GROUP 3:
STORE AND ORGANIZE DATA

Wyatt Priddy

Suraj Kunthu

Mohammad Atif Siddiqui

DS 5110 | Spring 2024

UVA | SCHOOL of DATA SCIENCE

Introduction
Use Case #2: Store and Organize Data

- Data Engineering focused project
  - Setting up ETL pipeline for data scientists
  - Compare query processes for optimal performance when working with Big Data sets

Importance of Storage And Organization

Improved Performance:

- ***Faster query execution:*** *Organized data with proper partitioning and indexing allows data processing tools to locate relevant information quickly, leading to faster query execution times.*
- ***Reduced processing overhead:*** *Compressed data files require less storage space, which translates to less data to transfer and process during data pipelines. This reduces the overall processing overhead and improves efficiency.*

Enhanced Scalability:

- ***Easier data management:*** *Organized data with clear structure facilitates adding new data or scaling existing datasets efficiently. This makes it easier to handle growing data volumes without performance degradation.*
- ***Cost-effective scaling:*** *Lower storage requirements from compression allow for scaling your data processing pipelines with minimal impact on storage costs.*

Reduced Storage Costs:

- ***Efficient storage utilization:*** *Organized data often eliminates redundancy and allows for optimized storage formats, significantly reducing the amount of physical storage needed.*
- ***Compression benefits:*** *Compressed data files occupy less space, leading to lower storage costs on AWS services like S3. This is especially beneficial for large datasets.*

Improved Data Quality:

- ***Reduced errors:*** *Organized data structures help mitigate errors during data ingestion and processing due to inconsistencies or ambiguity.*
- ***Data validation:*** *Organizing data allows for easier implementation of data validation checks to ensure the integrity and accuracy of your data before processing.*

The Data

- Court records data available from across Virginia
- Formed from existing real world court data:
  - Hope to predict outcome of misdemeanor and felony court cases
  - General district court
  - Utilizing case type and demographics surrounding the defendant
- subsection of the data 2010 – 2020
- ~93 different files Merged together used for feature Engineering .

https://virginiacourtdata.org/#download

The Data

- 7,196,322 data points
- 11 variables
- CSV format

Variable	Description	Data Type
Final Disposition	The outcome of the court case.	str
Court	The court system within Virginia where the case was heard.	str
Complanaint	The arrest officers in the case.	str
Public Defender	Whether a public defender was assigned to the case	bool
Gender	Gender of the defendant	str
Race	Race of the defendant	str
Case Type	Indication of whether the case was a mideameanor or felony	str
Class	The level of misdemeanor or felony committed	str
CodeSection	A reference to the law that was violated — can be state or local law	str
ChargeAmended	Whether the original charge was amended	bool
SentenceTime	The duration of a jail or prison sentence from the outcome of the case	int

Data Preprocessing

Feature Engineering Highlights:

Public Defender Variable:

Manipulation of free-form text entry of the *Defense Attorney* field in the raw data to discern whether a defendant was assigned a public defender.

Example Entries: "*Public Defender", "Public Def", "P Def", "Pub Def", "PD"*

Court Variable:

Distinction by court was given in the Federal Information Processing Standard (FIPS) format. Supplemental data from the Federal Communication Commission was used to map FIPS code to court name

Example Conversion: *"51540" -> "Charlottesville City"*

Data Preprocessing

Data Cleaning Highlights:

Final Disposition:

The response variable for a potential classification model had ~89k records with null values representing 1.2% of the dataset. These values were removed from the dataset.

SentenceTime:

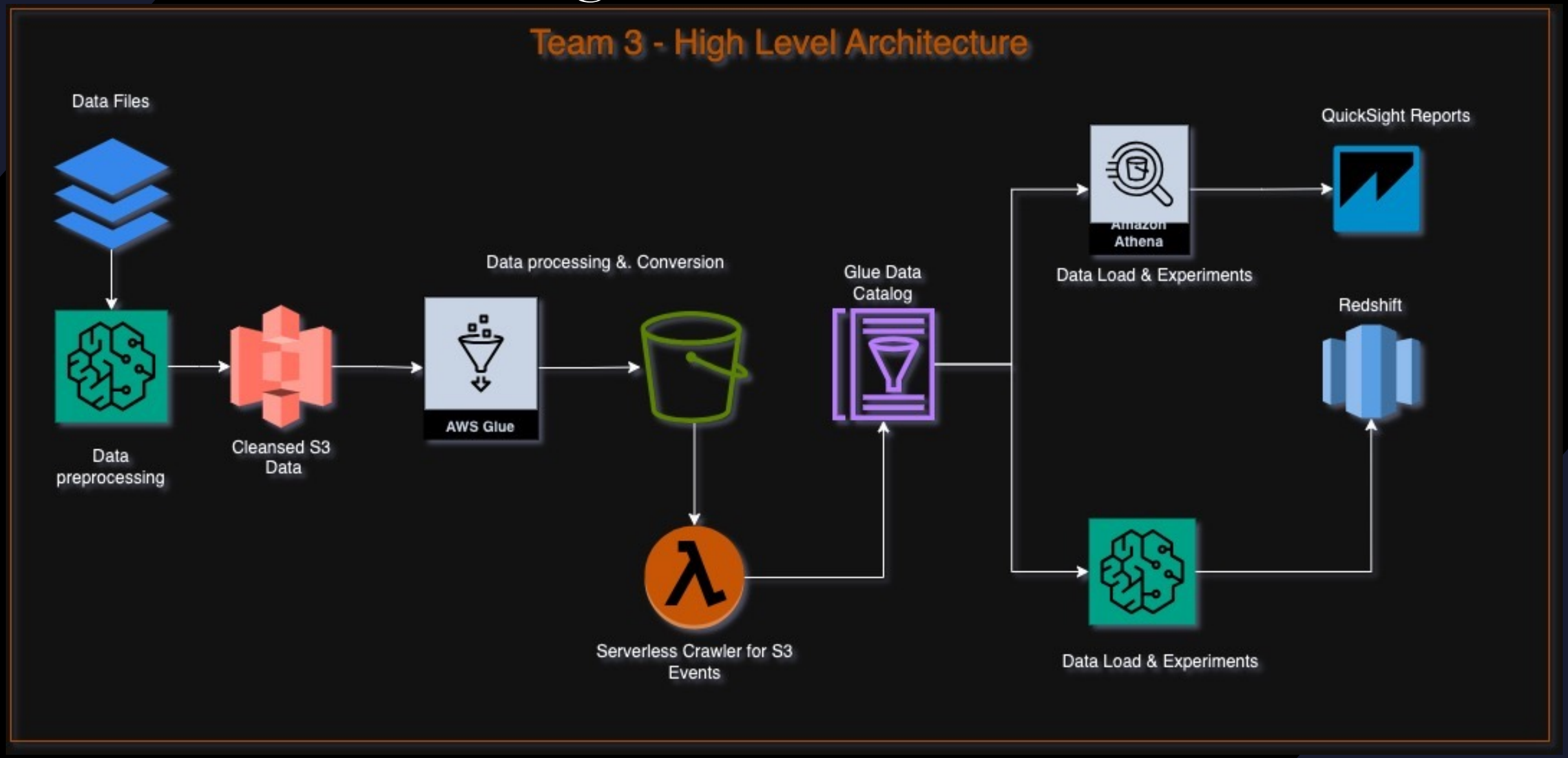
The response variable for a regression model had ~447k records with null values representing 6.2% of the dataset. 0 values were input to represent no sentence time for the case.

Charge Class:

The *Class* feature had numerous entries (~2.4 million) with null values. It is not always applicable that a charge has a class therefore a value of *"None"* was inputed for these values.

DATA PROCESSING

Data Processing



Processing

Data Preprocessing	SageMaker / Notebooks	Clean and Merge source files
Data Processing	Glue	Job to Compress and Load data into different File formats
	Glue Crawlers	Schema definition for all different formats
	Lambda	Execute Automated crawlers whenever S3 event is created
	Athena	Read data from S3 and Glue Catalog
	Redshift Spectrum	Read data from S3 as external objects
	Redshift	Load data into different formats for experimentation
	SageMaker / Notebooks	Experimental Analysis
	Quicksight	Dashboard for Result visualization

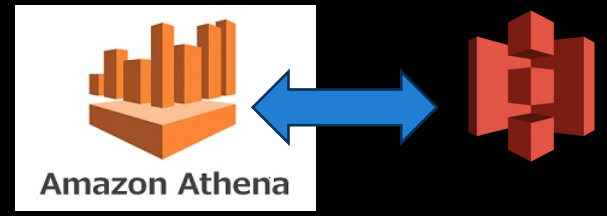
Experimental Design

- Measure Performance of big data reports based on the different file and compression formats supported by AWS

Format	File extension (optional)	Extensions for compressed files
Comma-separated values	.csv	.gz .snappy .lz4 .bz2 .deflate
Microsoft Excel workbook	.xlsx	No compression support
JSON (JSON document and JSON lines)	.json, .jsonl	.gz .snappy .lz4 .bz2 .deflate
Apache ORC	.orc	.zlib .snappy
Apache Parquet	.parquet	.gz .snappy .lz4

https://docs.aws.amazon.com/databrew/latest/dg/supported-data-file-sources.html

Experimentation



- For tables created, execution time was retrieved from CloudWatch Logs
- Leveraged Athena to query different compression formats in S3 buckets
  - 3 READ queries performed of varying sizes, 3 times each:
    - 10 records <-- (ex: `head` or `tail` of data)
    - 10,000 records <-- (ex: sampling)
    - All records <-- (ex: full data query)
  - Query times (seconds) and Data Scanned (MB) was recorded
- Data was then averaged, exported as a .csv, and loaded onto QuickSight for visualization

