# NYC Taxi and Citi Bike Data Analysis

## Project Report

**Data - 228 (Big Data Technologies and Applications)**

Team Dataluation
Abinaya Seshadre, Neelima Jagtap, Rishikumar Ravichandran, Sruthi Mallarapu

## I. Abstract

Yellow taxis have been a well-renowned symbol and phenomenon of New York city. They serve the 5 major boroughs of NYC: Manhattan, Queens, Bronx, Brooklyn, and Staten Island. NYC TLC is the government agency that oversees the operations of taxi services in NYC including controlling and handing out licenses to all taxi businesses, even to competitors to yellow taxis like Uber. Before the pandemic hit, there were about 1 million taxi trips made every day in NYC. TLC collects the trips data, stores them in its repository and shares those for public use.

Citi bike is the bike sharing program of NYC. These bikes are available throughout the day, throughout the year and can be picked up from any station and returned any of the docking locations, which is very convenient for commuters with various purposes and generate tons of transaction data. Currently, there are about 21500 bikes with 1400+ stations in and around the major boroughs.

## II. Project Goals

We had the following goals in mind for this term project:

- Find and use big data - datasets with million rows of data.
- Implement data modelling and data processing for data analysis and visualizations.
- Explore and use cloud technology for all our requirements.
- Uncover exciting insights with analysis and visualizations.

Later sections will describe, in detail, how each of our project goals were achieved.

## III. Motivation

NYC TLC vehicles make about 100000 trips every day. This generates huge data for TLC that can be analyzed to answer a multitude of questions.

1. Identify hotspots.
2. Identify various trends.
3. Compare taxi and bike data.
4. Capture the effects of the pandemic.

# IV. Methodology

This section gives a detailed description of the methods we followed to accomplish our project goals.

The first step we took was that of requirements gathering which consisted of the following steps:

## Project Planning

- Probing the internet for open datasets.
- Discussing about our goals and requirement of the project.
- Fixing the datasets
- Choosing our tools
- Choosing our methodology
- Following class lectures, homework, and online resources to understand concepts and help with the project implementation.

We ended up with the following high-level decisions, after which we came up with our end-to-end data flow chart.

- We chose to do our project using AWS services.
- We decided our datasets and took up combined analysis of Citi bike and Taxi data in NYC.
- We also wanted to showcase our NYC TLC Data Analysis for insights on peak hour, areas, and times of services and COVID effects.

The below flow chart depicts the process we followed for data ingestion from the data sources to our pipelines and to the final stages of data analysis and visualization.

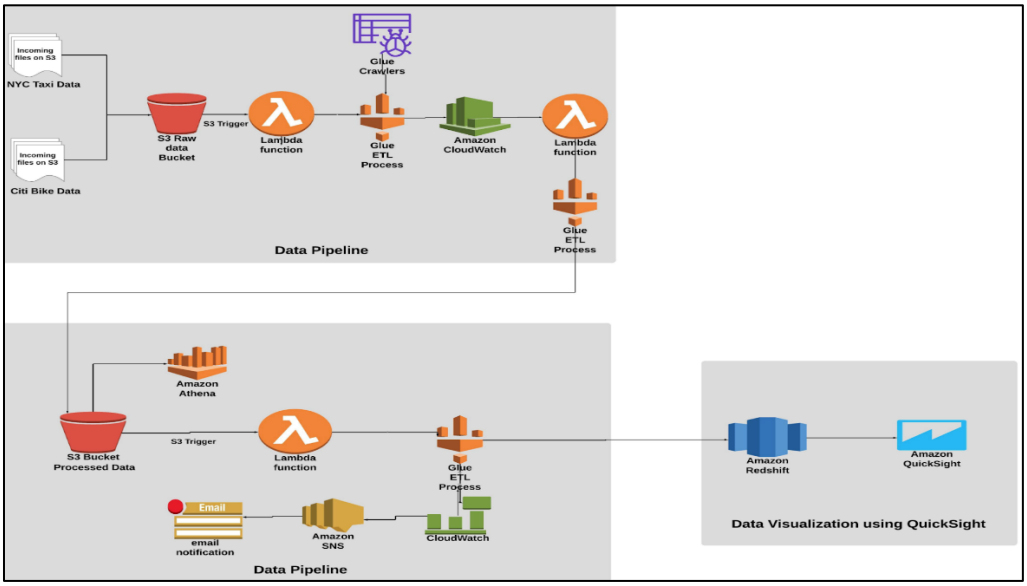



*Figure 1: Data flow from source to destination via pipelines*

## Process Overview

The below steps give an overview of the processes undertaken for data ingestion to our pipelines:

- Imported Data to S3 buckets using **AWS CLI.**
- Created an **S3 Trigger** that invoked the **Lambda function** which crawls the data in S3.
- Created **Cloud Watch** Event that triggers another Lambda function which perform the **ETL job** and stores the processed data in S3.
- Created an S3 Trigger in processed bucket that invoked the Lambda function that performed the ETL job of transferring processed data from S3 to **Redshift.**
- Created Cloud Watch Event which invokes **AWS SNS** and sends email notification on successful execution.
- Connected Redshift to QuickSight and created visualization dashboards.

## Process Description

### Data sources

Our data source for NYC Taxi was from the public repository available in public S3 buckets where each month's data is loaded in batches after the month is over. Instead of taking the same years data, we chose data that spanned the months of March to May for the years 2019 and 2020 so we will be able to capture the effects of the pandemic on our chosen business. The chosen datasets had 26.6 million data rows in total.

The datasets had details about the taxi trips including fields like pickup location, drop location, timestamp, fares, distance and so on.

Our data for New York Citi bikes was procured from NYC Open data S3 buckets. This dataset contains data about the pickup location, drop location, timestamp, duration, distance and so on. For Citi bike, we chose data that spanned the months of March to May for the year 2019, for comparison with our taxi data. The chosen dataset had 5 million data rows in total.

We imported the NYC Yellow taxi data and Citi bike data to our project S3 buckets using AWS CLI.

### Data Storage

Amazon Simple Storage Service, S3 acts as a file storage system and provides storage space for various data and objects. S3 has scalable infrastructure and is a public resource in the cloud provided by Amazon Web Services. The advantages of using S3 bucket is storing all data in one location with access to any of the services and tools provided by AWS. Also, the scalability factor makes it more beneficial for the implementation of projects.

Our data files were loaded into our S3 bucket for easy storage and further loading into Amazon Redshift. We decided on Amazon S3 due to its better compatibility with Amazon Redshift and QuickSight.

### Cleansing (Data Pre-processing)

Raw data almost always has redundancies and gaps. Such inconsistencies should be removed by either deletion or replacement before further processing can be done so the results generated from the data are accurate and not misleading. Cleansing can be accomplished by using Glue scripts. We did not want to go for other online data Cleansing tools as we wanted to experiment and achieve as much as possible with the cloud solutions and AWS.

Amazon Glue is a serverless processing and computing service provided by AWS. It is useful for data integration and formation of metadata for table data. It acts as a crawler which crawls through the data to detect the schema and composes the metadata for the data, after which data is loaded into the chosen target. Data can be pre-processed using Glue. Glue provides with a data catalog, which could be used to analyze and understand our data better. It is an Extract Transform Load (ETL) and event driven service to build our data in a better shape. The data is ingested into Glue with the help of a crawler and a job.

**Data Loading**

All the data, once cleansed, must be loaded into a common repository for easy access. We had our raw data and processed data in S3 and loaded our processed data to Redshift for further analysis using SQL queries.

**Data Modeling**

After the necessary, processed data are available, we used tools like draw.io to identify the entities, attributes, relationships and generate the data model, as the AWS data modeler available as part of DynamoDB was not available in the free tier. This data model is important to understand the fields in the data sets and relationships between the entities, if available and would also assist us in generating novel ideas of performing analysis on the data.

**Data Analysis**

Various types of analytics, including ‘Descriptive analysis, Exploratory analysis and Predictive analysis’ can be carried out on the data loaded into our AWS database, Amazon Redshift.

Amazon Redshift is a fully managed, cloud-based data warehouse provided by AWS. It is SQL compatible and SQL queries are used to perform DDL and DML operations on the tables and the large datasets. Redshift can store and operate (process joins) on even petabyte sized data.

We performed data analysis using SQL queries in Redshift and later also in QuickSight using graphs and charts.

**Visualization**

Finally, the results were presented in a graphical manner for easy and comprehensive visualization by using Amazon’s visualization tool, QuickSight.

Amazon QuickSight is AWS's visualization tool. It is highly scalable and consists of business intelligence and machine learning capabilities built in to easily create insights, visuals

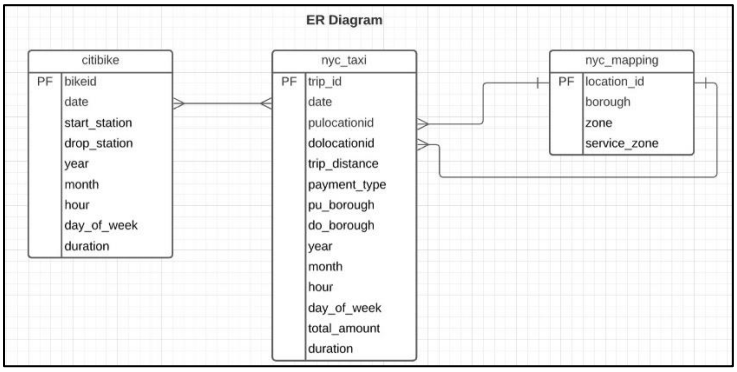
and dashboards for our applications, data models and projects.

QuickSight provides access to load data from many sources like S3, Redshift which are AWS services and sources external to AWS like PostgresSQL, Oracle, Teradata and so on. We can create graphs and charts using our own custom SQL queries.

# V. Implementation

This section dives deeper into our implementation steps and elaborates more on our described methodologies. It includes a detailed description of every step with appropriate screenshots showing execution in AWS.

## Our Data Model




Our data model has the entities for the Yellow Taxi and Citi Bikes data tables and also describes the attributes required in each of the entity.

## Implementation Steps:

### Data Ingestion
### Automated ETL pipeline

We automated the ETL pipeline from creating the data catalog to transferring the processed data to Redshift. Whenever new data is uploaded into our S3 buckets, below steps are automatically triggered:

1. Creation of the Data Catalog
2. ETL job Execution
3. Data transfer to Redshift.
4. Email Notification upon Successful Transfer.

1. **Creating the Data Catalog**

   To determine the schema of the raw data in S3, data is crawled using AWS Glue. This creates a data catalog which has the schema of the data. We automated this step by

creating a trigger in S3 bucket that invokes the AWS Lambda function. This function starts the AWS Glue crawler and catalogs the data.

2. **ETL JOB Execution**

   Upon Successful completion of Crawler, the second AWS Lambda function is triggered by using Amazon CloudWatch Events rule. This function invokes the ETL job.

   The ETL job is created using AWS Glue studio that cleans the data and transfers it to processed S3 bucket. Cleaning process involved dropping unnecessary columns, changing data types, filtering the data, and performing the joins with mapping files.

3. **Transferring the data to Redshift.**

   A Trigger is created in the S3 bucket with processed data that invokes the third AWS Lambda function. This function triggers another ETL job that transfers the data from S3 to Redshift. This ETL job is created using AWS Glue and as a pre-requisite, we also configured the Redshift connection and created necessary, predefined tables in Redshift.

4. **Email Notification upon Successful Transfer.**

   Created a rule in CloudWatch Events rule in such a way that when the status of the AWS Glue ETL job is 'Success', it sends an email notification using an SNS topic. Configured and set the email field for AWS to know which email IDs, notifications need to be sent in SNS Topic.

## Step by Step with screenshots

Data Ingestion into S3 buckets.

We first created separate buckets in S3 to store our raw Yellow taxi data and Citi Bike data and then loaded the data files using the CLI.

