# **COVID-19 Analytics (AWS) — End-to-End**

### Data description

A centralised repository of up-to-date and curated datasets on or related to the spread and characteristics of the novel coronavirus (SARS-CoV-2) and its associated illness, COVID-19. Globally, there are several efforts underway to gather this data, and we are working with partners to make this crucial data freely available and keep it up-to-date. Hosted on the AWS cloud, we have seeded our curated data lake with COVID-19 case tracking data from Johns Hopkins and The New York Times, hospital bed availability from Definitive Healthcare, and over 45,000 research articles about COVID-19 and related coronaviruses from the Allen Institute for AI.

Note - Data is available here

<https://registry.opendata.aws/aws-covid19-lake/>

Data links

<https://covid19-lake.s3.us-east-2.amazonaws.com/enigma-jhu/>

<https://covid19-lake.s3.us-east-2.amazonaws.com/enigma-nytimes-data-in-usa/>

<https://covid19-lake.s3.us-east-2.amazonaws.com/rearc-covid-19-testing-data/>

<https://covid19-lake.s3.us-east-2.amazonaws.com/rearc-usa-hospital-beds/>

<https://covid19-lake.s3.us-east-2.amazonaws.com/static-datasets/>

This project is an end-to-end data pipeline to analyze COVID-19 data using **AWS S3, AWS Glue, Athena, and Redshift**.

The pipeline follows a **Bronze → Silver → Gold** architecture:

* **Bronze:** Raw, unprocessed data.
* **Silver:** Cleaned and standardized data in parquet format.
* **Gold:** Curated, analytics-ready fact and dimension tables.

The goal was to ingest raw datasets, standardize and partition them, and make them query able for insights like daily cases, testing trends, and positivity rates.

**1. Setting up the S3 Data Lake**

I started by creating an S3 bucket called covid19-analytics-shweta with default settings. Inside this bucket, I created three main folders to represent the different layers:

* covid-bronze
* covid-silver
* covid-gold

Inside the **Bronze folder**, I created subfolders to organize the raw datasets:

* covid\_tracking → uploaded states\_daily.csv
* nytimes → uploaded us\_states.csv
* static → uploaded states\_abv.csv

This gave me a clear structure for raw data ingestion:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**2. Creating IAM Role for Glue**Next, I created an **IAM role** named AWSGlueServiceRole-CovidProject so that AWS Glue

could access my S3 bucket and write logs to CloudWatch. I attached the following policies:

* AmazonS3FullAccess
* AWSGlueServiceRole
* CloudWatchFullAccess

A screenshot of a computer

AI-generated content may be incorrect.

**3. Bronze → Silver ETL**I then built the first ETL job in **AWS Glue Studio** called covid-bronze-to-silver:

* Type: Spark
* Language: Python
* IAM Role: AWSGlueServiceRole-CovidProject
* Workers: 2 (small dataset)

**Script Highlights:**

* Read raw CSV files from Bronze:

o nytimes/us\_states.csv  
o covid\_tracking/states\_daily.csv o static/states\_abv.csv

* Standardized data:

o Trimmed and uppercased state codes  
o Converted dates to proper format  
o Calculated cumulative cases, deaths, and tests

* Joined with states\_abv.csv to get full state names.
* Partitioned data by state code, year, month, day.
* Written output to Silver folder in parquet:

o cases\_standardized o testing\_standardized

**Verification:** Checked S3 Silver folder to confirm partitioned parquet files were created correctly.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Script being used in Glue;

from pyspark.sql import SparkSession, functions as F

# Define S3 locations

BUCKET = "covid19-analytics-awsproject"

BRONZE = f"s3://{BUCKET}/covid/bronze"

SILVER = f"s3://{BUCKET}/covid/silver"

# Initialize Spark

spark = SparkSession.builder.appName("covid-minimal-silver").getOrCreate()

# Helper functions

upper\_trim = lambda c: F.upper(F.trim(F.col(c)))

to\_date = lambda c: F.to\_date(F.col(c).cast("string"))

# --- Lookups ---

states = (

spark.read.option("header", True)

.csv(f"{BRONZE}/static/states\_abv.csv") # ✅ all quotes closed correctly

.select(

upper\_trim("Abbreviation").alias("state\_code"),

F.initcap(F.col("State")).alias("state\_name")

)

)

# --- Cases (NYT state file: date,state,cases,deaths) ---

cases\_raw = spark.read.option("header", True).csv(f"{BRONZE}/nytimes/us\_states.csv")

cases\_std = (

cases\_raw

.withColumn("full\_date", F.to\_date("date"))