Results Compression – S3

Source Data Size after Data cleaning and preparation	Data Format	Compression	Comp S3 Size	Redshift Size
1 GB	csv	gzip	63 MB	388 MB
	csv	snappy	129 MB	360MB
	json	gzip	78 MB	224MB
	json	snappy	80 MB	224MB
	orc	snappy	54.5 MB	210MB
	orc	zlib	41.1. MB	224MB
	parquet	Gzip	38.7 MB	224MB
	parquet	lz4	50.4 MB	--
	parquet	lzo	49.9 MB	--
	parquet	snappy	50.3. MB	220MB
	parquet	zstd	39.1 MB	210MB

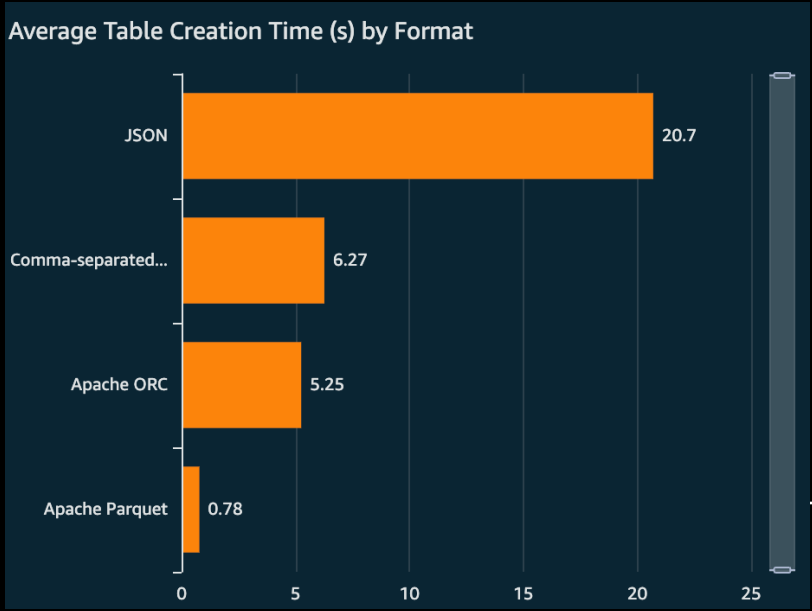
*** Lz compression is not supported with Athena for Csv and Json*
*** Redshift Deflates the data and then stores it in its cluster – Until Unless column level compression is defined.*

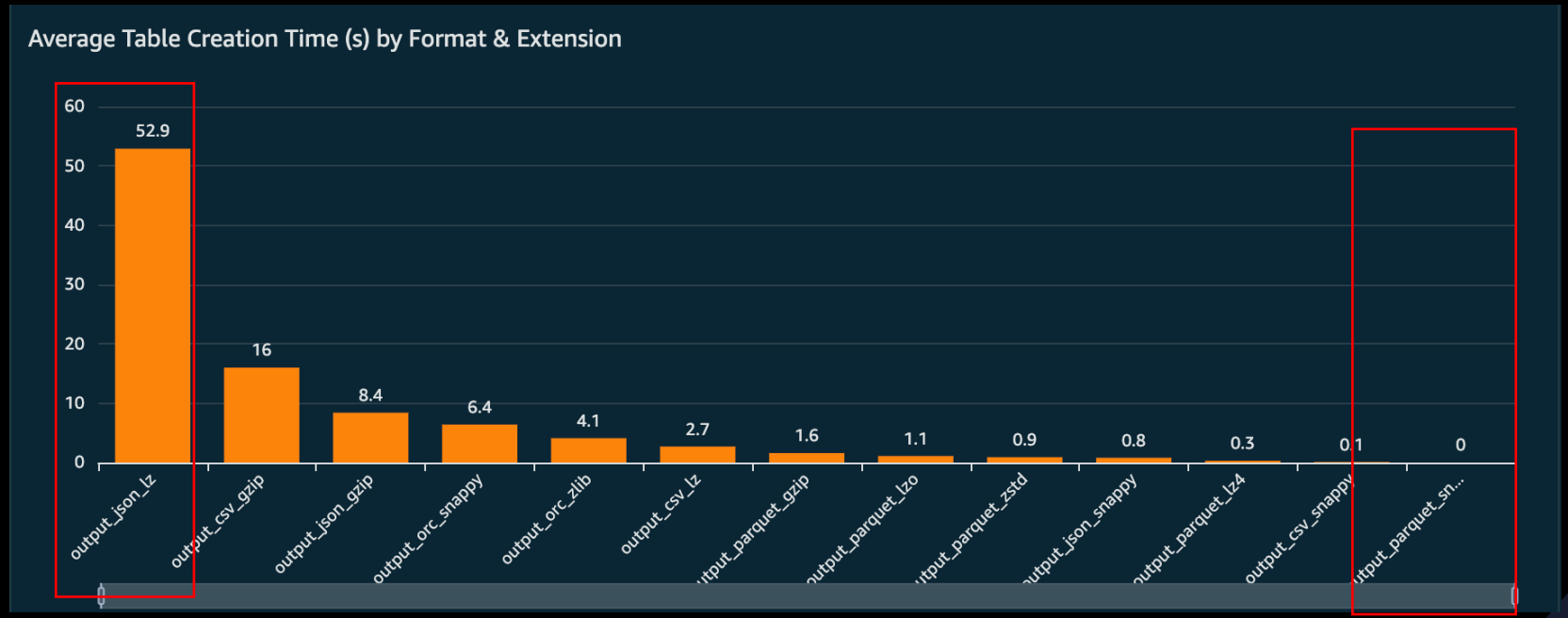
Results

- Table Creation

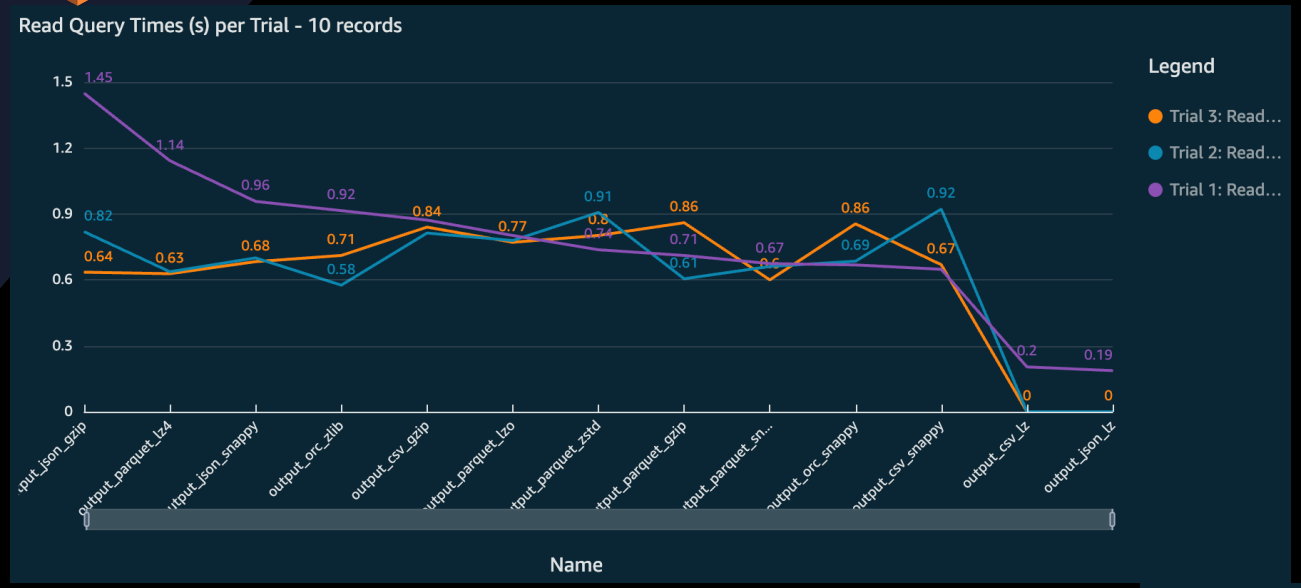
Count of Records by Format and Name

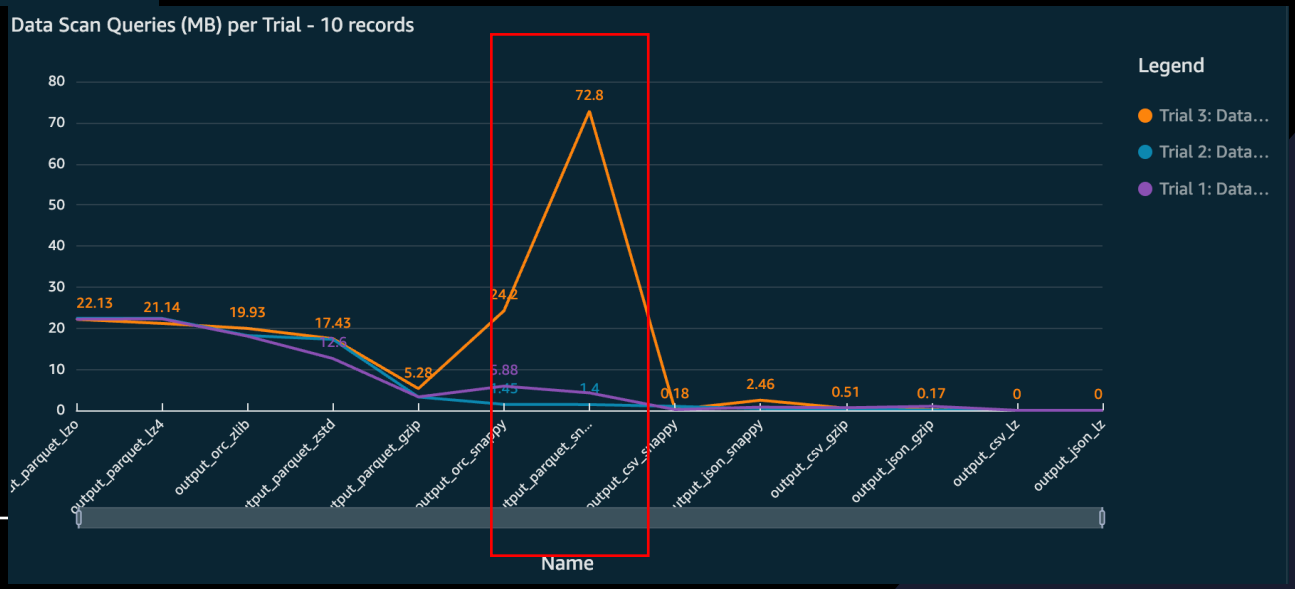
Rows	Count
⊞ Apache ORC	2
⊞ Apache Parquet	5
⊞ Comma-separated values	3
⊞ JSON	3



