SCHOOL OF INFORMATICS & IT

Individual Solution

Student Name (Matric Number)	: 2302276G
Tutorial Group	: PC02
Tutor	: Goh Rui Quan
Submission Date	: 27/7/24

Declaration of Originality

I am the originator of this work and I have appropriately acknowledged all other original sources used as my references for this work.
I understand that Plagiarism is the act of taking and using the whole or any part of another person's work, including work generated by AI, and presenting it as my own.

I understand that Plagiarism is an academic offence
and if I am found to have committed or abetted the offence of plagiarism in relation to this submitted work, disciplinary action will be enforced.

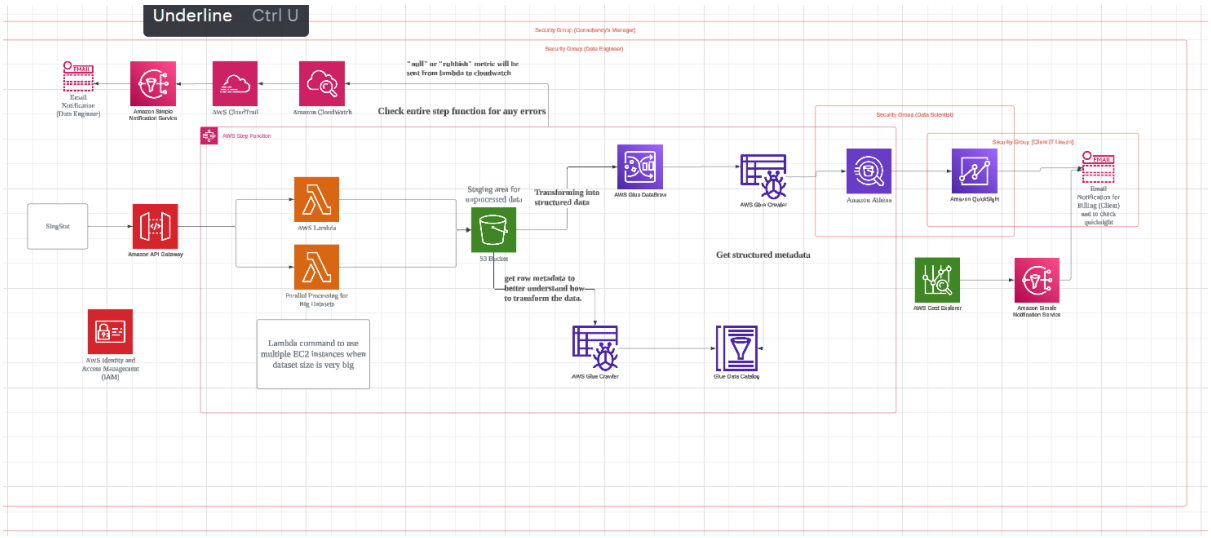
Declaration on the use of Generative AI tools for assignments

Describe how you have used Generative AI tools such as ChatGPT or Dall.E-2 in your assignment. Show snapshots of the conversations with the AI tool (i.e., the prompts you used and the response you get from the AI tool).
https://chatgpt.com/share/ae6db300-c143-4f03-9d81-6b6bc328197a Used it for steps on how to download the request library in AWS Lambda https://chatgpt.com/share/f47c0c9d-10db-4ad5-a0fb-97099f2fb843 Asked ChatGPT for ideas on how to fix the error where the data does not show up in Athena.
How do you indicate the reference? The content generated by AI tools are not retrievable except by the user who generated them, so they are considered non-recoverable sources. Although non-recoverable data or quotations in APA Style papers are usually cited as personal communications, with ChatGPT-generated text there is no person communicating. Quoting text from ChatGPT chat is therefore more like sharing the output of an algorithm, with a reference list entry and the corresponding in-text citation. According to the official APA Style site, ChatGPT references should be cited as: E.g. OpenAI. (2023). *ChatGPT* (Sep 25 version) [Large language model]. https://chat.openai.com/chat

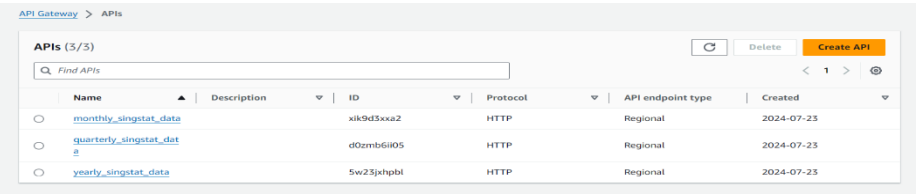
Important Note:

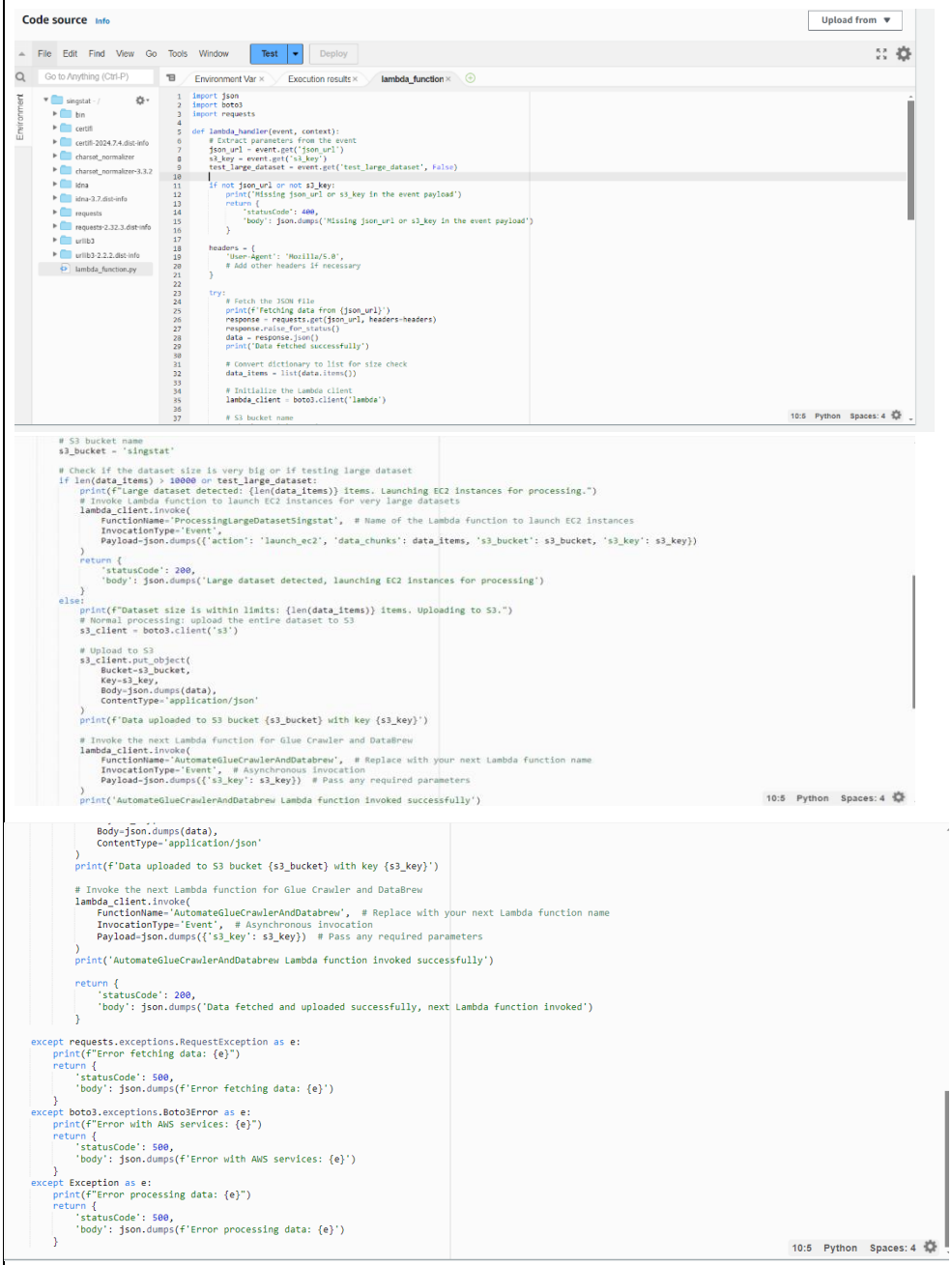
- Do not copy answers produced by the AI tool in totality as it is considered as plagiarism.
- Do not rely on any information produced by the AI tool blindly. You should always verify the answer with other sources. Do not assume that these answers provided by the AI tool are correct.
- To achieve quality outputs from the AI tool, you should provide good prompt that is clear and specific. Be precise and provide context. Avoid asking open-ended questions.

