| 3 | 840 | 2.23140018E8 | 2.82398554E8 | 3.11182845E8 | 3.382885768 | 3.39996563E8 | 3.52162301E8 | 3.75391963E8 | United States | 9372610.0 | 9147420.0 |
| 4 | 360 | 1.48177096E8 | 2.14072421E8 | 2.44016173E8 | 2.75501339E8 | 2.77534122E8 | 2.921501E8 | 3.17225213E8 | Indonesia | 1904569.0 | 1877519.0 |
| 5 | 586 | 8.0624057E7 | 1.54369924E8 | 1.94454498E8 | 2.35824862E8 | 2.40485658E8 | 2.74029836E8 | 3.67808468E8 | Pakistan | 881912.0 | 770880.0 |
| 6 | 566 | 7.2951439E7 | 1.22851984E8 | 1.60952853E8 | 2.18541212E8 | 2.23804632E8 | 2.6258042599999997E8 | 3.77459883E8 | Nigeria | 923768.0 | 910770.0 |
| 7 | 76 | 1.22288383E8 | 1.7587372E8 | 1.96353492E8 | 2.16422446E8 | 2.23908968E8 | 2.30885725E8 | 2.3908968E8 | Brazil | 8515767.0 | 8358140.0 |
| 8 | 50 | 8.3929765E7 | 1.29193327E8 | 1.49391139E8 | 1.71186372E8 | 1.72954319E8 | 1.8424144E8 | 2.039049E8 | Bangladesh | 147570.0 | 130170.0 |
| 9 | 643 | 1.3825742E8 | 1.46844839E8 | 1.43242599E8 | 1.44713314E8 | 1.44444359E8 | 1.41432741E8 | 1.33133035E8 | Russia | 1.7098242E7 | 1.637687E7 |

Query:

Data | **Query 1** | **Query 4** | **Query 5** | **Query 6**

1 `SELECT *`
2 `FROM "AwsDataCatalog"."de-population-cleaned-db"."cleaned_population_reference_data"`
3 `where country = 'India'`

Data source: AwsDataCatalog

Database: de-population-cleaned-db

Tables and views | Create | **Tables (1)**

Result:

Results (1)

Copy | Download results

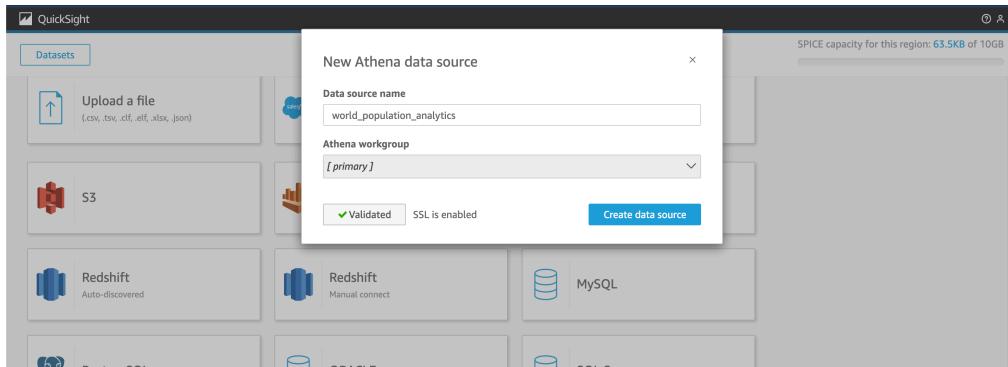
Search rows

| # | place | pop1980 | pop2000 | pop2010 | pop2022 | pop2023 | pop2030 | pop2050 | country |
|---|-------|--------------|---------------|--------------|---------------|---------------|--------------|---------------|---------|
| 1 | 356 | 6.96828385E8 | 1.059633675E9 | 1.24061362E9 | 1.417173173E9 | 1.428627663E9 | 1.51499408E9 | 1.670490596E9 | India |