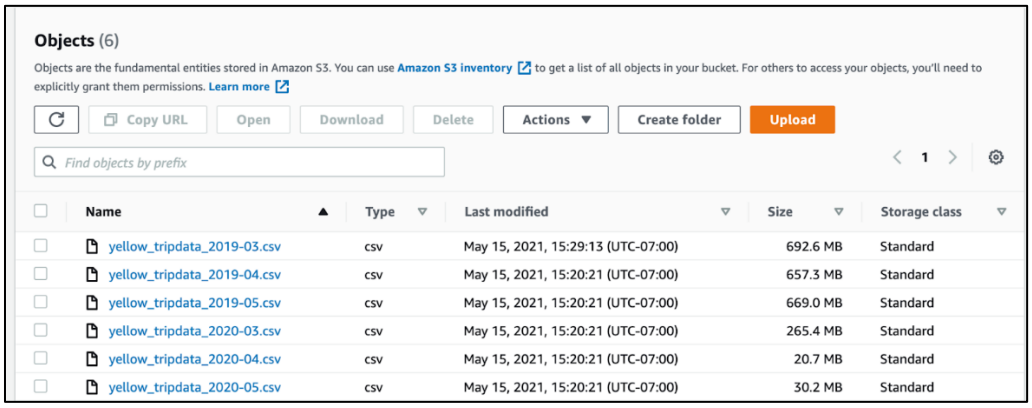



*Figure 2: Yellow Taxi Data of 6 months period (Mar2019-May2019, Mar2020-May2020) uploaded to our S3 bucket -* ***yellow-taxi-data-ny****.*

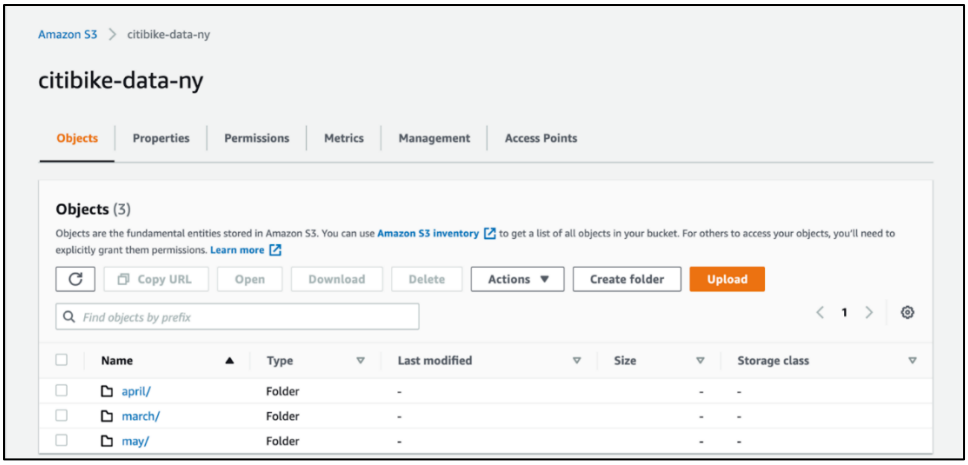


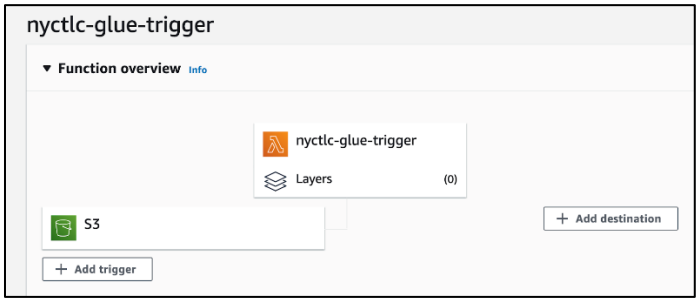

*Figure 3: Citi bike data of three months period (Mar2019-May2019) has been uploaded into another S3 bucket -* ***citibike-data-ny****.*

## Data Pipelines

We created the demo pipeline which we used for our project presentation and replicated this pipeline on our raw data buckets. Attached are the screenshots for the step-by-step process for both our project as well as demo pipeline.

Created the S3 trigger name **nyctlc-crawler-trigger** in the "**yellow-taxi-data-ny**" bucket. This trigger invokes the lambda function named "**nyctlc-glue-trigger**" whenever new file is uploaded in the bucket.

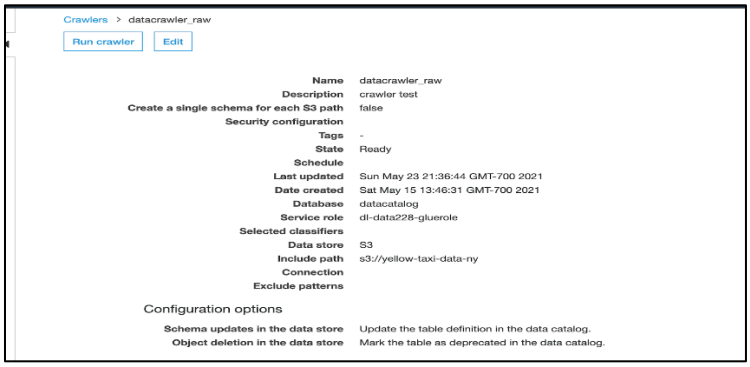

The Lambda function invoked above is used to create the crawler named "**datacrawler_raw**" and then creates the data catalog.



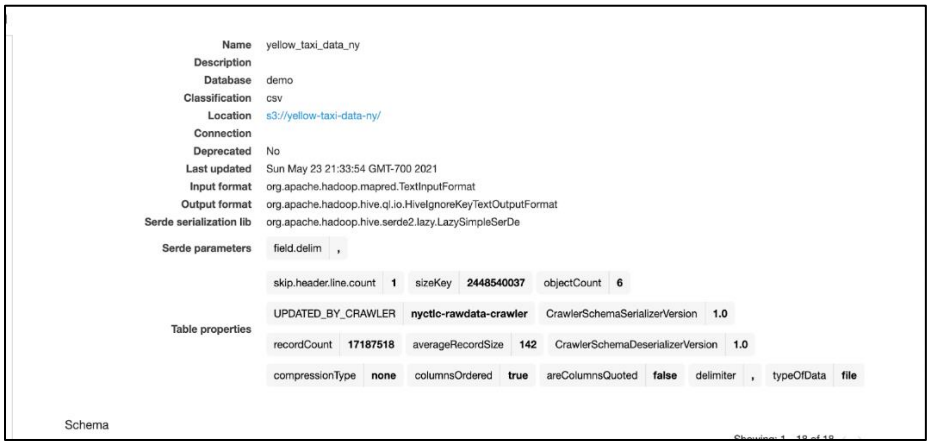

Code Snapshot for the above lambda function

```
var AWS = require('aws-sdk');
var glue = new AWS.Glue();
    exports.handler = function(event, context,callback) {
        console.log(JSON.stringify(event, null, 3));
        var dbName = 'datacatalog';
        var params = {
            DatabaseInput: {
                Name: dbName,
                Description: 'project',
            }
        };
        glue.createDatabase(params, function(err, data) {
            var params1 = {
                DatabaseName: dbName,
                Name: 'datacrawler_raw',
                Role: 'service-role/dl-data228-gluerole',
                Targets: {
                    S3Targets: [{ Path: 's3://yellow-taxi-data-ny' }]
                },
                Description: 'crawler test'
            };
            glue.createCrawler(params1, function(err1, data1) {
                startCrawler('datacrawler_raw', function(err2,data2){
                    if(err2) callback(err2)
                    else callback(null,data2)
                })
            });
        });
};
function startCrawler(name,callback){
    var params = {
        Name: name,
    };
    glue.startCrawler(params, function(err, data) {
        callback(null, data)
    });
```

Next, **datacrawler_raw** crawler is created by the invocation of the above lambda function.



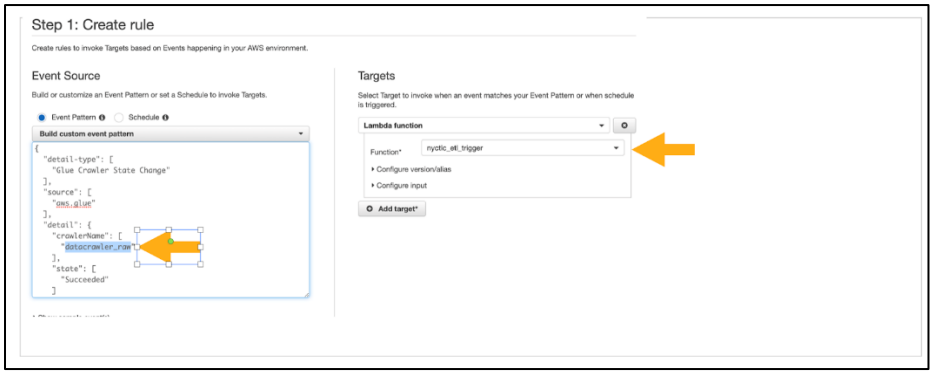

After the datacrawler_raw crawls the data, “yellow_taxi_data_ny” catalog table is created.



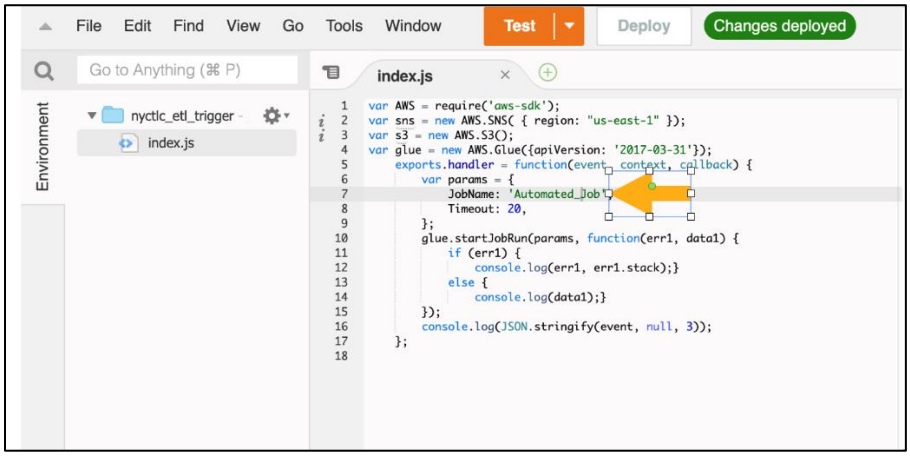

Schema of the raw data

| | Column name | Data type | Partition k |
|---|---|---|---|
| 1 | vendorid | bigint | |
| 2 | tpep_pickup_datetime | string | |
| 3 | tpep_dropoff_datetime | string | |
| 4 | passenger_count | bigint | |
| 5 | trip_distance | double | |
| 6 | ratecodeid | bigint | |
| 7 | store_and_fwd_flag | string | |
| 8 | pulocationid | bigint | |
| 9 | dolocationid | bigint | |
| 10 | payment_type | bigint | |
| 11 | fare_amount | double | |
| 12 | extra | double | |
| 13 | mta_tax | double | |
| 14 | tip_amount | double | |
| 15 | tolls_amount | double | |
| 16 | improvement_surcharge | double | |
| 17 | total_amount | double | |
| 18 | congestion_surcharge | double | |

An event rule is created in Cloud Watch that invokes another lambda function upon successful crawling of the data.

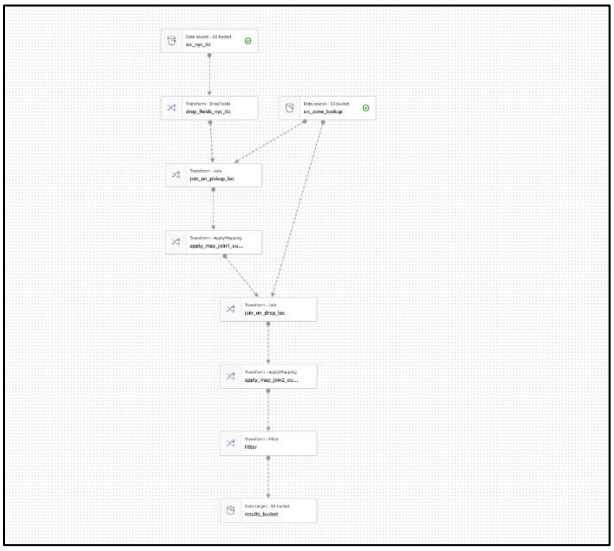



Code snapshot for the second lambda function "**nyc_tlc_etl_trigger**". It invokes the ETL job named '**Automated_Job**'.

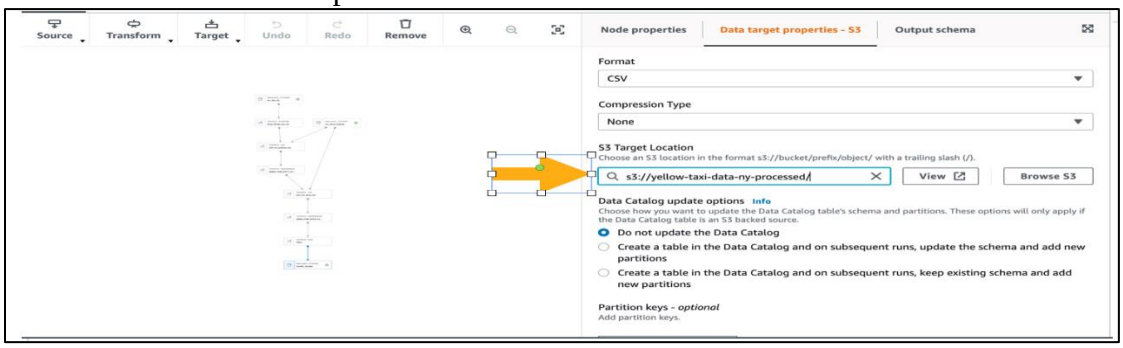


The ETL job is created using AWS Glue studio that cleans the data and transfers it to processed S3 bucket. Cleaning process involved dropping unnecessary columns, changing data types, filtering the data, and performing the joins with mapping files. Mapping files are present in S3 bucket named "**mapping-file-nyctlc-nvirginia**".