.withColumn("state\_name\_raw", F.initcap(F.col("state")))

.join(states, states.state\_name == F.col("state\_name\_raw"), "left")

.withColumn("cases\_cum", F.col("cases").cast("long"))

.withColumn("deaths\_cum", F.col("deaths").cast("long"))

.withColumn("year", F.year("full\_date"))

.withColumn("month", F.month("full\_date"))

.withColumn("day", F.dayofmonth("full\_date"))

.select(

"full\_date", "state\_code", "state\_name",

"cases\_cum", "deaths\_cum", "year", "month", "day"

)

.dropna(subset=["full\_date", "state\_code"])

)

(

cases\_std.write.mode("overwrite")

.partitionBy("state\_code", "year", "month", "day")

.parquet(f"{SILVER}/cases\_standardized") # ✅ closed properly

)

# --- Testing (COVID Tracking Project) ---

tests\_raw = spark.read.option("header", True).csv(f"{BRONZE}/covid\_tracking/states\_daily.csv")

tests\_std = (

tests\_raw

.withColumn("full\_date", F.to\_date(F.col("date").cast("string"), "yyyyMMdd"))

.withColumn("state\_code", upper\_trim("state"))

.join(states.select("state\_code", "state\_name"), "state\_code", "left")

.withColumn("tests\_total\_cum", F.col("totalTestResults").cast("long"))

.withColumn("tests\_pos\_cum", F.col("positive").cast("long"))

.withColumn("tests\_neg\_cum", F.col("negative").cast("long"))

.withColumn("year", F.year("full\_date"))

.withColumn("month", F.month("full\_date"))

.withColumn("day", F.dayofmonth("full\_date"))

.select(

"full\_date", "state\_code", "state\_name",

"tests\_total\_cum", "tests\_pos\_cum", "tests\_neg\_cum",

"year", "month", "day"

)

.dropna(subset=["full\_date", "state\_code"])

)

(

tests\_std.write.mode("overwrite")

.partitionBy("state\_code", "year", "month", "day")

.parquet(f"{SILVER}/testing\_standardized") # ✅ closed properly

)

print("✅ Silver complete.")

**4. Registering Silver Tables in Athena**

After the Bronze → Silver ETL, I registered the Silver data in Athena:

* Created a folder in S3: covid-athenaresult to store Athena query results.
* Opened **AWS Athena → Query Editor**.

o In the **Settings** tab, set the S3 location (covid-athenaresult) to save query results.

• Created external tables in Athena pointing to the Silver parquet files:

**Scripts that I have used:**

CREATE DATABASE IF NOT EXISTS covid\_silver\_db;  
CREATE EXTERNAL TABLE IF NOT EXISTS covid\_silver\_db.cases\_standardized (

full\_date date, state\_name string, cases\_cum bigint, deaths\_cum bigint

)  
PARTITIONED BY (state\_code string, year int, month int, day int)

STORED AS PARQUET  
LOCATION 's3://covid19-analytics-shweta/covid-silver/cases\_standardized/';

MSCK REPAIR TABLE covid\_silver\_db.cases\_standardized;

CREATE EXTERNAL TABLE IF NOT EXISTS covid\_silver\_db.testing\_standardized ( full\_date date,  
state\_name string,  
tests\_total\_cum bigint,

tests\_pos\_cum bigint,

tests\_neg\_cum bigint )

PARTITIONED BY (state\_code string, year int, month int, day int)  
STORED AS PARQUET  
LOCATION 's3://covid19-analytics-shweta/covid-silver/testing\_standardized/'; MSCK REPAIR TABLE covid\_silver\_db.testing\_standardized;

**Verified Silver data using queries like:**

• SELECT MIN(full\_date), MAX(full\_date), COUNT(\*) FROM covid\_silver\_db.cases\_standardized; # \_col0 \_col1 \_col2

1 2020-05-01 2020-06-03 102

• SELECT state\_code, COUNT(\*) FROM covid\_silver\_db.testing\_standardized GROUP BY 1 ORDER BY 2 DESC;

# state\_code \_col1 1 CA 34  
2 NY 34  
3 FL 34

A screenshot of a computer

AI-generated content may be incorrect.