Pipeline Overview:



Solution Implementation

Component	Setup, reason, additional notes (for data integrity, budget, etc)
API Gateway	 Created three Api's for the 3 datasets
Lambda	**Singstat function** 



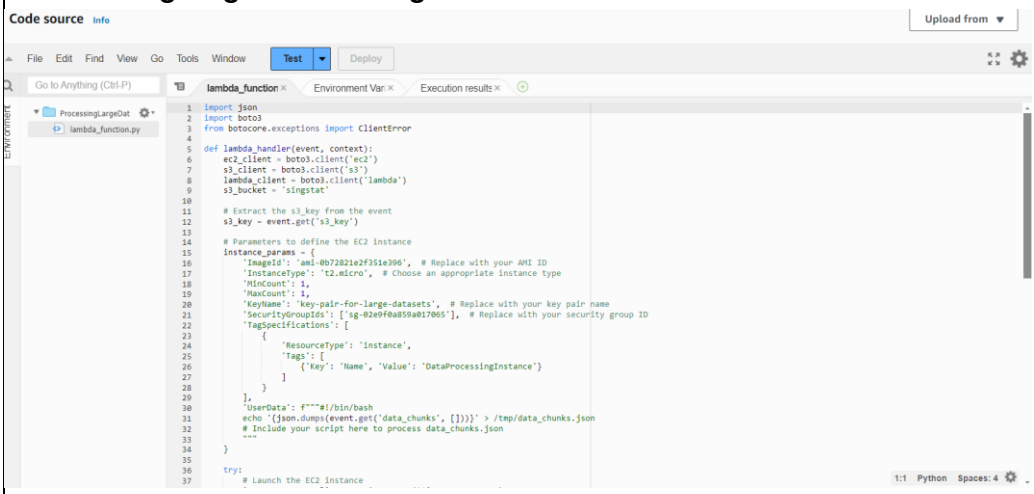
Created a Lambda function named Singstat which uploads JSON data to an S3 bucket after processing it from a given URL. It begins by obtaining the parameters (s3_key and json_url) from the test event. This is where the user will put the link to the json data from singstat and the s3 destination. If either is absent, an error is returned.

To check the amount of the dataset, it then uses the requests library to retrieve the JSON data and converts it to a list of objects. It calls another Lambda function (ProcessingLargeDatasetSingstat) to process the data using EC2 instances if the dataset is large.

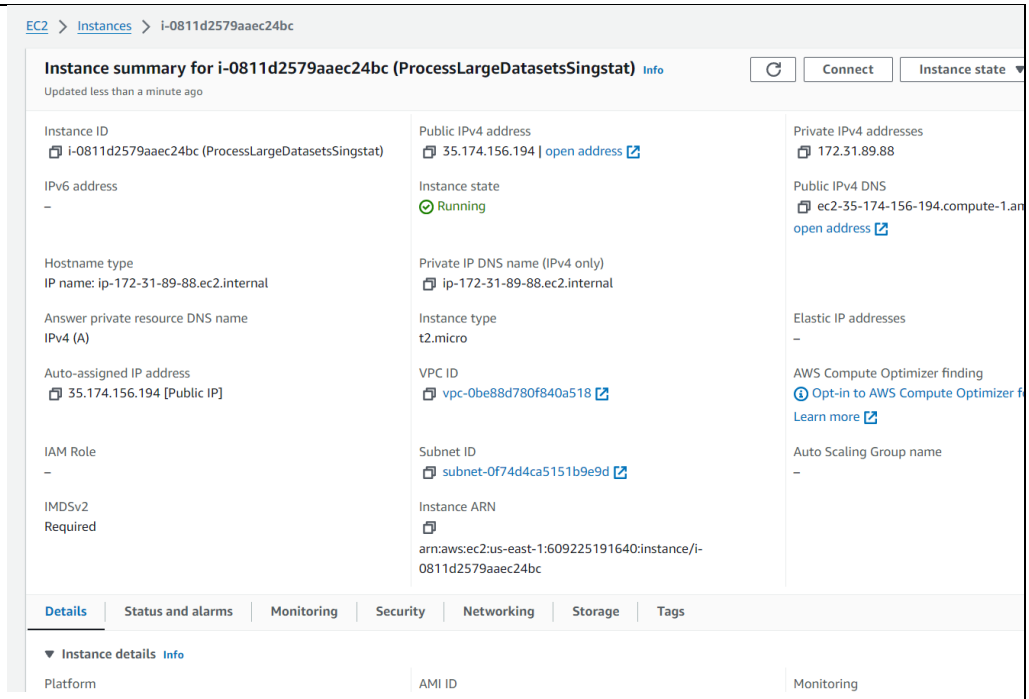
For datasets of a reasonable size, it uploads the information to the S3 bucket (singstat) and into a folder called untransformedsingstatdata, in either the monthly,yearly or quarterly folder. The user specifies which folder in the test event. This is done because most singstat data are either monthly, annually, or quarterly.

It then uses the AutomateGlueCrawlerAndDatabrew Lambda function for more processing.

ProcessingLargeDatasetSingstat function




```
except Exception as e:
    error_message = str(e)
    print(f"Error launching EC2 instance: {error_message}")
    return {
        'statusCode': 500,
        'body': json.dumps({'error': 'Exception', 'message': error_message})
    }
```



This AWS Lambda function launches an EC2 instance to process large datasets, uploads the data to an S3 bucket and then calls another Lambda function to process the data even more.

The first step involves obtaining the s3_key from the test event in the singstat function and specifying the AMI ID, instance type, key pair, security group, and user data script to manage the data chunks for the EC2 instance.

After that, the function tries to start the EC2 instance and uploads the data chunks to the designated S3 bucket (singstat) and into the untransformedsingstatdata folder and into either monthly, quarterly, or yearly just like the singstat function when it succeeds. Following a successful upload of the data, it immediately calls the AutomateGlueCrawlerAndDatabrew Lambda function to handle the data further.

When problems occur, the method returns the relevant error messages and status codes and handles possible errors using ClientError from boto3 and general exceptions.

AutomateGlueCrawlerAndDatabrew function

```python
import json
import boto3
import time

def lambda_handler(event, context):
    glue_client = boto3.client('glue')
    databrew_client = boto3.client('databrew')
    athena_client = boto3.client('athena')

    # Extract parameters from the event
    s3_key = event.get('s3_key')

    if not s3_key:
        print("Error: Missing s3_key in the event payload")
        return {
            'statusCode': 400,
            'body': json.dumps('Missing s3_key in the event payload')
        }

    print(f"Received s3_key: {s3_key}")

    # Convert s3_key to lowercase for case insensitive comparison
    s3_key_lower = s3_key.lower()

    # Names for the Glue Crawler and DataBrew Job
    untransformed_crawler_name = 'crawl_untransformed_singstat_data'  # Name of your Glue Crawler for untransformed data
    transformed_crawler_name = 'crawl_transformed_singstat_data'  # Name of your Glue Crawler for transformed data

    # Determine the job name and Athena query based on the prefix
    if s3_key_lower.startswith('singstat_data/untransformedsingstatdata/monthly/'):
        databrew_job_name = 'MonthlyTransformSingstatData'
        athena_query = "SELECT * FROM monthly"
    elif s3_key_lower.startswith('singstat_data/untransformedsingstatdata/quarterly/'):
        databrew_job_name = 'QuarterlyTransformSingstatData'
        athena_query = "SELECT * FROM quarterly"
    elif s3_key_lower.startswith('singstat_data/untransformedsingstatdata/yearly/'):
        databrew_job_name = 'YearlyTransformSingstatData'
```

1:1 Python Spaces: 4

```python
    databrew_job_name = 'YearlyTransformSingstatData'
    athena_query = "SELECT * FROM yearly"
else:
    print("Error: File does not match any known prefixes")
    return {
        'statusCode': 400,
        'body': json.dumps('File does not match any known prefixes')
    }

try:
    # Start the Glue Crawler for untransformed data
    glue_client.start_crawler(Name=untransformed_crawler_name)
    print('Glue Crawler for untransformed data started successfully')

    # Wait for the untransformed data crawler to finish
    while True:
        response = glue_client.get_crawler(Name=untransformed_crawler_name)
        state = response['Crawler']['State']
        if state == 'READY':
            print('Glue Crawler for untransformed data has finished')
            break
        print('Waiting for the untransformed data crawler to finish...')
        time.sleep(30)

    # Start the DataBrew Job
    databrew_client.start_job_run(Name=databrew_job_name)
    print('DataBrew job started successfully')

    # Wait for the DataBrew job to finish
    while True:
        job_run = databrew_client.list_job_runs(Name=databrew_job_name, MaxResults=1)
        state = job_run['JobRuns'][0]['State']
        if state == 'SUCCEEDED':
            print('DataBrew job has finished successfully')
            break
        elif state in ['FAILED', 'STOPPED']:
            print(f"DataBrew job did not complete successfully. State: {state}")
            return {
```