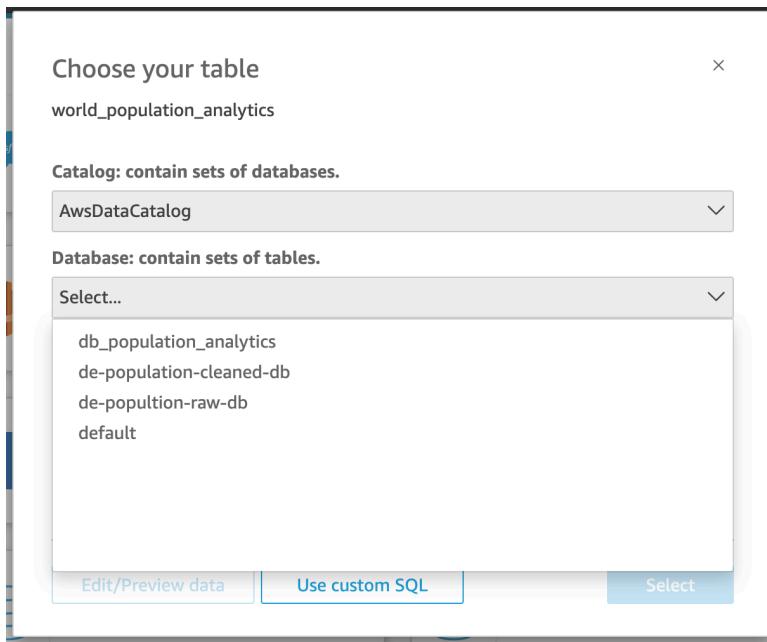
5.5 AWS QuickSight

Athena can be integrated with AWS QuickSight, allowing us to create dashboards and reports based on real-time or historical data stored in Amazon S3.

Integrating Athena as a data source for QuickSight:



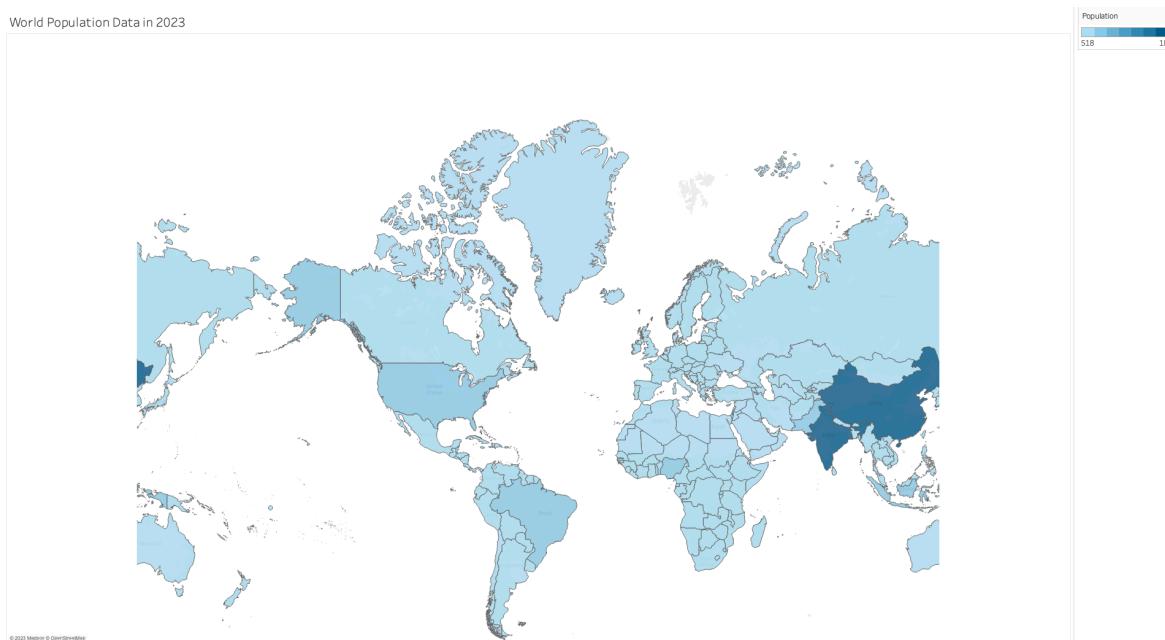
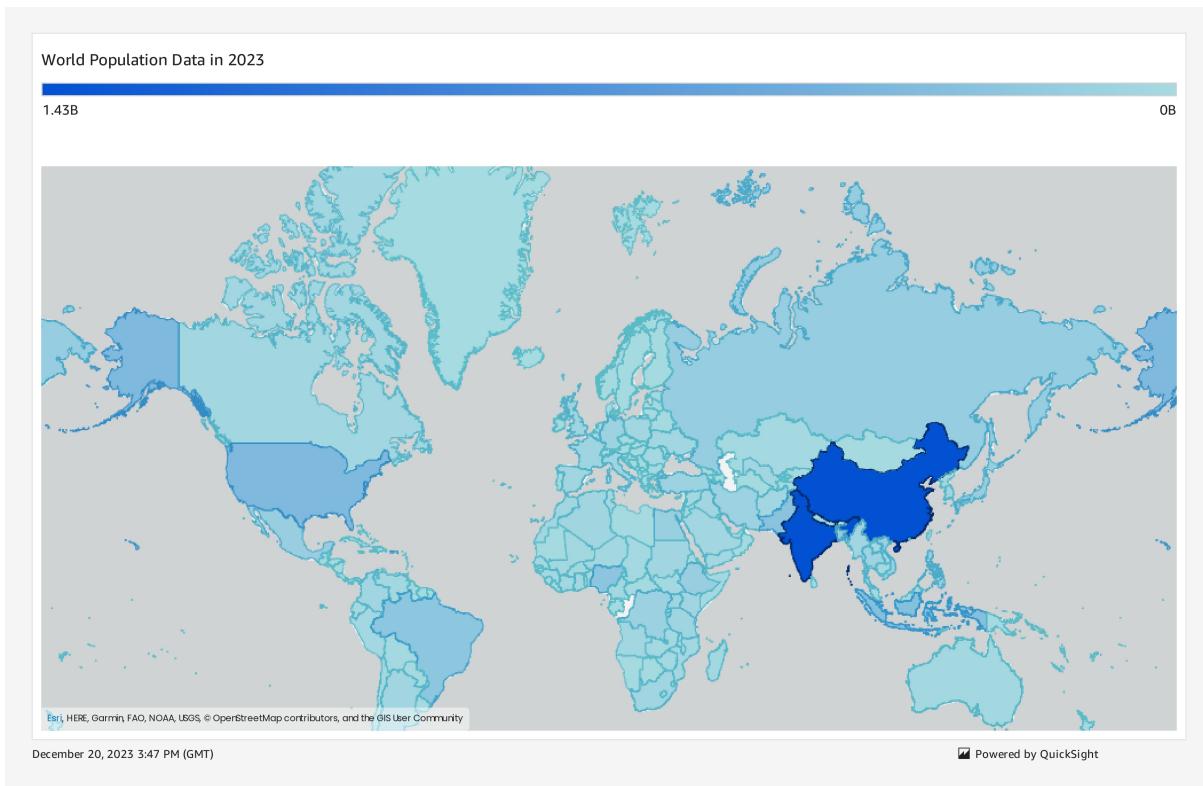
Here we get a list of all the databases and the data tables:



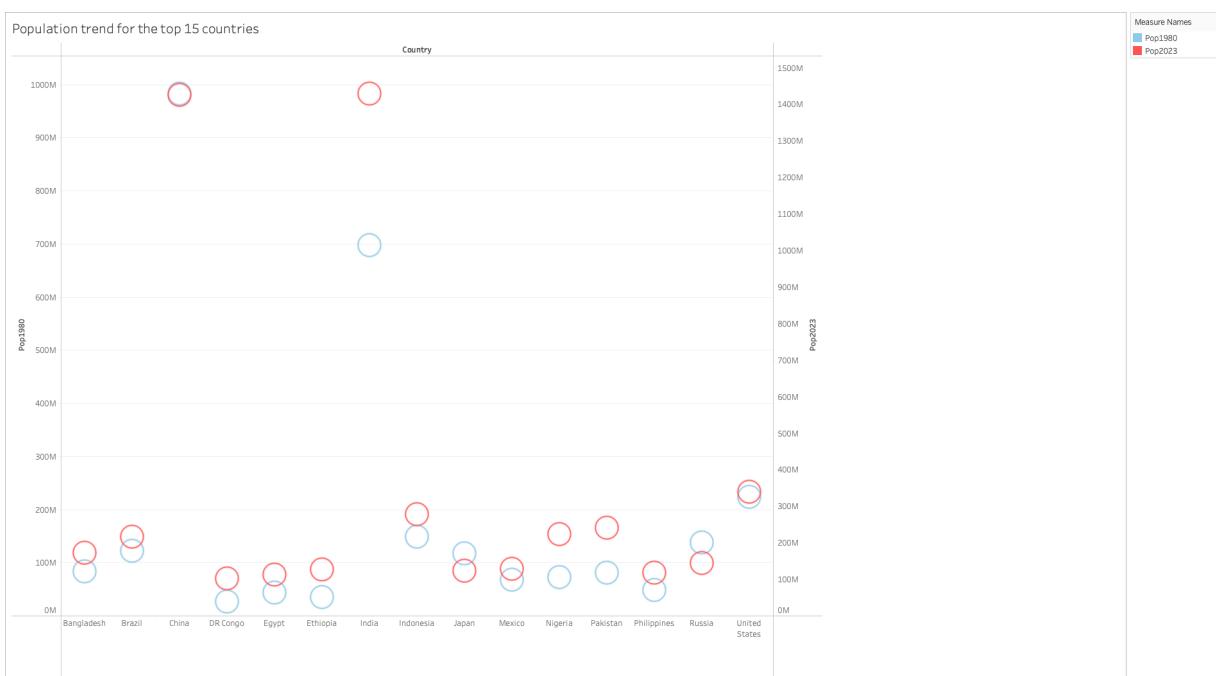
