SmartInSightGen: Serverless Review Analysis with AWS NLP Services

Krishna Varma Vetukuri

TABLE OF CONTENTS

1. Problem Statement 2
2. AWS services and justification (with Architecture Diagram) 2
3. Implementation details 4
4. Results and Insights 66
5. Challenges faced and solutions 71
6. Future enhancements 72

LIST OF FIGURES

Figure 4.1 Daily Sentiment Volume 66

Figure 4.2 Positive Sentiment Growth Line 67

Figure 4.3 Positive Sentiment Growth Bar 67

Figure 4.4: Pie Chart Analysis 67

Figure 4.5: Average Ratings and Review Counts per Product 68

Figure 4.6: Sentiment Distribution per Product 68

Figure 4.7: Average Rating Visualization 69

Figure 4.8: Review Heatmap 69

Figure 4.9: Feedback Summary (Pivot Table) 70

Figure 4.10: Key Phrase Analysis – Negative Feedback 70

1. Problem Statement:

E-commerce platforms receive thousands of customer reviews daily, but manually analyzing this large volume of unstructured text is inefficient and doesn't scale. The challenge is to build an automated system that can process raw review data, analyze sentiment, extract key phrases, and generate product-level summaries to help businesses quickly understand customer feedback. This project implements such a solution using AWS services like S3 for data storage, Lambda for processing, Comprehend for NLP tasks, DynamoDB for storing results, and QuickSight for visualization, enabling scalable and real-time insights from customer reviews.

2. AWS services and justification (with Architecture Diagram):

- **Amazon S3:**
  Used for storing raw review CSV files uploaded by users, as well as all intermediate and cleaned files (JSON, CSV). It's simple, cost-effective, and integrates easily with other AWS services like Lambda and QuickSight.

- **AWS Lambda:**
  Used throughout the project for processing reviews, exporting data, cleaning CSVs, generating summaries, and preparing datasets for QuickSight. Lambda was chosen because it's serverless, event-driven, and doesn't require managing any infrastructure. I wrote custom Python code for each function to automate data flow and transformation.

- **Amazon Comprehend:**
  Used for sentiment analysis, entity recognition, and key phrase extraction. Comprehend helped me apply NLP to each review automatically without training any models, making it ideal for fast and accurate text analysis.

- **Amazon DynamoDB:**
  Used to store processed review data including sentiment, key phrases, and entities. Also used for storing product-level and sentiment-level summaries. DynamoDB's NoSQL structure fit perfectly since each review had a flexible data format and fast access was needed for summarization.

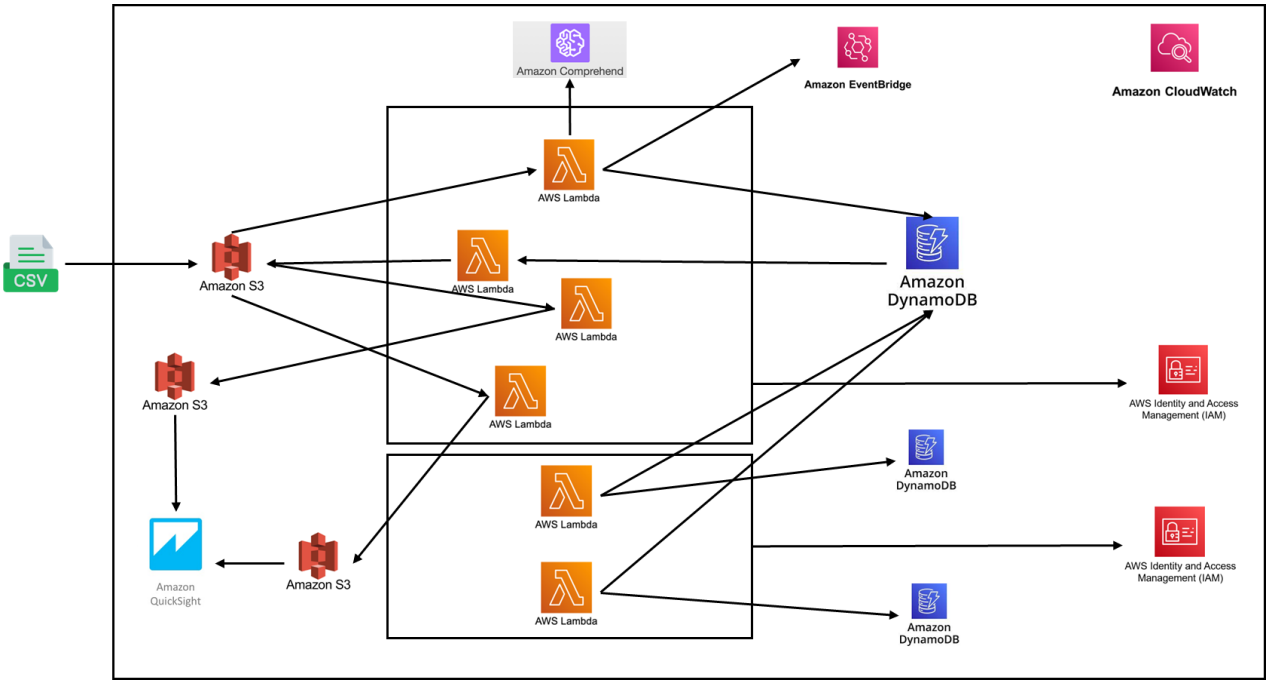
- **Amazon EventBridge:**
  Used to trigger the summarization Lambda functions daily without any manual effort. It helped automate routine tasks like generating summaries at fixed intervals.

- **Amazon CloudWatch:**
  Used to monitor Lambda execution, view logs, and debug issues. It helped me track function timeouts and performance during testing.

- **AWS Identity and Access Management (IAM):**
  Used to securely grant Lambda functions the right permissions to access S3, Comprehend, and DynamoDB. I created custom IAM roles for each function.

- **Amazon QuickSight:**
  Used to create dashboards and visualizations from the cleaned review data. It helped me display trends, summaries, and key insights for business understanding. I generated manifest files and prepared cleaned CSVs to make it compatible.

Architecture Diagram



3. Implementation details

In this step, a dataset of customer reviews is created. This dataset is used later by AWS services to analyze and summarize reviews.

3.1. Data Ingestion Using Amazon S3

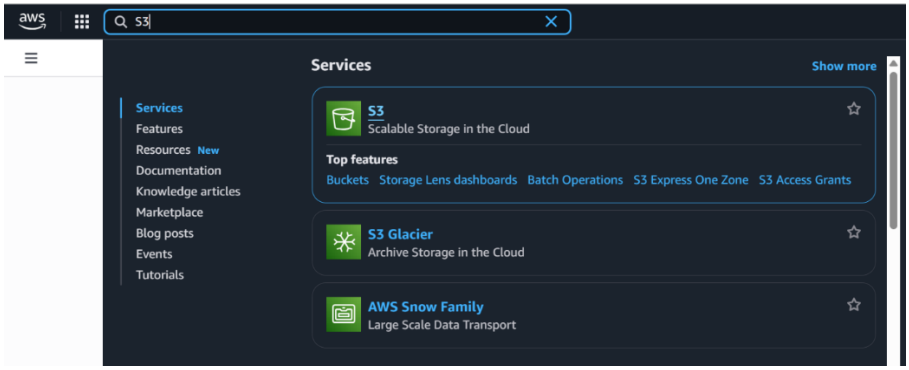
To begin the automated review processing workflow, a centralized data storage mechanism was established using Amazon S3 (Simple Storage Service). This serves as the input layer of the pipeline where customer review data is initially uploaded and monitored for processing.

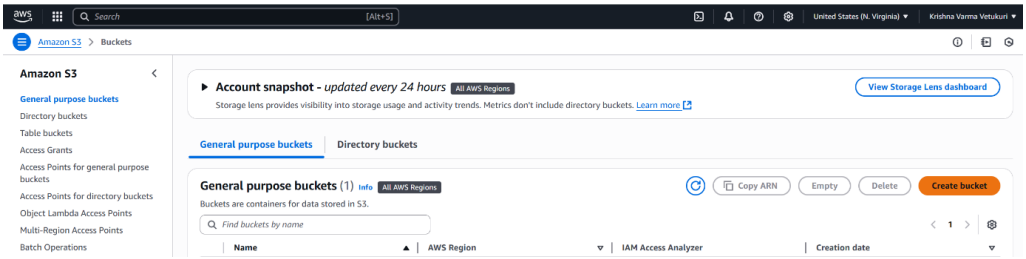
3.1.1. S3 Bucket Creation:

To store the raw customer review files for processing, an S3 bucket named **ecommerce-customer-reviews-rawdata** was created. These files mimic customer reviews in a real-world e-commerce system and serve as the input layer of the analysis pipeline.

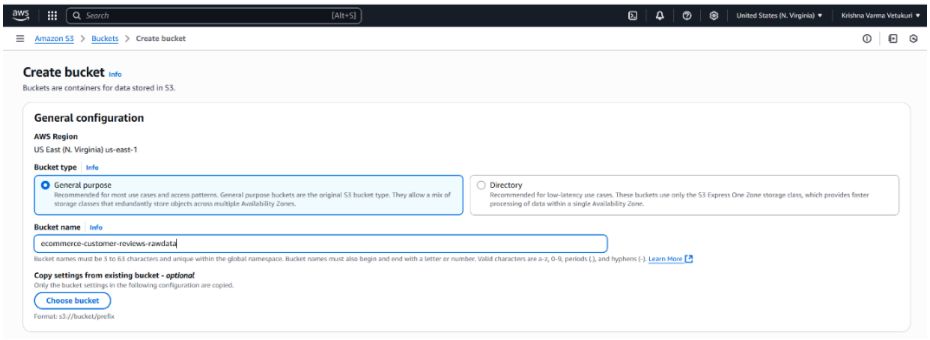
Steps to Create the S3 Bucket:

1. Open the AWS Console and navigate to **Amazon S3**.

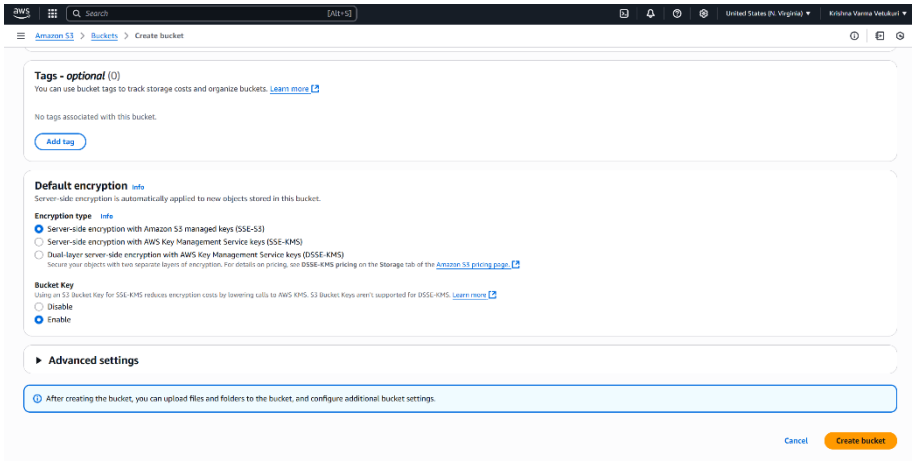




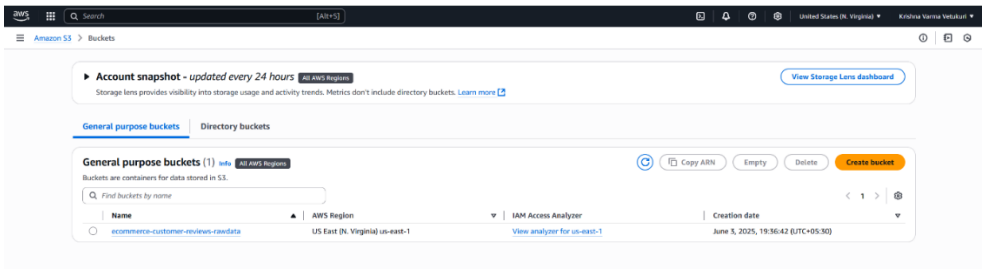
2. Click on the **"Create Bucket"** button.
3. In the **Bucket Name** field, enter: **ecommerce-customer-reviews-rawdata**



4. Leave all other settings (Region, Object Ownership, Block Public Access, etc.) as default.



5. Scroll down and click **"Create Bucket"** to finalize.



This bucket will store the raw review data and is used later by AWS Lambda and Comprehend.

3.1.2. CSV File Structure:

The raw customer review data is organized in a .csv format, with each row representing a customer review. This structured format enables automated ingestion and analysis using AWS services.

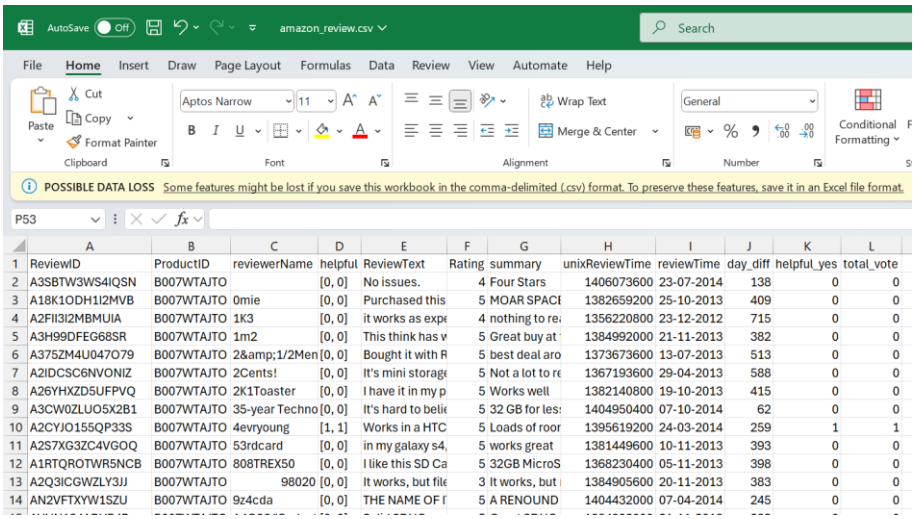
CSV Fields Included:

- reviewerID: Unique identifier of the reviewer
- asin: Amazon Standard Identification Number for the product

- reviewerName: Name of the reviewer
- reviewText: Full text of the review
- rating: Numerical star rating (1 to 5)
- summary: Short summary of the review
- unixReviewTime: Review timestamp in UNIX format
- reviewTime: Human-readable date of the review
- day_diff: Days since the review was written
- helpful_yes: Number of users who found the review helpful
- total_vote: Total number of votes received

Dataset Source and Preprocessing Notes:

- The dataset was originally sourced from Kaggle: Amazon Review Dataset on Kaggle
- The original file contained over 3,000 reviews but was associated with only one product ID.
- To simulate a more realistic multi-product e-commerce scenario, five distinct Product IDs were manually assigned across the dataset.
- To stay within AWS Free Tier constraints, the dataset was reduced to 285 reviews.
- The final working dataset was saved as: **amazon_review.csv**



	A	B	C	D	E	F	G	H	I	J	K	L
1	ReviewID	ProductID	reviewerName	helpful	ReviewText	Rating	summary	unixReviewTime	reviewTime	day_diff	helpful_yes	total_vote
2	A3SBTW3WS4IQSN	B007WTAJTO		[0, 0]	No issues.	4	Four Stars	1406073600	23-07-2014	138	0	0
3	A18K1ODH1I2MVB	B007WTAJTO	0mie	[0, 0]	Purchased this	5	MOAR SPACE	1382659200	25-10-2013	409	0	0
4	A2FII3I2MBMUIA	B007WTAJTO	1K3	[0, 0]	it works as expe	4	nothing to re	1356220800	23-12-2012	715	0	0
5	A3H99DFEG68SR	B007WTAJTO	1m2	[0, 0]	This think has v	5	Great buy at	1384992000	21-11-2013	382	0	0
6	A375ZM4U047O79	B007WTAJTO	2&1/2Men	[0, 0]	Bought it with R	5	best deal aro	1373673600	13-07-2013	513	0	0
7	A2IDCSC6NVONIZ	B007WTAJTO	2Cents!	[0, 0]	It's mini storage	5	Not a lot to re	1367193600	29-04-2013	588	0	0
8	A26YHXZD5UFPVQ	B007WTAJTO	2K1Toaster	[0, 0]	I have it in my p	5	Works well	1382140800	19-10-2013	415	0	0
9	A3CW0ZLUO5X2B1	B007WTAJTO	35-year Techno	[0, 0]	It's hard to belie	5	32 GB for less	1404950400	07-10-2014	62	0	0
10	A2CYJO155QP33S	B007WTAJTO	4evryoung	[1, 1]	Works in a HTC	5	Loads of roor	1395619200	24-03-2014	259	1	1
11	A2S7XG3ZC4VGOQ	B007WTAJTO	53rdcard	[0, 0]	in my galaxy s4,	5	works great	1381449600	10-11-2013	393	0	0
12	A1RTQROTWR5NCB	B007WTAJTO	808TREX50	[0, 0]	I like this SD Ca	5	32GB MicroS	1368230400	05-11-2013	398	0	0
13	A2Q3ICGWZLY3JJ	B007WTAJTO	98020	[0, 0]	It works, but file	3	It works, but	1384905600	20-11-2013	383	0	0
14	AN2VFTXYW1SZU	B007WTAJTO	9z4cda	[0, 0]	THE NAME OF I	5	A RENOUND	1404432000	07-04-2014	245	0	0

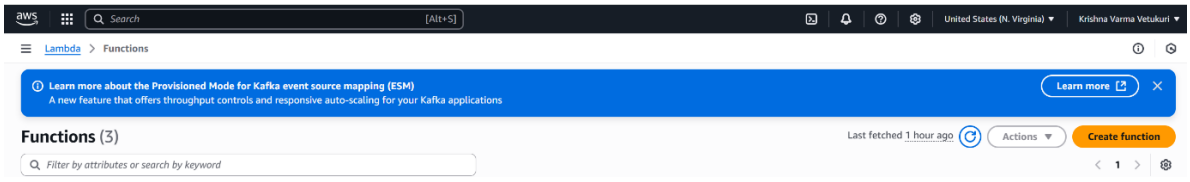
3.2. Lambda for Review Processing

To automate the analysis of customer reviews upon CSV upload, an AWS Lambda function named processReviewCSV was developed. This function uses Amazon Comprehend for NLP tasks like sentiment analysis, key phrase extraction, and named entity recognition. The results are stored in Amazon DynamoDB for structured querying.