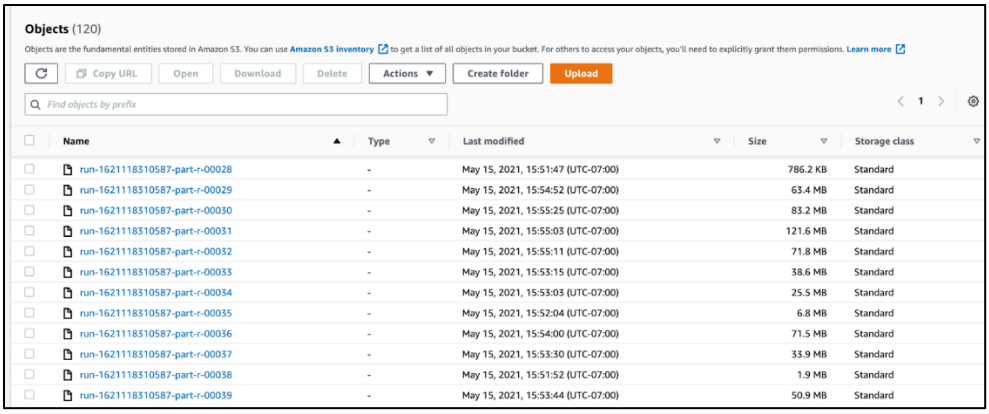

**ETL FLOW**



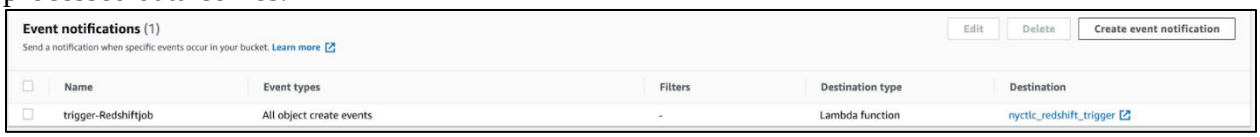
Processed Data is stored in S3 processed buckets.



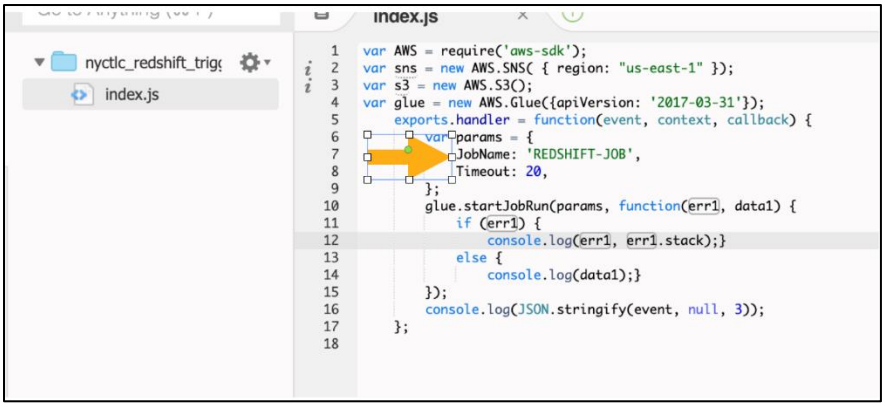

The cleaned data is partitioned and is stored in a processed S3 bucket named "**yellow-taxi-data-ny-processed**".



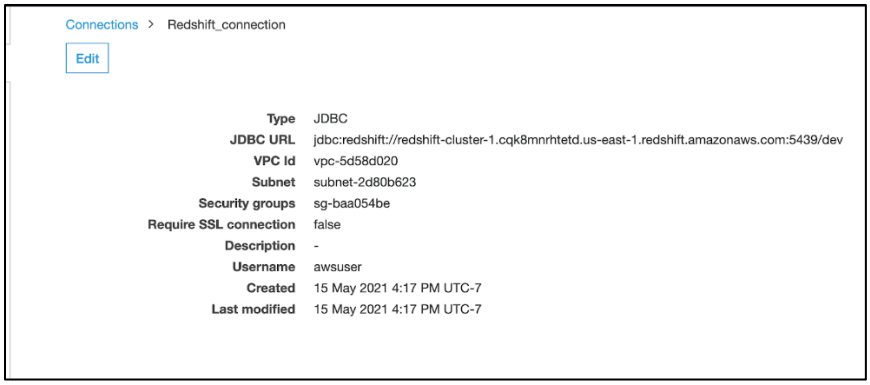

We have used Athena service to check if the processed data is in correct format.
Created the S3 trigger name **trigger-Redshiftjob** in the "**yellow-taxi-data-ny-processed**" bucket. This trigger invokes the "**nyctlc_redshift_trigger**" lambda function as soon as the processed data comes.



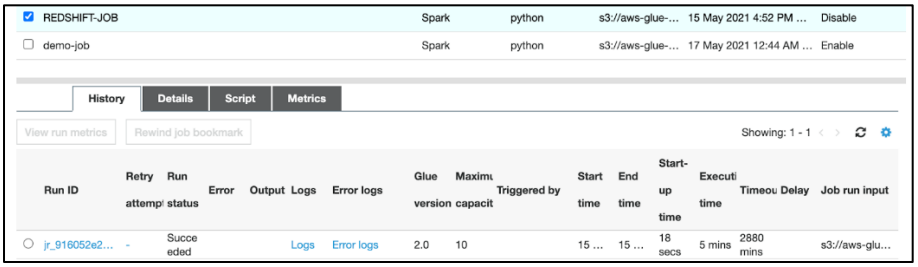

This Lambda function invokes "**REDSHIFT-JOB**".




Configured the redshift Connection that is required by the **REDSHIFT-JOB** to transfer the data from processed S3 bucket to Redshift.

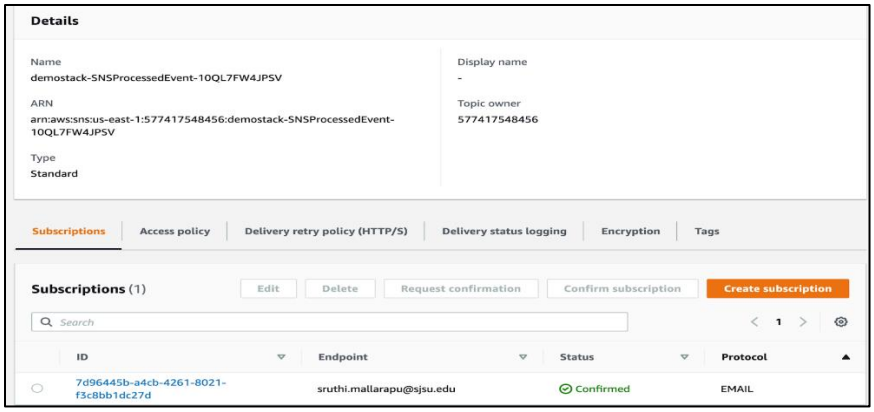



Successful execution of **REDSHIFT-JOB**. It means data has transferred to Redshift tables.

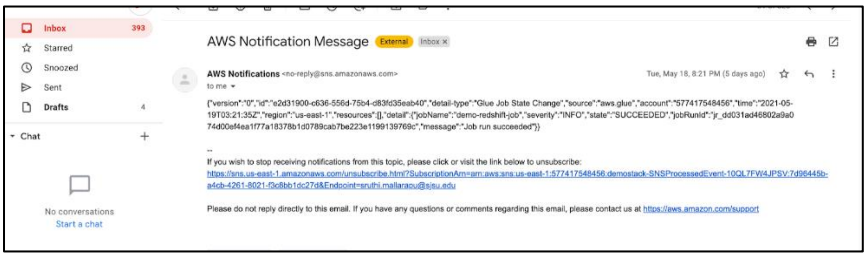



Another event rule is created in Cloud Watch that invokes SNS services. This service notifies through an email upon successful completion of the **REDSHIFT-JOB**.

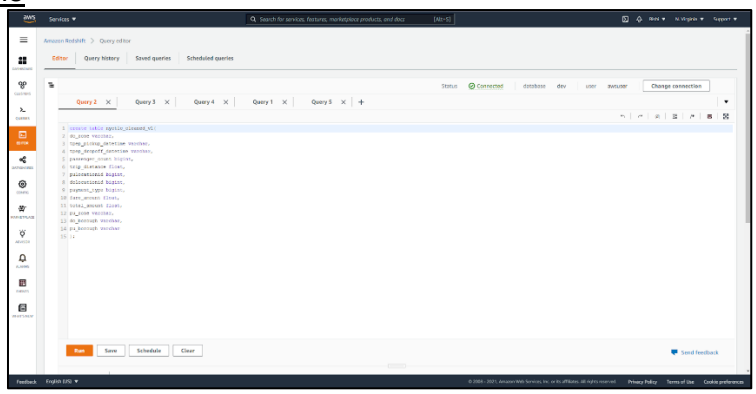

Configured Email in AWS SNS.

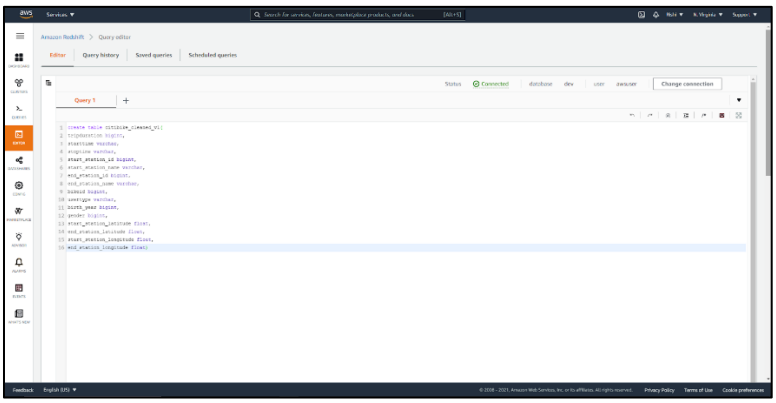



Successful Email Upon Completion of the Job




Redshift create table



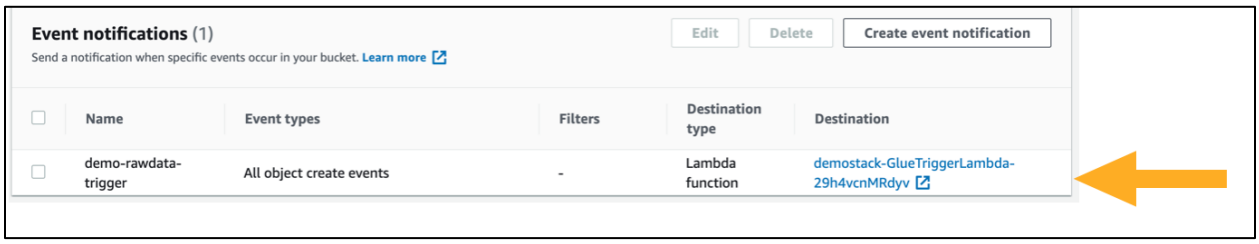


Demo data flow

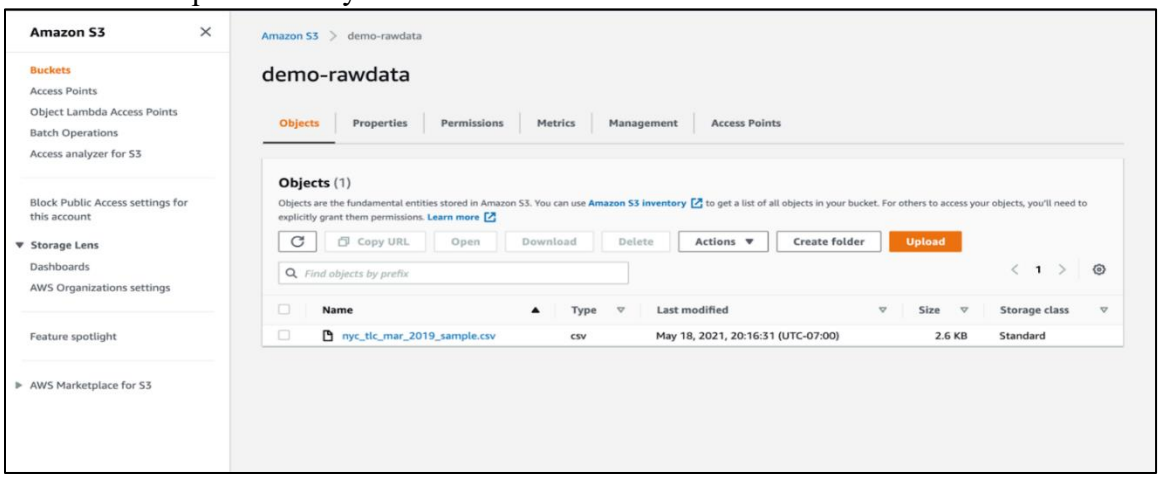
Created two S3 buckets namely “demo-rawdata” and “demo-processed-data”.
“demo-rawdata” stores the raw data and “demo-processed-data” bucket stores the processed data.

| | | | | |
|---|---|---|---|---|
| ○ | demo-processed-data | US East (N. Virginia) us-east-1 | Objects can be public | May 17, 2021, 00:13:12 (UTC-07:00) |
| ○ | demo-rawdata | US East (N. Virginia) us-east-1 | Objects can be public | May 17, 2021, 00:12:47 (UTC-07:00) |

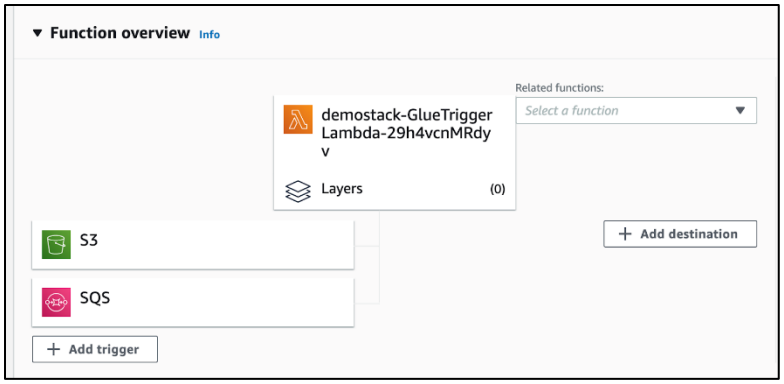
Created the S3 trigger name **demo-rawdata-trigger** in the”**demo-rawdata**” bucket. This trigger invokes the lambda function whenever new file is uploaded in the bucket.