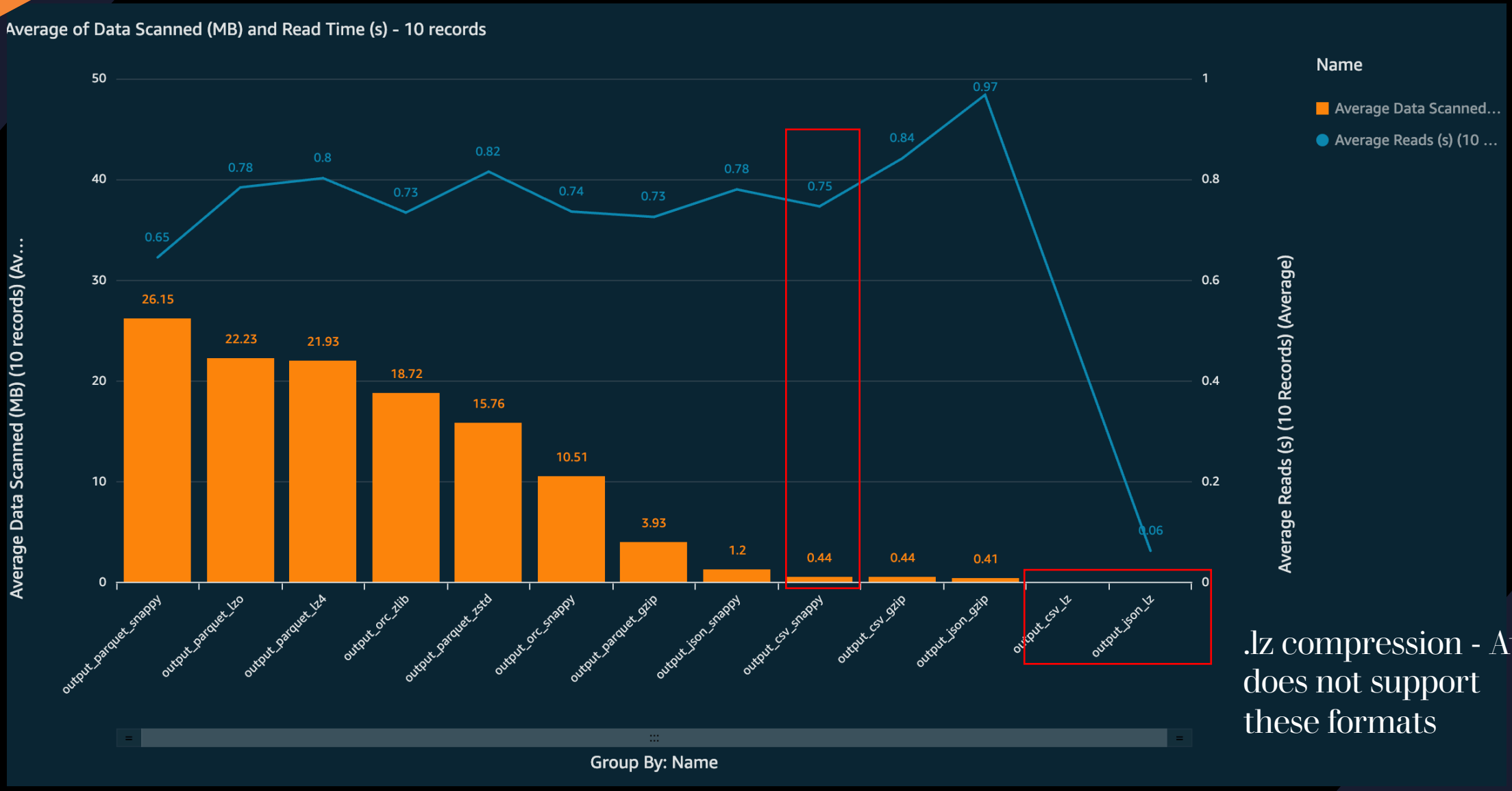


Results – 10 records



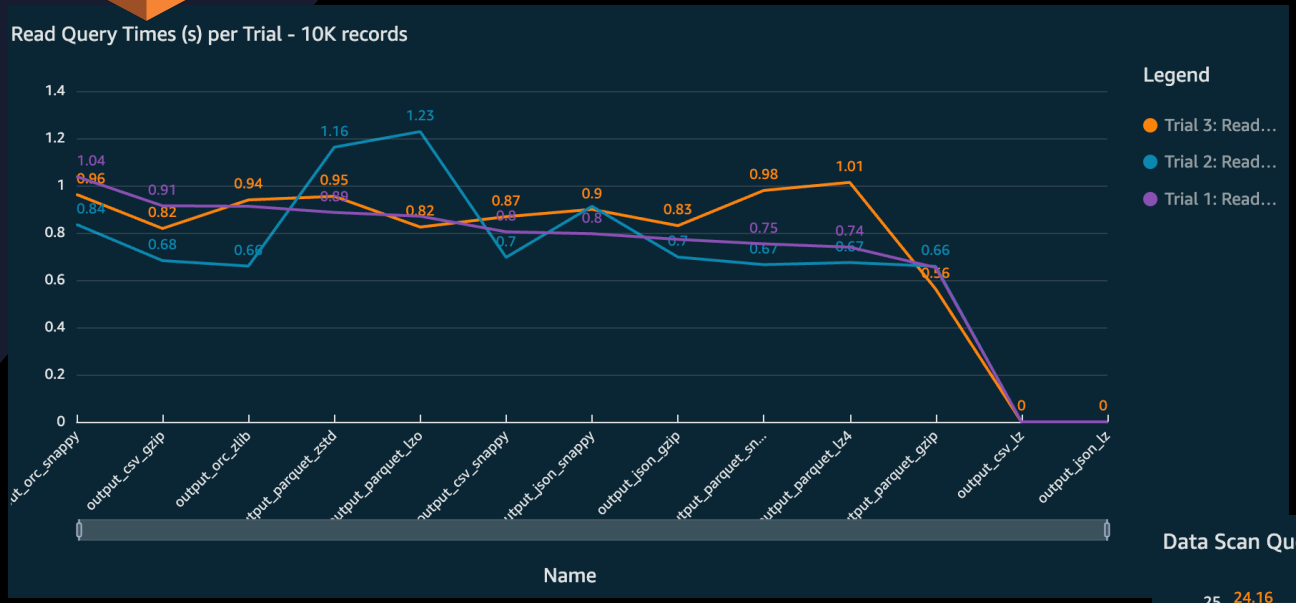


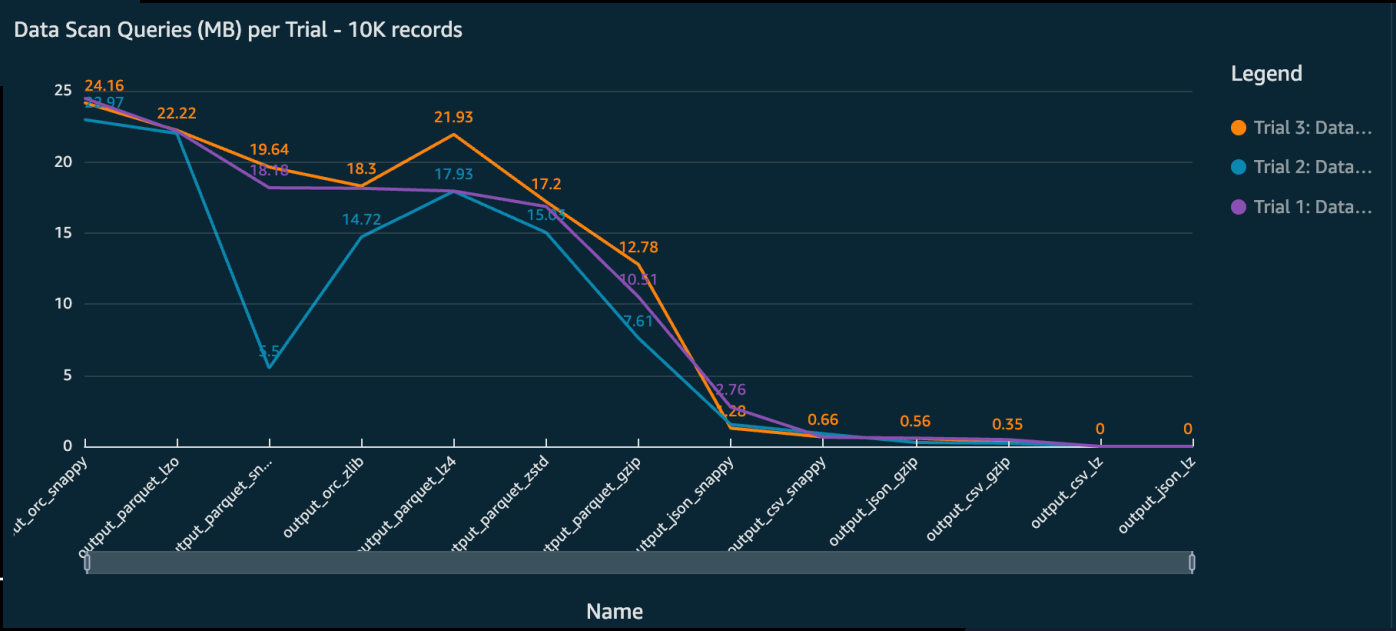