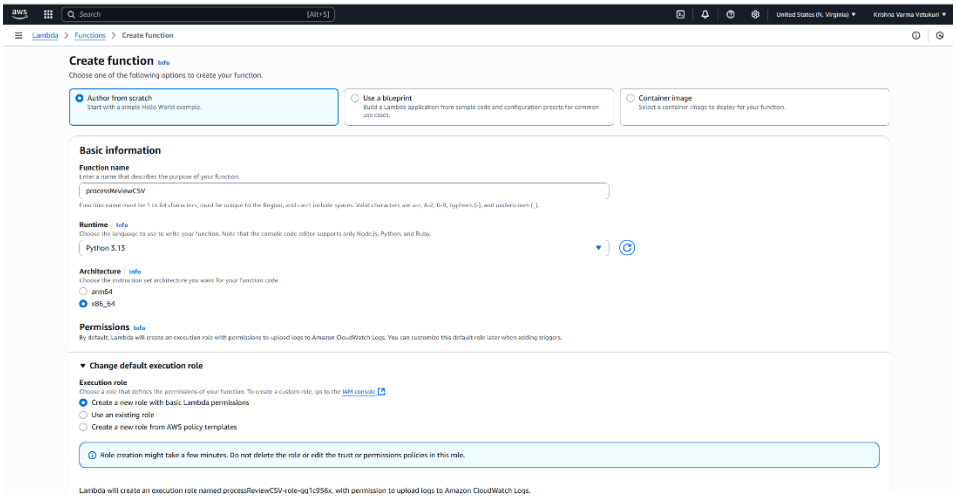
This function is automatically triggered every time a new CSV file is uploaded to a specified S3 bucket (ecommerce-customer-reviews-rawdata), enabling a fully serverless and real-time processing pipeline.

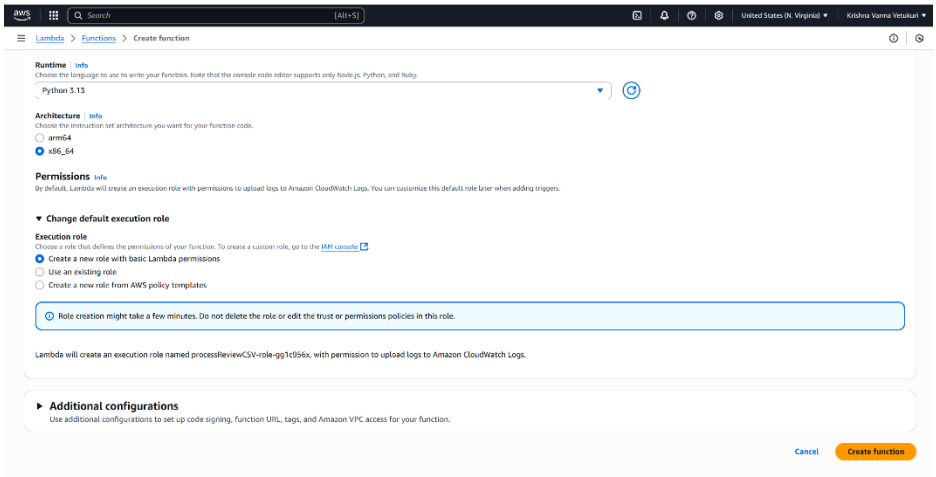
3.2.1. Lambda Function Creation

- Open **AWS Lambda**, click **Create function**.



- Set:
  - **Function name**: processReviewCSV
  - **Runtime**: Python 3.13



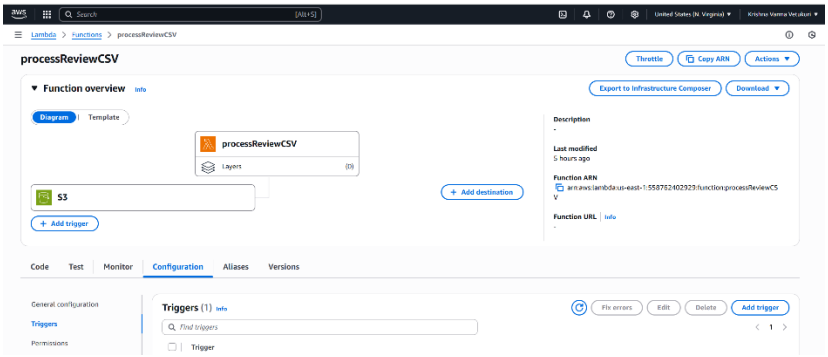


- Leave all other settings as default and click **Create Function**.

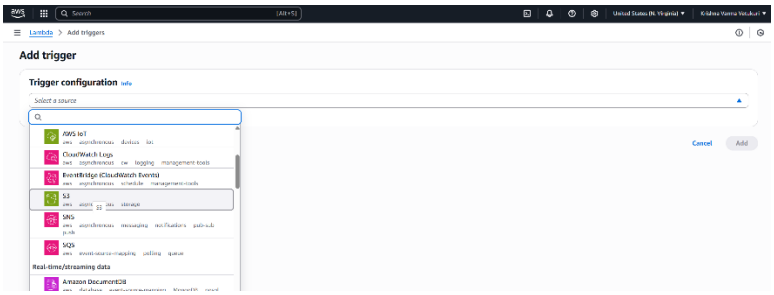
3.2.2. Configure Lambda Trigger (S3 Event)

To make the function automatically respond to new file uploads:

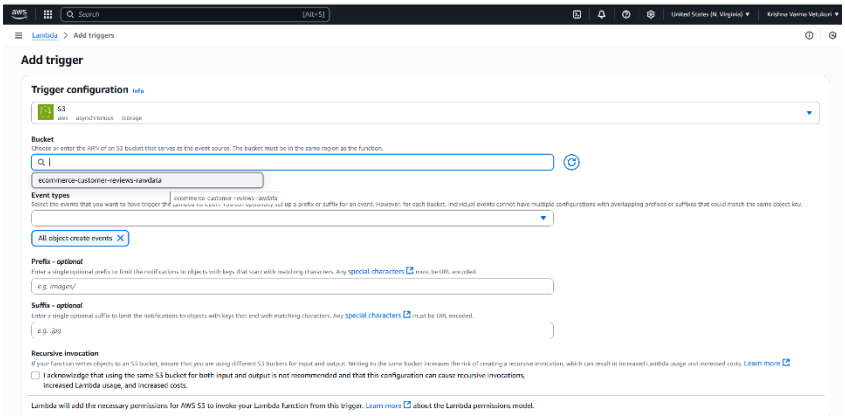
- Go to the **Configuration** tab of the Lambda function.

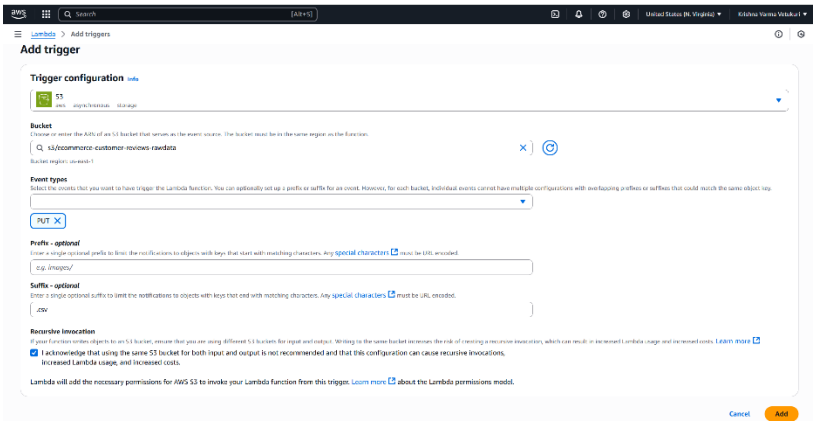


- Under **Triggers**, click **Add trigger**.
- Set:
  - **Trigger source**: S3



  - **Bucket**: ecommerce-customer-reviews-rawdata
  - **Event Type**: *PUT* only (uncheck others)
  - **Suffix (optional)**: .csv – restricts the trigger to CSV files.



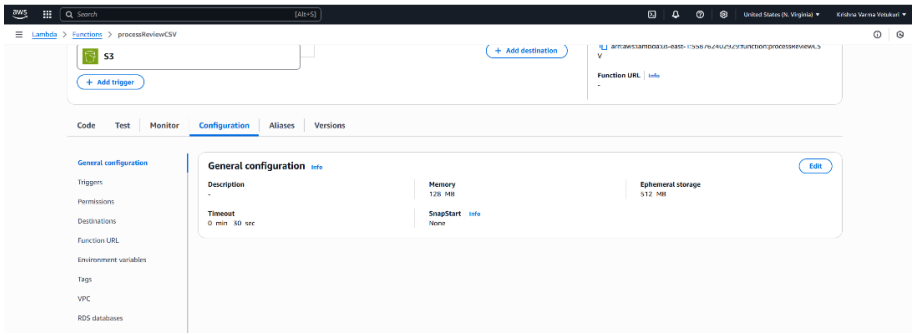


- Click **Add** to complete the trigger setup.

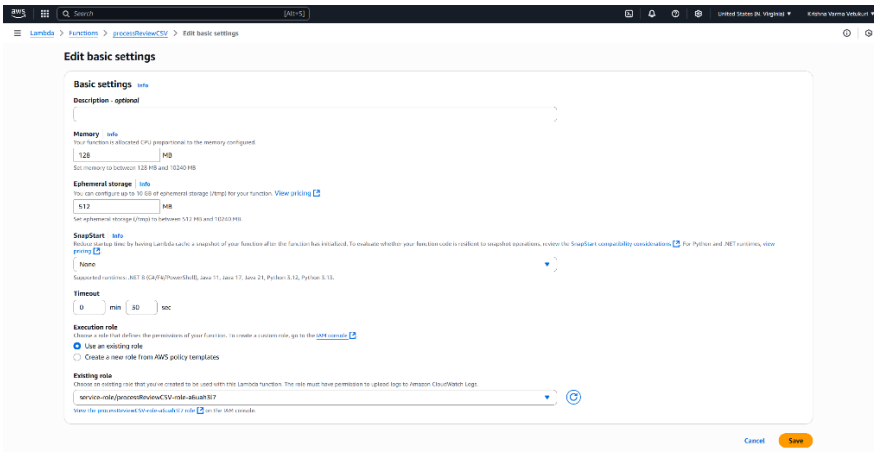
3.2.3. Adjust General Configuration (Timeout)

Since processing review data may take several seconds, the function timeout needs to be increased:

- Navigate to Configuration → General configuration.



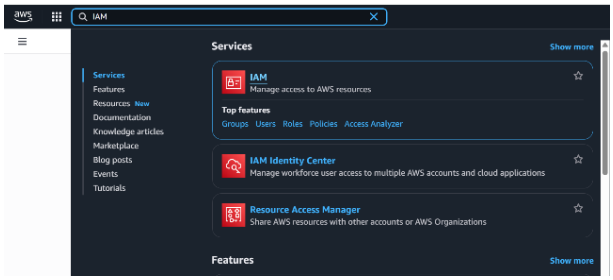
- Click Edit.
- Change Timeout from 3 seconds (default) to 30 seconds.



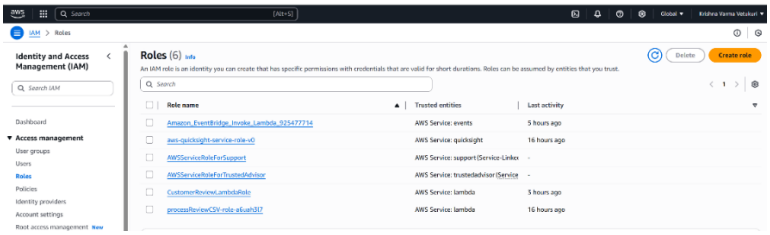
- Click Save.

3.2.4. IAM Role and Permissions

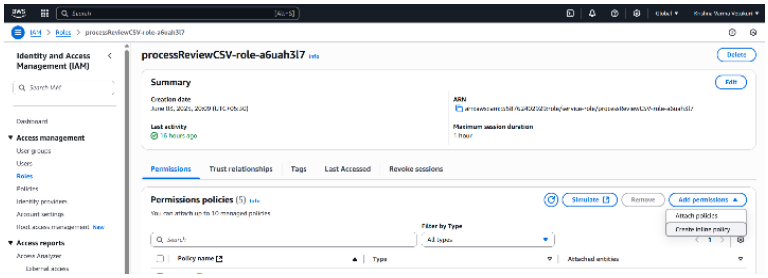
- The Lambda function needs permissions to access DynamoDB, S3, Comprehend, and CloudWatch.



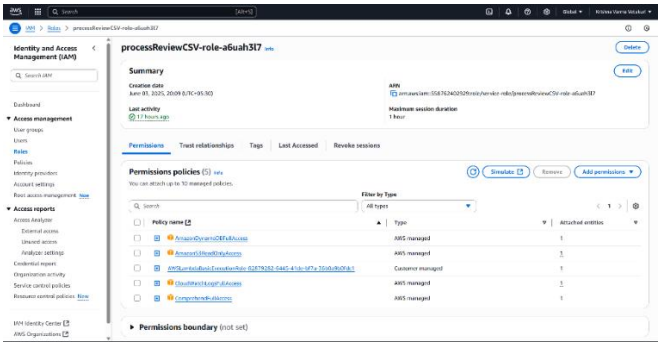
- Go to IAM → Roles.



- Find the role automatically created with the function (processReviewCSV-role-xxxx).
- Click Add permissions → Attach policies.



- Search for and attach the following managed policies:
  - **AmazonDynamoDBFullAccess**
  - **AmazonS3ReadOnlyAccess**
  - **ComprehendFullAccess**
  - **CloudWatchLogsFullAccess**
- Click Add permissions.



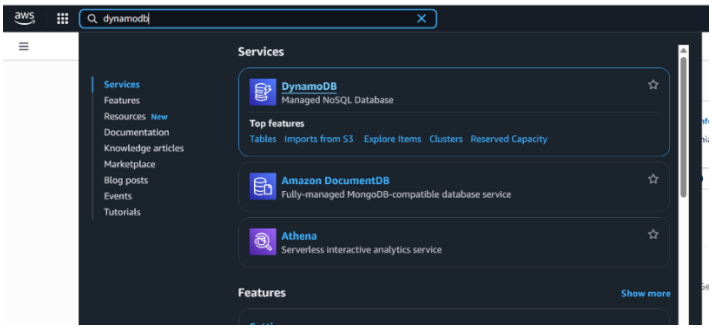
3.3. DynamoDB Table: CustomerReviewsAnalysis

To store the analysis results from the Lambda function, a DynamoDB table named CustomerReviewsAnalysis was created. Each record in the table represents a single processed customer review.