Uploaded the Sample Data of yellow taxi in the S3 bucket named “demo-rawdata”.



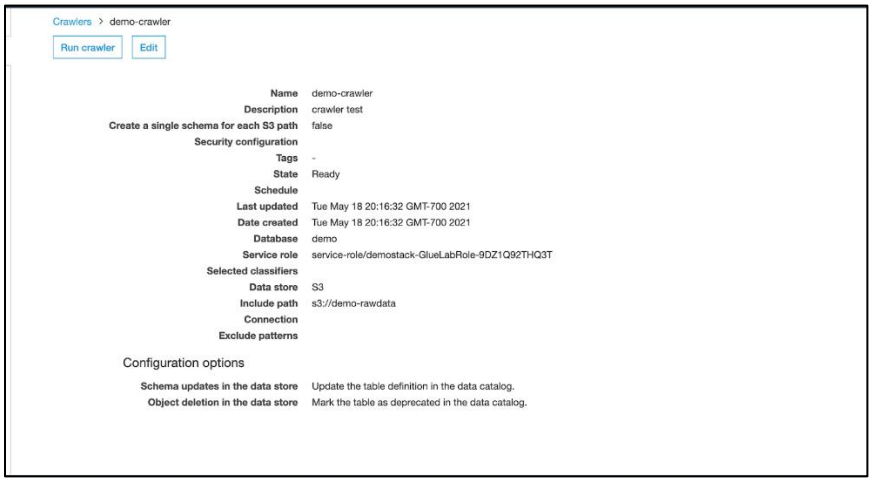

The Lambda function invoked above is used to create the crawler named “demo-crawler” and then creates the data catalog.  SQS trigger is implemented for the retry logic.




Code Snapshot for the above lambda function.

```
    })
}else{
var dbName = 'demo';
var params = {
    DatabaseInput: {
        Name: dbName,
        Description: 'Blog Post database',
    }
};
glue.createDatabase(params, function(err, data) {
        var params1 = {
            DatabaseName: dbName,
            Name: 'demo-crawler',
            Role: 'service-role/dl-data228-gluerole',
            Targets: {
                S3Targets: [{ Path: 's3://demo-rawdata' }]
            },
            Description: 'crawler test'
        };
        glue.createCrawler(params1, function(err1, data1) {
            startCrawler('demo-crawler', function(err2,data2){
                if(err2) callback(err2)
                else callback(null,data2)
            })
        });
```

demo-crawler is created by the invocation of the above lambda function.

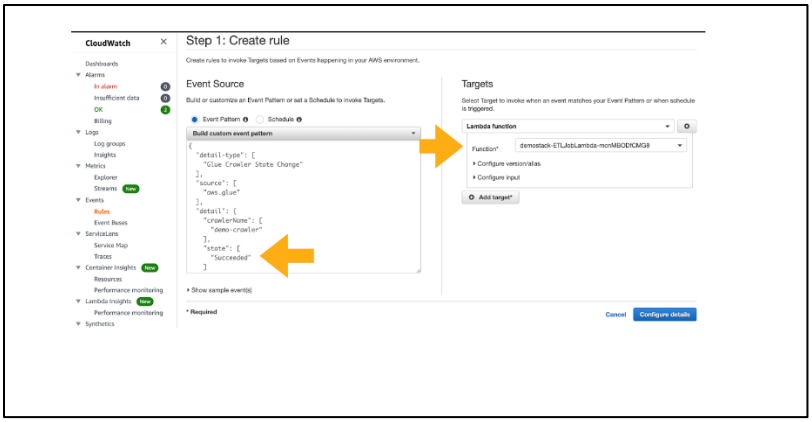



After the demo crawler crawls the data “demo-rawdata” catalog table is created.


Schema of the raw data

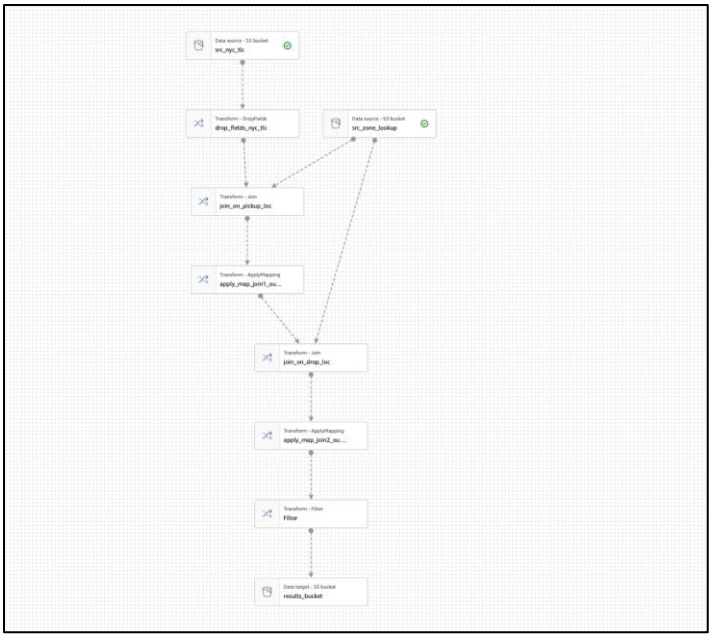
| | Column name | Data type | Partition ke |
|---|---|---|---|
| 1 | vendorid | bigint | |
| 2 | tpep_pickup_datetime | string | |
| 3 | tpep_dropoff_datetime | string | |
| 4 | passenger_count | bigint | |
| 5 | trip_distance | double | |
| 6 | ratecodeid | bigint | |
| 7 | store_and_fwd_flag | string | |
| 8 | pulocationid | bigint | |
| 9 | dolocationid | bigint | |
| 10 | payment_type | bigint | |
| 11 | fare_amount | double | |
| 12 | extra | double | |
| 13 | mta_tax | double | |
| 14 | tip_amount | double | |
| 15 | tolls_amount | double | |
| 16 | improvement_surcharge | double | |
| 17 | total_amount | double | |
| 18 | congestion_surcharge | double | |

An event rule is created in Cloud Watch that invokes another lambda function upon successful crawling of the data.

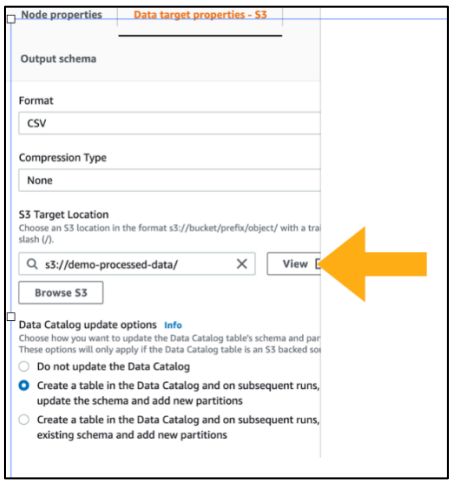



Code snapshot for the second lambda function. It invoke the ETL job named ‘demo-job’

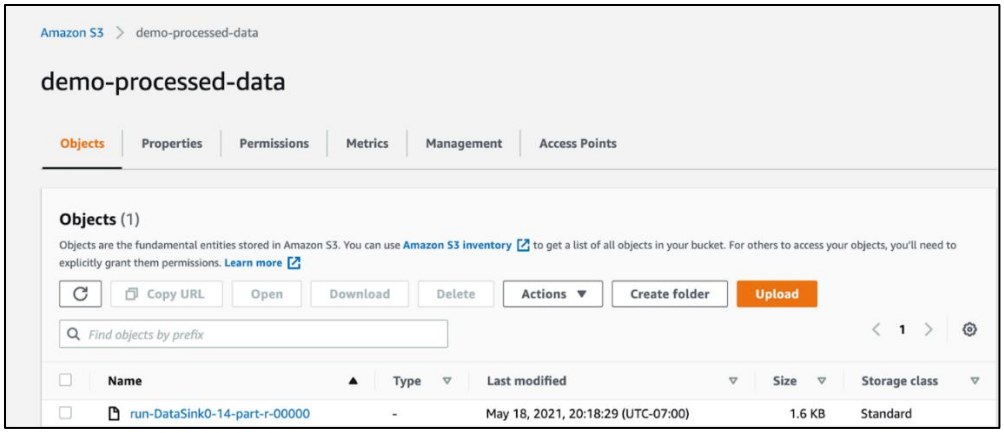

The ETL job is created using AWS Glue studio that cleans the data and transfers it to processed S3 bucket. Cleaning process involved dropping unnecessary columns, changing data types, filtering the data, and performing the joins with mapping files.



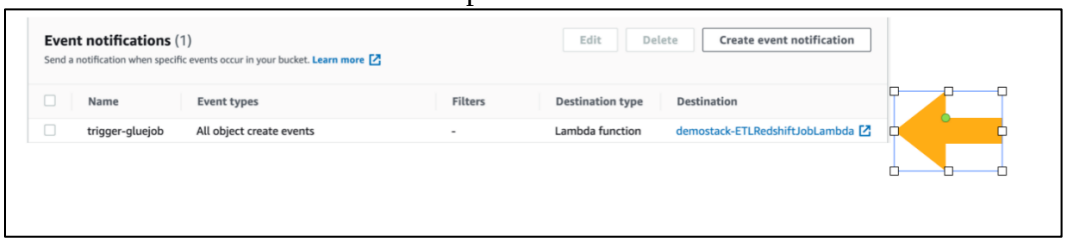
Processed Data is stored in S3 processed buckets.




The data is stored in processed S3 bucket.



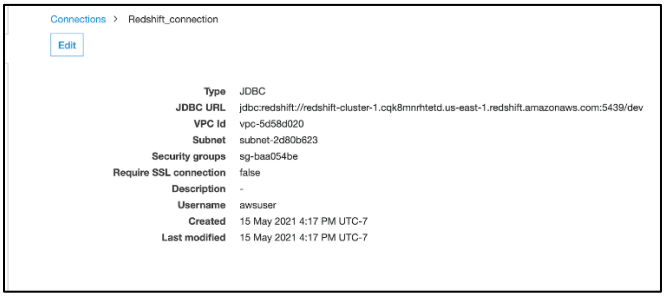

Created the S3 trigger name **trigger-gluejob** in the"**demo-processed-data**" bucket.This trigger invokes the lambda function as soon as the processed data comes.



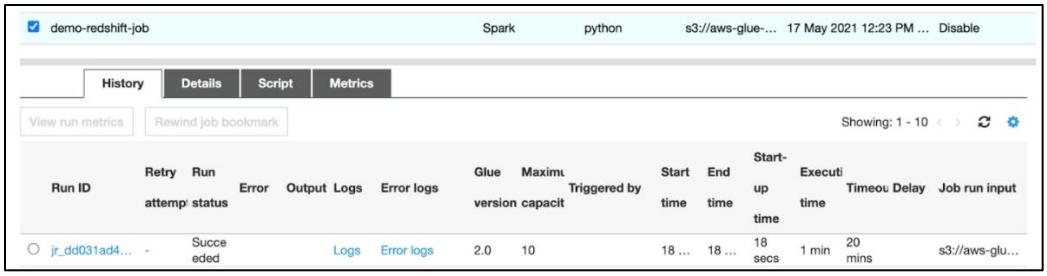

This Lambda function invokes" **demo-redshift-job**"

```
var AWS = require('aws-sdk');
var sns = new AWS.SNS( { region: "us-east-1" });
var s3 = new AWS.S3();
var glue = new AWS.Glue({apiVersion: '2017-03-31'});
    exports.handler = function(event, context, callback) {
        var params = {
            JobName: 'demo-redshift-job',
            Timeout: 20,
        };
        glue.startJobRun(params, function(err1, data1) {
            if (err1) {
                console.log(err1, err1.stack);}
            else {
                console.log(data1);}
        });
        console.log(JSON.stringify(event, null, 3));
    };
```

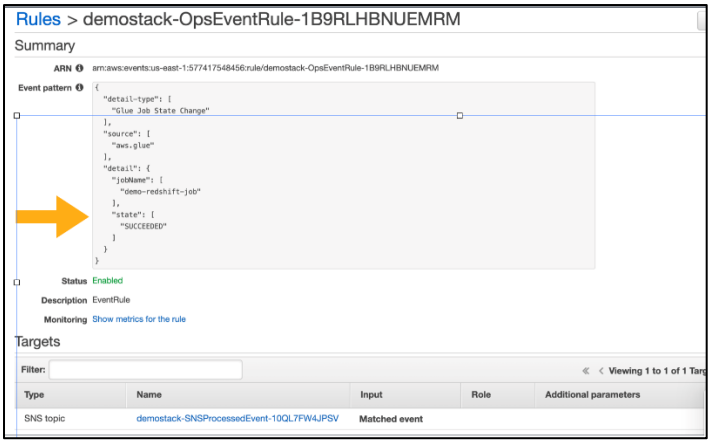
Configured the redshift Connection that is required by the **demo-redshift-job** in order to transfer the data from processed S3 bucket to Redshift.