**5. Silver → Gold ETL**Created **AWS Glue Job** covid-silver-to-gold in Glue Studio:

* Type: Spark
* Language: Python
* IAM Role: AWSGlueServiceRole-CovidProject
* Requested workers: 2

A screenshot of a computer

AI-generated content may be incorrect.

ScriotA screenshot of a computer program

AI-generated content may be incorrect.

Script, I used in glue;

from pyspark.sql import SparkSession, functions as F, Window as W

# -----------------------------

# Config

# -----------------------------

BUCKET = "covid19-analytics-awsproject"

SILVER = f"s3://{BUCKET}/covid/silver"

GOLD = f"s3://{BUCKET}/covid/gold"

spark = SparkSession.builder.appName("covid-minimal-gold").getOrCreate()

# -----------------------------

# Read Silver parquet

# -----------------------------

cases\_path = f"{SILVER}/cases\_standardized"

tests\_path = f"{SILVER}/testing\_standardized"

print(f"Reading Silver data from:\n{cases\_path}\n{tests\_path}")

cases = spark.read.parquet(cases\_path)

tests = spark.read.parquet(tests\_path)

# -----------------------------

# dim\_date

# -----------------------------

all\_dates = (

cases.select("full\_date")

.union(tests.select("full\_date"))

.dropDuplicates()

.dropna(subset=["full\_date"])

)

dim\_date = (

all\_dates

.withColumn("year", F.year("full\_date"))

.withColumn("month", F.month("full\_date"))

.withColumn("day", F.dayofmonth("full\_date"))

.withColumn("dow", F.dayofweek("full\_date")) # 1=Sunday, 7=Saturday

.withColumn("is\_weekend", F.col("dow").isin([1, 7]))

.withColumn(

"date\_id",

(F.col("year") \* 10000 + F.col("month") \* 100 + F.col("day")).cast("int")

)

)

dim\_date.write.mode("overwrite").parquet(f"{GOLD}/dim\_date")

# -----------------------------

# dim\_state

# -----------------------------

dim\_state = cases.select("state\_code", "state\_name").dropDuplicates()

dim\_state.write.mode("overwrite").parquet(f"{GOLD}/dim\_state")

# -----------------------------

# fact\_cases\_state\_daily

# -----------------------------

w\_cases = W.partitionBy("state\_code").orderBy("full\_date")

fact\_cases = (

cases

.withColumn(

"date\_id",

(F.year("full\_date") \* 10000 + F.month("full\_date") \* 100 + F.dayofmonth("full\_date")).cast("int")

)

.withColumn(

"new\_cases",

F.greatest(F.col("cases\_cum") - F.lag("cases\_cum").over(w\_cases), F.lit(0))

)

.withColumn(

"new\_deaths",

F.greatest(F.col("deaths\_cum") - F.lag("deaths\_cum").over(w\_cases), F.lit(0))

)

.select("date\_id", "state\_code", "cases\_cum", "deaths\_cum", "new\_cases", "new\_deaths")

)

fact\_cases.write.mode("overwrite").partitionBy("state\_code").parquet(f"{GOLD}/fact\_cases\_state\_daily")

# -----------------------------

# fact\_testing\_state\_daily

# -----------------------------

w\_tests = W.partitionBy("state\_code").orderBy("full\_date")

fact\_tests = (

tests

.withColumn(

"date\_id",

(F.year("full\_date") \* 10000 + F.month("full\_date") \* 100 + F.dayofmonth("full\_date")).cast("int")

)

.withColumn(

"new\_tests",

F.greatest(F.col("tests\_total\_cum") - F.lag("tests\_total\_cum").over(w\_tests), F.lit(0))

)

.withColumn(

"positivity\_rate",

F.when(

F.col("tests\_total\_cum") > 0,

(F.col("tests\_pos\_cum") / F.col("tests\_total\_cum")).cast("double")

)

)

.select(

"date\_id",

"state\_code",

"tests\_total\_cum",

"tests\_pos\_cum",

"tests\_neg\_cum",

"new\_tests",

"positivity\_rate"

)

)

fact\_tests.write.mode("overwrite").partitionBy("state\_code").parquet(f"{GOLD}/fact\_testing\_state\_daily")

print("✅ Gold layer successfully created.")