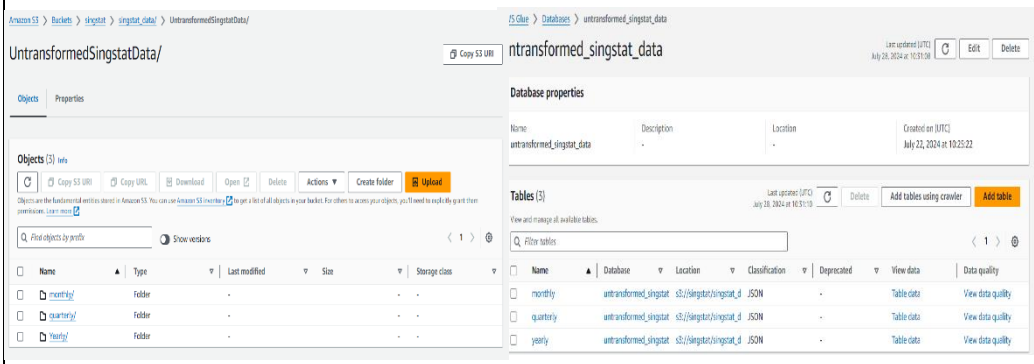
1:1 Python Spaces: 4

lambda_function × Environment Var × Execution results ×

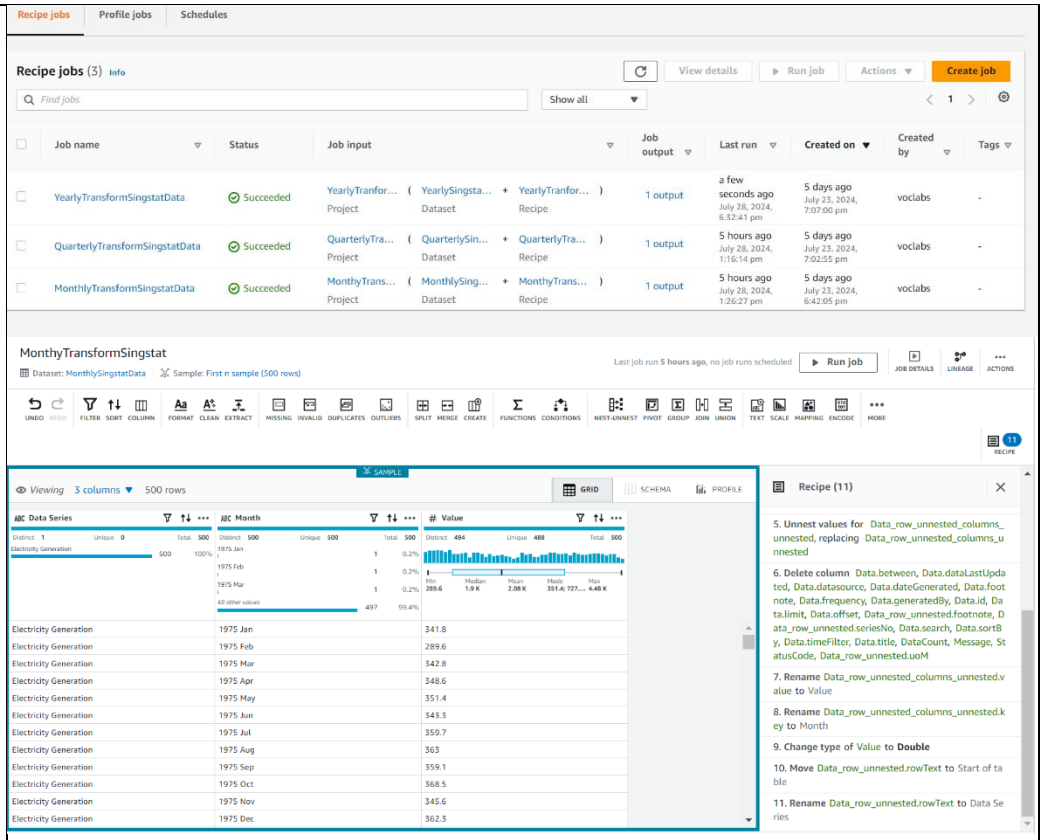
```
                return {
                    'statusCode': 500,
                    'body': json.dumps(f"DataBrew job did not complete successfully. State: {state}")
                }
            print('Waiting for the DataBrew job to finish...')
            time.sleep(30)

        # Start the Glue Crawler for transformed data
        glue_client.start_crawler(Name=transformed_crawler_name)
        print('Glue Crawler for transformed data started successfully')

        # Wait for the transformed data crawler to finish
        while True:
            response = glue_client.get_crawler(Name=transformed_crawler_name)
            state = response['Crawler']['State']
            if state == 'READY':
                print('Glue Crawler for transformed data has finished')
                break
            print('Waiting for the transformed data crawler to finish...')
            time.sleep(30)

        # Run the Athena query
        print(f"Running Athena query: {athena_query}")

        # Start the Athena query execution
        response = athena_client.start_query_execution(
            QueryString=athena_query,
            QueryExecutionContext={
                'Database': 'transformed_singstat_data'
            },
            ResultConfiguration={
                'OutputLocation': 's3://singstat/singstat_data/QueriedSingstatData/'
            }
        )
        query_execution_id = response['QueryExecutionId']
        print(f"Started Athena query with execution ID: {query_execution_id}")
```

```
    # Check the Athena query execution status
    state = 'RUNNING'
    while state in ['RUNNING', 'QUEUED']:
        response = athena_client.get_query_execution(QueryExecutionId=query_execution_id)
        state = response['QueryExecution']['Status']['State']
        if state == 'SUCCEEDED':
            print("Athena query succeeded!")
            break
        elif state in ['FAILED', 'CANCELLED']:
            print(f"Athena query did not complete successfully. State: {state}")
            return {
                'statusCode': 500,
                'body': json.dumps(f"Athena query did not complete successfully. State: {state}")
            }
        print('Waiting for the Athena query to finish...')
        time.sleep(10)

    return {
        'statusCode': 200,
        'body': json.dumps('Glue Crawler, DataBrew job, and Athena query started and finished successfully')
    }

except Exception as e:
    print(f"Error: {e}")
    return {
        'statusCode': 500,
        'body': json.dumps(f"Error: {e}")
    }
```

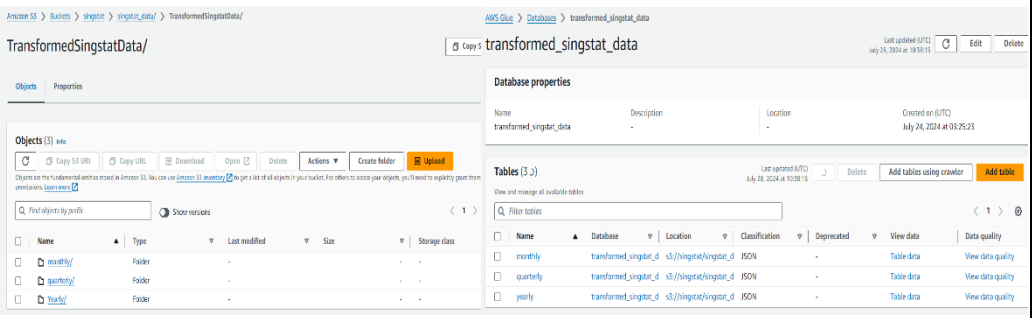
This AWS Lambda function consists of AWS Glue, DataBrew, and Athena operations to process and query data stored in the untransformedsingstatdata folder.



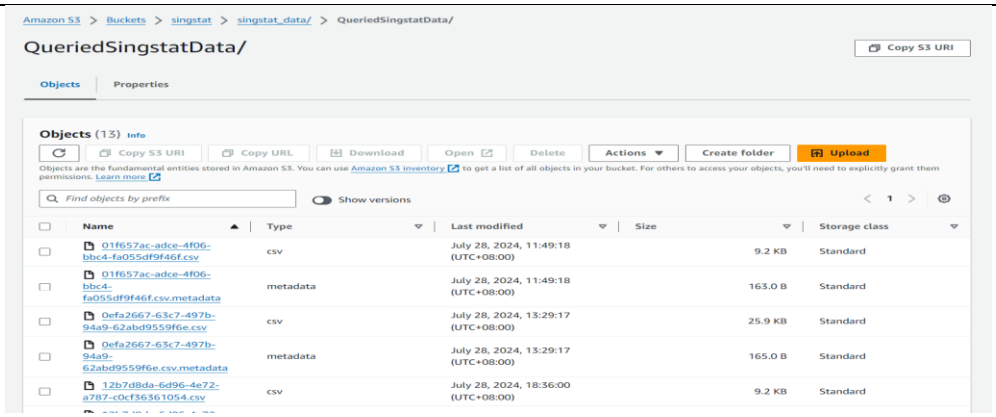
First, it crawls the untransformedsingstatdata folder using the crawl_untransformed_singstat_data crawler and uploads it to the untransformed_singstat_data database.



It then takes the s3_key out of the test event in singstat function and uses its prefix to identify which DataBrew task and Athena query to use. For example, singstat_data/untransformedsingstatdata/monthly/ for the MonthlyTransformSingstatData databrew job.



Once the DataBrew job completes successfully, the transformed data is then uploaded to the transformedsingstatdata folder and into either the monthly, quarterly, or yearly folder. Then, the crawl_transformed_singstat_data crawler is used to crawl the transformedsingstat data folder and uploads it to the transformed_singstat_data database.



To analyse the data that has been processed, a select Athena query is finally run. The queries are different for each type of dataset. For example, SELECT * FROM monthly for month and SELECT * FROM quarterly for quarter. All the Athena queries go to the QueriedSingstatData folder.

In addition to providing suitable error handling and status updates throughout, the function incorporates checks at each stage to guarantee that the tasks are completed successfully.