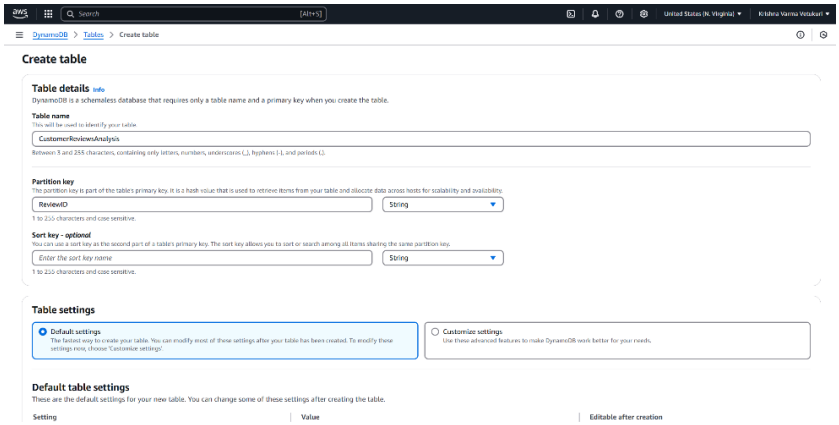
This table stores both the **original review details** and the **NLP-derived insights**, such as sentiment, sentiment scores, key phrases, and named entities.

3.3.1 Creating the DynamoDB Table

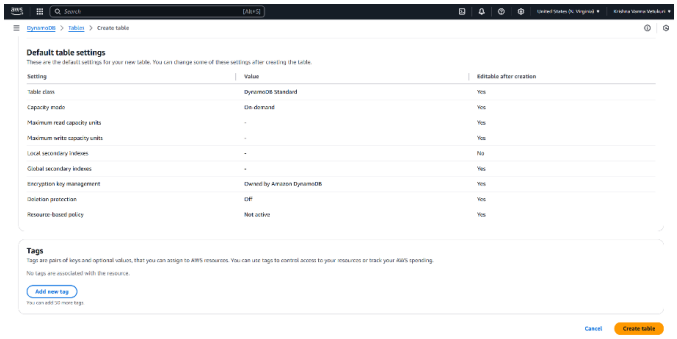
- Go to Amazon DynamoDB.



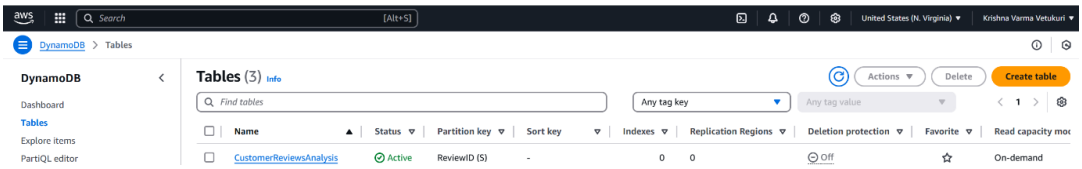
- Click Create Table.
- Set:
  - Table Name: CustomerReviewsAnalysis
  - Partition key: ReviewID (type: String)



- Leave all other settings as default.

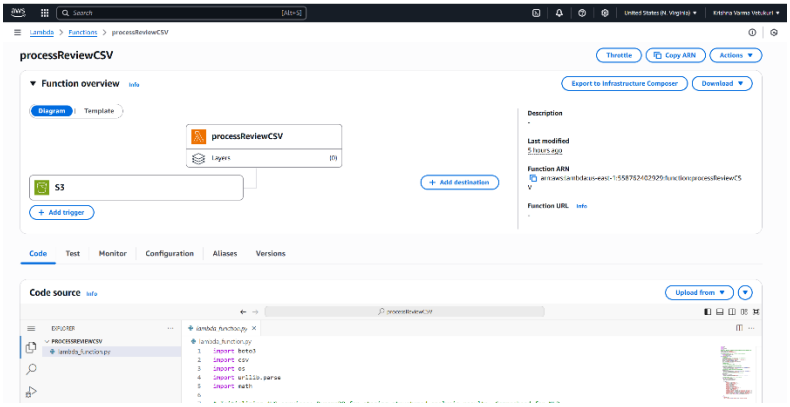


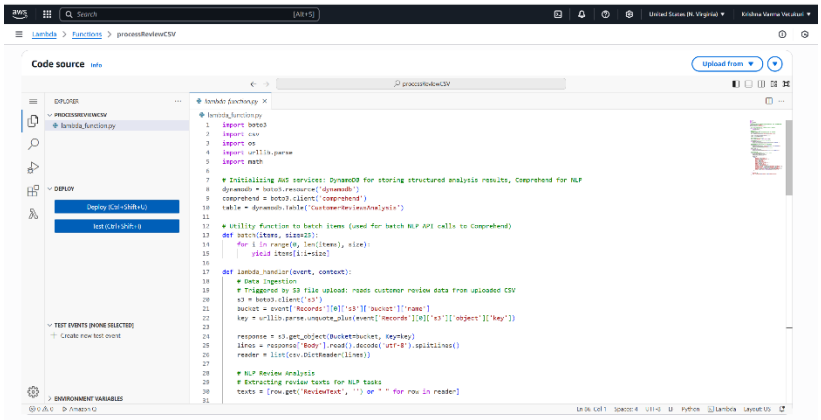
- Click Create Table.



3.3.2 Writing and Deploying Lambda Code

With the table and permissions in place, the Lambda function code can now be added.





- In the Function code section of processReviewCSV, replace the default code with the custom Python script that:
    - Reads CSV from S3.
    - Uses Amazon Comprehend to extract:
        - Sentiment
        - Key Phrases
        - Entities
- Stores each review and its analysis into DynamoDB.

processReviewCSV lambda_function.py

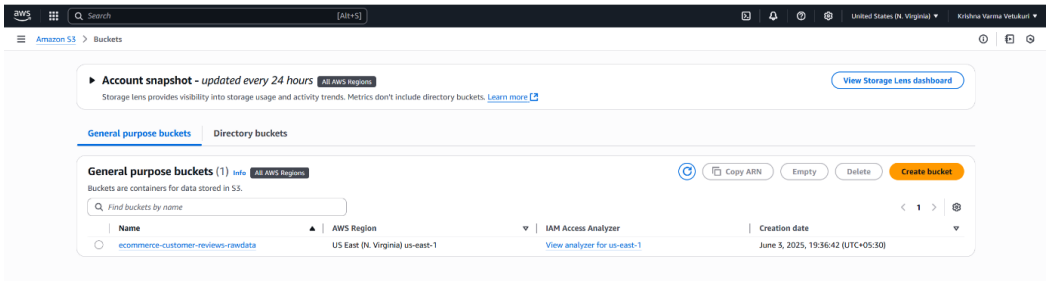
```
import boto3
import csv
import os
import urllib.parse
import math

# Initializing AWS services: DynamoDB for storing structured analysis results, Comprehend for NLP
dynamodb = boto3.resource('dynamodb')
comprehend = boto3.client('comprehend')
table = dynamodb.Table('CustomerReviewsAnalysis')

# Utility function to batch items (used for batch NLP API calls to Comprehend)
def batch(items, size=25):
    for i in range(0, len(items), size):
        yield items[i:i+size]

def lambda_handler(event, context):
    # Data Ingestion
    # Triggered by S3 file upload: reads customer review data from uploaded CSV
    s3 = boto3.client('s3')
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'])

    response = s3.get_object(Bucket=bucket, Key=key)
    lines = response['Body'].read().decode('utf-8').splitlines()
    reader = list(csv.DictReader(lines))

    # NLP Review Analysis
    # Extracting review texts for NLP tasks
    texts = [row.get('ReviewText', '') or " " for row in reader]

    # Performing batch sentiment analysis using Amazon Comprehend
    sentiment_results = []
    for batch_texts in batch(texts):
        response = comprehend.batch_detect_sentiment(TextList=batch_texts, LanguageCode='en')
        sentiment_results.extend(response['ResultList'])

    # Performing batch key phrase extraction using Amazon Comprehend
    key_phrases_results = []
    for batch_texts in batch(texts):
        response = comprehend.batch_detect_key_phrases(TextList=batch_texts, LanguageCode='en')
        key_phrases_results.extend(response['ResultList'])

    # Performing batch named entity recognition using Amazon Comprehend
    entities_results = []
    for batch_texts in batch(texts):
        response = comprehend.batch_detect_entities(TextList=batch_texts, LanguageCode='en')
        entities_results.extend(response['ResultList'])

    # Structured Storage
    # Stored review data (sentiment, phrases, entities) in DynamoDB
    for idx, row in enumerate(reader):
        sentiment_data = sentiment_results[idx]
        key_phrases_data = key_phrases_results[idx]
        entity_data = entities_results[idx]

        table.put_item(
```

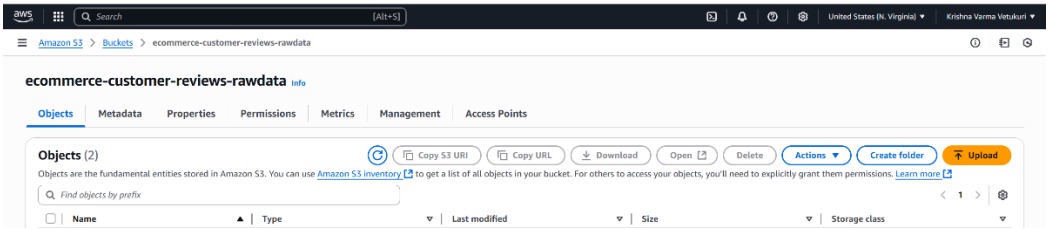
```
        Item={
            'ReviewID': row.get('ReviewID', ''),
            'ProductID': row.get('ProductID', ''),
            'ReviewerName': row.get('reviewerName', ''),
            'Helpful': row.get('helpful', ''),
            'ReviewText': texts[idx],
            'Rating': row.get('Rating', ''),
            'Summary': row.get('summary', ''),
            'ReviewTime': row.get('reviewTime', ''),
            'UnixReviewTime': row.get('unixReviewTime', ''),
            'DayDiff': row.get('day_diff', ''),
            'HelpfulYes': row.get('helpful_yes', ''),
            'TotalVote': row.get('total_vote', ''),
            'Sentiment': sentiment_data['Sentiment'],
            'SentimentScore_Positive': str(sentiment_data['SentimentScore'].get('Positive', '')),
            'SentimentScore_Negative': str(sentiment_data['SentimentScore'].get('Negative', '')),
            'SentimentScore_Neutral': str(sentiment_data['SentimentScore'].get('Neutral', '')),
            'SentimentScore_Mixed': str(sentiment_data['SentimentScore'].get('Mixed', '')),
            'KeyPhrases': [phrase['Text'] for phrase in key_phrases_data.get('KeyPhrases', [])],
            'Entities': [entity['Text'] for entity in entity_data.get('Entities', [])]

        }
    )

return {
    'statusCode': 200,
    'body': f'Processed file {key} from bucket {bucket} using batch processing.'
}
```

3.3.3 Uploading the CSV to S3 (Triggering the Lambda)

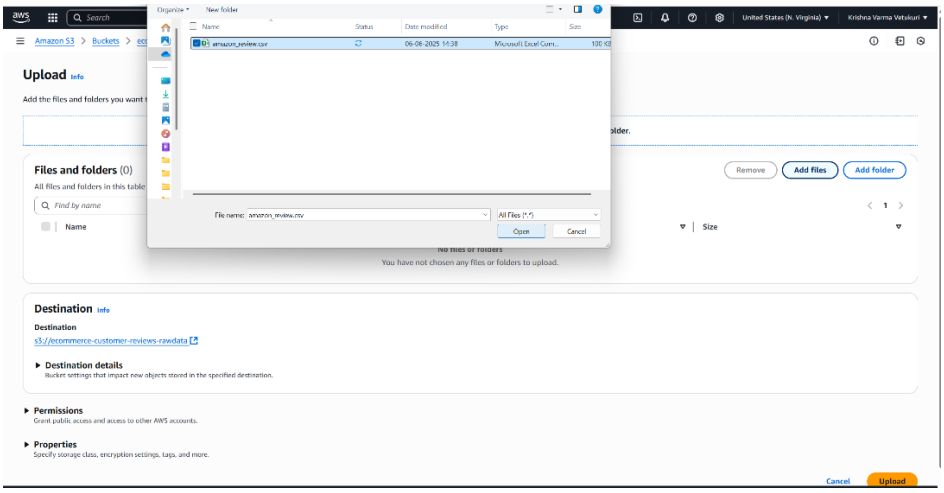
To test the end-to-end flow:

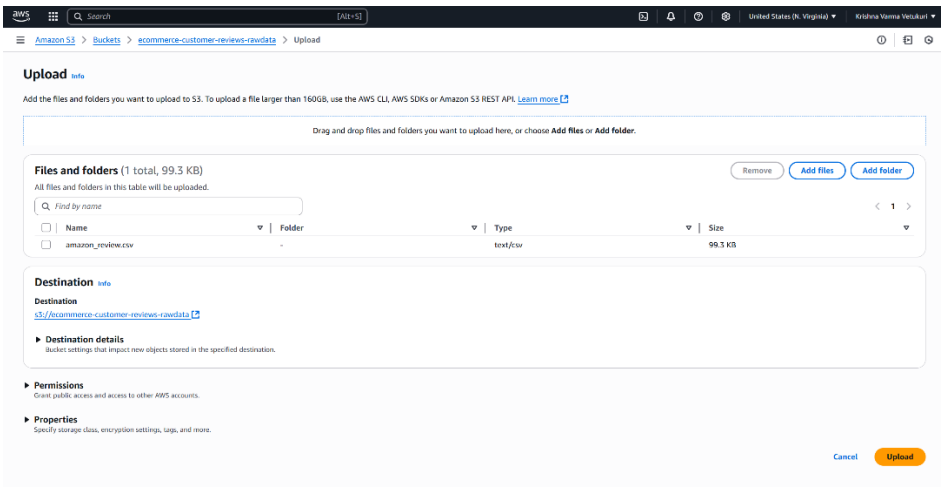
- Go to S3 → ecommerce-customer-reviews-rawdata

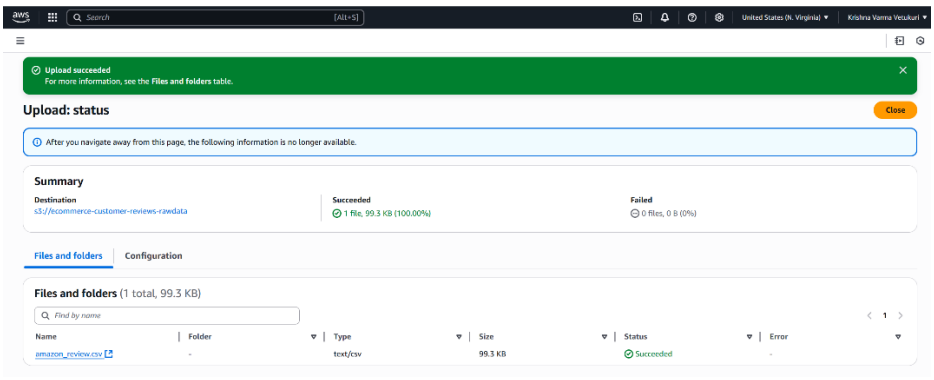


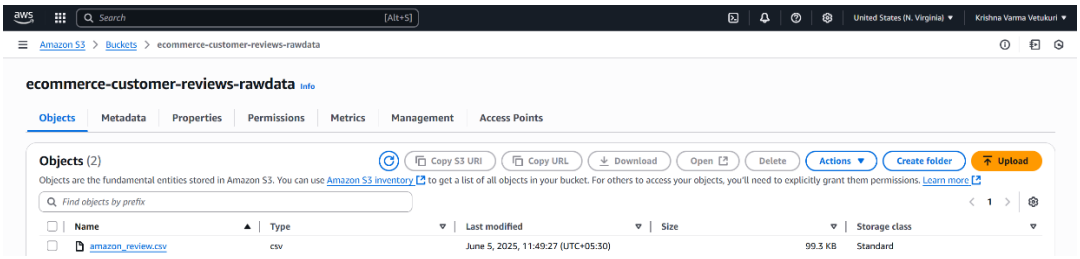


- Click Upload → Add files → Select amazon_review.csv → Click Upload





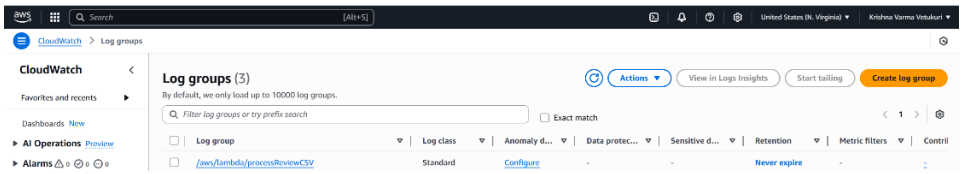




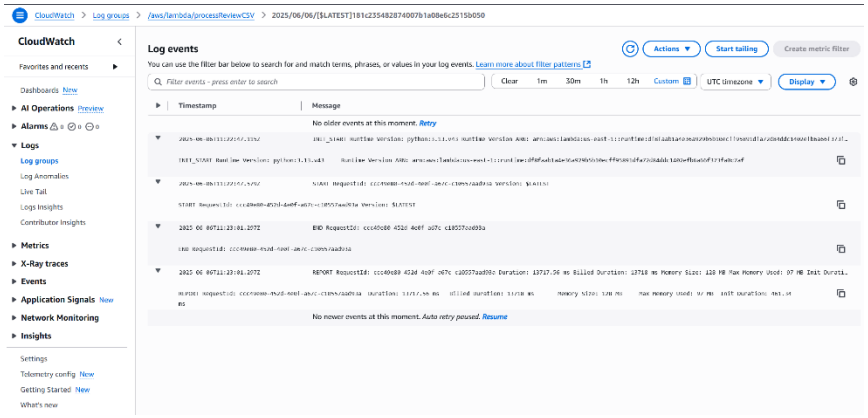
- Once uploaded, the Lambda function will be triggered.