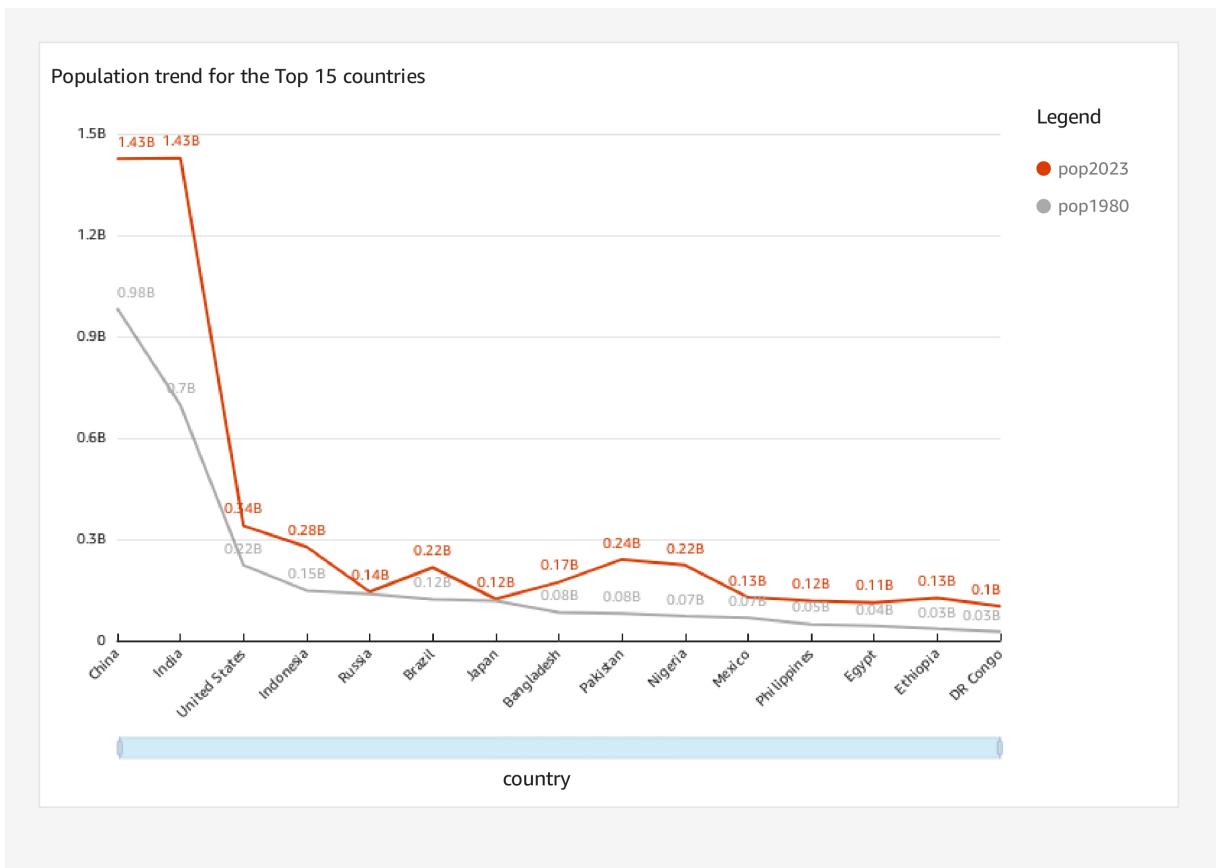
6. RESULTS