SingstatNotifications function

```
import boto3
import json
import datetime
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    # Set up AWS clients
    s3_client = boto3.client('s3')
    ce_client = boto3.client('ce')
    sns_client = boto3.client('sns')

    # Define the S3 bucket and file keys
    s3_bucket = 'singstat'  # Replace with your S3 bucket name
    powerbi_chart_key = 'singstat_data/PowerBIChartCostExplorerReport/YearlyTransformSingstatData.pbix'  # Replace with the path to your Power BI file in S3

    # Generate the AWS Cost Explorer report
    try:
        end = datetime.date.today()
        start = end - datetime.timedelta(days=5)
        response = ce_client.get_cost_and_usage(
            TimePeriod={
                'Start': start.strftime('%Y-%m-%d'),
                'End': end.strftime('%Y-%m-%d')
            },
            Granularity='DAILY',
            Metrics=['UnblendedCost'],
            Filter={
                'Not': {
                    'Dimensions': {
                        'Key': 'RECORD_TYPE',
                        'Values': ['Credit']
                    }
                }
            }
```

```
    )
except Exception as e:
    logger.error(f"Error fetching cost and usage data: {e}")
    return {
        'statusCode': 500,
        'body': json.dumps('Error fetching cost and usage data')
    }

# Extract the total cost
try:
    total_cost = response['ResultsByTime'][0]['Total']['UnblendedCost']['Amount']
except KeyError as e:
    logger.error(f"Key error while extracting total cost: {e}")
    return {
        'statusCode': 500,
        'body': json.dumps('Error extracting total cost')
    }

# Create the cost report content
cost_report_content = f"Total cost for {start} to {end} (excluding credits): ${total_cost}"

# Generate a unique key for the cost report based on the date
cost_report_key = f"singstat_data/PowerBIChartCostExplorerReport/cost_report_{end.strftime('%Y-%m-%d')}.csv"

# Upload the cost report to S3
try:
    s3_client.put_object(Bucket=s3_bucket, Key=cost_report_key, Body=cost_report_content)
except Exception as e:
    logger.error(f"Error uploading cost report to S3: {e}")
    return {
        'statusCode': 500,
        'body': json.dumps('Error uploading cost report to S3')
    }

# Generate a pre-signed URL for the cost report (expires in 7 days)
try:
    cost_report_url = s3_client.generate_presigned_url(
```

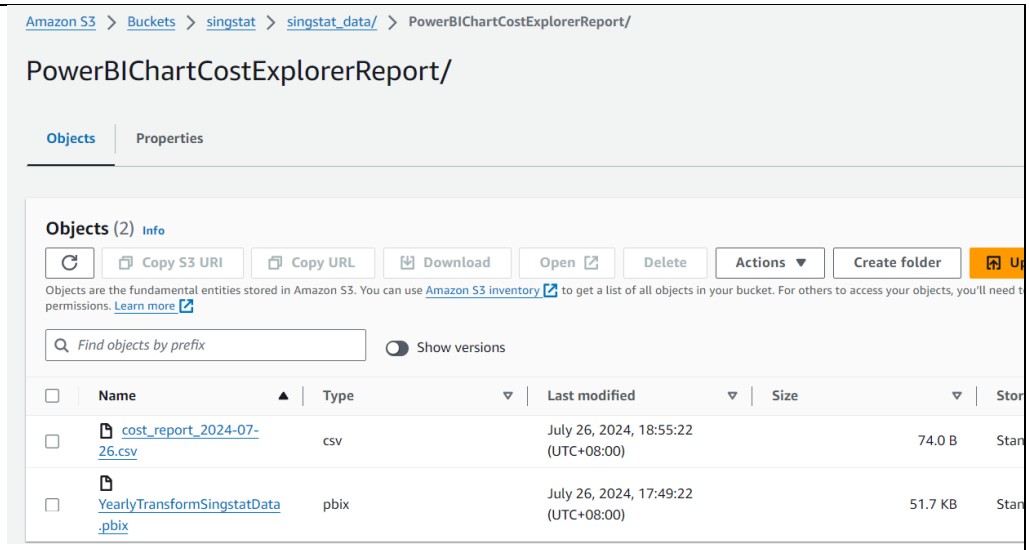
12:35 Python Spaces: 4

```
try:
    cost_report_url = s3_client.generate_presigned_url(
        'get_object',
        Params={'Bucket': s3_bucket, 'Key': cost_report_key},
        ExpiresIn=604800  # Link expires in 7 days
    )
except Exception as e:
    logger.error(f"Error generating pre-signed URL: {e}")
    return {
        'statusCode': 500,
        'body': json.dumps('Error generating pre-signed URL')
    }

# Generate a pre-signed URL for the Power BI chart (expires in 7 days)
try:
    powerbi_chart_url = s3_client.generate_presigned_url(
        'get_object',
        Params={'Bucket': s3_bucket, 'Key': powerbi_chart_key},
        ExpiresIn=604800  # Link expires in 7 days
    )
except Exception as e:
    logger.error(f"Error generating pre-signed URL for Power BI chart: {e}")
    return {
        'statusCode': 500,
        'body': json.dumps('Error generating pre-signed URL for Power BI chart')
    }

# Construct the SNS message
message = f"Total cost for {start} to {end} (excluding credits): ${total_cost}\n\n"
message += "Please check the updated Power BI charts and AWS Cost Explorer reports:\n"
message += f"Power BI Chart: {powerbi_chart_url}\n"
message += f"Cost Explorer Report: {cost_report_url}\n"

# Publish the message to SNS
try:
    sns_client.publish(
        TopicArn='arn:aws:sns:us-east-1:609225191640:SingstatNotifications',
        Message=message
```

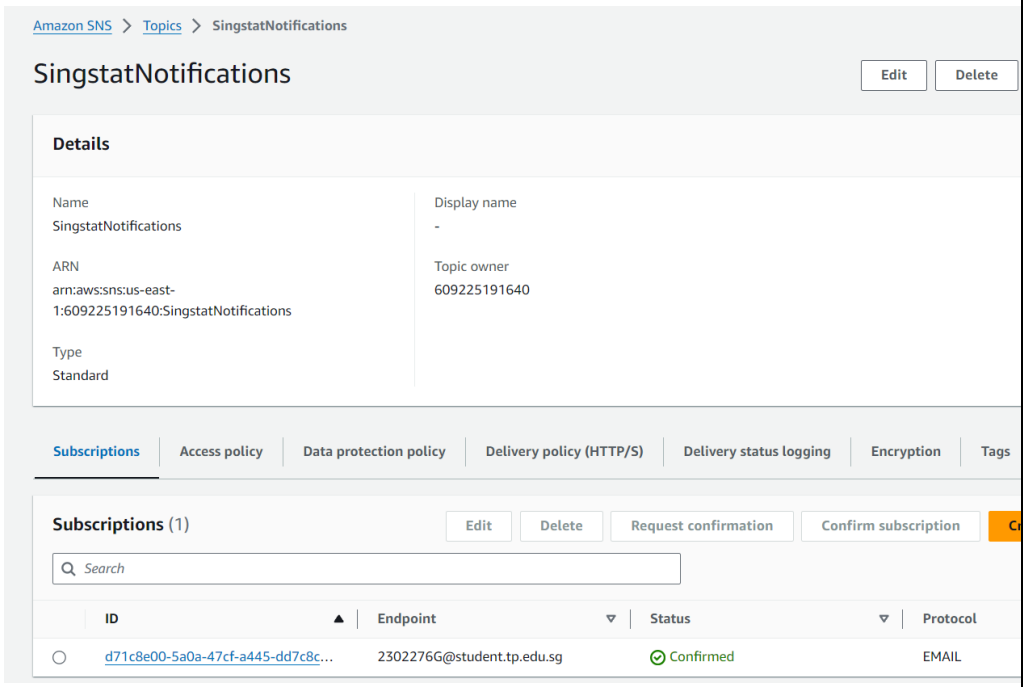
12:35 Python Spaces: 4

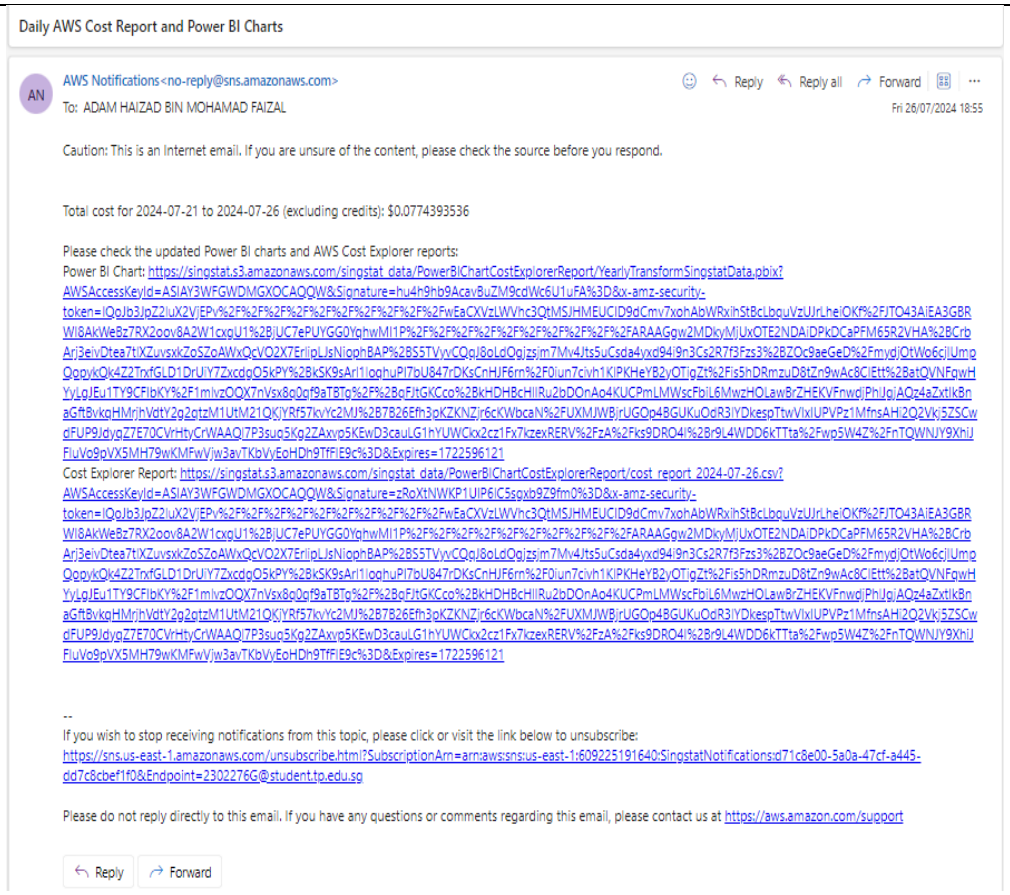
```
# Publish the message to SNS
try:
    sns_client.publish(
        TopicArn='arn:aws:sns:us-east-1:609225191640:SingstatNotifications',
        Message=message,
        Subject='Daily AWS Cost Report and Power BI Charts'
    )
except Exception as e:
    logger.error(f"Error publishing message to SNS: {e}")
    return {
        'statusCode': 500,
        'body': json.dumps('Error publishing message to SNS')
    }

return {
    'statusCode': 200,
    'body': json.dumps('Notification sent successfully')
}
```

Using AWS Cost Explorer, this Lambda function creates a daily cost report and uploads it to the s3 folder named PowerBIChartCostExplorerReport. It then pre-signs URLs for the report and a Power BI chart and sends an SNS notification.