3.3.4 Verifying Lambda Execution in CloudWatch

- Navigate to CloudWatch → Log groups.



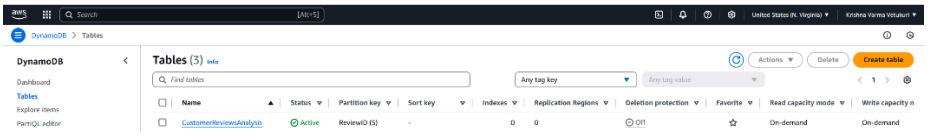
- Locate the log group for the processReviewCSV Lambda function.
- Open the most recent log stream.



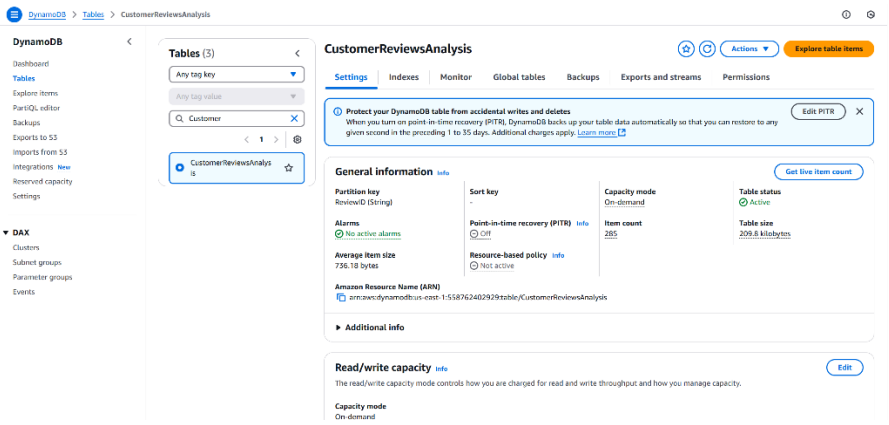
- Confirm successful execution (no errors in logs).

3.3.5 Confirming Data Storage in DynamoDB

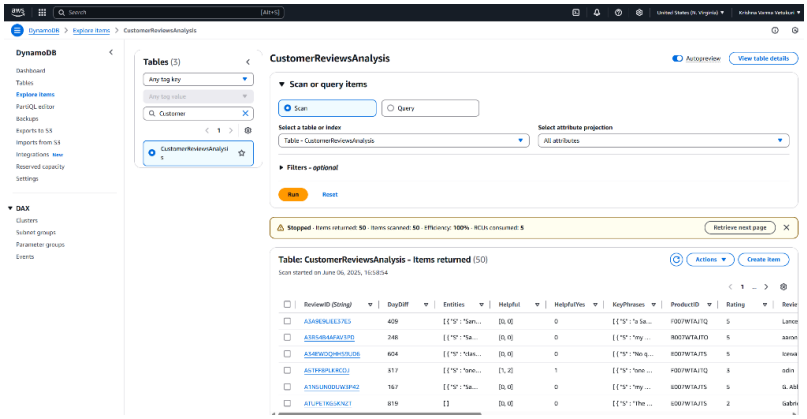
- Go to DynamoDB → Tables → CustomerReviewsAnalysis.



- Confirm that the Item count reflects the number of reviews (e.g., 285).



- Click Explore table items to browse the records.



- Each item should include:
    - Original fields: ReviewID, DayDiff, Helpful, HelpfulYes, ProductID, Rating, ReviewerName, ReviewText, ReviewTime, Summary, TotalVote, UnixReviewTime.

    - Derived fields: Entities, KeyPhrases, Sentiment, SentimentScores, SentimentScore_Mixed, SentimentScore_Negative, SentimentScore_Neutral, SentimentScore_Positive

3.4. Product Review Summaries

To help business teams quickly grasp customer feedback trends per product, we implemented an automated system that generates daily product-level review summaries using AWS services. These summaries include top key phrases mentioned by customers and sentiment distribution (positive, negative, etc.) for each product. The system leverages:

- Amazon DynamoDB to store summary outputs,
- AWS Lambda for summarization logic, and
- Amazon EventBridge for daily automation.

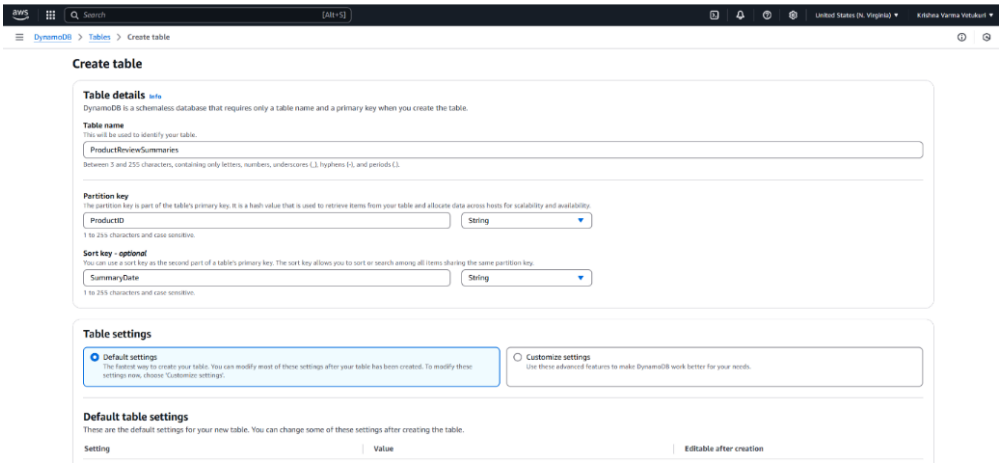
This component plays a crucial role in transforming unstructured text data into actionable product-specific insights.

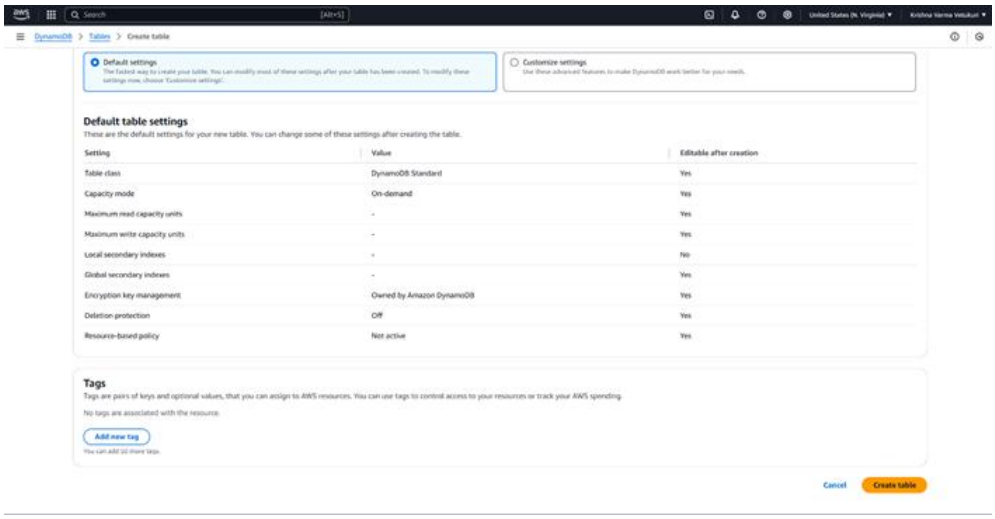
3.4.1. Create ProductReviewSummaries Table

We use DynamoDB to store each product's daily review summary.

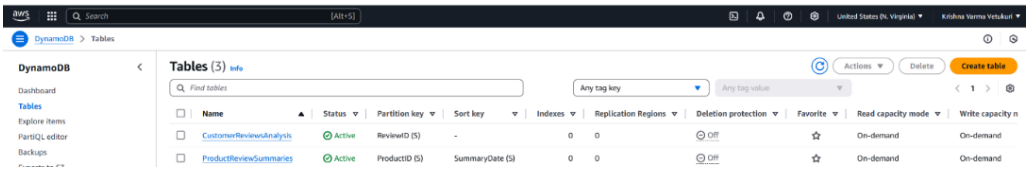
Steps:

- Navigate to DynamoDB > Tables > Create Table
- Set:
    - Table Name: ProductReviewSummaries
    - Partition Key: ProductID (String)
    - Sort Key: SummaryDate (String)





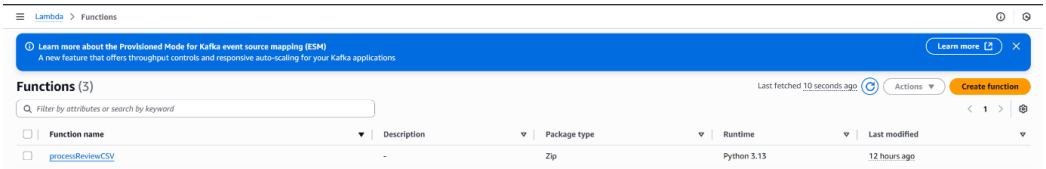
- Leave all other settings default and click Create Table



- Wait until the table is successfully created.

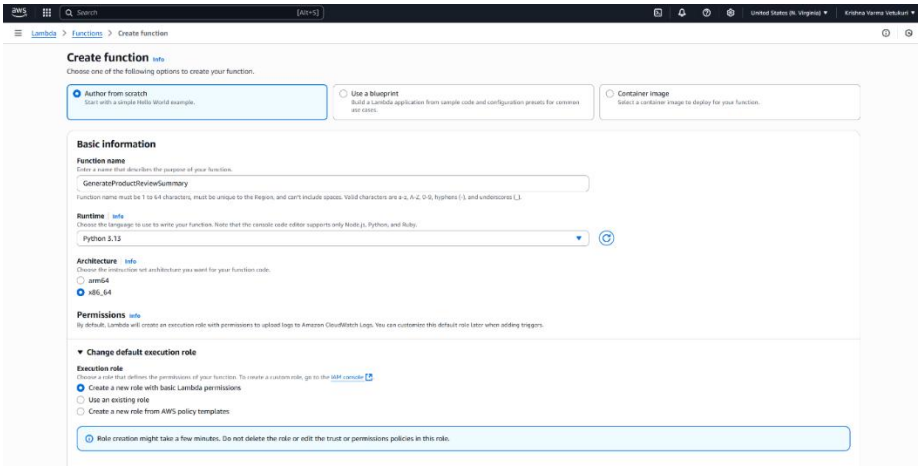
3.4.2. Create GenerateProductReviewSummary Lambda Function

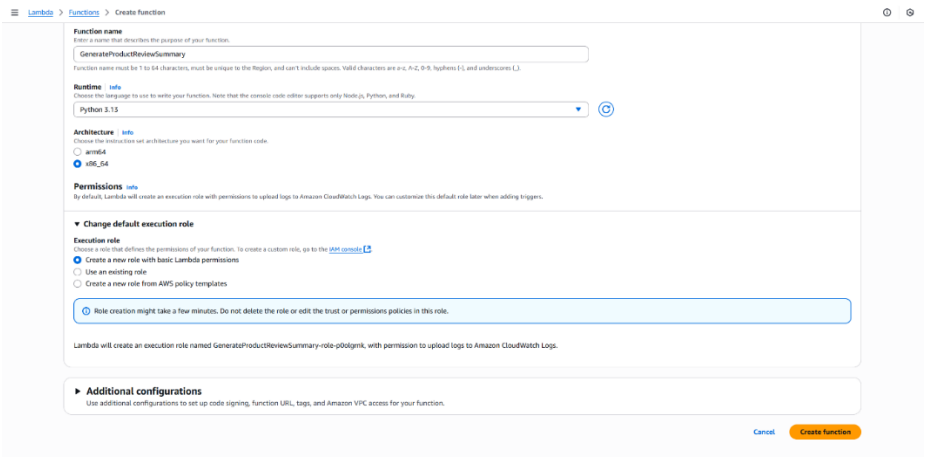
This function scans the CustomerReviewsAnalysis table, aggregates sentiment and key phrases per product, and stores the summaries in ProductReviewSummaries.



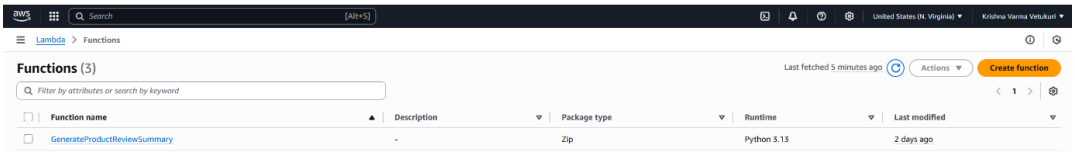
Steps:

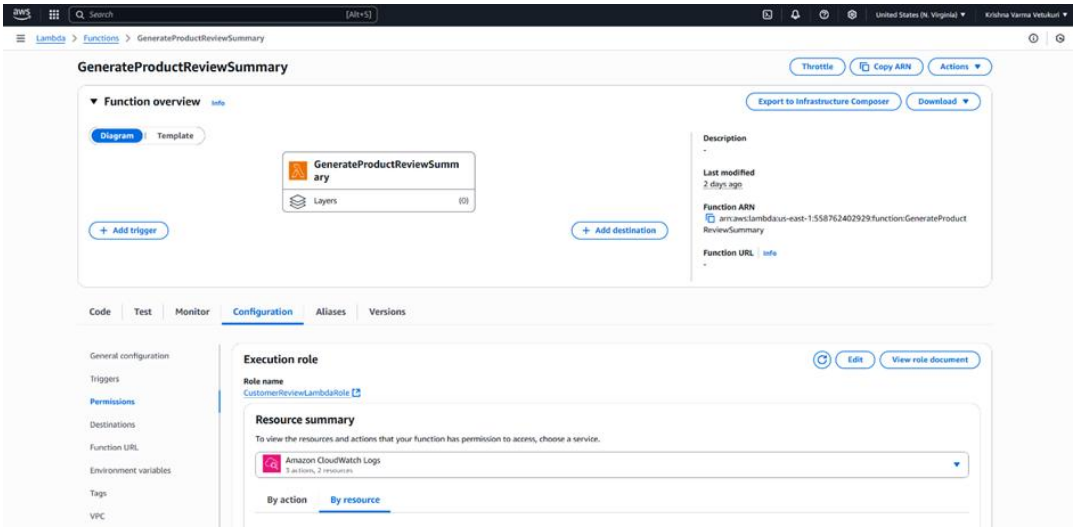
- Go to Lambda > Create Function
- Set:
    - Function Name: GenerateProductReviewSummary
    - Runtime: Python 3.13
    - Execution Role: Create a new role with basic Lambda permissions





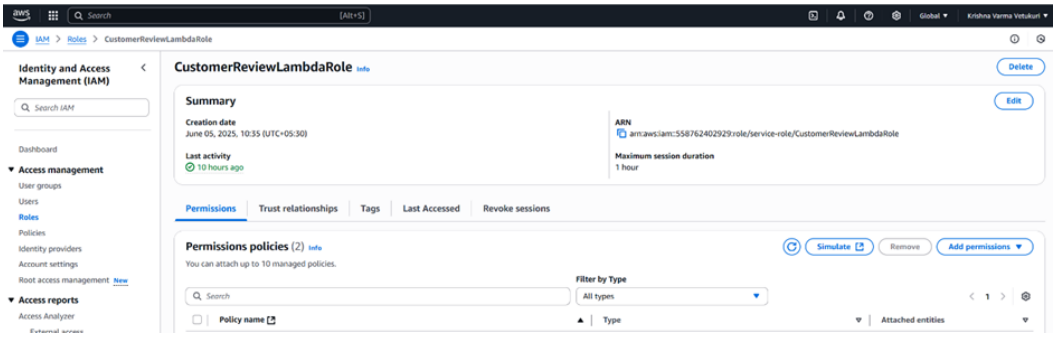
- Click Create Function and the GenerateProductReviewSummary Function is created and visible like below