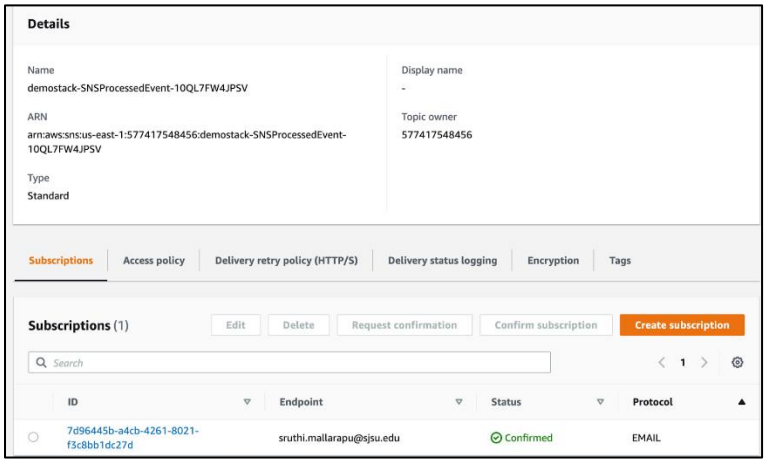

Successful execution of **demo-redshift-job**. It means data has transferred to Redshift tables



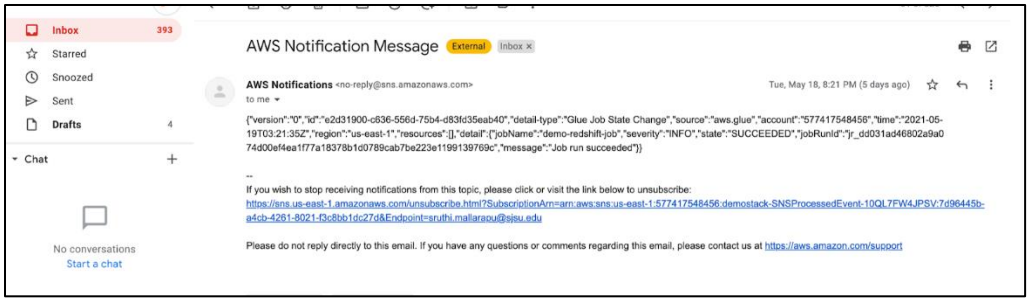

Another event rule is created in Cloud Watch that invokes SNS services. This service notifies through an email upon successful completion of the **demo-redshift-job**.




Configured Email in AWS SNS.




Successful Email Upon Completion of the Job




## Data analysis in Redshift

We stored our processed data as tables in Redshift and executed SQL queries for analysis.

As a first step, a cluster must be created to host the database instance. Subsequently, a database is created with name and password and becomes functional.

Redshift Cluster

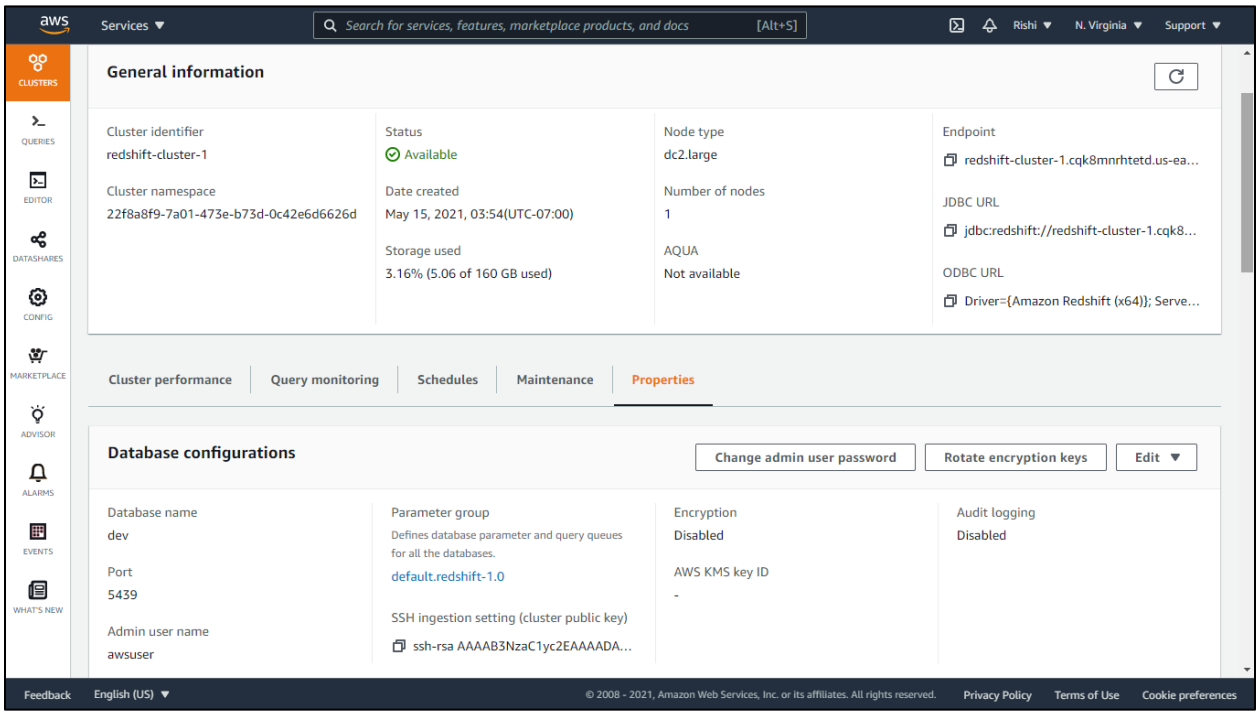



*Figure 4: Cluster created along with the database.*

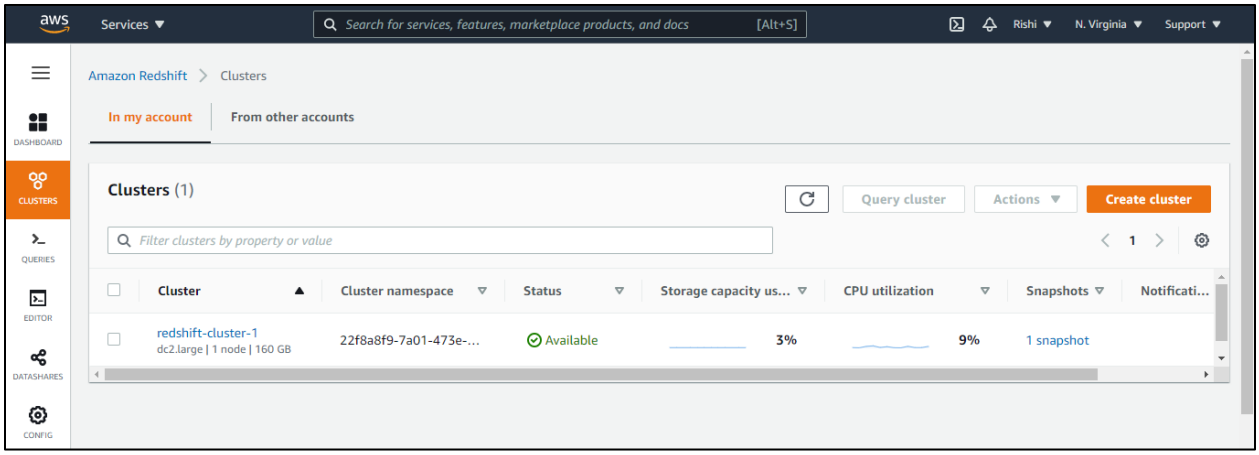



*Figure 5: Redshift cluster is in status Available, ready to query data for analysis.*