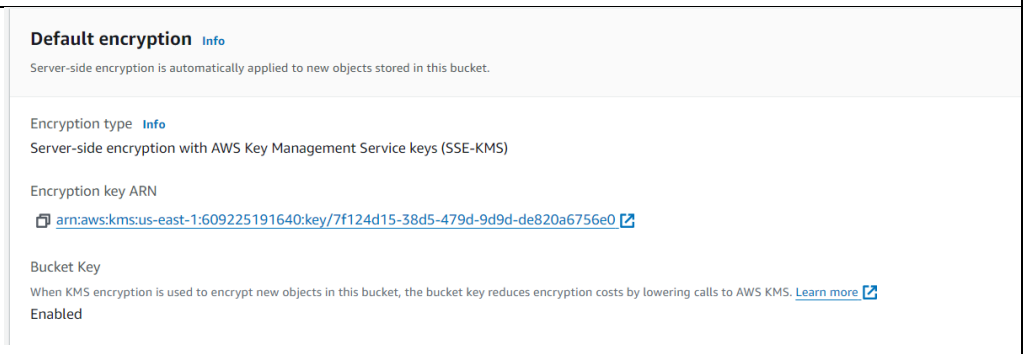
It retrieves the last five days' worth of expenses, calculates the overall cost, and uploads the report to S3. Both the report and the Power BI chart have pre-signed URLs created by the function that make them available for seven days.



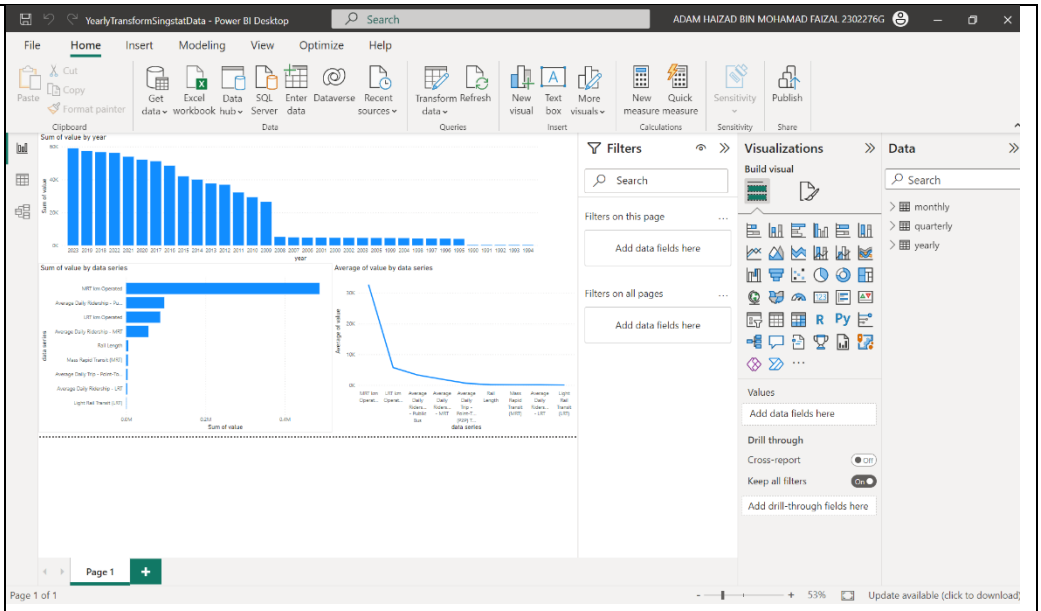
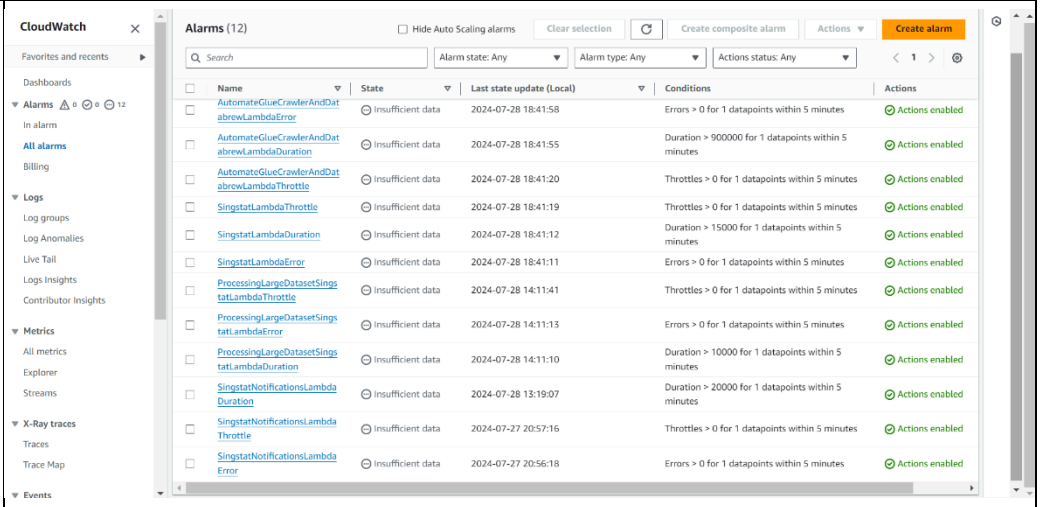


It then constructs a message with these URLs and the total cost and sends it via SNS to the email. The function offers error management for data fetching, S3 operations, URL generation, and SNS publishing, logging problems and returning appropriate status codes and warnings.

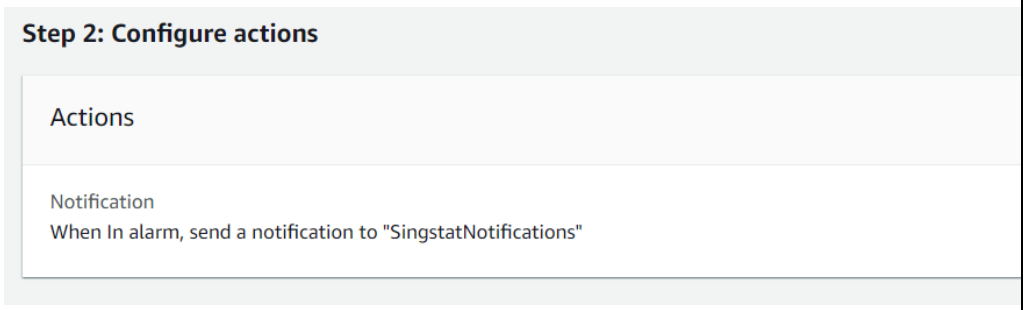
S3



Enabled versioning for protecting against accidental deletions and overwrites, providing an additional layer of data integrity. For encryption, AWS Key Management Service (KMS) has been implemented. As a result, there is strong protection against unwanted access and potential data breaches since our data is encrypted while it is at rest using a key that is owned by the customer.

Quicksight (Power BI)	 After querying in Athena, the queried data is then displayed in Power BI. The connection from Athena to Power BI is done through ODBC driver. After creating visualizations, the Power BI file is saved and uploaded to the PowerBIChartCostExplorerReport s3 folder.
CloudWatch	 Created alarms for all the 4 lambda functions. The alarms include errors, throttles, and duration to guarantee the dependability and efficiency of our Lambda services. When a Lambda function experiences execution errors, including exceptions or timeouts, the error alarms are set off, signalling possible problems with the code or environment that need to be fixed. Throttle alarms help us detect and handle capacity restrictions by alerting us when a function above its concurrency limits and rejects or delays further invocations. Duration alarms monitor how long Lambda functions execute and notify us if they take longer than anticipated. This could be a sign of wasteful code or

performance bottlenecks. When combined, these alerts offer thorough oversight to preserve the effectiveness and well-being of our Lambda operations.



ALARM: "SingstatLambdaError" in US East (N. Virginia)

AN

AWS Notifications<no-reply@sns.amazonaws.com>

Reply | Reply all | Forward

To: ADAM HAIZAD BIN MOHAMAD FAIZAL

Sun 28/07/2024 21:02

Caution: This is an Internet email. If you are unsure of the content, please check the source before you respond.

You are receiving this email because your Amazon CloudWatch Alarm "SingstatLambdaError" in the US East (N. Virginia) region has entered the ALARM state, because "Threshold Crossed: 1 out of the last 1 datapoints [1.0 (28/07/24 12:57:00)] was greater than the threshold (0.0) (minimum 1 datapoint for OK -> ALARM transition)." at "Sunday 28 July, 2024 13:02:11 UTC".