Results – 10 records



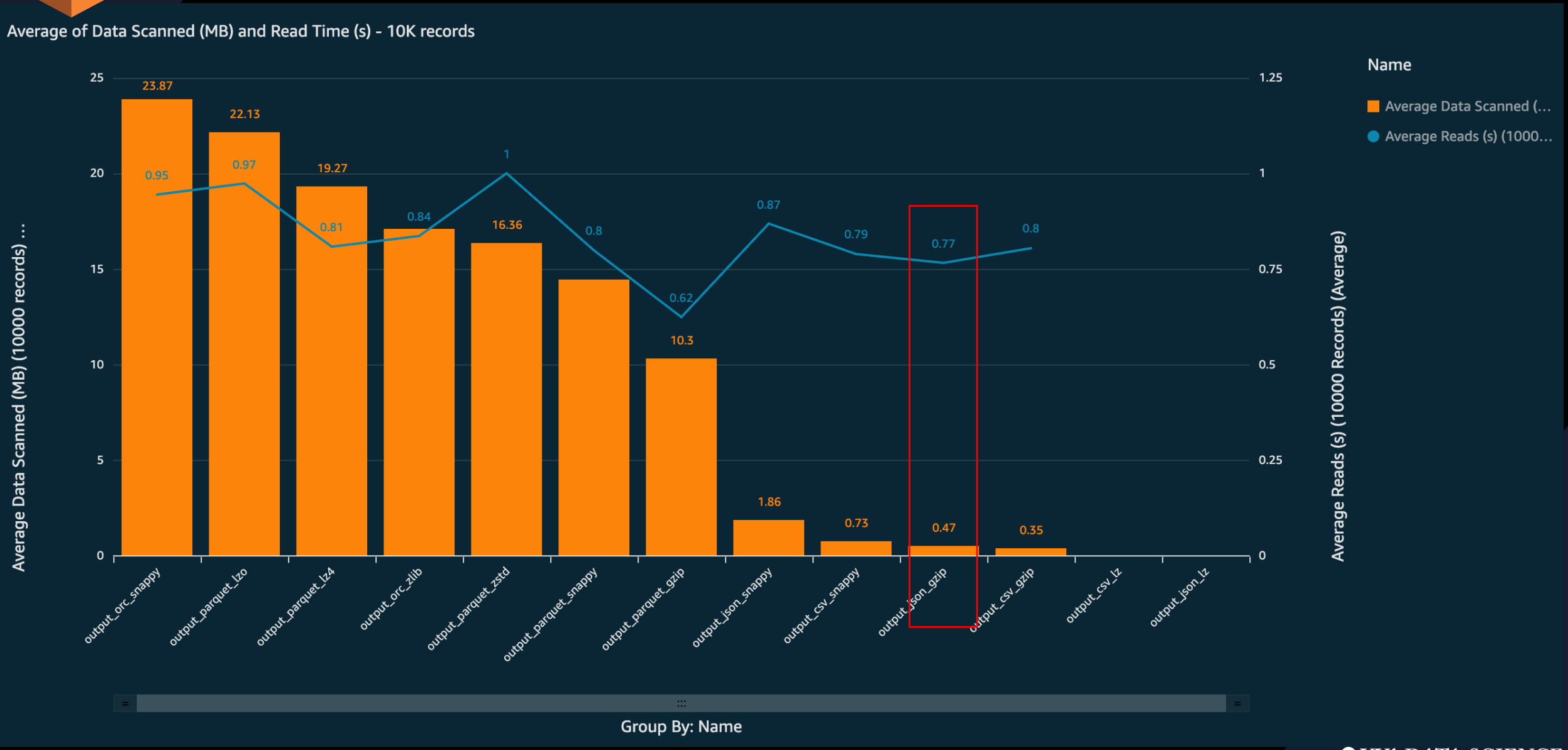
.lz compression - Athena does not support these formats