Script Highlights:

* Read Silver parquet files.
* Created **dimension tables**:

o dim\_date → full\_date, year, month, day, day-of-week, weekend flag, date\_id o dim\_state → state\_code, state\_name

• Created **fact tables**:  
o fact\_cases\_state\_daily → cumulative and daily new cases and deaths

o fact\_testing\_state\_daily → cumulative and daily tests, positivity rate  
• Wrote output to Gold folder in parquet format in S3, partitioned as needed:

o dim\_date  
o dim\_state  
o fact\_cases\_state\_daily o fact\_testing\_state\_daily

**Verification:** Checked S3 Gold folder to confirm correct creation of parquet files.

A screenshot of a computer

AI-generated content may be incorrect.

**6. Registering Gold Tables in Athena**

* Used the existing folder covid-athenaresult for Athena query results.
* Registered Gold tables in Athena. Done similar as done for silver table. Athena > Query

Editor. In the **Settings** tab, S3 location (covid-athenaresult) to save query results has already been integrated before.

**Query that I have used:**

CREATE DATABASE IF NOT EXISTS covid\_gold\_db;  
CREATE EXTERNAL TABLE IF NOT EXISTS covid\_gold\_db.dim\_date (

date\_id int, full\_date date, year int, month int, day int,

dow int,

is\_weekend boolean )

STORED AS PARQUET  
LOCATION 's3:// covid19-analytics-awsproject /covid-gold/dim\_date/'; CREATE EXTERNAL TABLE IF NOT EXISTS covid\_gold\_db.dim\_state (

state\_code string,

state\_name string )

STORED AS PARQUET  
LOCATION 's3:// covid19-analytics-awsproject /covid-gold/dim\_state/';

-- Create database (run once)

CREATE DATABASE IF NOT EXISTS covid\_gold\_db;

-- ----------------------------

-- dim\_date table

-- ----------------------------

CREATE EXTERNAL TABLE IF NOT EXISTS covid\_gold\_db.dim\_date (

date\_id INT,

full\_date DATE,

year INT,

month INT,

day INT,

dow INT,

is\_weekend BOOLEAN

)

STORED AS PARQUET

LOCATION 's3:// covid19-analytics-awsproject /covid-gold/dim\_date/';

-- ----------------------------

-- dim\_state table

-- ----------------------------

CREATE EXTERNAL TABLE IF NOT EXISTS covid\_gold\_db.dim\_state (

state\_code STRING,

state\_name STRING

)

STORED AS PARQUET

LOCATION 's3:// covid19-analytics-awsproject /covid-gold/dim\_state/';

-- ----------------------------

-- fact\_cases\_state\_daily table

-- ----------------------------

CREATE EXTERNAL TABLE IF NOT EXISTS covid\_gold\_db.fact\_cases\_state\_daily (

date\_id INT,

cases\_cum BIGINT,

deaths\_cum BIGINT,

new\_cases INT,

new\_deaths INT

)

PARTITIONED BY (state\_code STRING)

STORED AS PARQUET

LOCATION 's3:// covid19-analytics-awsproject /covid-gold/fact\_cases\_state\_daily/';

MSCK REPAIR TABLE covid\_gold\_db.fact\_cases\_state\_daily;

-- ----------------------------

-- fact\_testing\_state\_daily table

-- ----------------------------

CREATE EXTERNAL TABLE IF NOT EXISTS covid\_gold\_db.fact\_testing\_state\_daily (

date\_id INT,

tests\_total\_cum BIGINT,

tests\_pos\_cum BIGINT,

tests\_neg\_cum BIGINT,

new\_tests INT,

positivity\_rate DOUBLE

)

PARTITIONED BY (state\_code STRING)

STORED AS PARQUET

LOCATION 's3:// covid19-analytics-awsproject /covid-gold/fact\_testing\_state\_daily/';

MSCK REPAIR TABLE covid\_gold\_db.fact\_testing\_state\_daily;

**7. Setting up Redshift IAM Role:**

To work with Redshift Serverless and load data from S3, I first created an **IAM role** with the service type set to **Redshift**. I attached the managed policy **AmazonS3ReadOnlyAccess** and gave the role a name: **RedshiftCopyRole**.

In the **Trust Relationship** tab, I updated the JSON to allow both Redshift and Redshift Serverless to assume the role:

**Code used:**

{  
"Version": "2012-10-17", "Statement": [

{  
"Effect": "Allow", "Principal": {

"Service": [  
"redshift.amazonaws.com", "redshift-serverless.amazonaws.com"

] },

"Action": "sts:AssumeRole" }

] }

I also added additional permissions via an **inline policy** to allow access specifically to my S3 bucket

A screenshot of a computer

AI-generated content may be incorrect.