View this alarm in the AWS Management Console:
https://us-east-1.console.aws.amazon.com/cloudwatch/deeplink.js?region=us-east-1#alarmsV2:alarm/SingstatLambdaError

Alarm Details:
- Name: SingstatLambdaError
- Description:
- State Change: INSUFFICIENT_DATA -> ALARM
- Reason for State Change: Threshold Crossed: 1 out of the last 1 datapoints [1.0 (28/07/24 12:57:00)] was greater than the threshold (0.0) (minimum 1 datapoint for OK -> ALARM transition).
- Timestamp: Sunday 28 July, 2024 13:02:11 UTC
- AWS Account: 609225191640
- Alarm Arn: arn:aws:cloudwatch:us-east-1:609225191640:alarm:SingstatLambdaError

Threshold:
- The alarm is in the ALARM state when the metric is GreaterThanThreshold 0.0 for at least 1 of the last 1 period(s) of 300 seconds.

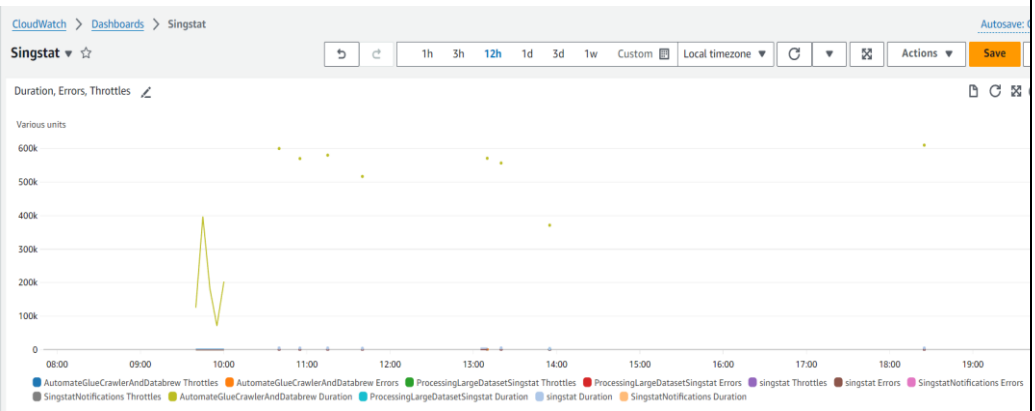
Monitored Metric:
- MetricNamespace: AWS/Lambda
- MetricName: Errors
- Dimensions: [FunctionName = singstat]
- Period: 300 seconds
- Statistic: Sum
- Unit: not specified
- TreatMissingData: missing

State Change Actions:
- OK:
- ALARM: [arn:aws:sns:us-east-1:609225191640:SingstatNotifications]
- INSUFFICIENT_DATA:

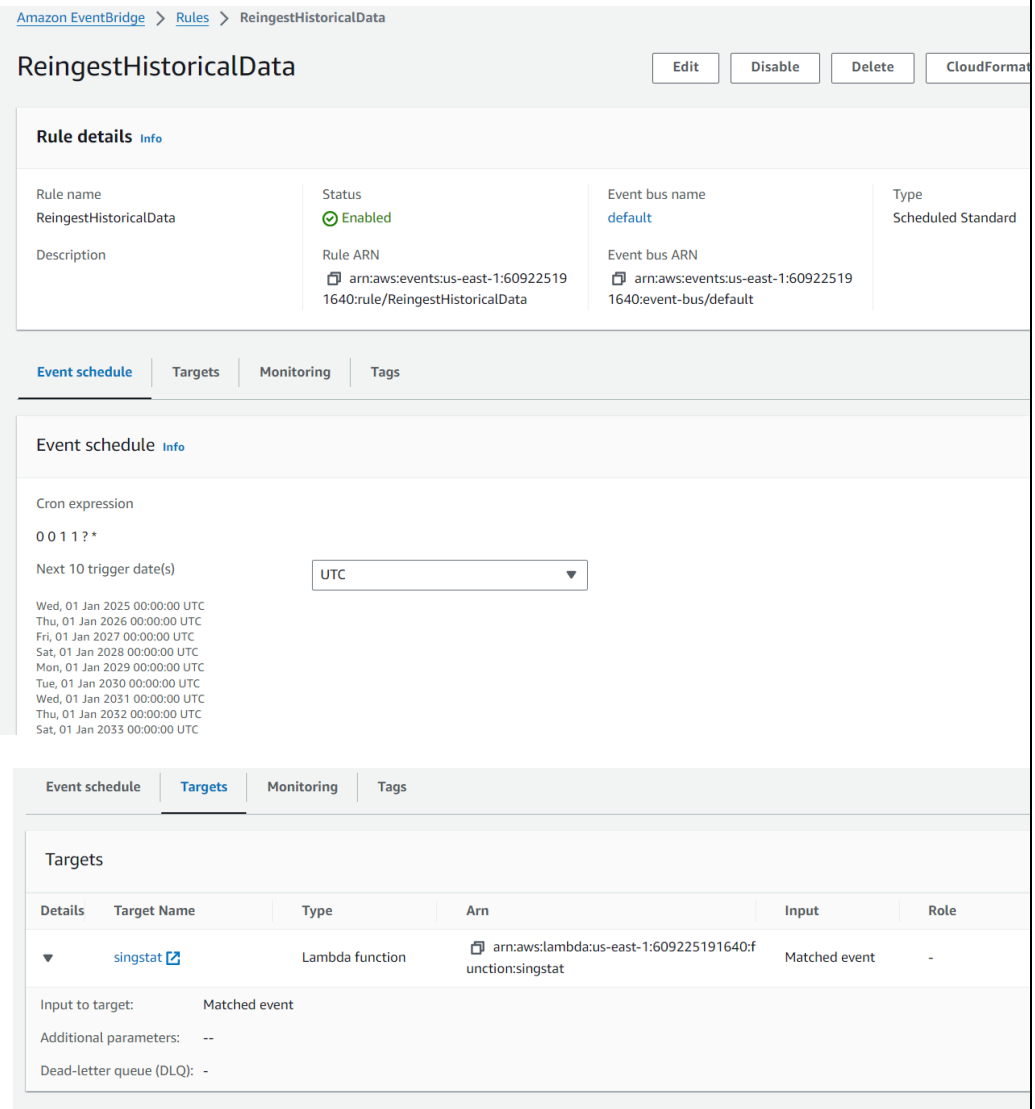
--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:609225191640:SingstatNotifications:d71c8e00-5a0a-47cf-a445-dd7c8cbef1f0&Endpoint=2302276G@student.tp.edu.sg

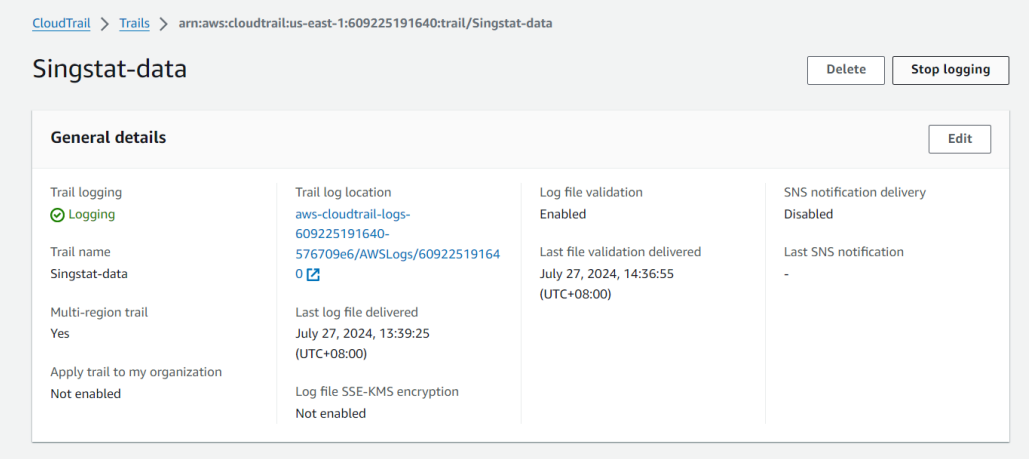
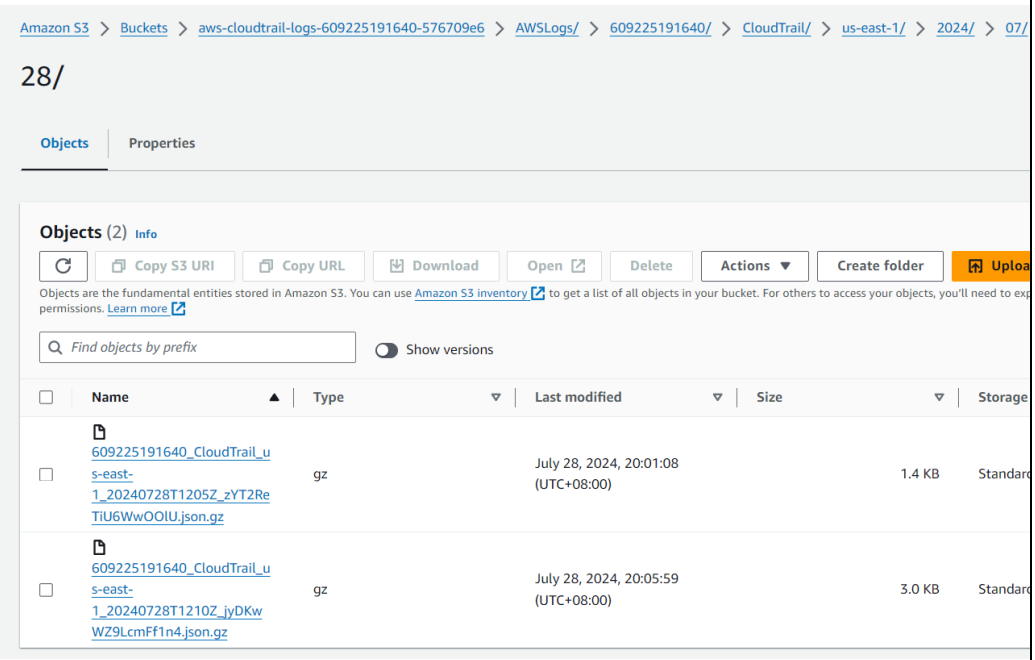
Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