Results – 10000 records

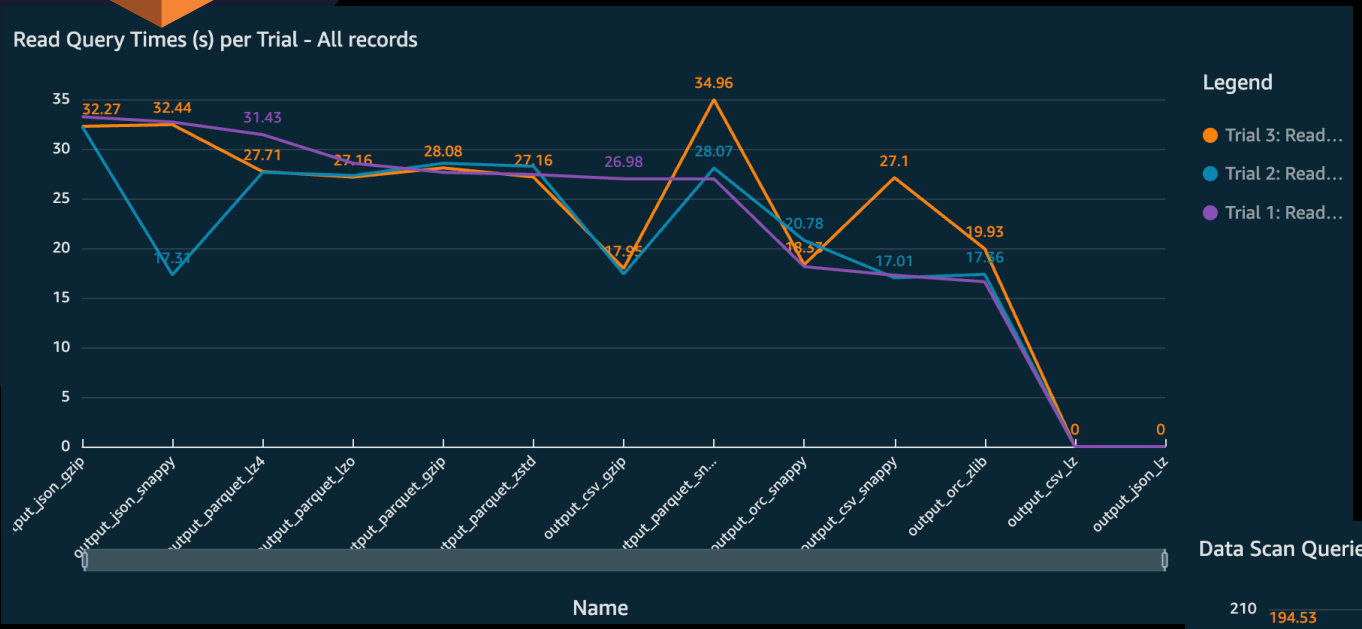


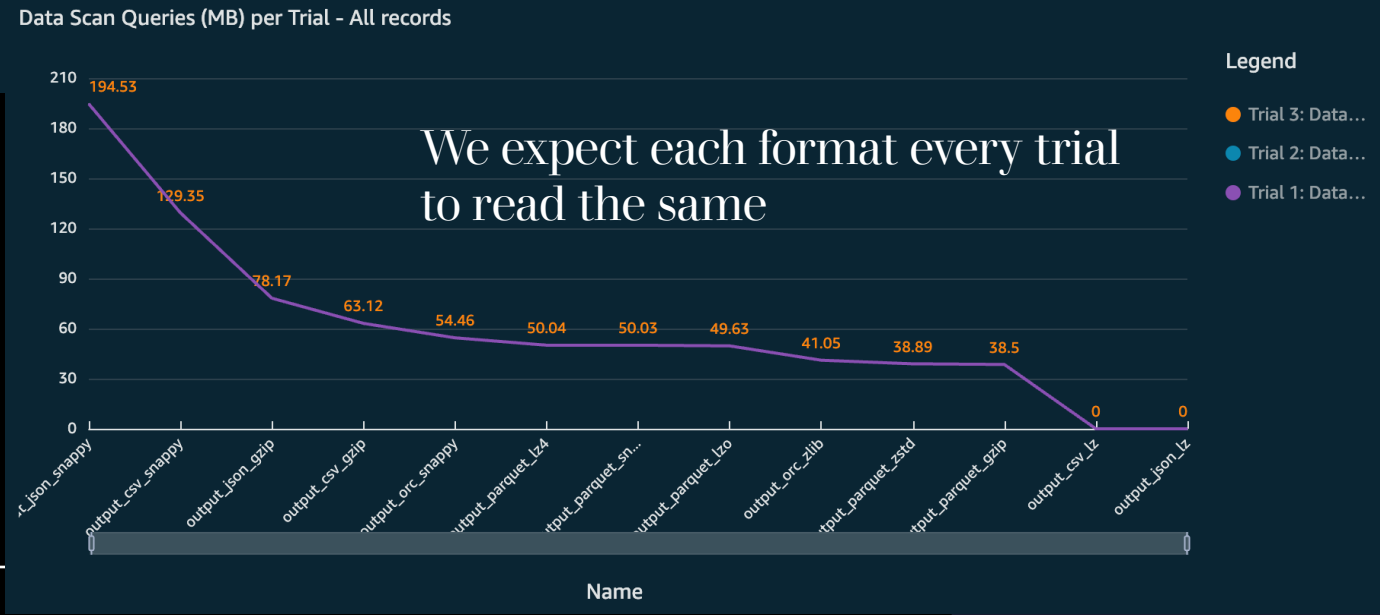


Results – 10000 records



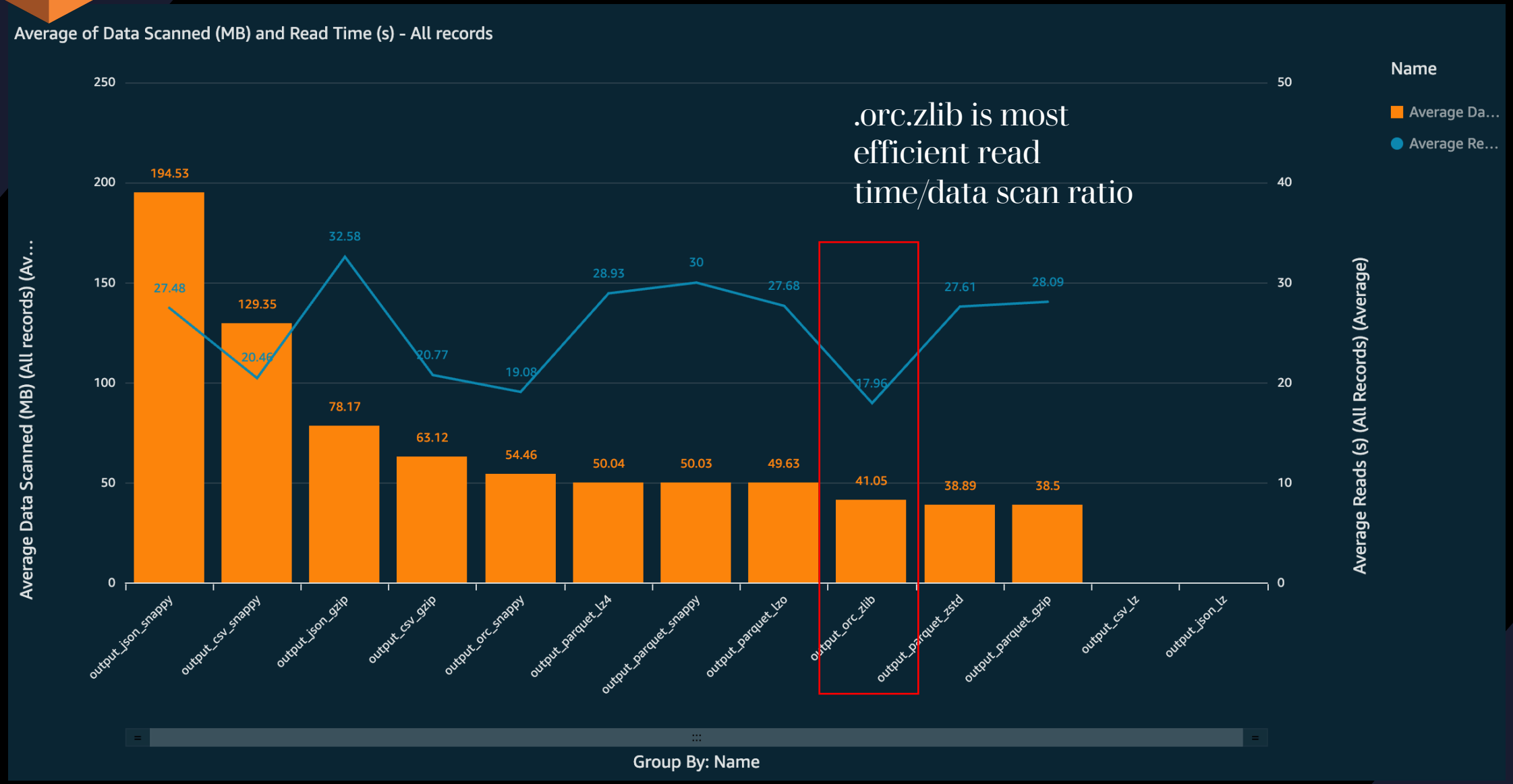
Results – All records





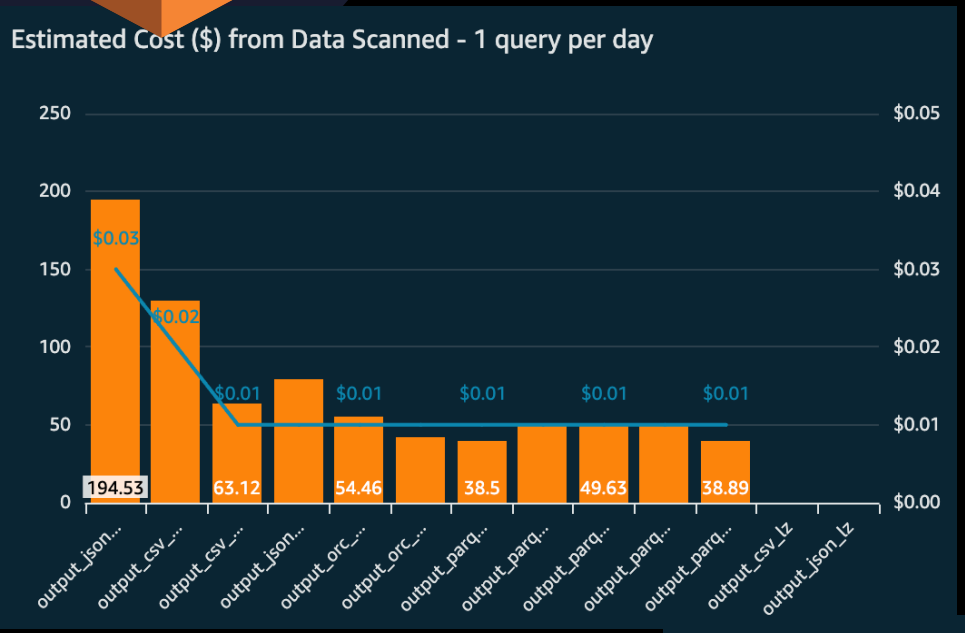
We expect each format every trial to read the same