The JSON for the inline policy is: {

"Version": "2012-10-17", "Statement": [

{  
"Effect": "Allow", "Action": [

"s3:GetObject",

"s3:ListBucket" ],

"Resource": [ "arn:aws:s3:::covid19-analytics-shweta", "arn:aws:s3:::covid19-analytics-shweta/\*"

] }

] }

This setup ensures Redshift can **read the Gold parquet data from S3** and perform COPY operations securely.

A screenshot of a computer

AI-generated content may be incorrect.

8. **Setting up Redshift Serverless Workgroup and Resolving Metadata Issues:** To work with Redshift, I navigated to Amazon Redshift → Serverless → Create Workgroup**. I created a workgroup named** covid-gold-wg**, leaving most configurations at default. I selected the** default VPC**, chose any** 2 subnets**, and used the** default security group**.**

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

For the namespace, I created one named **covid-gold-ns**, again using default settings. After creating the workgroup and namespace, I assigned the **IAM role (**RedshiftCopyRole**)** I had created earlier:

* Selected the namespace I created
* Went to Security & Encryption → Manage IAM Roles
* Attached the existing IAM role for Redshiftx

Once the role was synced, I clicked **Query Data**, selected the namespace and workgroup, and connected successfully.

A screenshot of a computer program

AI-generated content may be incorrect.

While executing queries, I noticed a **mismatch in metadata** between Athena and Redshift, which caused errors when performing COPY operations. To fix this, I created a **new Athena table** with a schema exactly matching Redshift’s table schema. This ensured consistency and allowed the COPY commands to execute without errors.

New table created in athena is added to the folder created in S3 bucket: covid-gold-clean.

A screenshot of a computer

AI-generated content may be incorrect.

Query that I have executed in Athena and Redshift query editor after fixing the issue.

--- New table created in athena : dim\_date -------- CREATE TABLE covid\_gold\_db.dim\_date\_clean WITH (

format = 'PARQUET',

external\_location = 's3:// covid19-analytics-awsproject /covid-gold-clean/dim\_date/' )AS

SELECT date\_id, full\_date,

year, month,  
day,  
dow, is\_weekend

FROM covid\_gold\_db.dim\_date;

**In redshift :**

A screenshot of a computer program

AI-generated content may be incorrect.

--- athena for dim\_state  
DROP TABLE IF EXISTS covid\_gold\_db.dim\_state\_clean;

CREATE TABLE covid\_gold\_db.dim\_state\_clean

WITH (  
format = 'PARQUET',  
external\_location = 's3:// covid19-analytics-awsproject /covid-gold-clean/dim\_state/'

)AS  
SELECT state\_code, state\_name FROM covid\_gold\_db.dim\_state;

-----redshift +copy ---------

A screenshot of a computer

AI-generated content may be incorrect.

---- athena fact\_cases\_state\_daily------  
DROP TABLE IF EXISTS covid\_gold\_db.fact\_cases\_state\_daily\_clean;

CREATE TABLE covid\_gold\_db.fact\_cases\_state\_daily\_clean WITH (

format = 'PARQUET',

external\_location = 's3:// covid19-analytics-awsproject /covid-gold-clean/fact\_cases\_state\_daily/' )AS

SELECT date\_id, state\_code, cases\_cum, deaths\_cum, new\_cases, new\_deaths FROM covid\_gold\_db.fact\_cases\_state\_daily;

----redshift+ copy --------------

A screenshot of a computer

AI-generated content may be incorrect.

---- athena fact\_testing\_state\_daily---------  
DROP TABLE IF EXISTS covid\_gold\_db.fact\_testing\_state\_daily\_clean;

CREATE TABLE covid\_gold\_db.fact\_testing\_state\_daily\_clean WITH (

format = 'PARQUET',

external\_location = ‘covid19-analytics-awsproject /covid-gold-clean/fact\_testing\_state\_daily/' )AS

SELECT date\_id, state\_code, tests\_total\_cum, tests\_pos\_cum, tests\_neg\_cum, new\_tests, positivity\_rate

FROM covid\_gold\_db.fact\_testing\_state\_daily;

--redshift +copy ----------

DROP TABLE IF EXISTS covid\_gold\_db.fact\_testing\_state\_daily;

CREATE TABLE covid\_gold\_db.fact\_testing\_state\_daily ( date\_id INT NOT NULL ,  
state\_code VARCHAR(2) NOT NULL ,  
tests\_total\_cum BIGINT,

tests\_pos\_cum BIGINT, tests\_neg\_cum BIGINT,  
new\_tests INT,  
positivity\_rate DOUBLE PRECISION

)  
DISTKEY(state\_code) SORTKEY(date\_id, state\_code);

COPY covid\_gold\_db.fact\_testing\_state\_daily  
FROM 's3://covid19-analytics-shweta/covid-gold-clean/fact\_testing\_state\_daily/' IAM\_ROLE 'arn:aws:iam::930797749854:role/RedshiftCopyRole'  
FORMAT AS PARQUET;

ANALYZE VERBOSE;

While copying the data I have used ARN of the IAM role created for Redshift : RedshiftCopyRole.