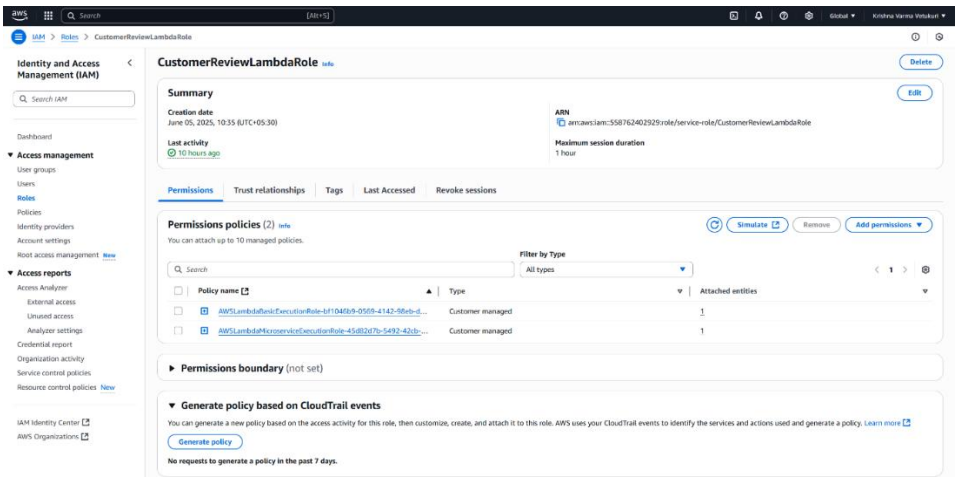


Update IAM Role Permissions:

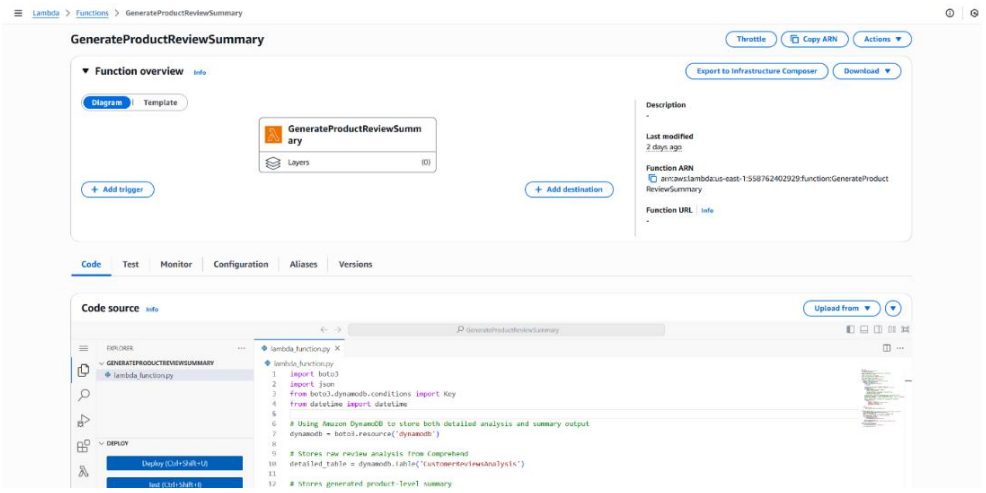
- Go to IAM > Roles > CustomerReviewLambdaRole



- Attach the following policies:
  - AWSLambdaBasicExecutionRole-bf1046b9-0569-4142-98eb-de925fb18328
  - AWSLambdaMicroserviceExecutionRole-45d82d7b-5492-42cb-9c30-063e9dfa2b39



Add Lambda Code:



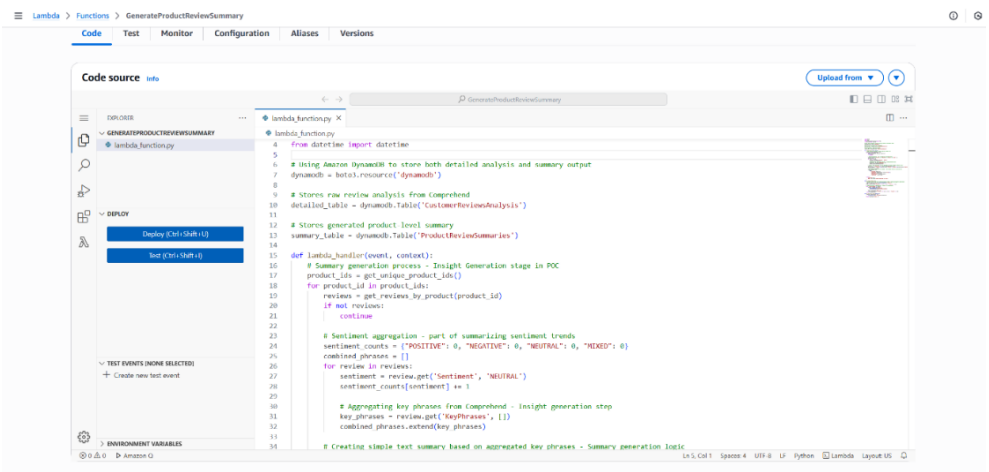
Paste the summarization code from the below.

GenerateProductReviewSummary lambda_function.py

```
import boto3
import json
from boto3.dynamodb.conditions import Key
from datetime import datetime

# Using Amazon DynamoDB to store both detailed analysis and summary output
dynamodb = boto3.resource('dynamodb')

# Stores raw review analysis from Comprehend
detailed_table = dynamodb.Table('CustomerReviewsAnalysis')

# Stores generated product-level summary
summary_table = dynamodb.Table('ProductReviewSummaries')

def lambda_handler(event, context):
    # Summary generation process - Insight Generation stage
    product_ids = get_unique_product_ids()
    for product_id in product_ids:
        reviews = get_reviews_by_product(product_id)
        if not reviews:
            continue

        # Sentiment aggregation - part of summarizing sentiment trends
        sentiment_counts = {"POSITIVE": 0, "NEGATIVE": 0, "NEUTRAL": 0, "MIXED": 0}
        combined_phrases = []
        for review in reviews:
            sentiment = review.get('Sentiment', 'NEUTRAL')
            sentiment_counts[sentiment] += 1

            # Aggregating key phrases from Comprehend - Insight generation step
            key_phrases = review.get('KeyPhrases', [])
            combined_phrases.extend(key_phrases)

        # Creating simple text summary based on aggregated key phrases - Summary generation logic
        summary_text = f"Top phrases: {', '.join(set(combined_phrases[:10]))}"
        sentiment_summary = f"Positive: {sentiment_counts['POSITIVE']}, Negative:
```

```
{sentiment_counts['NEGATIVE']}, Neutral: {sentiment_counts['NEUTRAL']}"
        total_reviews = len(reviews)

        # Storing summarized insights into DynamoDB - Final insight storage in structured form
        summary_table.put_item(
            Item={
                'ProductID': product_id,
                'SummaryDate': datetime.now().strftime('%Y-%m-%d'),
                'SummaryText': summary_text,
                'SentimentSummary': sentiment_summary,
                'TotalReviews': total_reviews
            }
        )
    return {
        'statusCode': 200,
        'body': json.dumps('Summaries generated and saved.')
    }

def get_unique_product_ids():
    # Fetching distinct ProductIDs from detailed analysis table - supports grouping for summarization
    response = detailed_table.scan(ProjectionExpression='ProductID')
    items = response.get('Items', [])
    product_ids = set(item['ProductID'] for item in items if 'ProductID' in item)

    # Handling pagination of DynamoDB scan - scalable for large datasets
    while 'LastEvaluatedKey' in response:
        response = detailed_table.scan(
            ProjectionExpression='ProductID',
            ExclusiveStartKey=response['LastEvaluatedKey']
        )
        items = response.get('Items', [])
        product_ids.update(item['ProductID'] for item in items if 'ProductID' in item)
    return list(product_ids)

def get_reviews_by_product(product_id):
    response = detailed_table.scan(
        FilterExpression=Key('ProductID').eq(product_id)
    )
    return response.get('Items', [])
```

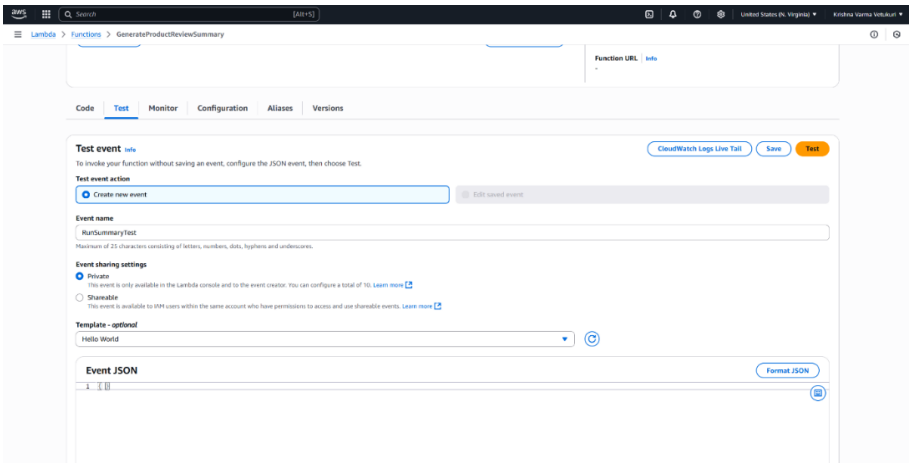
Click Deploy.



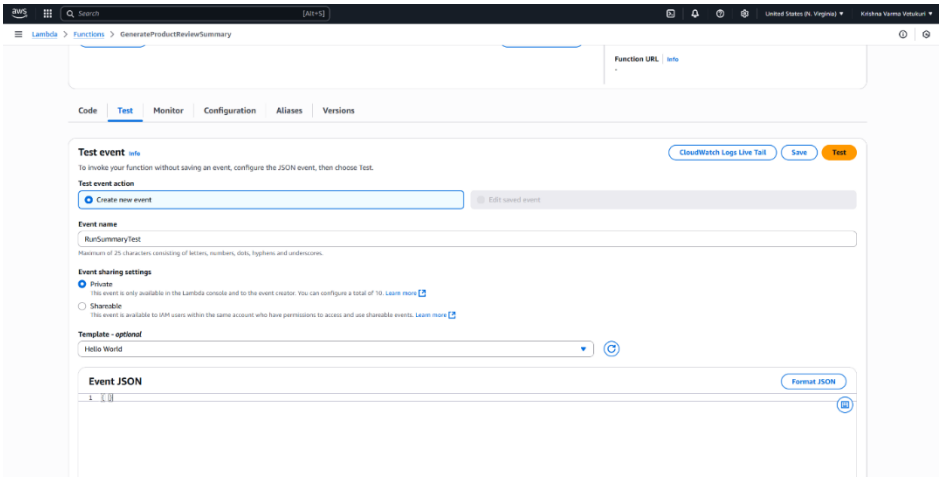
3.4.3. Manual Testing of Product Summarization

Steps:

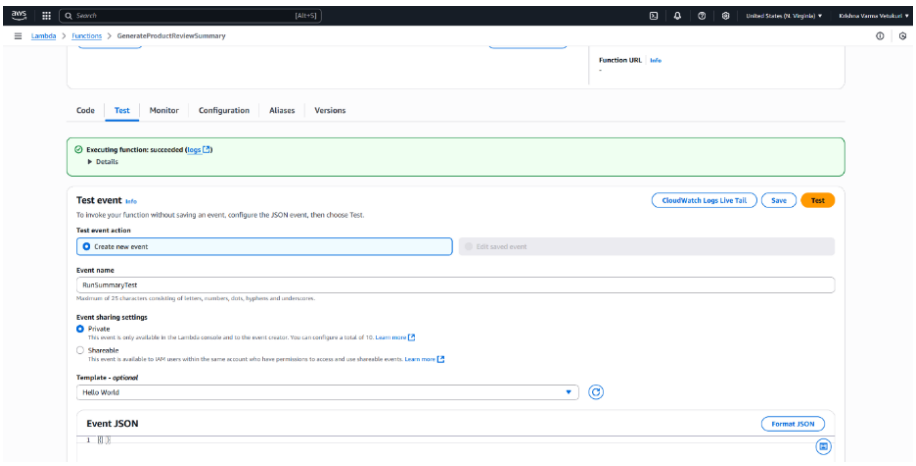
- In Lambda, click Test



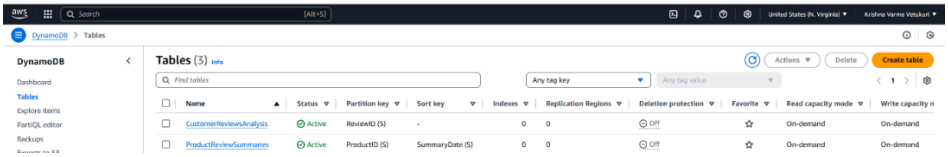
- Create a test event named: RunSummaryTest
- Payload: {} (empty JSON)

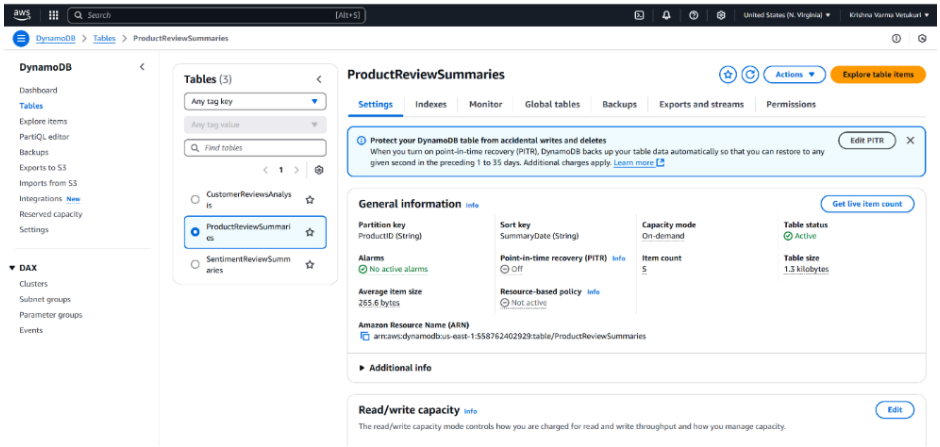


- Click Test
- On success, we'll see the message: Execution succeeded

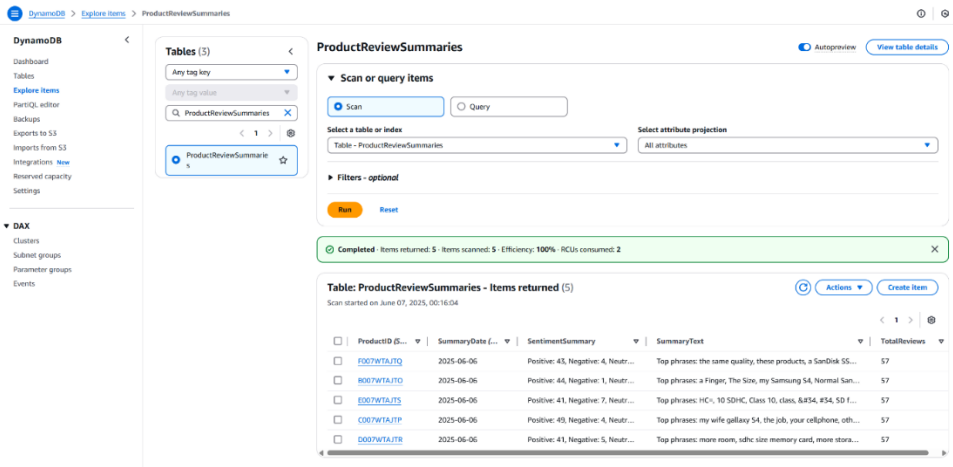


- Go to DynamoDB > ProductReviewSummaries > Explore Table Items





- Confirm that new entries are added.