Comparing Visualization in QuickSight vs Tableau

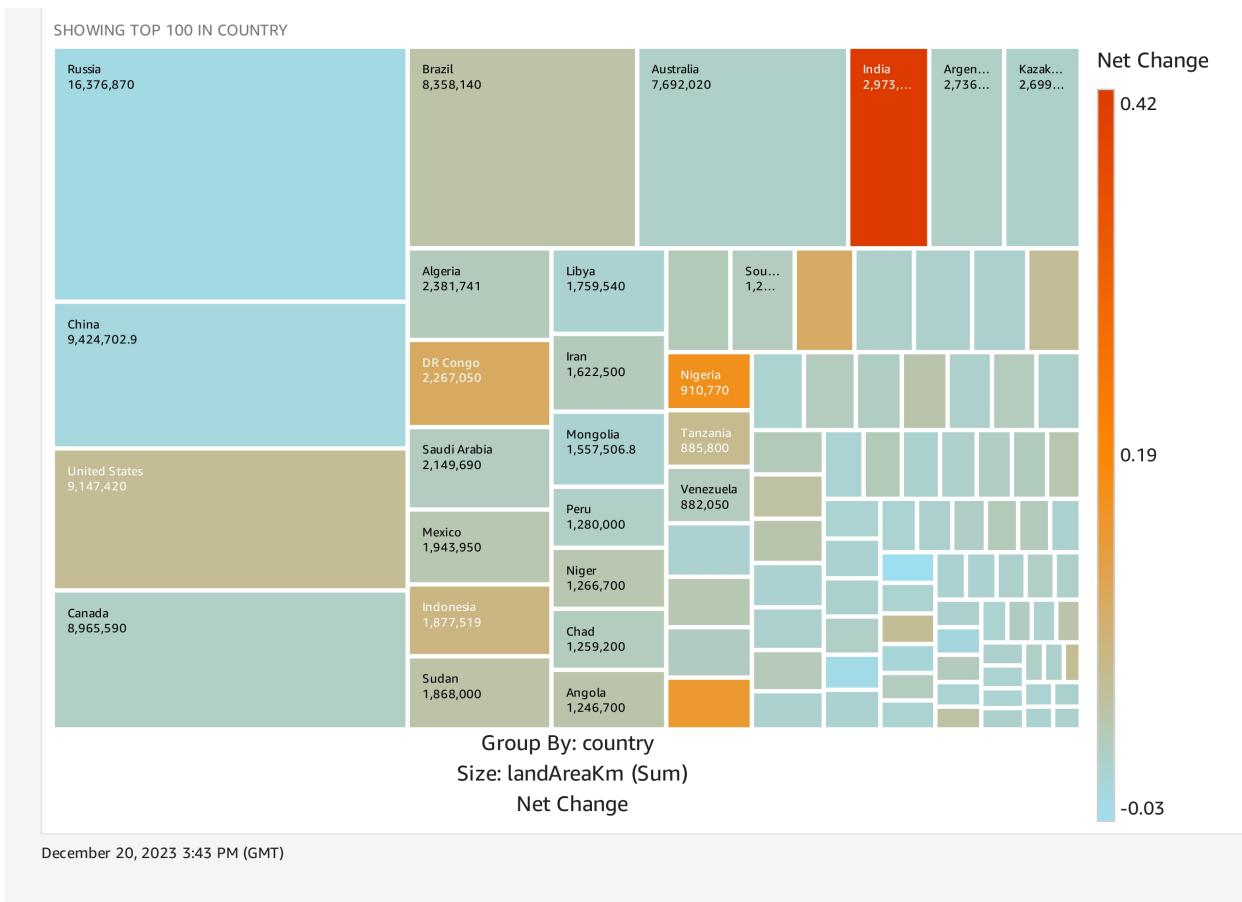
Representation of Population for all the countries in 2023 based on Longitude (generated) and Latitude (generated).



Population trend for the Top 15 countries having the highest population. The graph demonstrates a comparison between the population in 1908 and 2023.



This graph aims to compare the Net change in population for the top 100 countries vs the Land area under the country using AWS QuickSight.



7. CONCLUSION

7.1 ETL Pipeline Performance:

Our evaluation of the ETL pipeline revealed commendable performance in handling diverse data sources and processing volumes. The seamless integration of AWS services, from S3 storage to AWS Glue ETL jobs, contributed to efficient data processing. Athena's SQL-based querying proved invaluable for ad-hoc analyses, demonstrating the agility of the pipeline.

7.2 QuickSight Visualization:

Amazon QuickSight emerged as a powerful tool for data visualization. Its integration with AWS services allowed for the creation of interactive dashboards, providing a user-friendly interface for end-users. The incorporation of machine learning insights, such as anomaly detection and forecasting, added depth to the visualization experience.

The ETL pipeline and QuickSight implementation demonstrated a tangible impact on business decision-making. The accessibility of actionable insights, reduction in data processing times, and the ability to create real-time dashboards contributed to informed decision-making within the organization.

7.3 Comparative Analysis with Tableau:

- QuickSight seamlessly integrates with the broader AWS ecosystem, offering native connectivity to various AWS data sources.
- AWS QuickSight operates on a serverless architecture, eliminating the need for infrastructure management and offers pay-as-you-go pricing.
- QuickSight offers fast query performance even with large datasets. It leverages a cloud architecture, distributed processing, enabling users to analyze and visualize data with low latency.
- QuickSight incorporates machine learning (ML) insights. Hence any user on the business side with the basic knowledge of ML can enhance the depth of analysis without requiring to be ML experts.