•

Result: state\_name California

9. Now executing the query and showing the result :

**Top 5 states by new cases on a given date**

SELECT s.state\_name, f.new\_cases  
FROM covid\_gold\_db.fact\_cases\_state\_daily f  
JOIN covid\_gold\_db.dim\_state s ON s.state\_code = f.state\_code WHERE f.date\_id = 20200515  
ORDER BY f.new\_cases DESC  
LIMIT 5;

new\_cases 147158

7-day **positivity rate** trend for one state SELECT d.full\_date,

SELECT d.full\_date,

(t.tests\_pos\_cum::double precision / NULLIF(t.tests\_total\_cum, 0)) AS positivity\_rate FROM covid\_gold.fact\_testing\_state\_daily t  
JOIN covid\_gold.dim\_date d ON d.date\_id = t.date\_id  
WHERE t.state\_code = 'NY'

ORDER BY d.full\_date;

Daily **cases vs tests** for a state (sanity/trend)

SELECT d.full\_date, c.new\_cases,

t.new\_tests,  
(t.tests\_pos\_cum::double precision / NULLIF(t.tests\_total\_cum, 0)) AS positivity\_rate

FROM covid\_gold.fact\_cases\_state\_daily c JOIN covid\_gold.fact\_testing\_state\_daily t

ON t.date\_id = c.date\_id AND t.state\_code = c.state\_code JOIN covid\_gold.dim\_date d ON d.date\_id = c.date\_id WHERE c.state\_code = 'CA'  
ORDER BY d.full\_date;

Result: full\_date 2020-05-01 2020-05-02 2020-05-03 2020-05-04 2020-05-05 2020-05-06 2020-05-07 2020-05-08 2020-05-09 2020-05-10 2020-05-11 2020-05-12 2020-05-13 2020-05-14 2020-05-15 2020-05-16 2020-05-17

new\_cases 00 175909 00 199407 68018 0

new\_tests positivity\_rate 0.03855339657693313  
0 0.23057791436980665 0.48978498312472024  
1456537 0.03916757346792193 0.07380881306961197

0 508684 0.07283671925717493 00 0.26240801776508965  
257289 222448 0.19856095306755325 00 0.13074025510602671

5390 570905 0.01049917694199203 248868 0 0.18262875119986066 63098 0 0.25024931523282085  
0 845805 0.024347042070765136 0 0 0.6005198306338128

147158 2157417 0.11458532480225272 151689 0 0.12332111796885958  
0 406515 0.0801504579032255

2020-05-18 2020-05-19 2020-05-20 2020-05-21 2020-05-22 2020-05-23 2020-05-24 2020-05-25 2020-05-26 2020-05-27 2020-05-28 2020-05-29 2020-05-30 2020-05-31 2020-06-01 2020-06-02 2020-06-03

13666 0 0.11955913348904176  
42140 560724 0.028072658042899348 0 984530 0.05867550024705868

0 0  
16124 0  
78438 0  
79102 1535992 0.1098724799511063 31753 871679 0.00908384399398635 0 0 0.07519699695973196 171848 0 0.5243538924669484 0 395866 0.1196842844612638 95952 0 0.08321640640421683

0 2416595 0.1298880555176425  
196054 0 0.35155362849549215  
0 277086 0.19948292503808812 228294 317457 0.05188773671778036 0 0 0.5950308702719646

Key Learnings & Highlights

* Built a complete cloud ETL pipeline using AWS S3, Glue, Athena, and Redshift.
* Learned partitioning parquet files for efficient querying.
* Implemented daily metrics calculations: new\_cases, new\_deaths, new\_tests,

positivity\_rate.

* Handled metadata mismatches between Athena and Redshift.
* Verified and validated data with multiple queries for correctness.

This project showcases a full-fledged cloud analytics workflow, from raw ingestion to ready-to- query outputs.