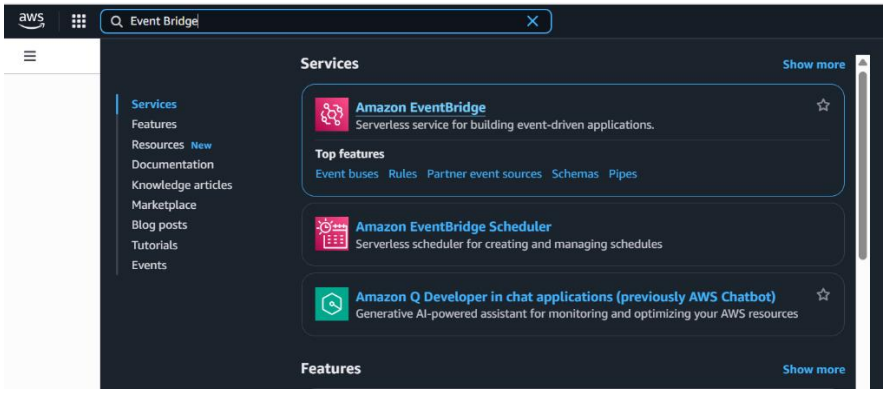


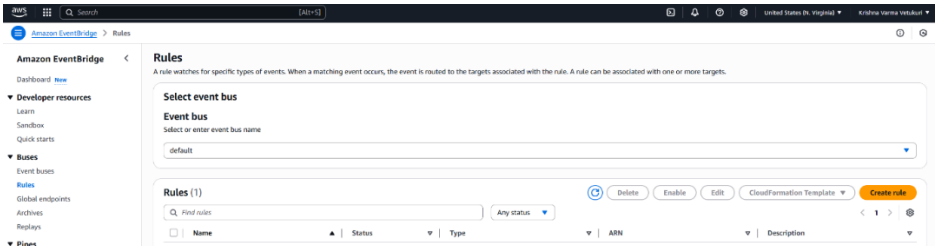
3.4.4. Automation via Amazon EventBridge

To avoid manual testing daily, we automate the Lambda execution with EventBridge.

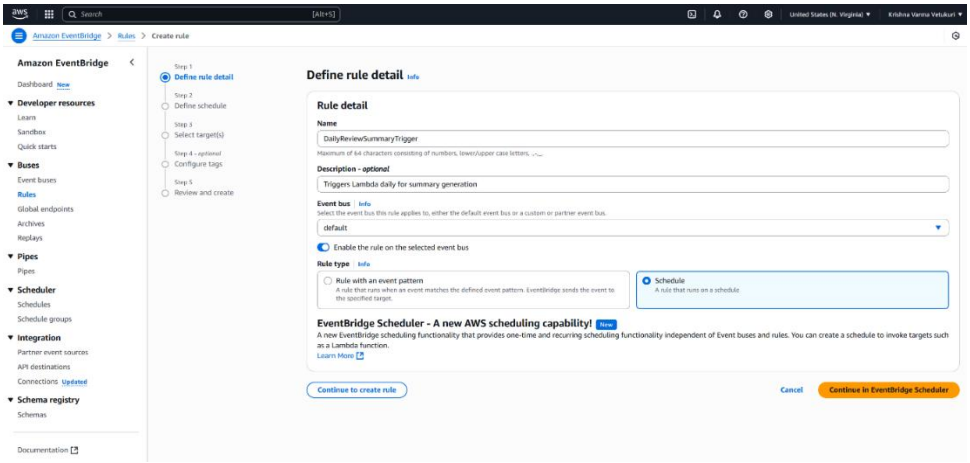
Steps:

- Go to Amazon EventBridge > Rules > Create Rule

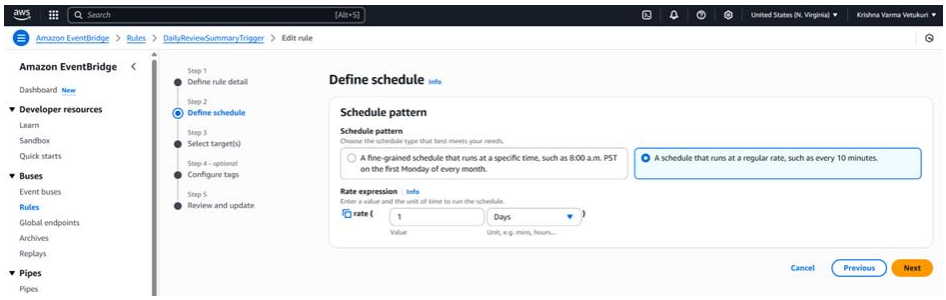


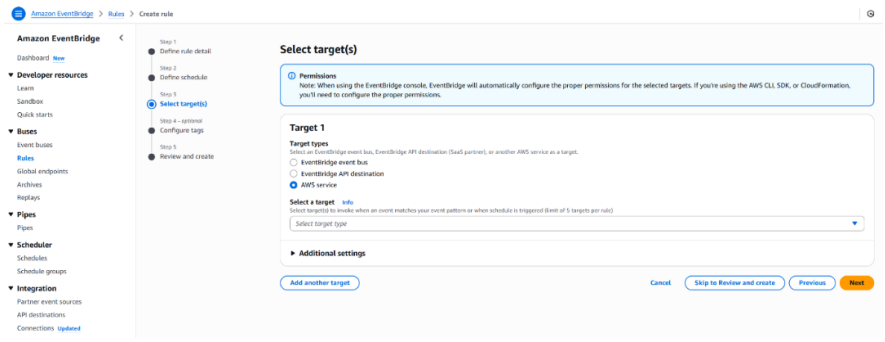


- Step 1: Name = DailyReviewSummaryTrigger, Rule Type = Schedule

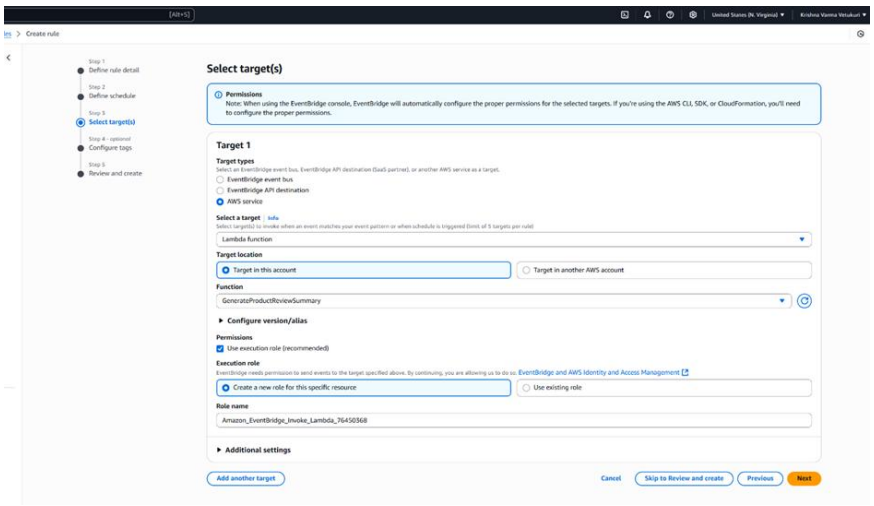


- Step 2: Choose schedule: Rate → Every 1 day

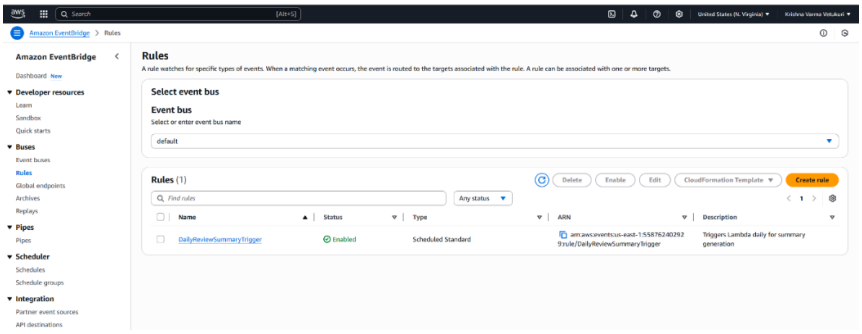




- Step 3: Set Target as AWS Lambda Function → Select GenerateProductReviewSummary



- Leave other settings as-is and click Create Rule



3.5. Product Review Summaries

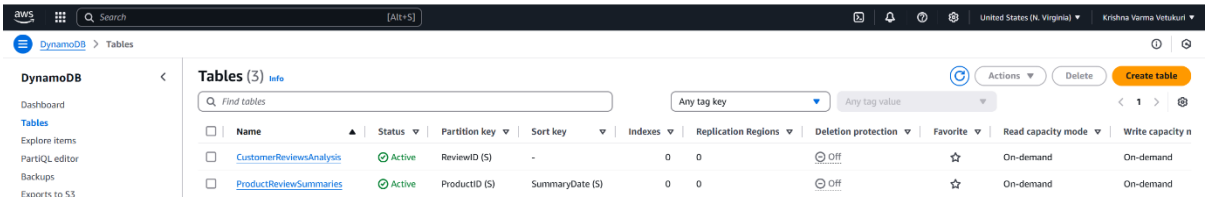
In addition to product-wise insights, understanding customer emotions across the platform is equally vital. To address this, we implemented sentiment-level summaries that group customer reviews by sentiment type (Positive, Negative, Neutral, Mixed). These summaries help stakeholders monitor sentiment trends and detect shifts in overall customer satisfaction.

We use Amazon DynamoDB to store daily sentiment summaries, AWS Lambda to process and aggregate key phrases for each sentiment, and EventBridge for scheduled execution.

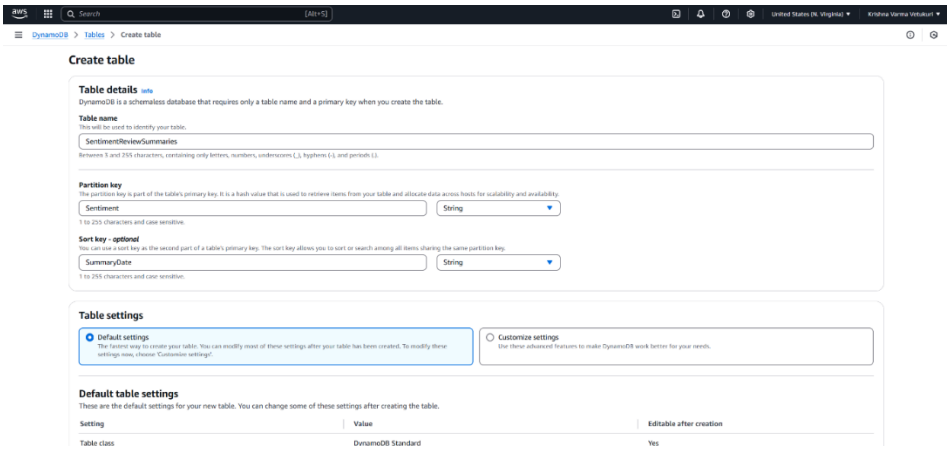
3.5.1. Create SentimentReviewSummaries Table

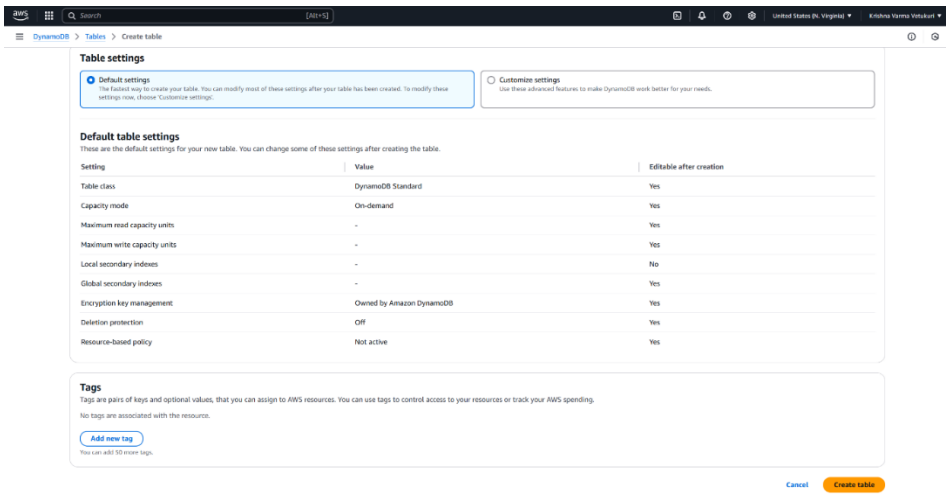
This table stores one summary per sentiment category each day.

Steps:

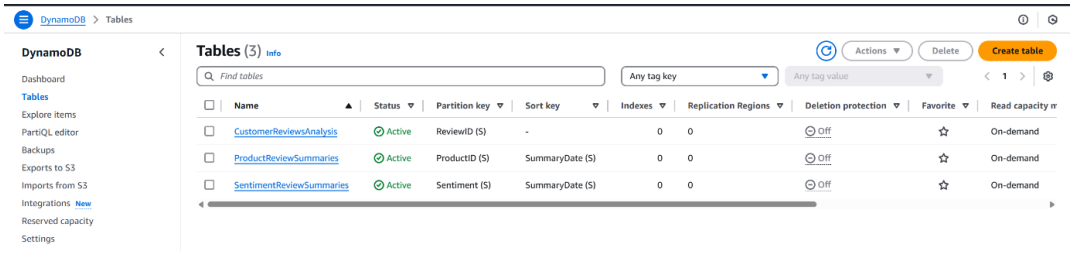


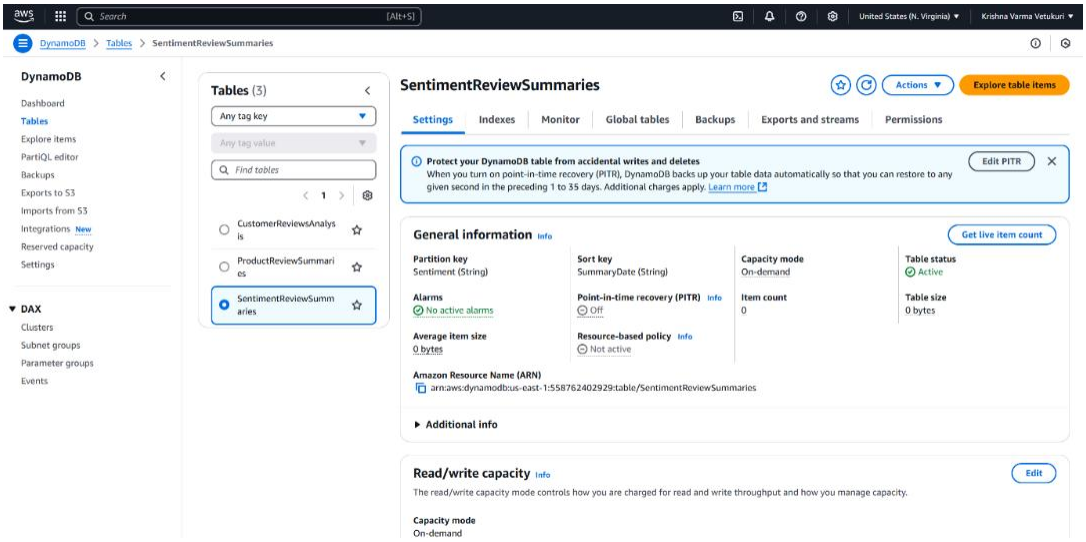
- Go to DynamoDB > Tables > Create Table
- Set:
    - Table Name: SentimentReviewSummaries
    - Partition Key: Sentiment (String)
    - Sort Key: SummaryDate (String)





- Leave all other settings default and click Create Table
- Wait until the table is created successfully.