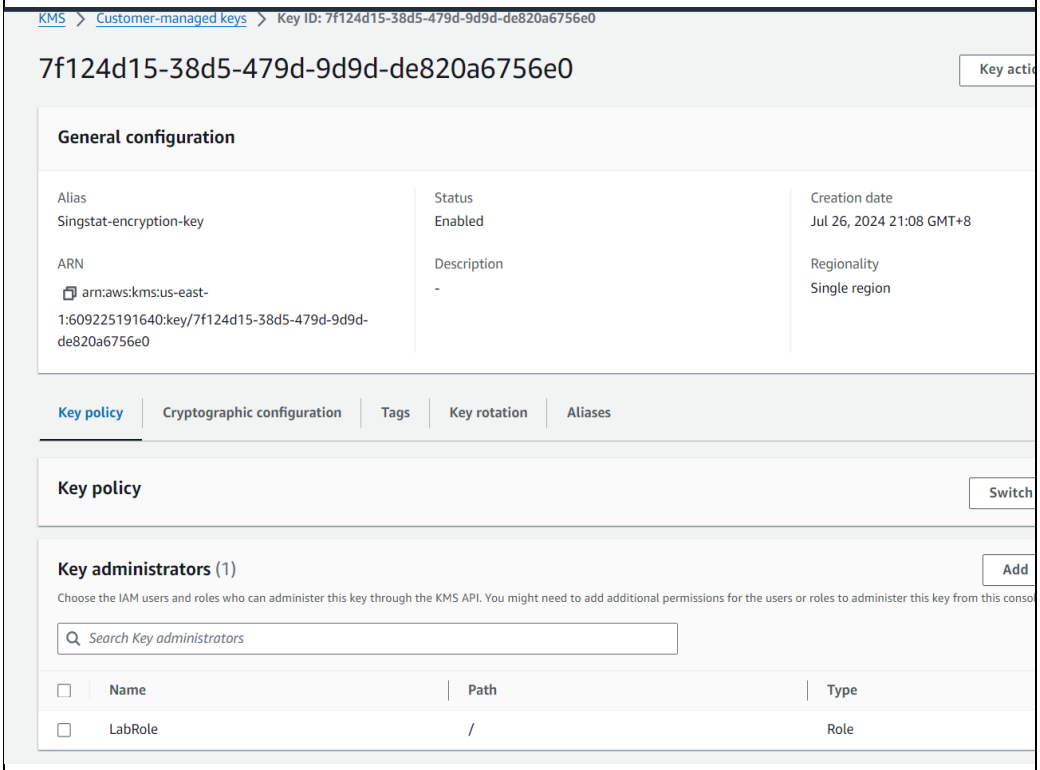
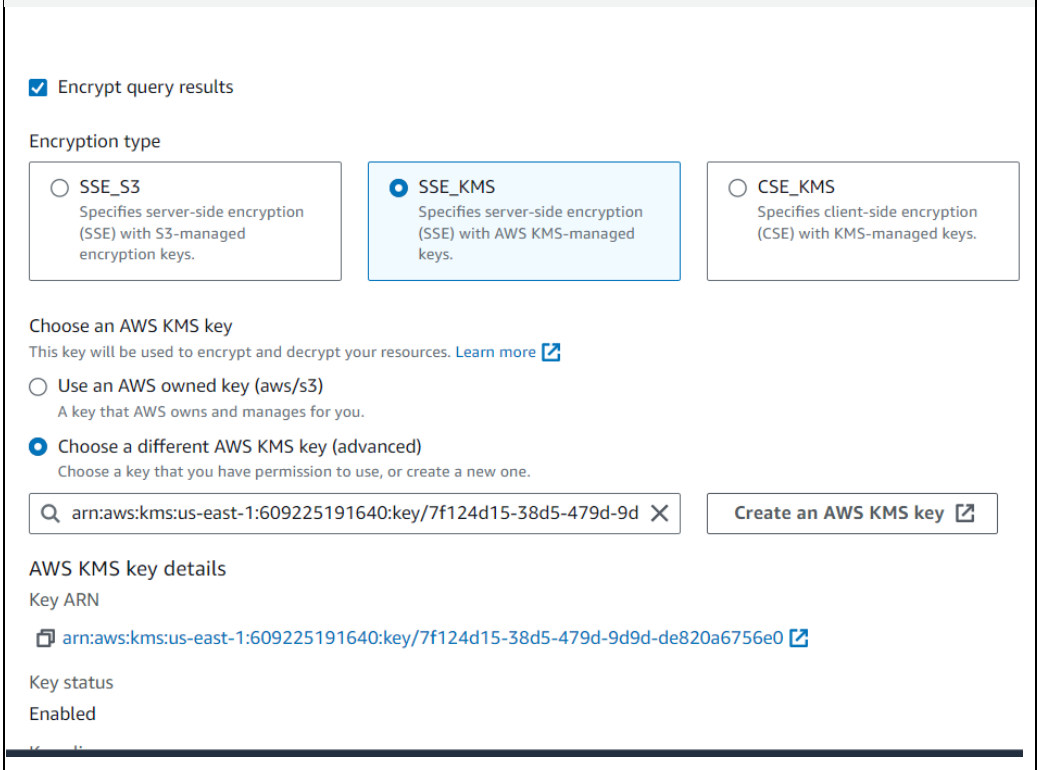
For all these alarms, when the alarm state goes into the ‘in alarm’ state, an email will be sent through SNS to notify about the alarm.

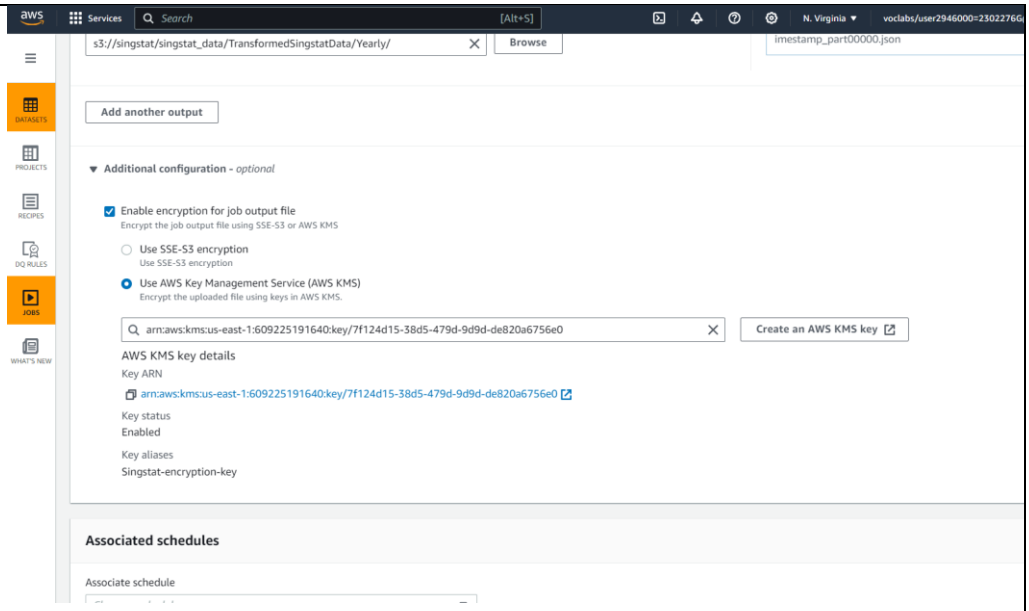


I made a dashboard to track all the data with alarms in addition to the alerts. We can swiftly detect and fix problems with this dashboard's real-time view of important metrics for each Lambda function, such as durations, throttles, and errors.



	Created a CloudWatch rule which will run the Singstat lambda function every year. The reason this is crucial is that Singstat refreshes its data on an annual basis, thus it ensures that the data is the most recent. This automated procedure reduces the need for human interaction while maintaining data relevance and accuracy.
CloudTrail	  To monitor and track user activity across different AWS regions, the "Singstat-data" AWS CloudTrail trail is actively logging events. This trail uses log file validation to guarantee the integrity of the logs and saves log files in a designated S3 bucket named aws-cloudtrail-logs-609225191640-576709e6. All activities inside the AWS environment are logged and can be examined for operational and security troubleshooting thanks to this configuration, which is utilised for auditing and compliance purposes. Additionally, Insights are enabled to get a deeper analysis of the API activity to detect unusual activities.

Key Management Service (KMS)	 

	 I made an encryption key that can be used with S3, Athena, and DataBrew, among other services named Singstat-encryption-key. By encrypting data both in transit and at rest, this key improves data security by guaranteeing that private data is safeguarded during various processing and storage phases. We adhere to data protection laws and uphold uniform security standards by combining this encryption key with different services.