## Our Taxi and Citi bike data tables in Redshift

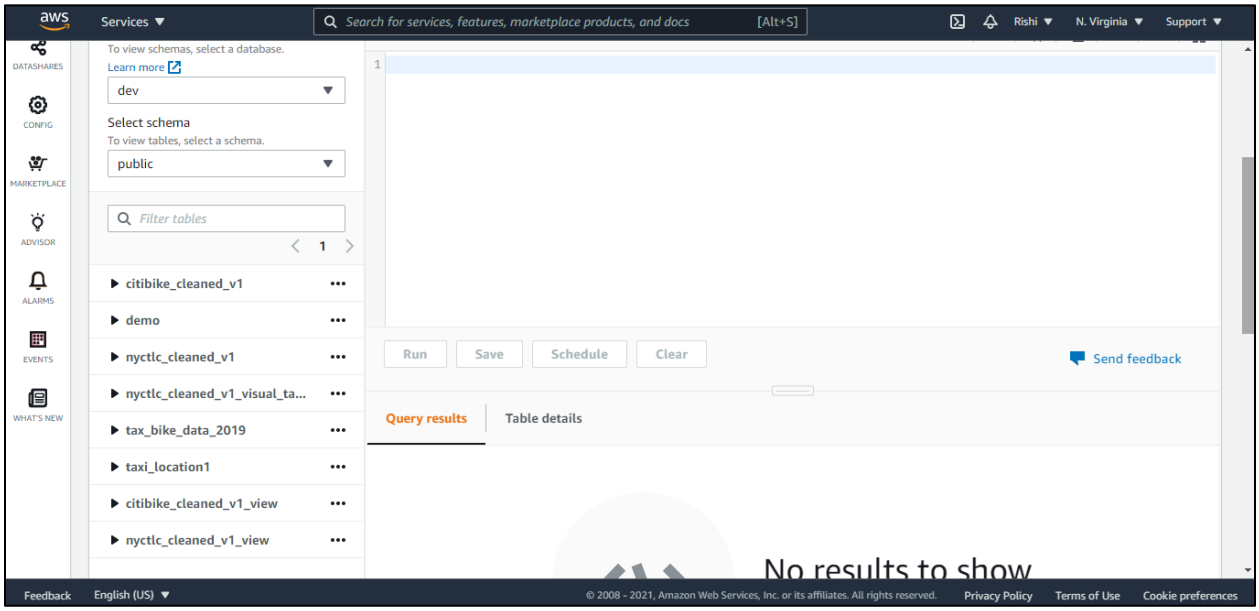



*Figure 6: Redshift data tables*

## Queries, loads and system performance parameters are tracked automatically by AWS

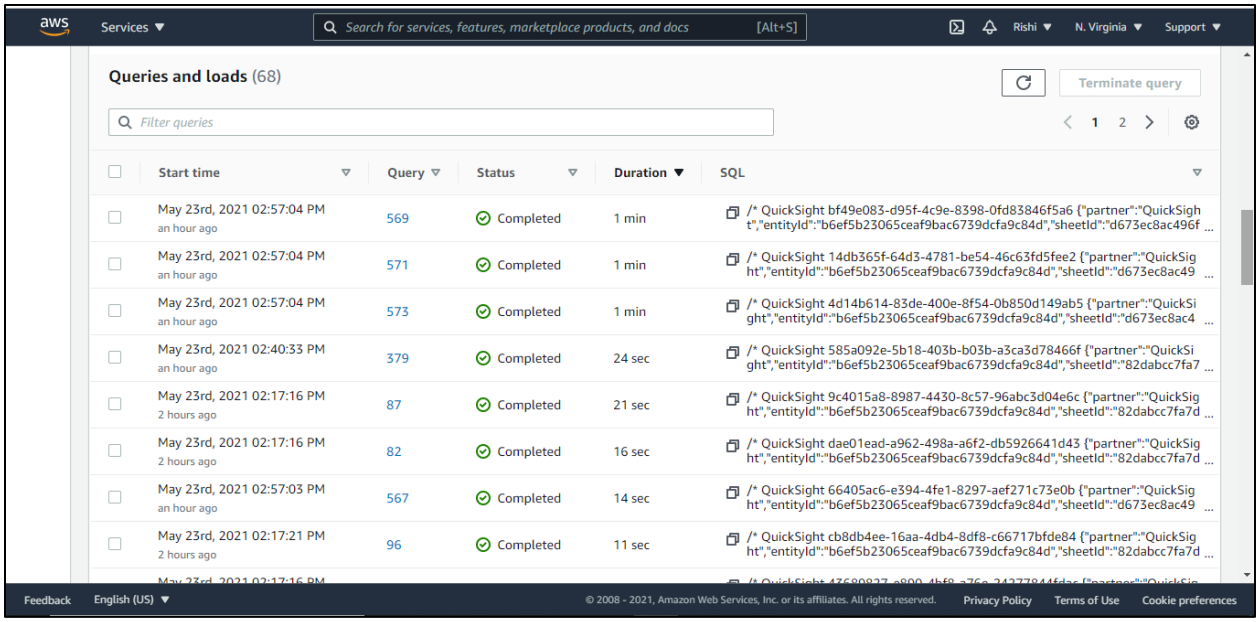



*Figure 7: Infrastructure managed by the cloud vendor*

## Some analysis using SQL:

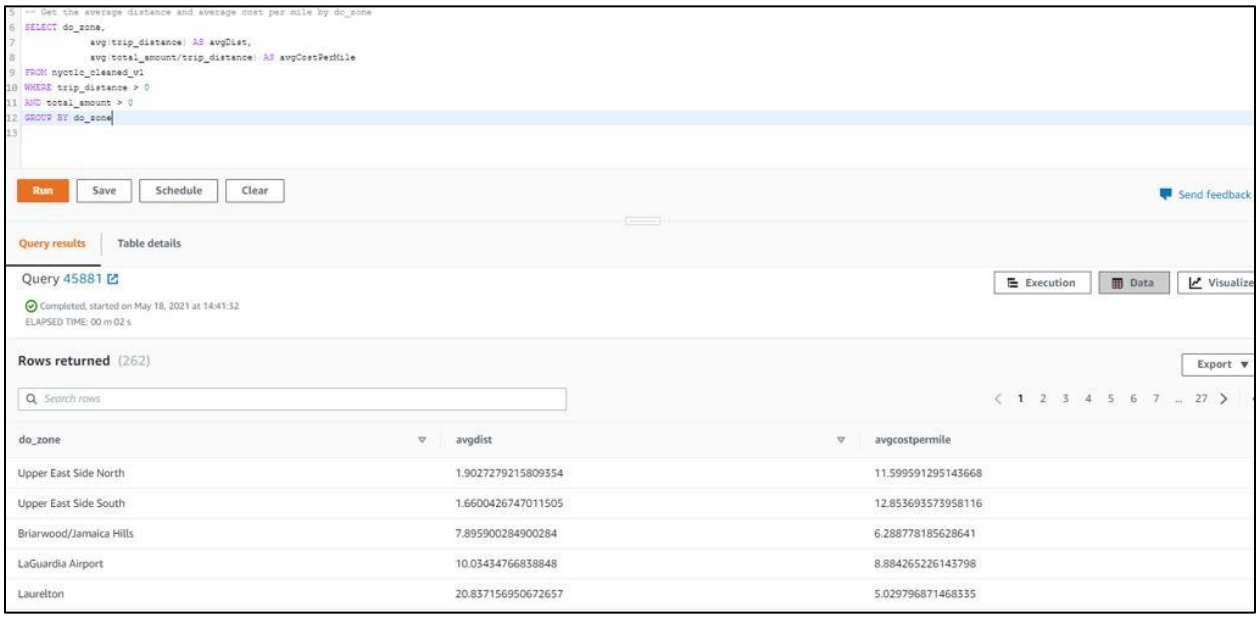



*Figure 8: SQL Analysis - Taxi data: Average distance, average cost per mile*

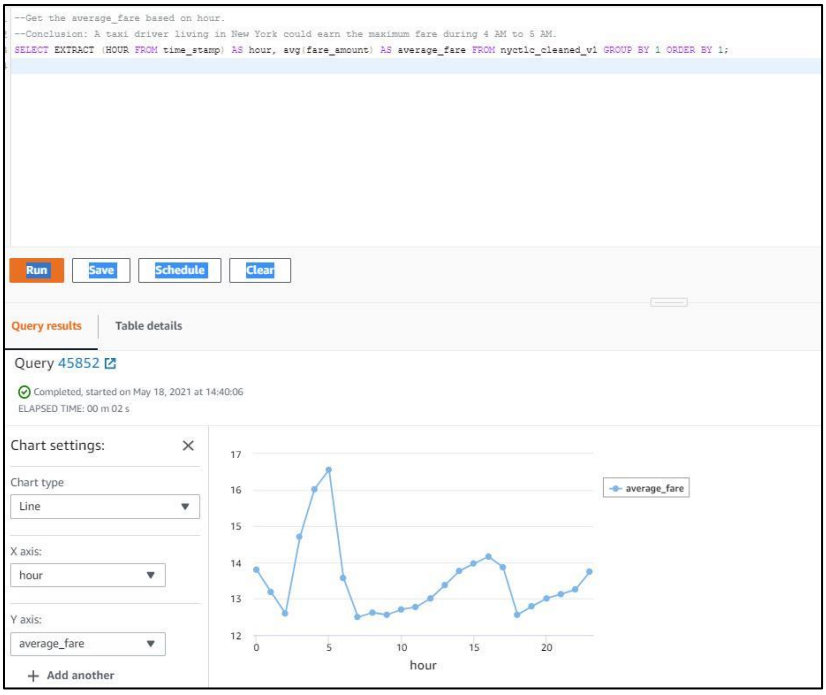



*Figure 9: SQL Analysis - Taxi data: Average fare based on hour.*

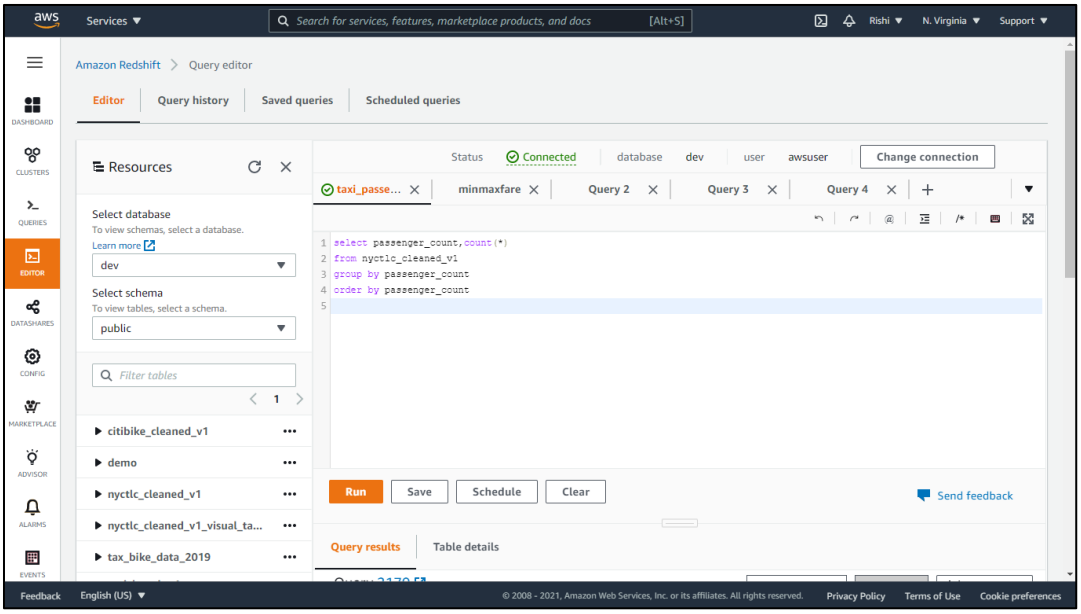



*Figure 10: SQL Query: Taxi data - Passenger count*

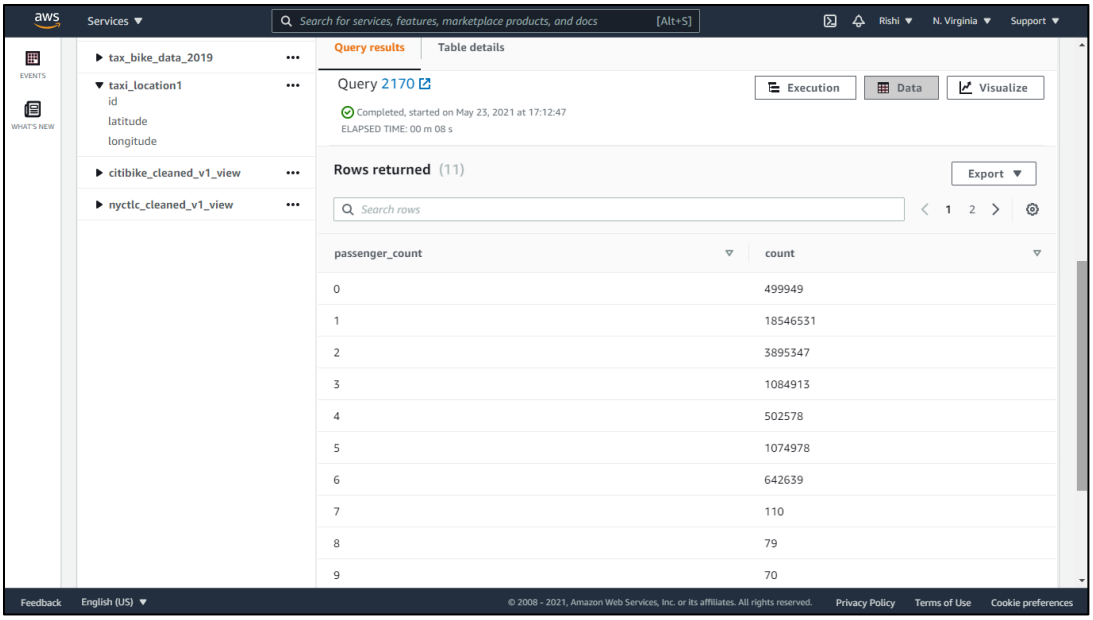



*Figure 11: SQL Query result - Passenger count*

## Data Visualizations in QuickSight

Data has been imported from Redshift to QuickSight.

### Connection process from Redshift to QuickSight:

Create an IAM role for Redshift to QuickSight and add the same role in VPC Security Group at Redshift.

### Datasets

The below datasets have been imported from AWS Redshift and will be used to generate our graphs and charts.

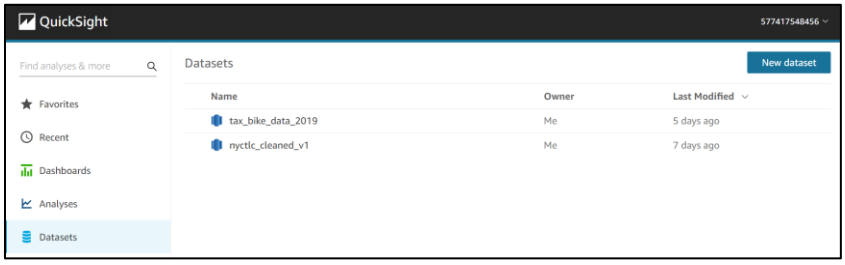



*Figure 12: Datasets with NYC yellow Taxi and Citi Bike data for visualizations*

### Analyses and Dashboard

From the above dataset, analysis was performed with the required visual type. We have options for filtering and performing any required action with the data.

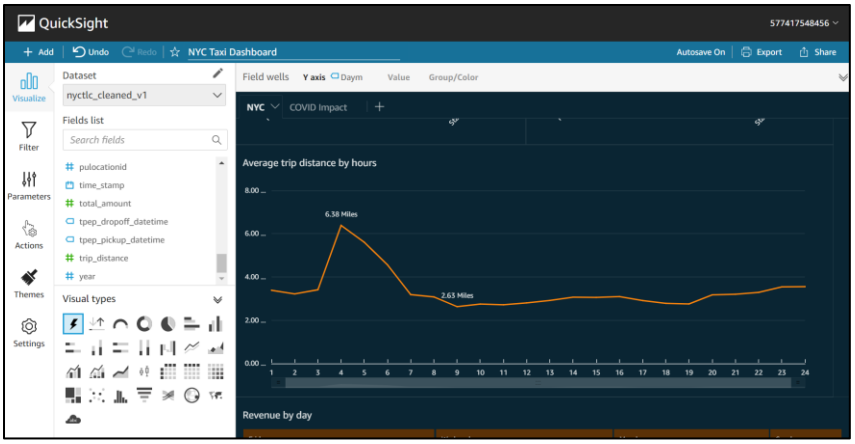



*Figure 13: Analysis window with menu options*

Once the graph and visualization are finalized, the same can be published as Dashboards using the ‘share’ option. The published dashboards can be viewed under Dashboards menu in QuickSight.