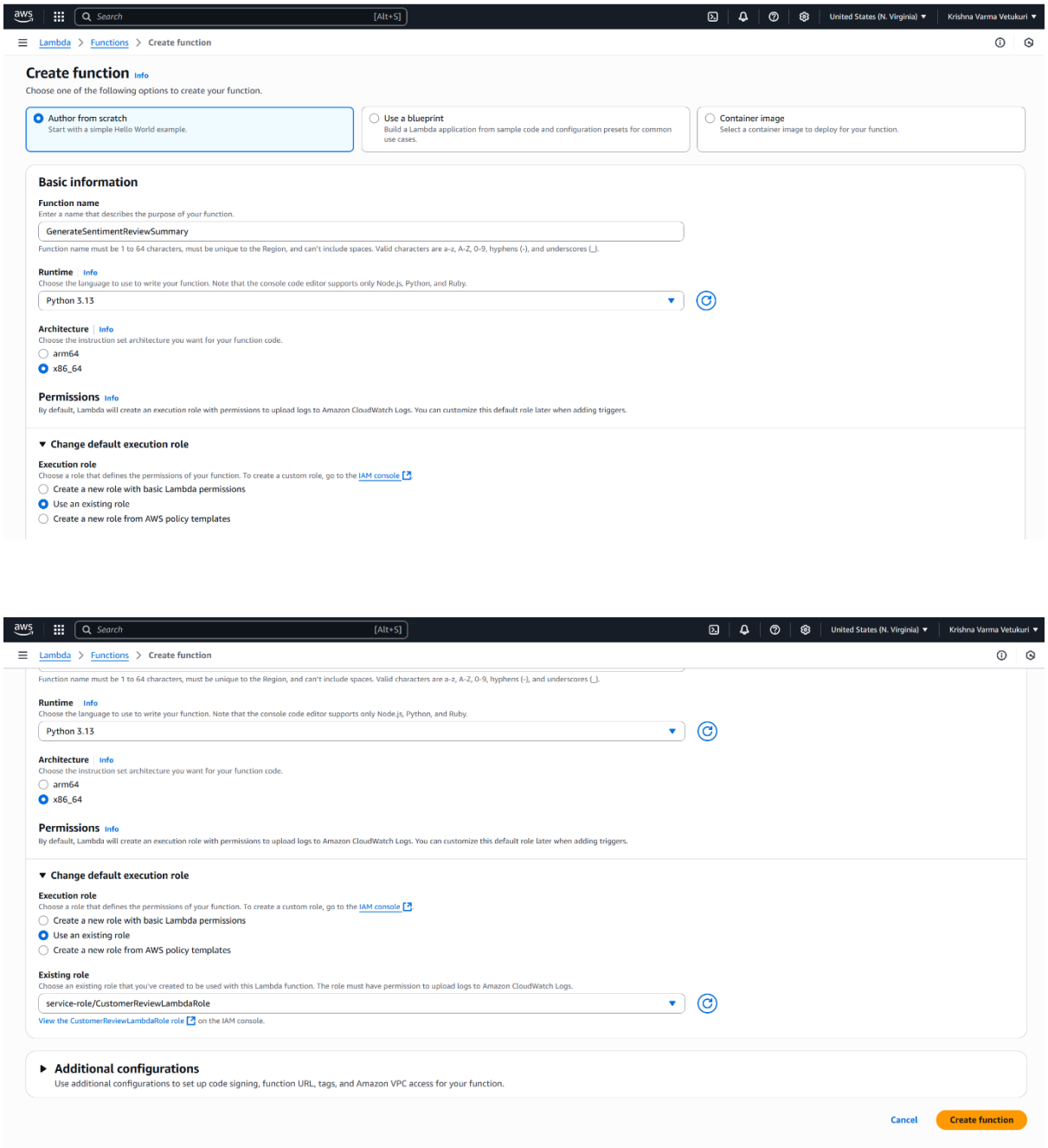


3.5.2. Create GenerateSentimentReviewSummary Lambda Function

This function scans all reviews from CustomerReviewsAnalysis, groups them by sentiment, aggregates key phrases for each group, and stores summaries in SentimentReviewSummaries.

Steps:

- Go to Lambda > Create Function
- Set:
    - Function Name: GenerateSentimentReviewSummary
    - Runtime: Python 3.13
    - Execution Role: Use existing role → service-role/CustomerReviewLambdaRole (used earlier)

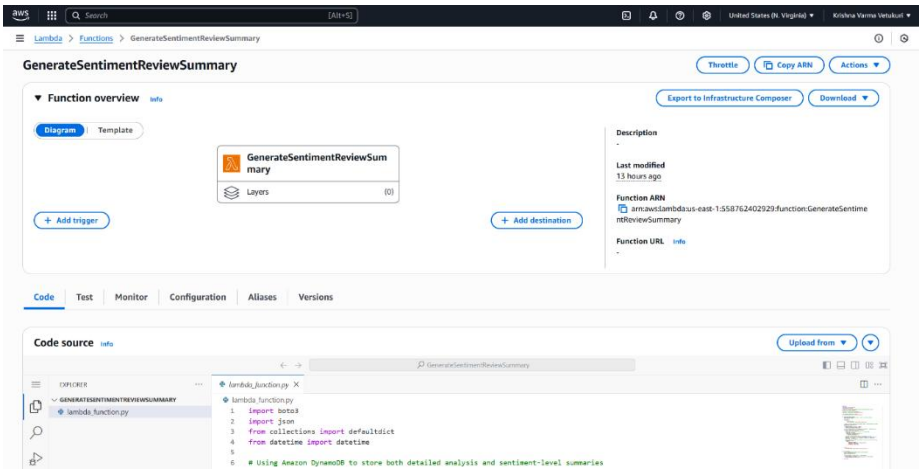


- Click Create Function

IAM Permissions

Since the reused role (CustomerReviewLambdaRole) already has the necessary permissions (DynamoDB, S3, Comprehend), no changes are needed in IAM.

Add Lambda Code:



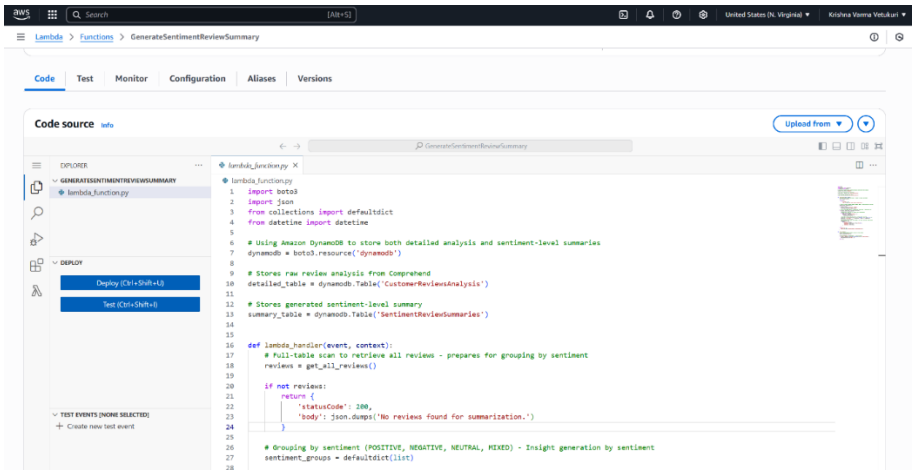
Paste the provided sentiment-level summarization code into the code editor.

GenerateSentimentReviewSummary lambda_function.py

```
import boto3
import json
from collections import defaultdict
from datetime import datetime

# Using Amazon DynamoDB to store both detailed analysis and sentiment-level summaries
dynamodb = boto3.resource('dynamodb')

# Stores raw review analysis from Comprehend
detailed_table = dynamodb.Table('CustomerReviewsAnalysis')

# Stores generated sentiment-level summary
summary_table = dynamodb.Table('SentimentReviewSummaries')

def lambda_handler(event, context):
    # Full-table scan to retrieve all reviews - prepares for grouping by sentiment
    reviews = get_all_reviews()

    if not reviews:
        return {
            'statusCode': 200,
            'body': json.dumps('No reviews found for summarization.')
        }

    # Grouping by sentiment (POSITIVE, NEGATIVE, NEUTRAL, MIXED) - Insight generation by sentiment
    sentiment_groups = defaultdict(list)

    for review in reviews:
        sentiment = review.get('Sentiment', 'NEUTRAL')
        sentiment_groups[sentiment].append(review)

    # For each sentiment group, generate a summary based on key phrases - Summarization logic
    for sentiment, group_reviews in sentiment_groups.items():
        key_phrase_counter = defaultdict(int)
```

```
        # Aggregating key phrases from all reviews in this sentiment group
        for review in group_reviews:
            key_phrases = review.get('KeyPhrases', [])
            for phrase in key_phrases:
                key_phrase_counter[phrase] += 1

        # Selecting top 10 most common key phrases - relevant insight extraction
        top_phrases = sorted(key_phrase_counter.items(), key=lambda x: x[1], reverse=True)[:10]
        summary_text = "Top phrases: " + ", ".join([phrase for phrase, _ in top_phrases])
        total_reviews = len(group_reviews)

        # Storing sentiment-wise summary into DynamoDB - Final stage: structured insight storage
        summary_table.put_item(
            Item={
                'Sentiment': sentiment,
                'SummaryDate': datetime.now().strftime('%Y-%m-%d'),
                'SummaryText': summary_text,
                'TotalReviews': total_reviews
            }
        )

    return {
        'statusCode': 200,
        'body': json.dumps('Sentiment summaries generated and saved.')
    }

def get_all_reviews():
    # Scans the detailed analysis table to retrieve all review entries
    response = detailed_table.scan()
    items = response.get('Items', [])

    # Handling pagination for larger datasets - scalability support
    while 'LastEvaluatedKey' in response:
        response = detailed_table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])
        items.extend(response.get('Items', []))

    return items
```

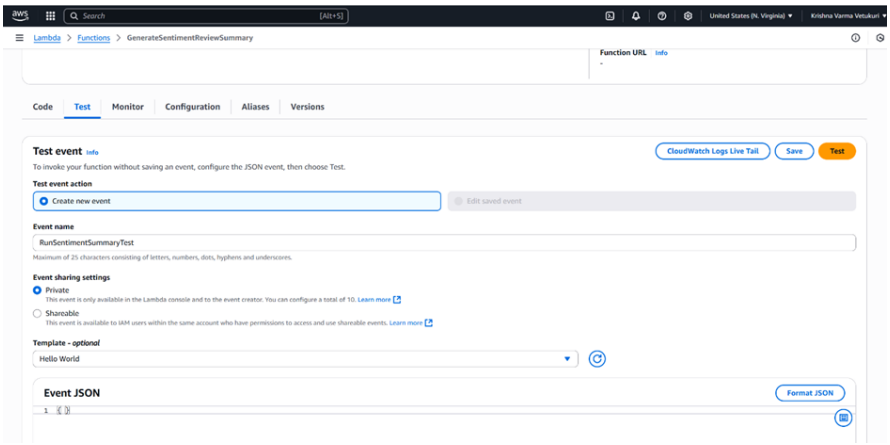
Click Deploy



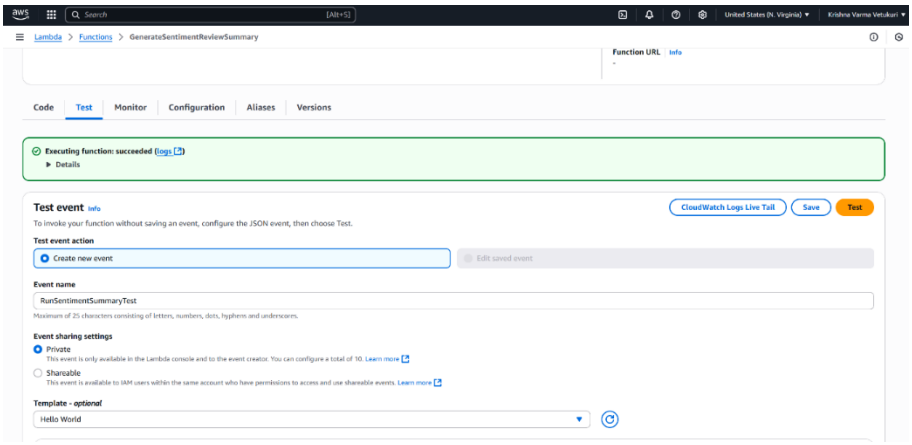
3.5.3. Manual Testing of Sentiment Summarization

Steps:

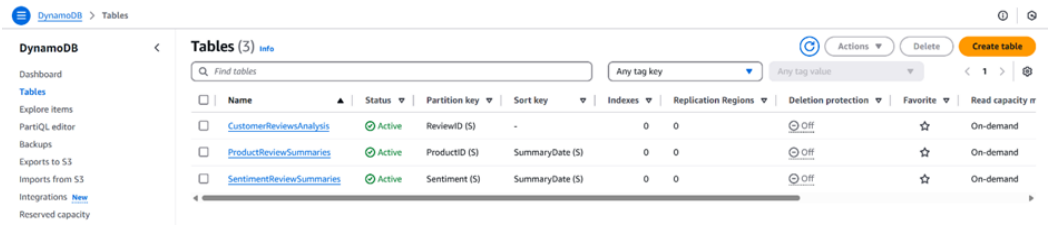
- In Lambda, click Test
- Create a test event named: RunSentimentSummaryTest
- Payload: {} (empty JSON)

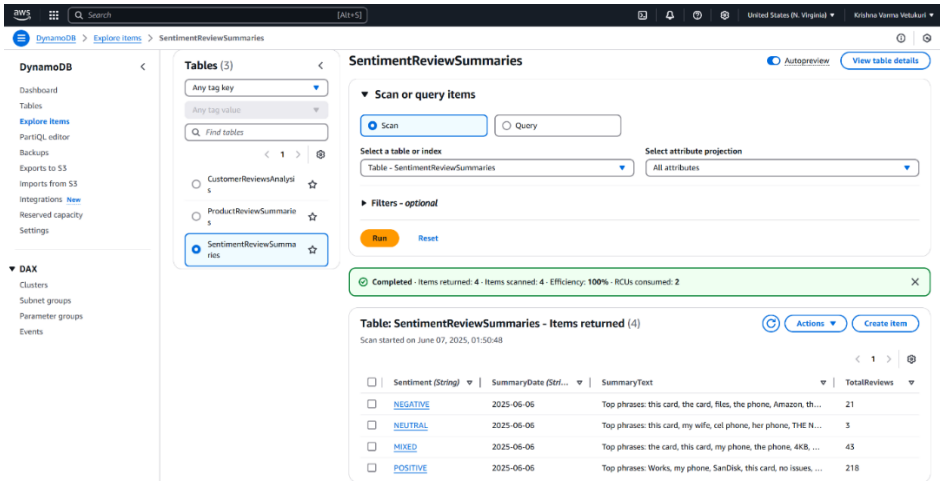


- Click Test
- On success, we'll see the message: Execution succeeded



- Go to DynamoDB > SentimentReviewSummaries > Explore Table Items





- Confirm that new entries are populated.

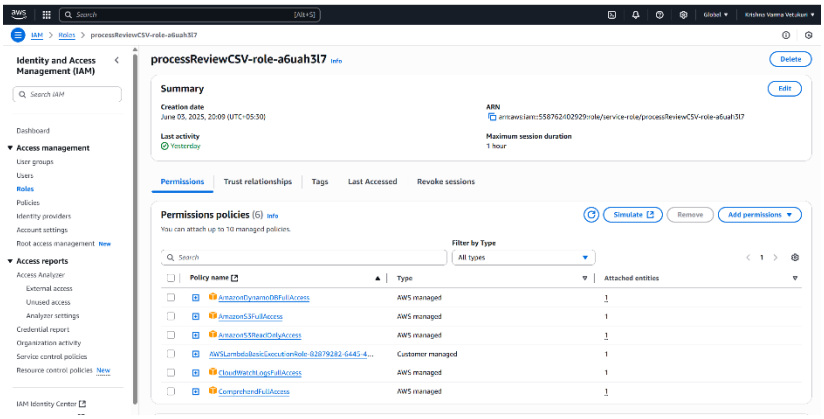
3.6 Exporting DynamoDB Data to S3 for Further Processing

To enable further data cleaning, visualization, and integration with services like QuickSight, we export the entire CustomerReviewsAnalysis DynamoDB table into both JSON and CSV formats in an S3 bucket. This allows us to easily manipulate and analyze the data using standard tools.

3.6.1 Updating IAM Role Permissions

To allow the Lambda function to upload files to Amazon S3, we first update the execution role's permissions.

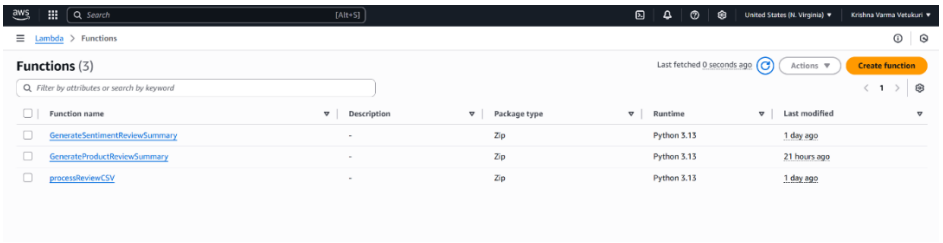
- Go to IAM > Roles
- Search and open the role: processReviewCSV-role-a6uah3l7
- Attach the policy: AmazonS3FullAccess



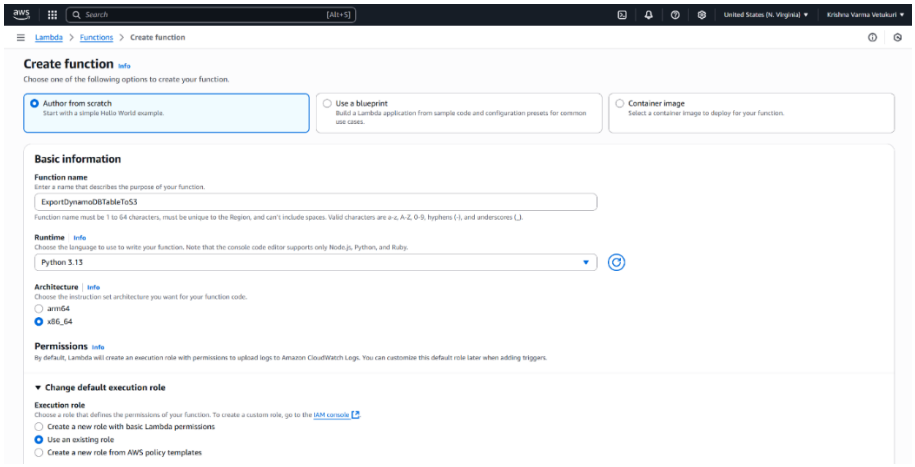
3.6.2 Creating Lambda Function: ExportDynamoDBTableToS3

Next, we create a new Lambda function that will handle the export task.