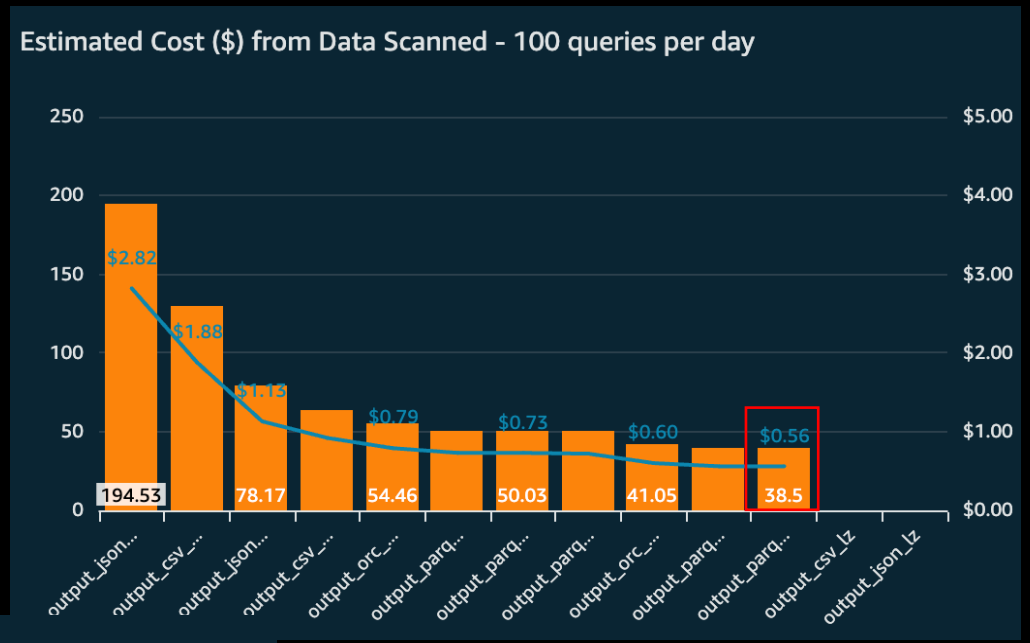
Results – All records

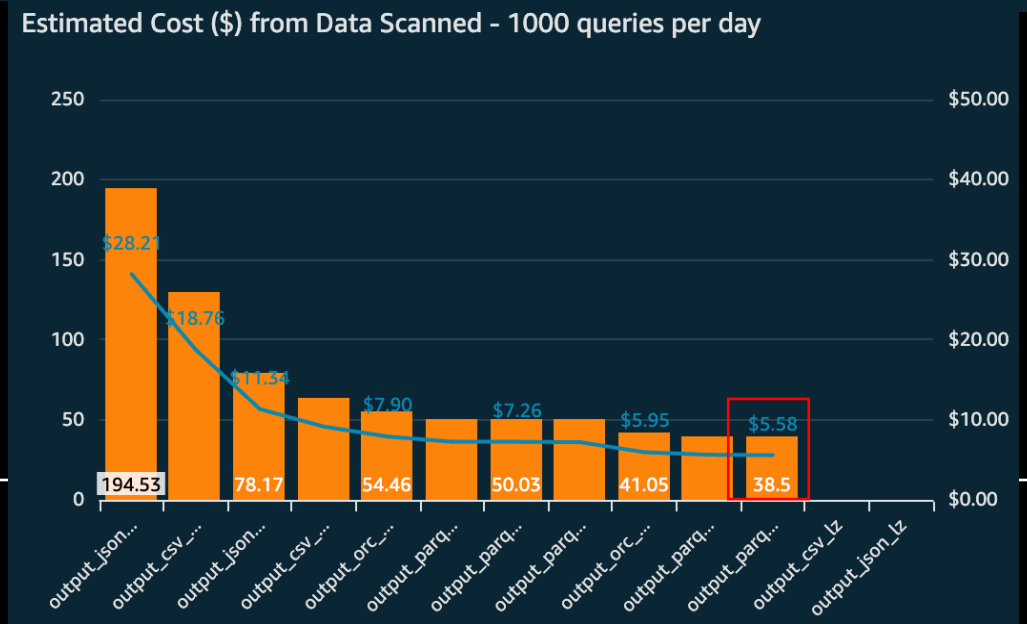


Results – Cost Estimation

Obtained from AWS Cost Estimator

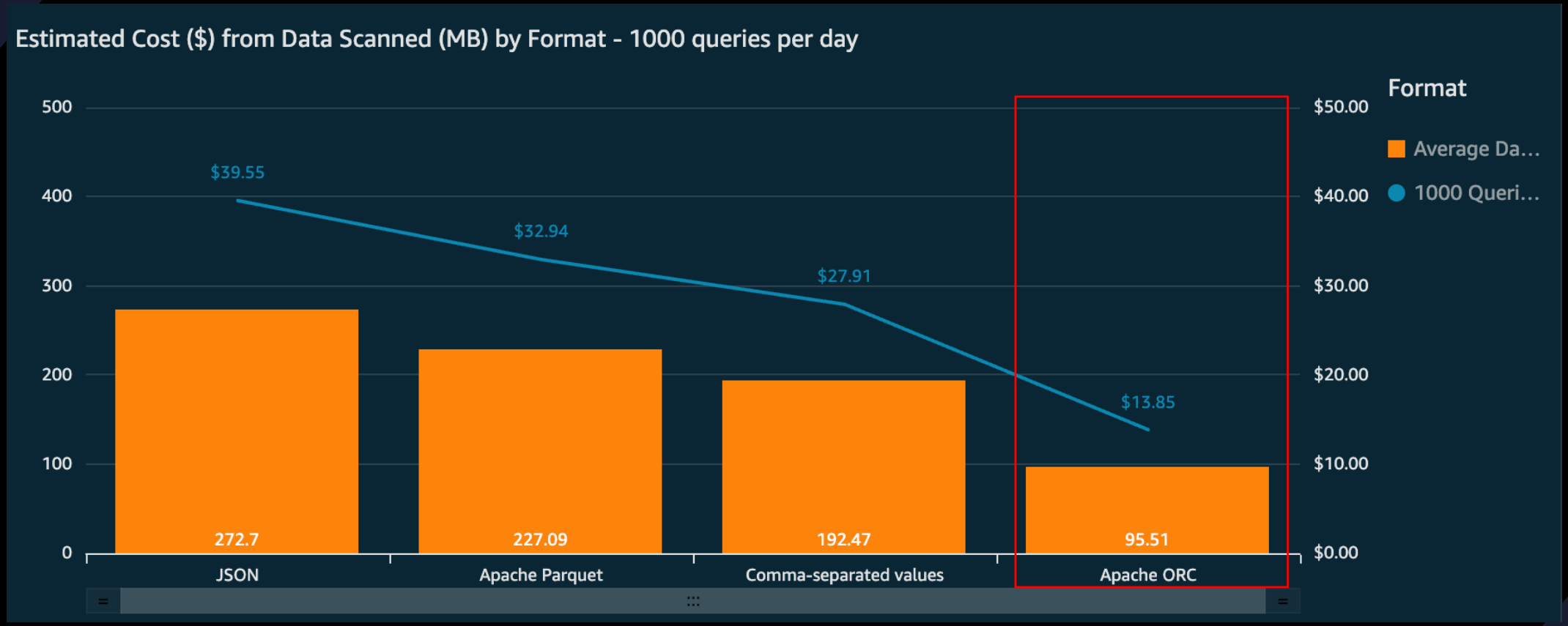






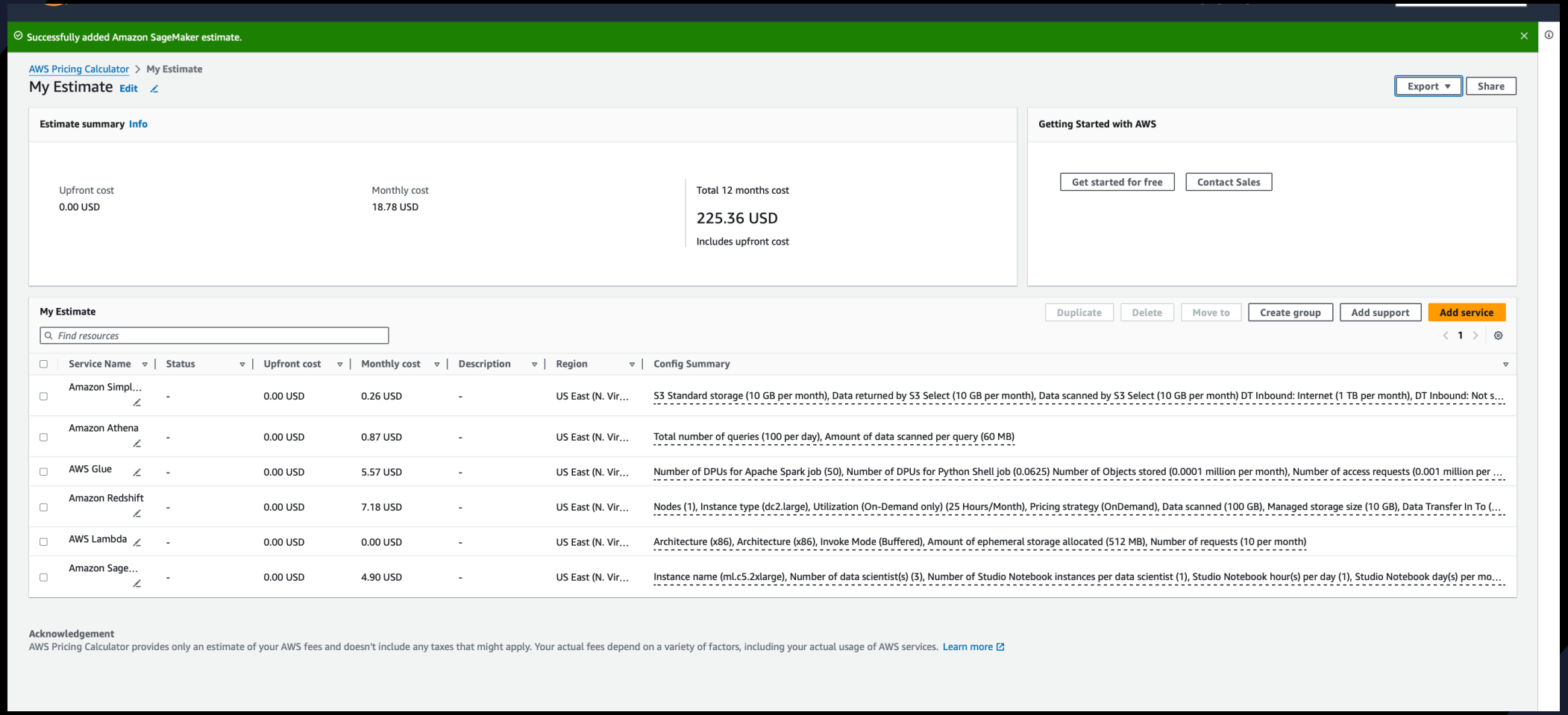
Parquet.gzip is the most cost effective format for Athena READs

Results – Cost Estimation



Link to Quicksight dashboard: https://us-east-1.quicksight.aws.amazon.com/sn/accounts/211125778552/dashboards/df177a27-eeaf-4861-ab23-9bd5c4da0aec/sheets/df177a27-eeaf-4861-ab23-9bd5c4da0aec_9cee886f-53d0-4865-9995-c78eaa190ebc