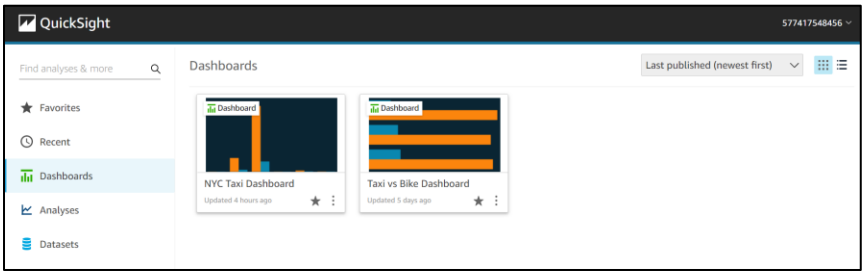


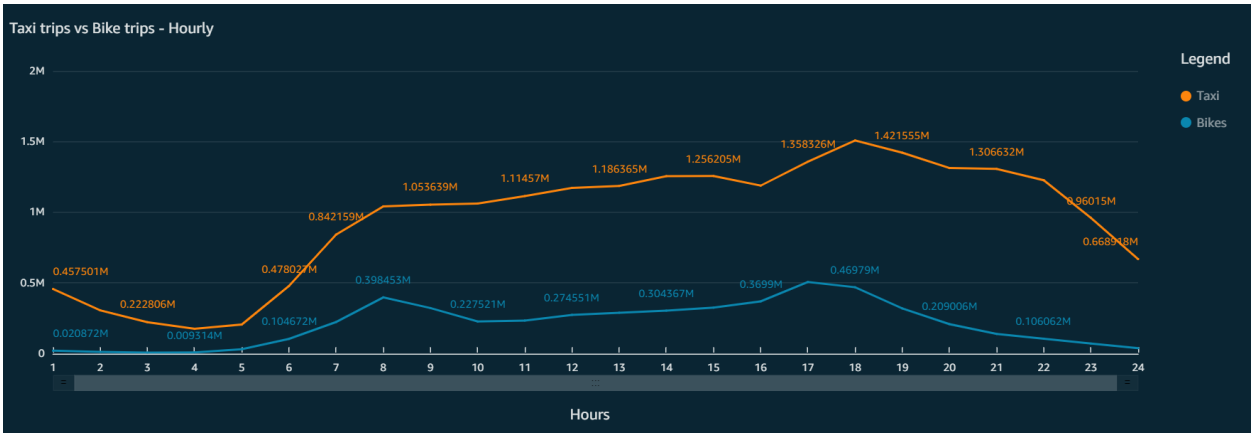

*Figure 14: Dashboards menu*

**Visualization for NYC – Yellow Taxi and Citi Bike:**

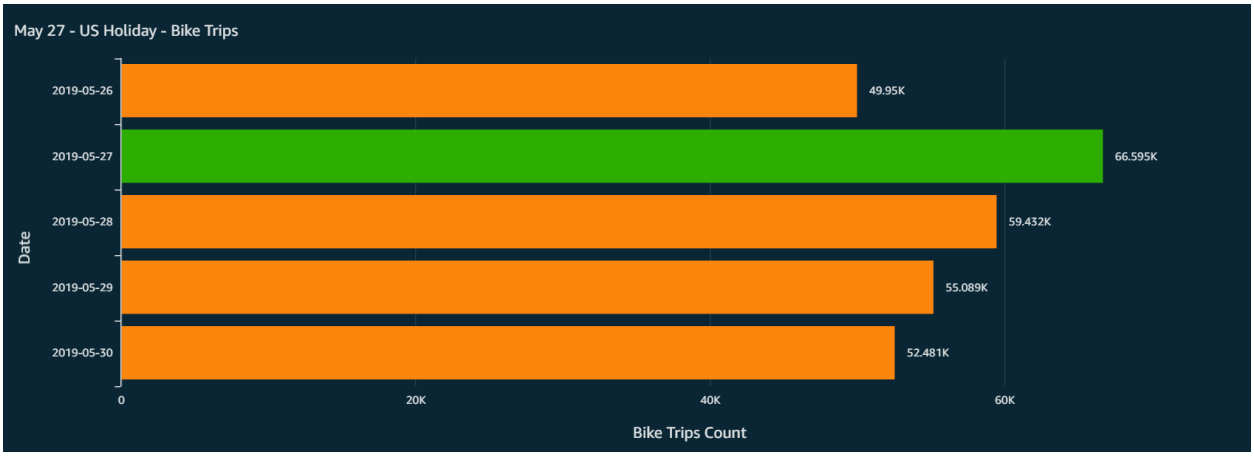
The below bar graph represents the count of trips comparison between the Taxi and Citi Bike. Here, we could see the increase in bike trips as month goes on.

In the below bar graph, we have segregated the trips of both Taxi and Bike based on the day. We have analyzed that, peak day for taxi is Friday and bike is Wednesday. The lowest trips for Taxi and Bike are on Sunday.

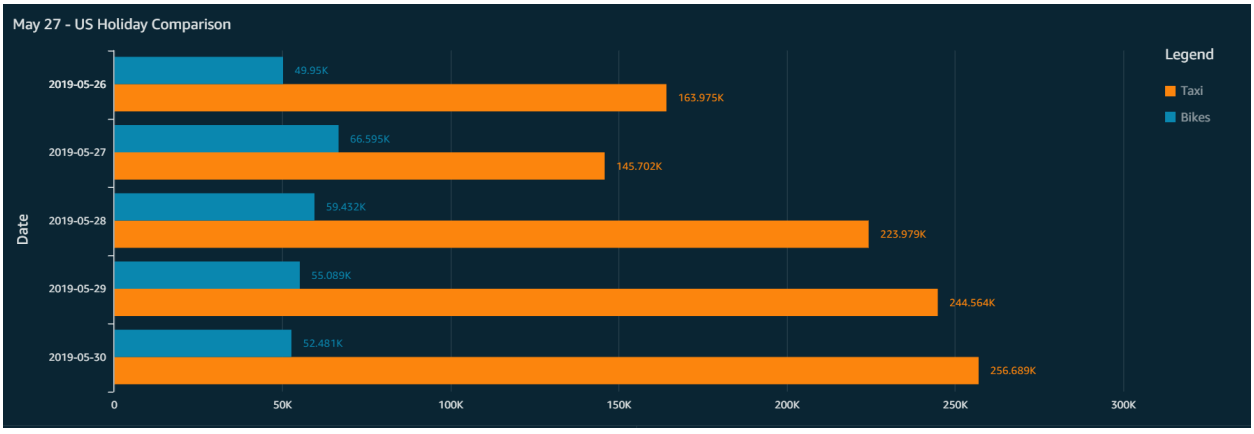

The below line graph implies the peak booking count based on hours and we analyzed that 8 am is the highest on morning and 5 pm for Bike and 6 pm for taxi is the highest trips witnessed hours of the day. This time is claimed to be the office and school's closing time.




We made analysis for May 27, 2019 from our dataset, which was a public holiday, and we could see, on that particular day, the Bike trips is comparatively high on the week.




The same public holiday trips count has been analyzed for both Taxi and Bike in a single graph and the outcome was negative to Taxi.

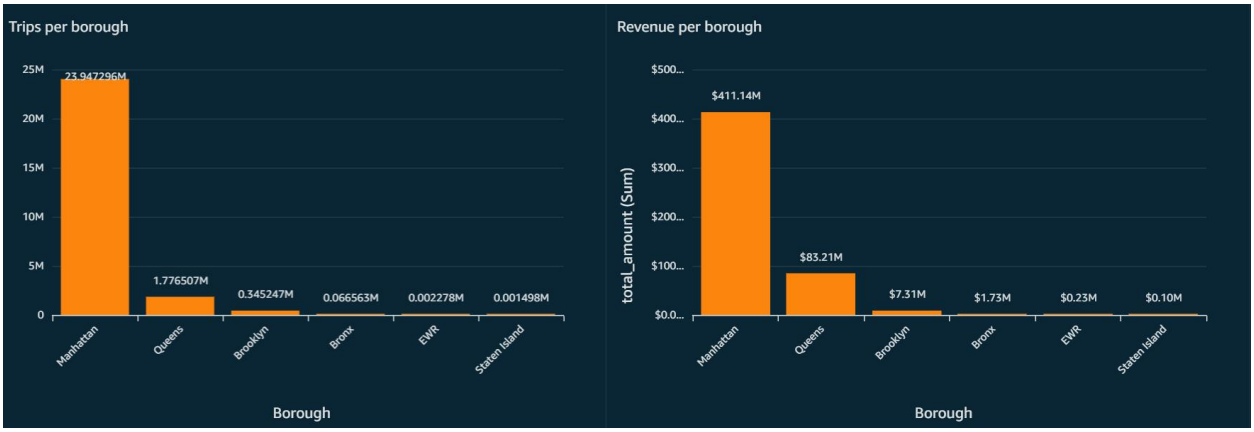



From our dataset, we have found that the average minutes for both bike and taxi to increase the availability.

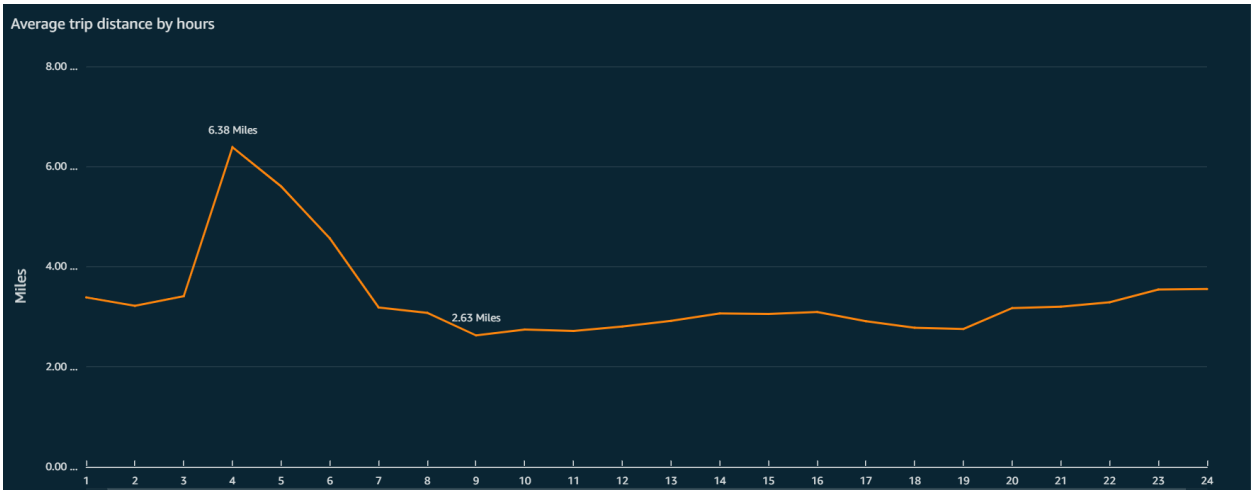

**Visualization for NYC yellow Taxi and COVID-19 impact:**

As yellow taxi generates significant revenue for NYC, we were additionally inspired to perform some analysis on the same and the effects of COVID on the business.

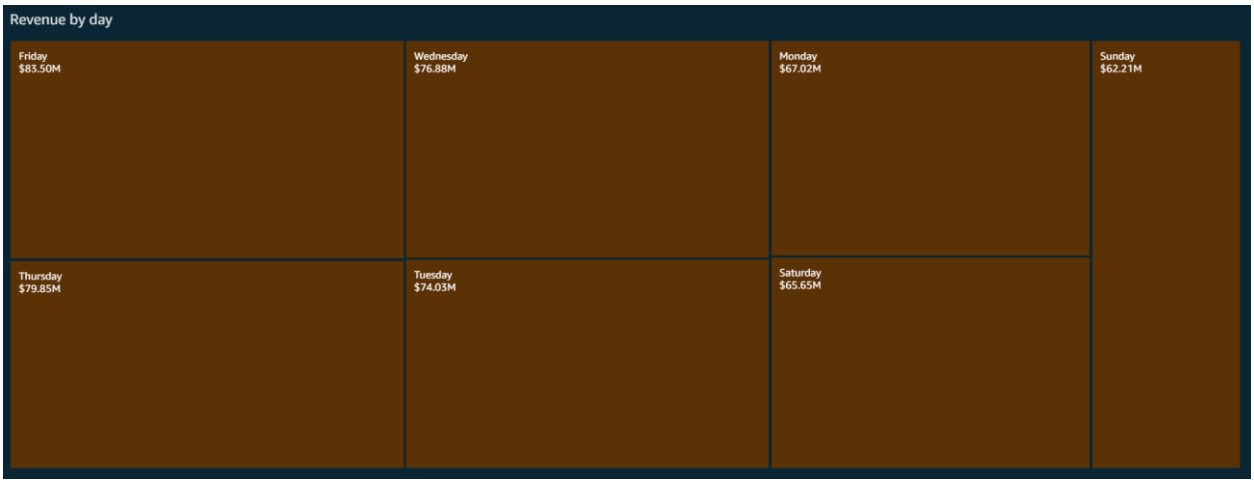
The below graph represents the trip count and revenue of top 6 boroughs. Manhattan City holds the highest trip count and the highest revenue.



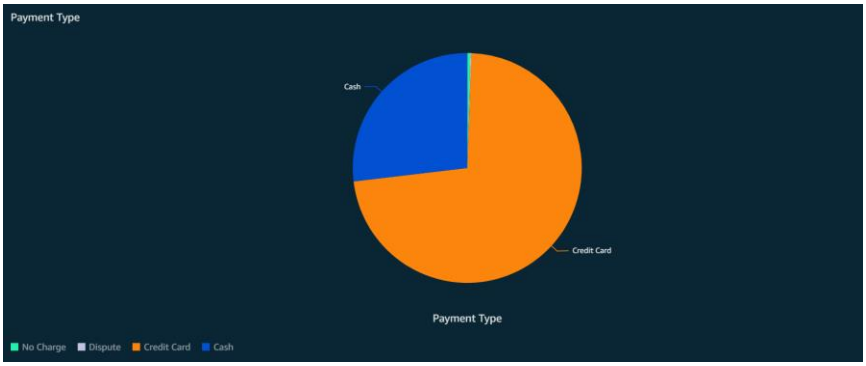

This graph provides information about the average distance (Miles) travelled based on hours and the result is that 4 am has the highest with average of 6.38 Miles and the lowest occurs at 9 am with 2.63 Miles.



The tree map below shows the revenue generated by NYC on the each day of the week. Friday, being the start of the weekend holds the highest revenue compared to other days.