User's process:

First, the user will need to create an API gateway to access the singstat data. Then, go to the singstat lambda function and edit any of the test event and modify the s3_key, which is the destination you want the file to be in. Make sure that the destination matches with the dataset. For example, monthly dataset goes to monthly folder. Also modify the json_url which is the link to your singstat data, Run the function and the function automates up till Athena for you.

Then, the user needs to connect Athena to Power BI using ODBC driver. To do this, the user needs to get the access key id, secret access key id and session token, which can be found on the details for the learner lab. After connecting to Athena, the user can perform any visualizations they like to send to the client.

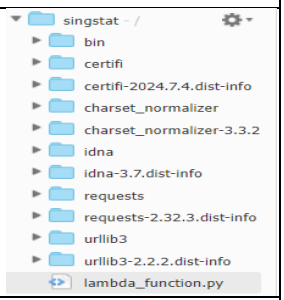
When the user is done, download the file and upload it to the PowerBIChartCostExplorerReport. Then go to the SingstatNotifications lambda function and make sure to replace the powerbi_chart key with the s3 link to the file. Then, run the function and the client will receive an email on the visualizations and costs.

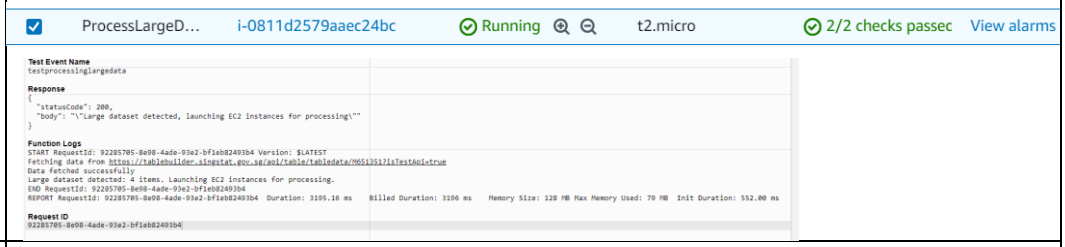
SDL Learning Journal

Task 1:

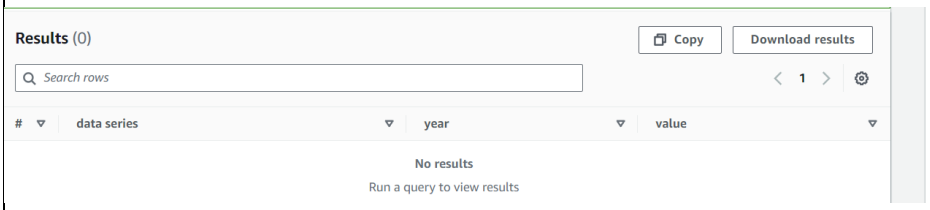
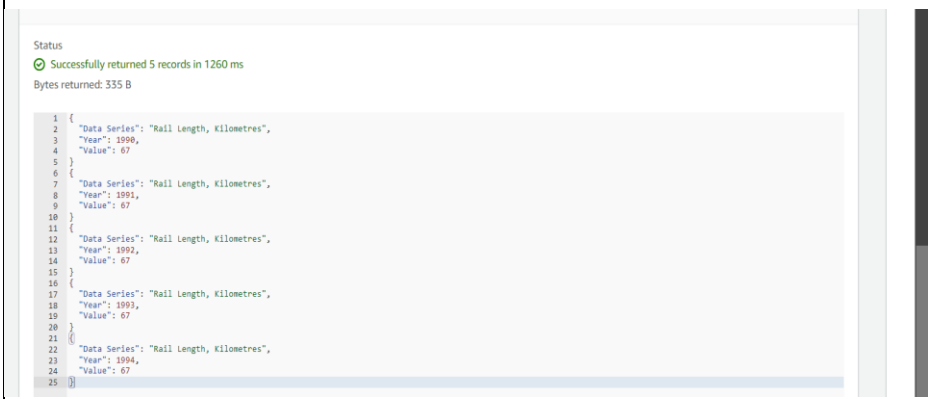
Cycle:	1

Plan: Task & Resources	Implement AWS Lambda for ingestion, planned to use AWS academy lab and materials and reference to code provided by the teacher on teams
Perform:	Carried out the task on 22 June 2024, modifying from the code given by the teacher.
Monitor:	It was not successful. Asked ChatGPT for more information on the error and suggestions on what to do. (https://chatgpt.com/share/ae6db300-c143-4f03-9d81-6b6bc328197a). The error was due to the requests library not being installed.
Reflect:	Learned that the requests library is not included in the default Lambda environment. Decided to continue with installing tof the library the next day.

Cycle:	2
Plan: Task & Resources	Install the request library with help from ChatGPT and YouTube videos
Perform:	Carried out the task on 23 June 2024, which required the use of the command prompt and jupyter notebook to download the library and zip the file to be uploaded to the lambda function. 
Monitor:	The function was executed with no errors and has been assigned to an s3 bucket for further processing down the pipeline.
Reflect:	Learned how to package and deploy third party libraries in AWS Lambda, which is a useful skill to have. The next cycle will focus on adding more features to the lambda function so that it will be able to handle larger datasets.
Cycle:	3
Plan: Task & Resources	Add another function that will be invoked in the main lambda function when there is a large dataset. I will be using aws materials and the lucid chart diagram done during the group project.

Perform:	Carried out the task on the same day and created ec2 instances which will be used to ingest the larger dataset in the lambda function and then upload it to the untransformedsingstatdata s3 folder, for data that is untransformed. 
Monitor:	The function was executed with no errors and has been assigned to an s3 bucket for further processing down the pipeline.
Reflect:	Learned how to integrate lambda with ec2 for scalable data processing.

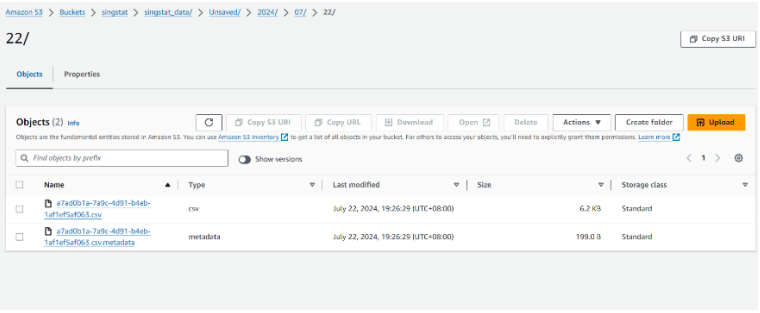
Task 2:

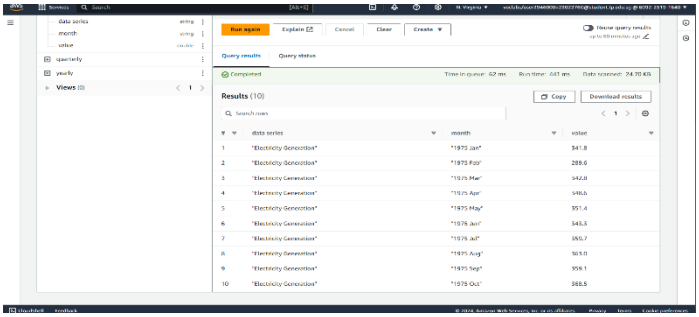
Cycle:	1
Plan: Task & Resources	After implementing the automategluecrawlerdatabrew lambda function, the next step would be to query the data in Athena. I will use aws learner lab and aws materials for the task.
Perform:	Performed the SELECT * FROM query in Athena to show the table and its data.
Monitor:	The table was shown but there was no data inside it. This was weird because I could view the data in s3 query select.   Asked for ChatGPT's help (https://chatgpt.com/share/f47c0c9d-10db-4ad5-a0fb-97099f2fb843). ChatGPT gave some ideas on what could have went wrong.

Reflect:	The issue could be due to the data format or something could be wrong with the steps I did in glue databrew. Decided to check all these the next day.

Cycle:	2
Plan: Task & Resources	Try out all the suggestions that ChatGPT gave and YouTube videos.
Perform:	Changed the format of the data and even remove the step where I merge the data which was a possible cause of error.
Monitor:	However, despite the multiple amount of attempts to try to figure out the error, I could not solve the problem.
Reflect:	I learned how to persevere and keep calm as I felt as if I wasted a lot of time trying to figure the error out but fail in the end. I planned to ask my teacher for help during the next class.

Cycle:	3
Plan: Task & Resources	Planned to ask my teacher for help during class.
Perform:	Asked my teacher and he told me that it could be a file error and that I should try again with a new crawler, new bucket, and new database.
Monitor:	I did what he asked me to do but the same thing happened. I then asked him again and after multiple attempts, it did not work. The problem was deeper than I thought.
Reflect:	Even though it was not working, it was already late, and I had to go home. I recognised that some issues are very complex and require a lot of patience and require multiple attempts to solve. I planned to go home and try fixing the error.

Cycle:	4
Plan: Task & Resources	Troubleshoot the error with the help of ChatGPT, YouTube and visiting websites for potential solutions.
Perform:	When going through the s3 bucket, I noticed an unsaved folder in the same folder as the data used for Athena. I deleted the unsaved folder as I felt it may be the root cause of the error. 

Monitor:	When I deleted the folder, I ran the automategluecrawlerdatabrew lambda function again and when I ran the SELECT * FROM query, the data was displayed.  The unsaved folder was likely interfering with the data processing and querying in Athena.
Reflect:	The experience taught me to carefully inspect all the components in the data pipeline to ensure I can identify the causes of error. It also taught me a minor issue such as a folder can have a significant impact.