Cost Estimation - Project



Conclusion – Athena & S3

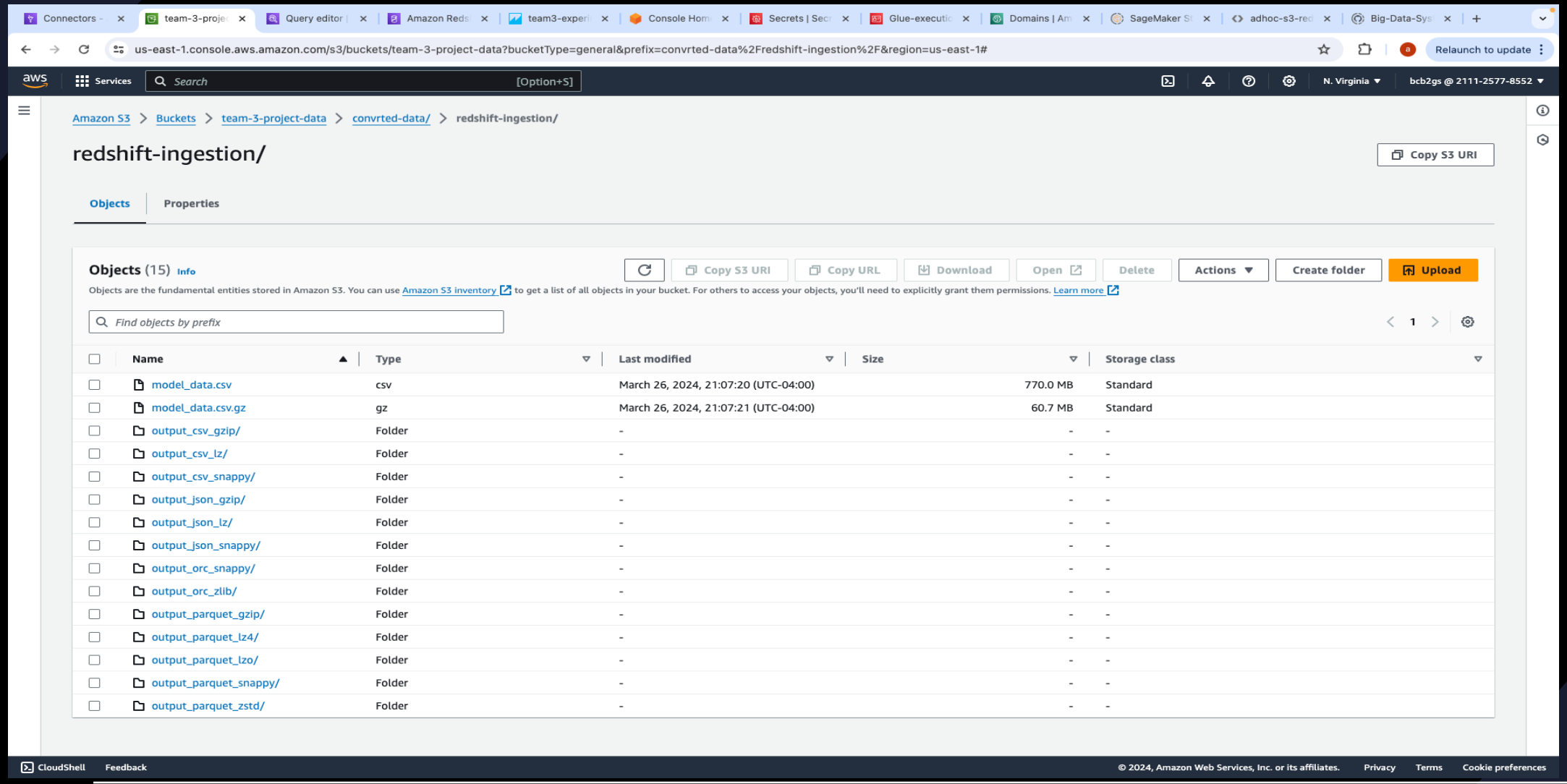
Data format and Compression	Select appropriate data formats (e.g., Parquet, ORC) and compression types (e.g., Snappy, Zstandard) based on your data characteristics and query patterns.
Organize Data in S3	Organize data in Amazon S3 using partitioning and bucketing techniques to improve query performance and reduce query costs.
Define Schema	Define the schema for your data in Athena either through AWS Glue Data Catalog or by creating tables using CREATE TABLE AS queries.
Optimize Query Performance	Optimize query performance by leveraging partition pruning, predicate pushdown, and column projection to reduce data scanned by queries.
Optimize Data Access Patterns	Optimize data access patterns by using AWS Glue Crawlers, updating partitions, and leveraging the AWS Glue Data Catalog for metadata management.
Monitor and Analyze	Monitor and analyze query performance using AWS CloudWatch metrics and Athena query execution plans to identify bottlenecks and areas for optimization.

Conclusion – Redshift

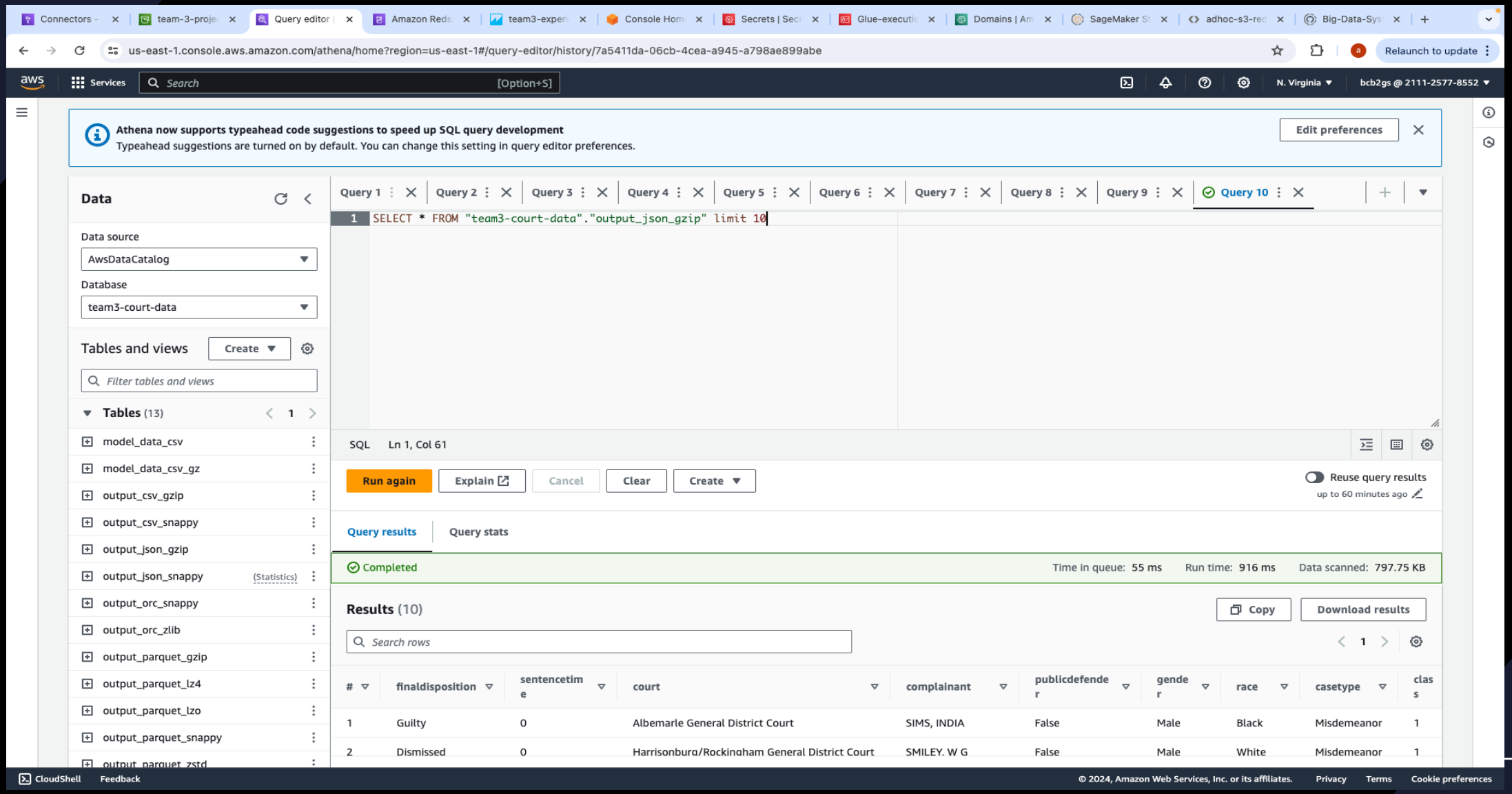
Data format and Compression	Choose Data Format and Compression: Select appropriate data formats (e.g., CSV, Parquet) and compression types (e.g., GZIP, ZSTD) based on your data characteristics and query patterns.
Load Data (Glue , COPY , Python)	Load Data to Amazon Redshift: Load data into Redshift using preferred methods such as COPY command, INSERT statements, or Glue from S3 for large datasets.
Optimize Data Distribution & sorting	Optimize Data Distribution & Sorting: Utilize appropriate distribution and sort keys (DISTKEY, SORTKEY) to optimize data distribution and storage on Redshift.
Analyze and Regular Vacuum	Perform Regular Vacuum and Analyze: Schedule regular Vacuum and Analyze operations to reclaim disk space and update statistics for query optimization.
Optimize Query Performance (Tuning)	Optimize Query Performance: Fine-tune and optimize queries for better performance, including indexing, query structure, and data distribution strategies.
Monitor and Analyze	Monitor and Analyze Query Performance: Continuously monitor query performance metrics and analyze query execution plans to identify bottlenecks and areas for improvement.