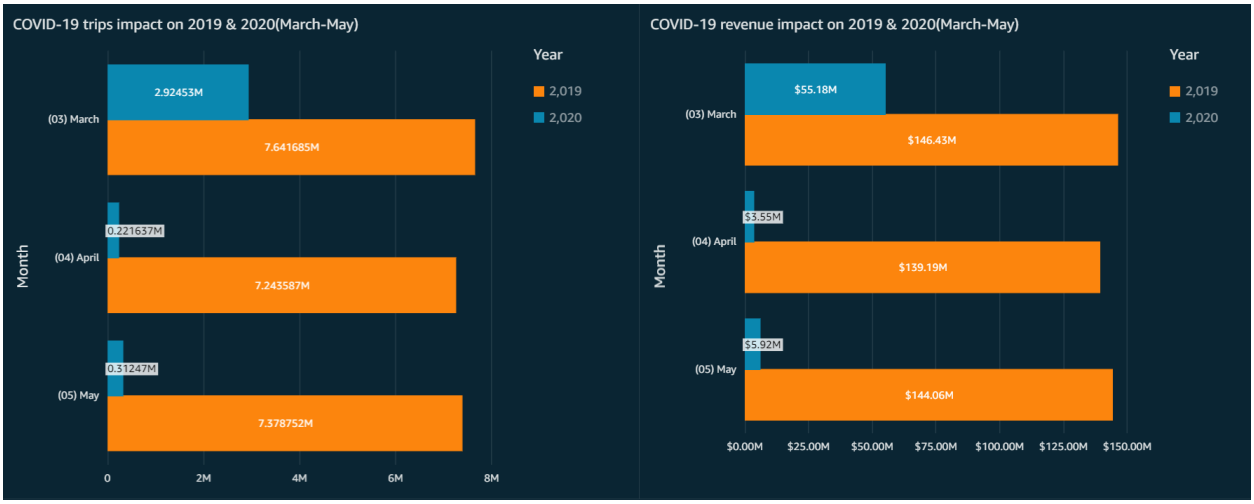



The pie chart below represents the payment types used by the customers.

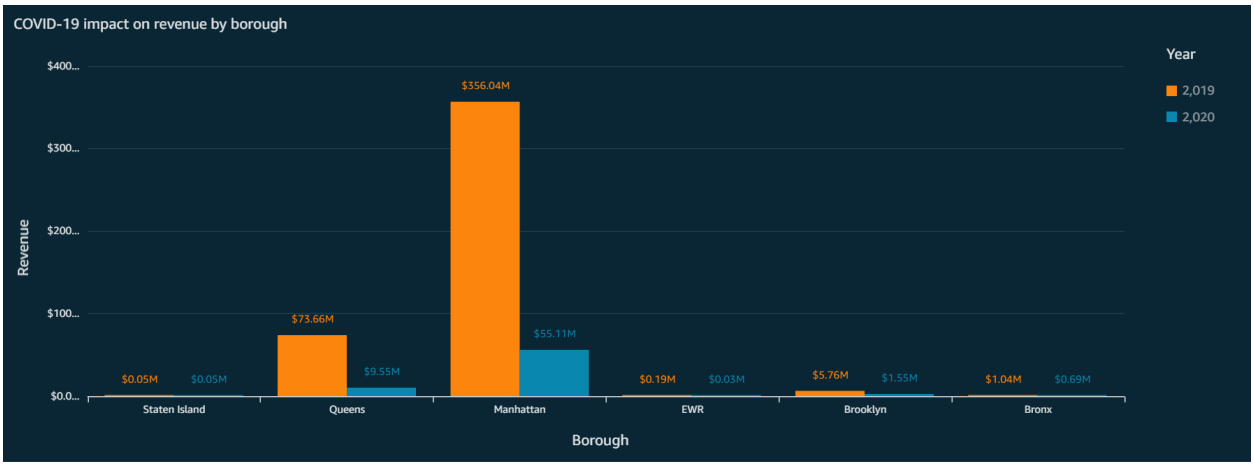



## COVID-19 Impact:

The bar graph below clearly shows the impact of the global pandemic and shows the trips and revenue impact of the month of March – May in 2019 and 2020.



The revenue impact on the business due to COVID-19 is analyzed for the top 6 boroughs on the below bar graph.




## VI. Team Members and Roles

As a group, we invested some time in fixing on the goals of this project and brainstorming ideas. It involved discussions from choosing the dataset, coming up with relevant use-cases and to finalizing the tools we will be using to implement the project.

Once fixed on our datasets and the tools, we tried to break up our project tasks into parts and assigned ourselves a significant portion of the project. For example, one of us proceeded to process the raw data, while the other tried to learn about visualization. In this way, many different components of our project were simultaneously developed.

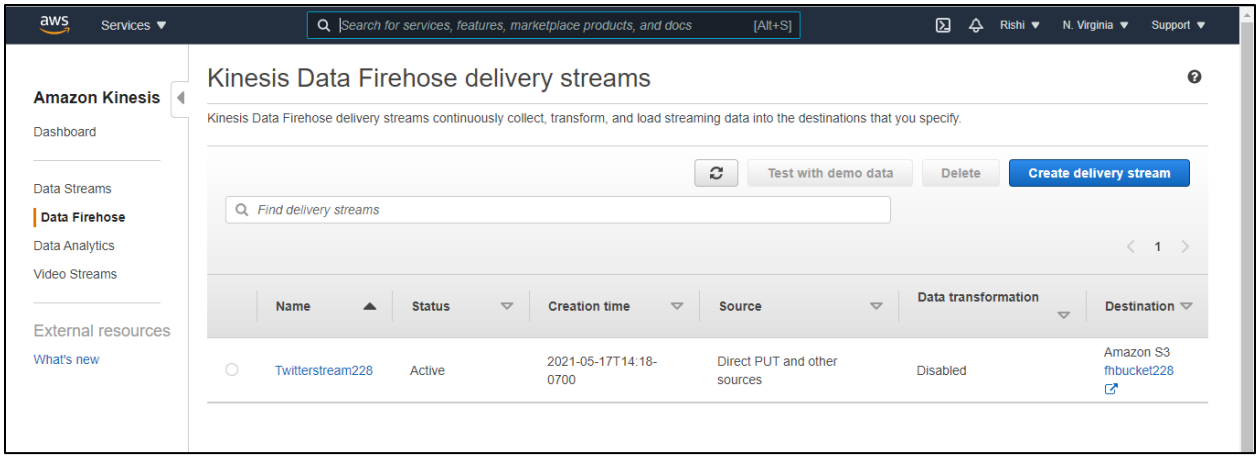
From time to time, we worked together by means of recurring meetings and weekly sprints. During the meetings, everyone shared their updates along with challenges faced and the entire team would brainstorm to solve those challenges. As a result, the entire team was always updated and involved in all parts of the project.

## VII. Future Scope

We believe this project can be improved further by adding even more data or processes. One such endeavor we took up was to combine real-time data with the data we already had.

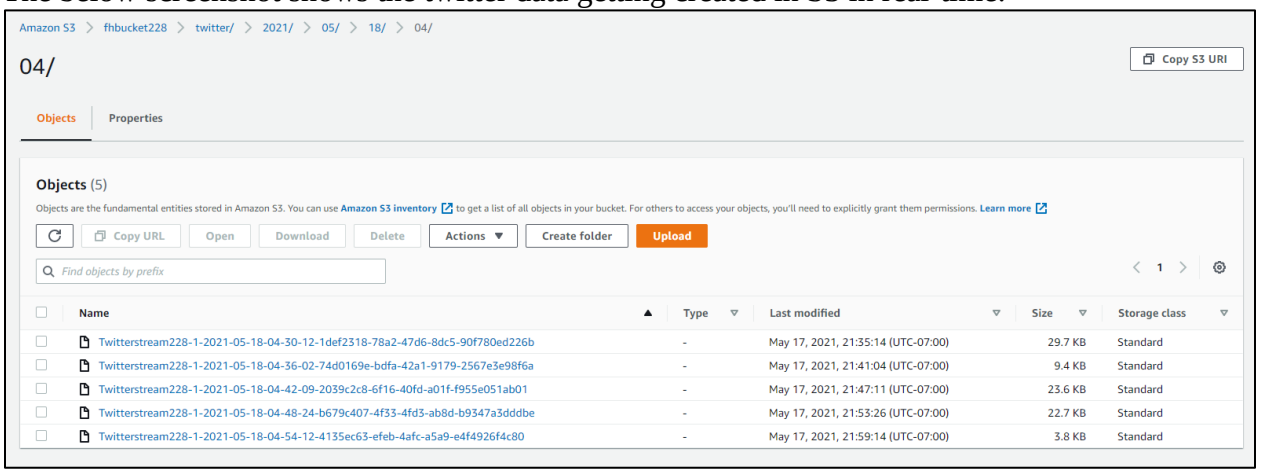
We wanted to know how useful the Twitter data of NYC TLC is to the public and the business. So we pulled real-time Twitter data using node.js scripts and loaded our S3 buckets with the tweets using Amazon Kinesis Data Firehose delivery stream.

The below screenshot shows the Kinesis data stream ‘Twitterstream228’, created to push real-time tweets to S3 bucket named ‘fhbucket228’.




The below screenshot shows the twitter data being pulled using Twitter API by running node.js scripts.

The below screenshot shows the twitter data getting created in S3 in real-time.




This twitter data can be loaded into Redshift or 'Elasticsearch' and Kibana for data analysis. This data analysis can be also integrated with our data for comparison and further analysis.

Additionally, we also converted shapefile available in the taxi data repository to geojson using online software convertors and loaded this data into DynamoDB for visualization. Unfortunately, the conversion was flawed and the latitude, longitude values were not helpful. We are planning to debug this more and look for alternative methods to extract location field values form shapefiles.

```
        geojson_name=r'C:\Users\vijay\Documents\A\SJSU\01.BigData\
        dynamodb_table_name='taxi1')
    print(f"{NUM_FEATURES_UPLOADED} features in data uploaded to d

Connecting to Dynamodb...
Enter your aws_access_key:
AKIAYM4GESKUNZEWT5UL
Enter your aws_secret_access_key:
HnRDOZCM8z0EJv4+uEvr4Nb1WxJA9O+mxenbmzjW
Creating new table...
Uploading segments to table...
263 features in data uploaded to dynamodb
```

*Figure 15: Python script uploaded data to DynamoDB*

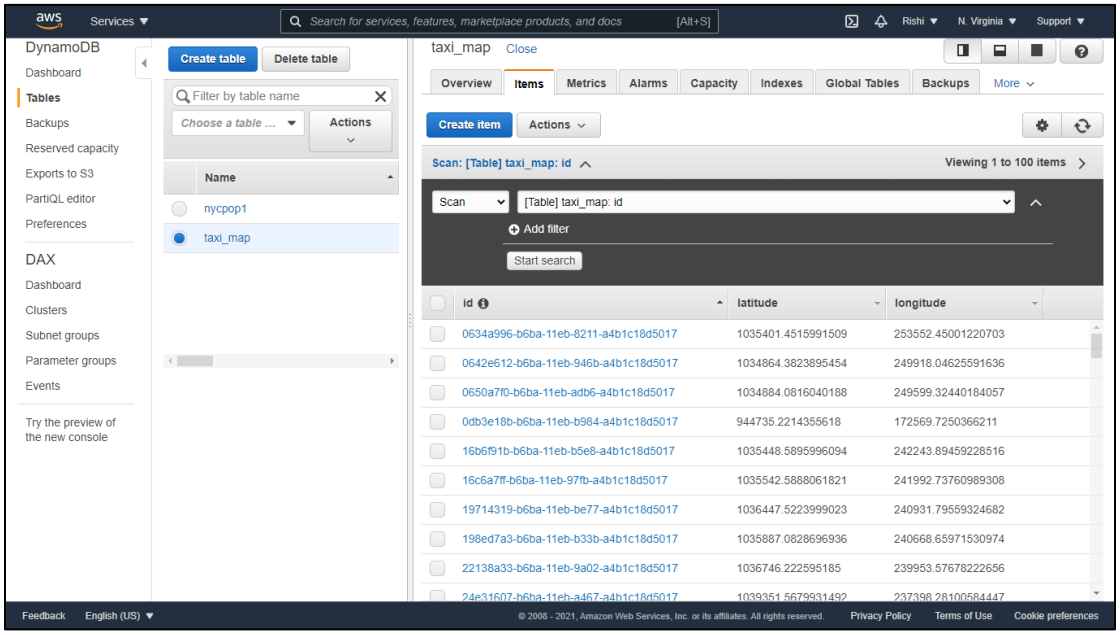



*Figure 16: Latitude, longitude data is invalid.*

## VIII. Learnings and Conclusion

We were able to achieve our goals by leveraging cloud services provided by AWS. We worked on big data and applied analysis on big data. As a result, this project helped us significantly in

- Exploring plethora of AWS services like automated ETL jobs that saves the time and manual effort and applies to huge datasets.
- Uncovering issues and debug errors related to cloud technology and software. IAM is one aspect that was totally new and was required to be properly configured for every step to work.

- Reading, understanding and interpreting logs - AWS creates logs for every action using its in-built service ‘CloudWatch’ which is useful in debugging errors with any of the services.
- Uncovering issues with big data and set those right. Loading, processing, debugging issues with big data took time and made us realize the significance of cloud infrastructures in today’s expansive data world.
- Exploring analytics by performing querying and join operations on tables with millions of rows made possible due to fast, fully managed, petabyte scale Redshift.
- Bringing about visualizations in the form of graphs and charts in AWS QuickSight.

By the end of the project, we were able to learn a lot about cloud technology and AWS and we attribute our successful completion to our professor and ISA. We sincerely thank the professor for this opportunity and his every lecture and guidance. We also thank our ISA for continued and detailed guidance during every stage of the project.

## IX. References


- https://www1.nyc.gov/site/tlc/index.page
- https://zacks.one/aws-athena-lab/
- https://docs.aws.amazon.com/ses/latest/DeveloperGuide/event-publishing-redshift-firehose-stream.html
- NYC TLC data source : https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2020-01.csv
- Citi bike data: https://s3.amazonaws.com/tripdata/index.html
- https://docs.aws.amazon.com/code-samples/latest/catalog/code-catalog-python-example_code-kinesis.html
- https://docs.aws.amazon.com/quicksight/latest/user/enabling-access-redshift.html