APPENDIX

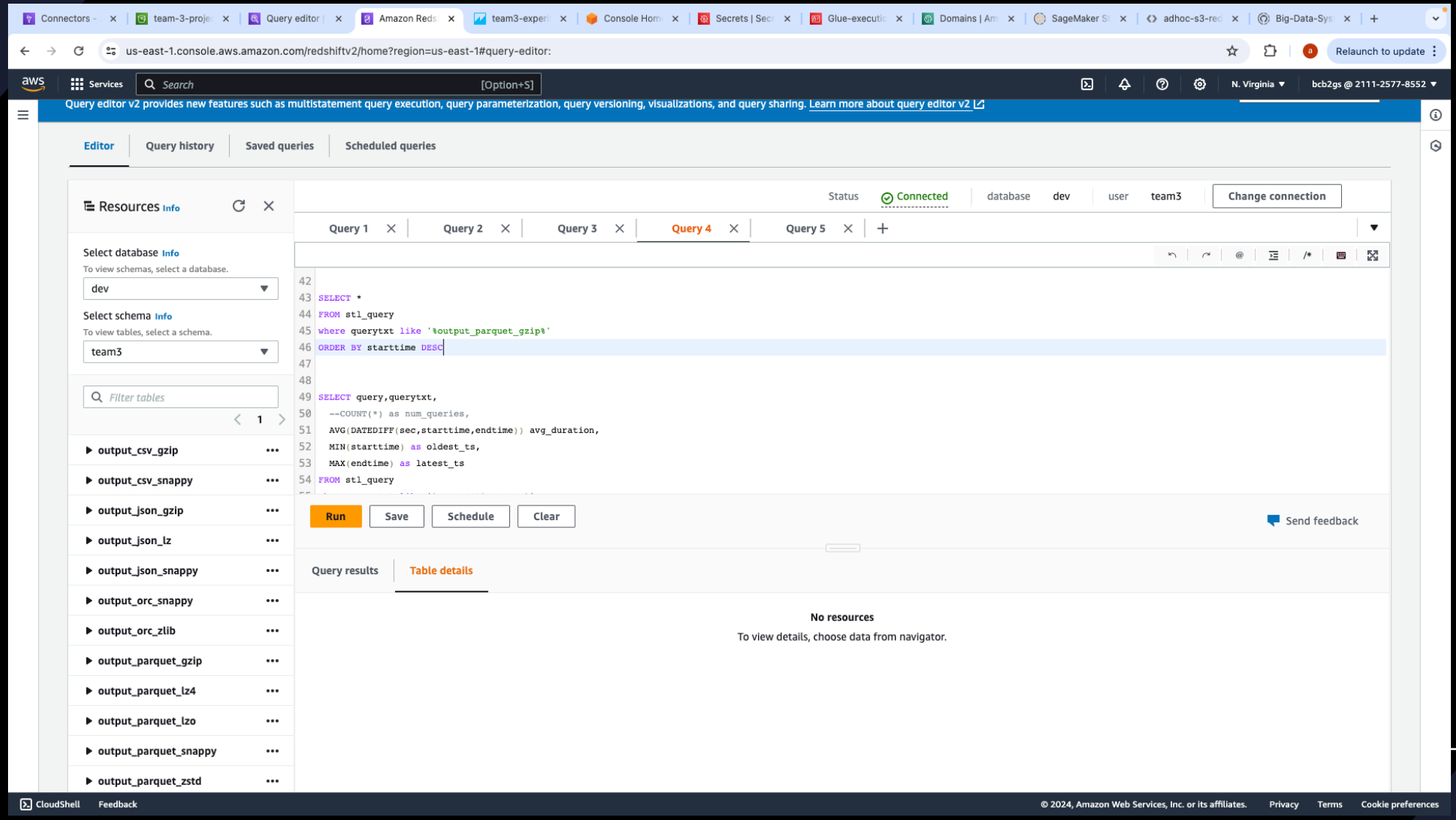
S3 – Compressed Data Snapshot



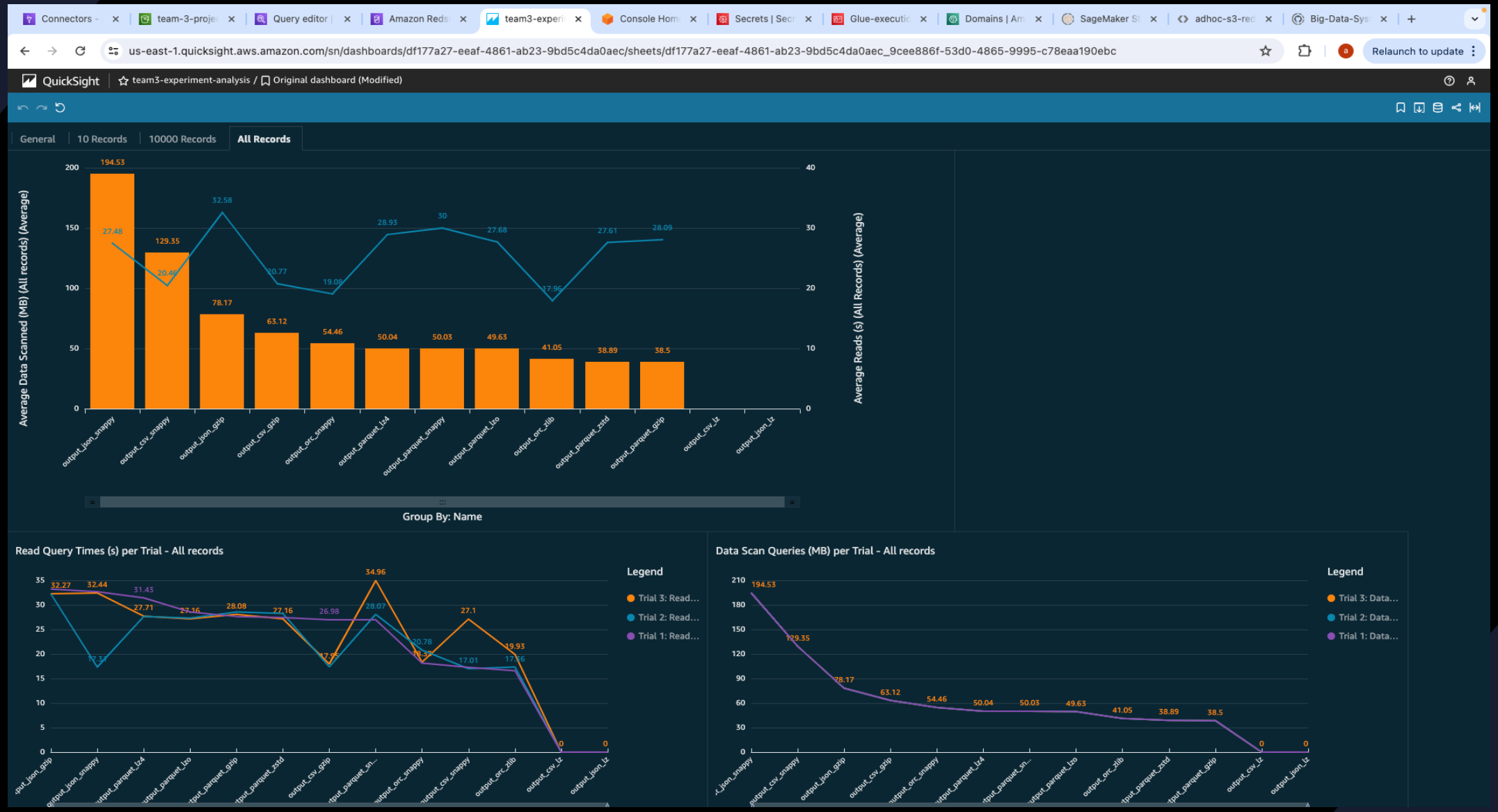
Athena – External Database



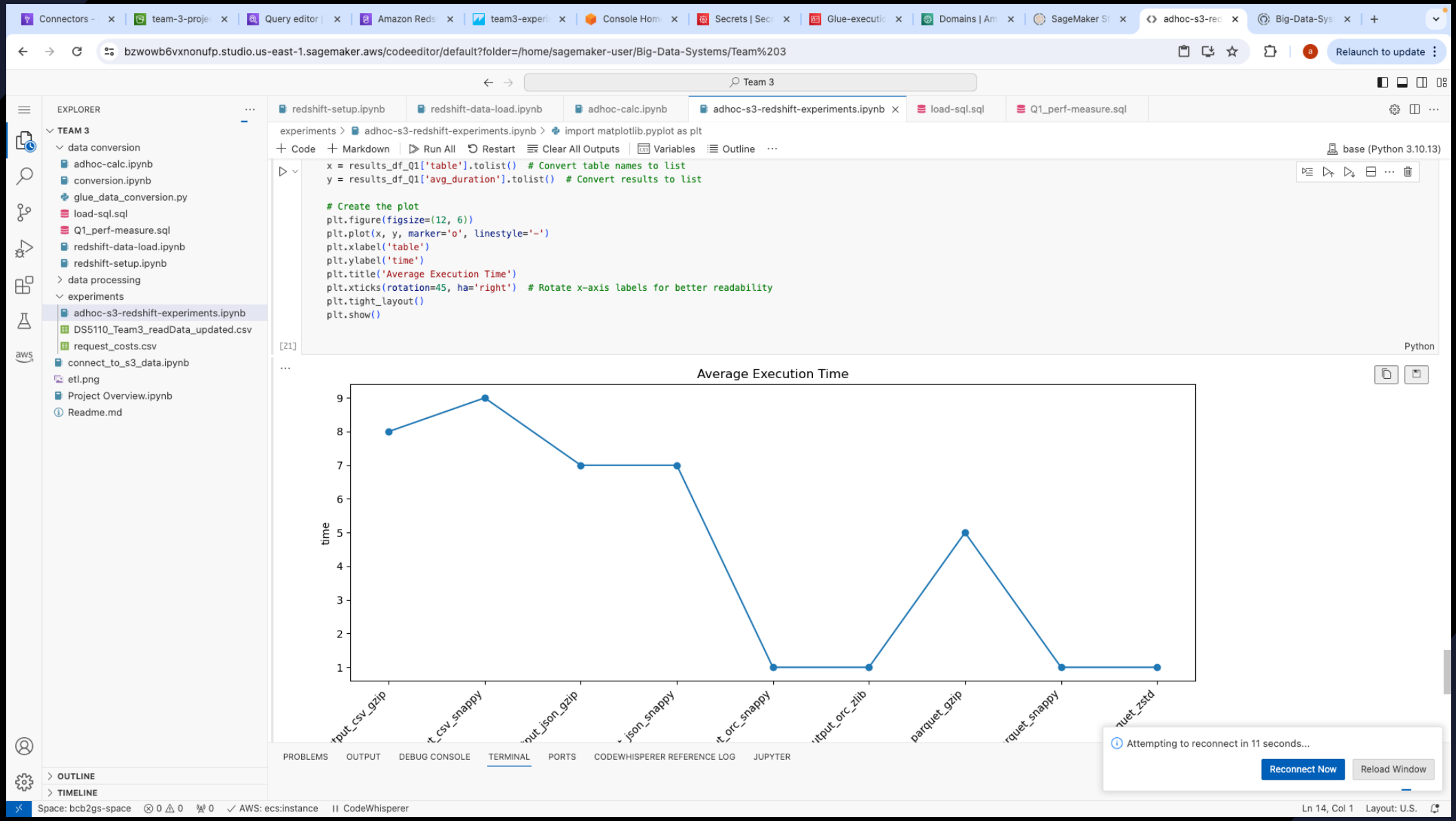
Redshift Database



QuickSight Reports



Sagemaker notebooks - Experiments



Thank you!