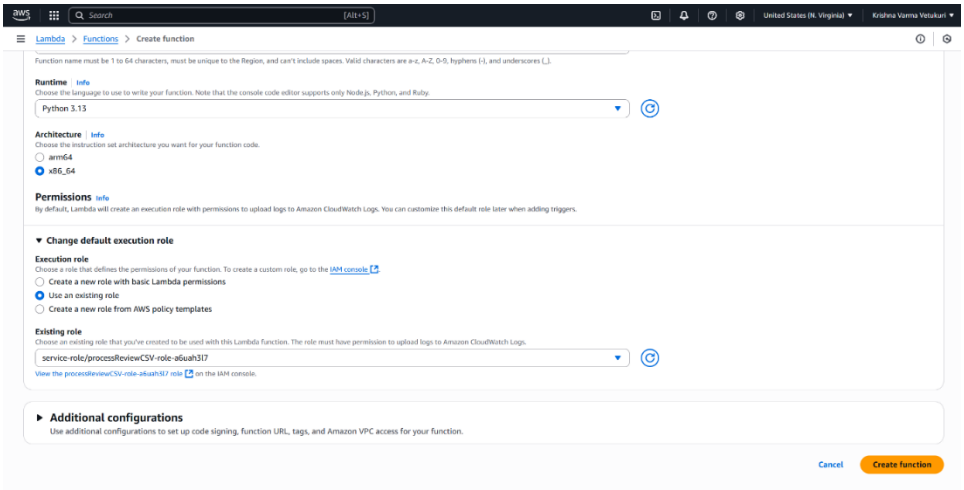
- Open AWS Lambda.



- Click Create Function
- Fill in:
    - Function Name: ExportDynamoDBTableToS3
    - Runtime: Python 3.13



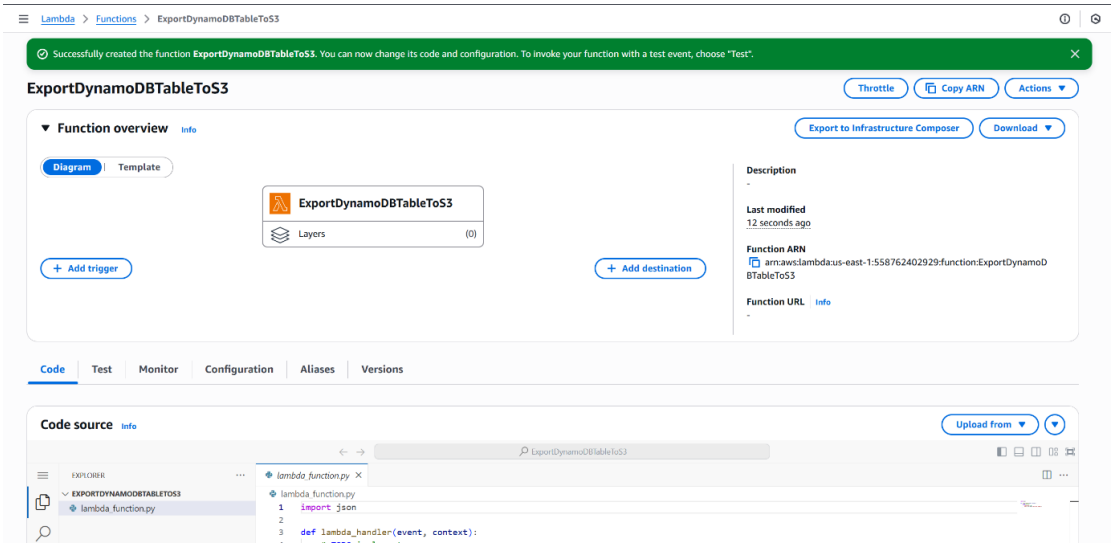
- Under Execution Role:
    - Choose: Use existing role
    - Select: processReviewCSV-role-a6uah3l7



- Click Create Function

3.6.3 Lambda Function Code Highlights

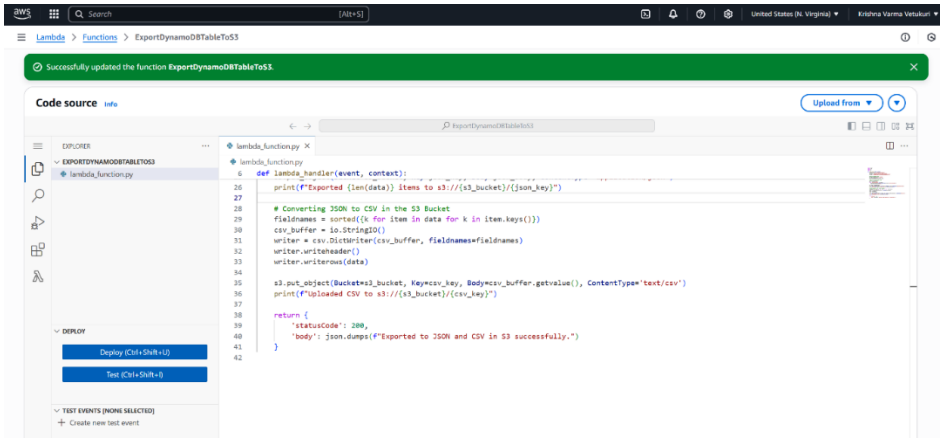
After the function is created, we navigate to the Code tab and paste the following code into lambda_function.py:



ExportDynamoDBTableToS3 lambda_function.py

```
import boto3
import json
import csv
import io

def lambda_handler(event, context):
    dynamodb_table = 'CustomerReviewsAnalysis'
    s3_bucket = 'ecommerce-customer-reviews-rawdata'
    json_key = 'dynamodb-export/CustomerReviewsAnalysis.json'
    csv_key = 'dynamodb-export/CustomerReviewsAnalysis.csv'

    # Initializing AWS clients
    dynamodb = boto3.resource('dynamodb')
    s3 = boto3.client('s3')
    table = dynamodb.Table(dynamodb_table)

    # Exporting from DynamoDB to JSON
    response = table.scan()
    data = response['Items']
    while 'LastEvaluatedKey' in response:
        response = table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])
        data.extend(response['Items'])

    json_body = json.dumps(data)
    s3.put_object(Bucket=s3_bucket, Key=json_key, Body=json_body, ContentType='application/json')
    print(f"Exported {len(data)} items to s3://{s3_bucket}/{json_key}")

    # Converting JSON to CSV in the S3 Bucket
    fieldnames = sorted({k for item in data for k in item.keys()})
    csv_buffer = io.StringIO()
    writer = csv.DictWriter(csv_buffer, fieldnames=fieldnames)
```

```
writer.writeheader()
writer.writerows(data)

s3.put_object(Bucket=s3_bucket, Key=csv_key, Body=csv_buffer.getvalue(), ContentType='text/csv')
print(f"Uploaded CSV to s3://{s3_bucket}/{csv_key}")

return {
    'statusCode': 200,
    'body': json.dumps(f"Exported to JSON and CSV in S3 successfully.")
}
```

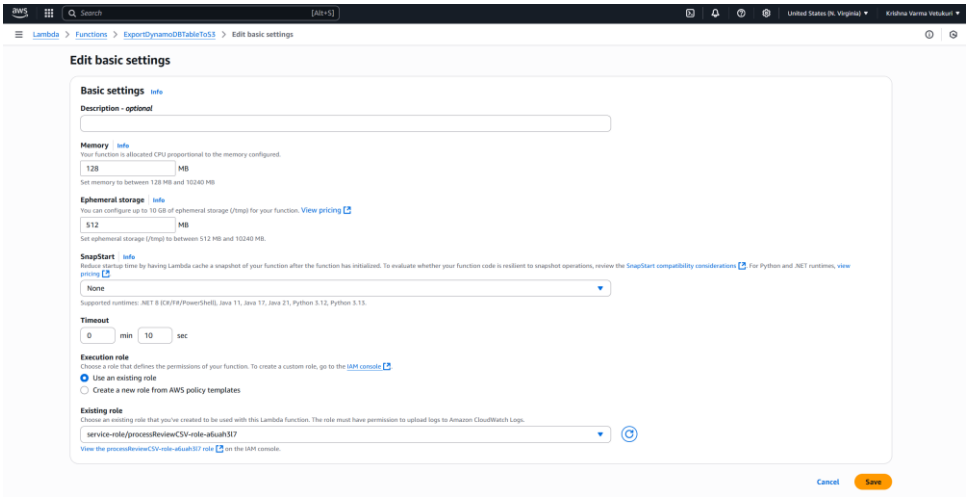


Once done, click Deploy and wait until we see a “Deployed successfully” message.

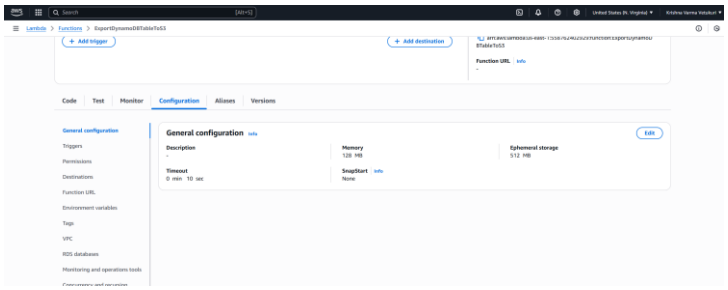
3.6.4 Adjusting Lambda Timeout

Since the export operation may take more than a few seconds (typically 5–7 seconds), we increase the timeout limit:

- Go to the Configuration tab
- Click General Configuration → Edit
- Change timeout from 3 seconds to 10 seconds



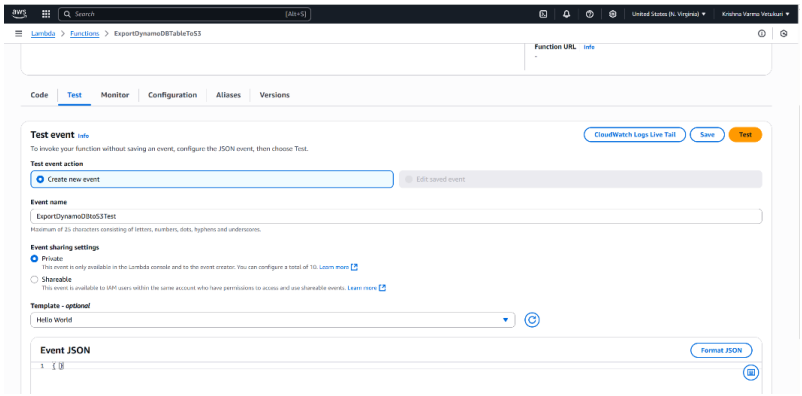
- Click Save



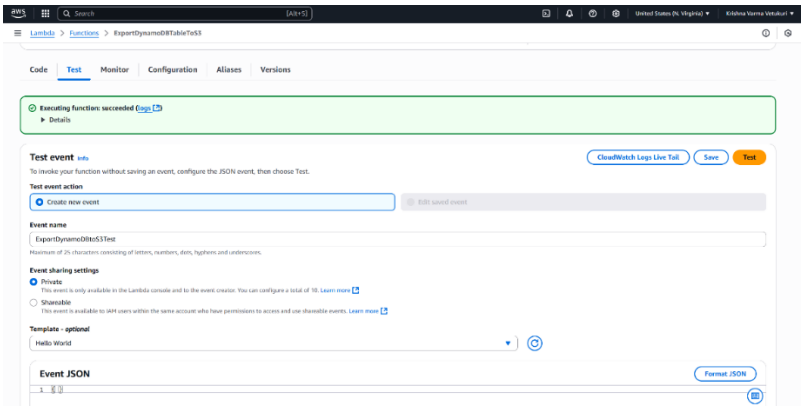
3.6.5 Testing the Function

Now we test the function:

- Go to the Test tab in Lambda
- Click Create New Test
- Set:
    - Name: ExportDynamoDBtoS3Test
    - Event JSON: { }



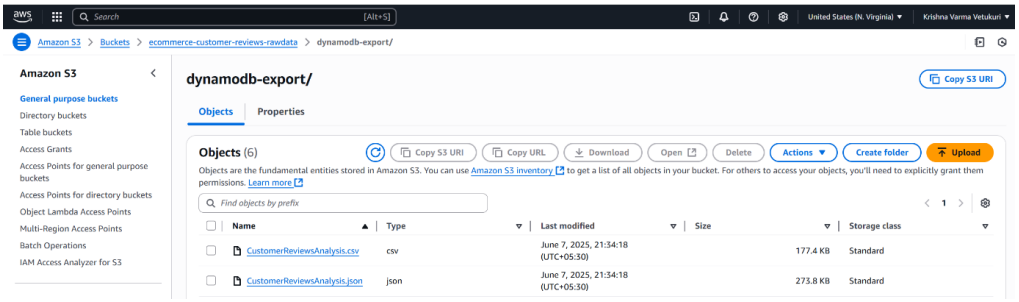
- Leave all other settings as-is
- Click Test



Wait for execution to complete. We should see a success response message and the logs confirming the number of items exported.

3.6.6 Verifying S3 Export

- To confirm that the data has been exported:
- Open the S3 Console
- Navigate to the bucket: ecommerce-customer-reviews-rawdata
- Open the folder: dynamodb-export/



- We should see:
  - CustomerReviewsAnalysis.json
  - CustomerReviewsAnalysis.csv

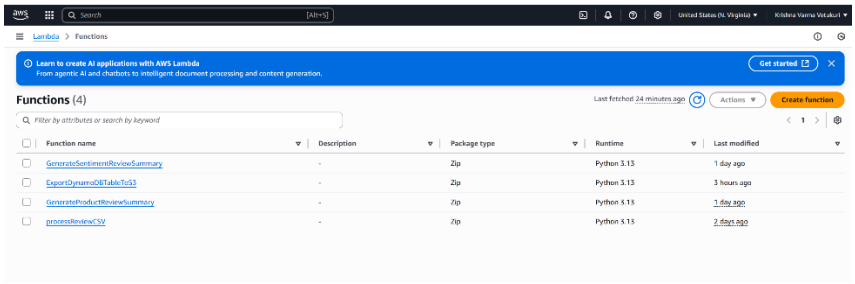
3.7 Preparing Cleaned CSV for Amazon QuickSight

To visualize customer review insights in Amazon QuickSight, the exported DynamoDB data must be cleaned and formatted properly. This step involves creating a Lambda function that processes the previously exported CustomerReviewsAnalysis.csv, cleans it by removing problematic characters, retains only relevant fields, and generates a corresponding manifest.json for QuickSight to interpret the CSV.

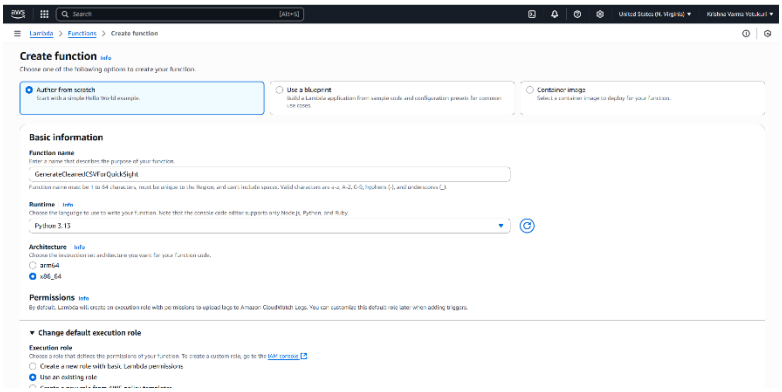
3.7.1 Creating the Lambda Function: GenerateCleanedCSVForQuickSight

Start by creating a new Lambda function that will clean the CSV and generate a manifest file.

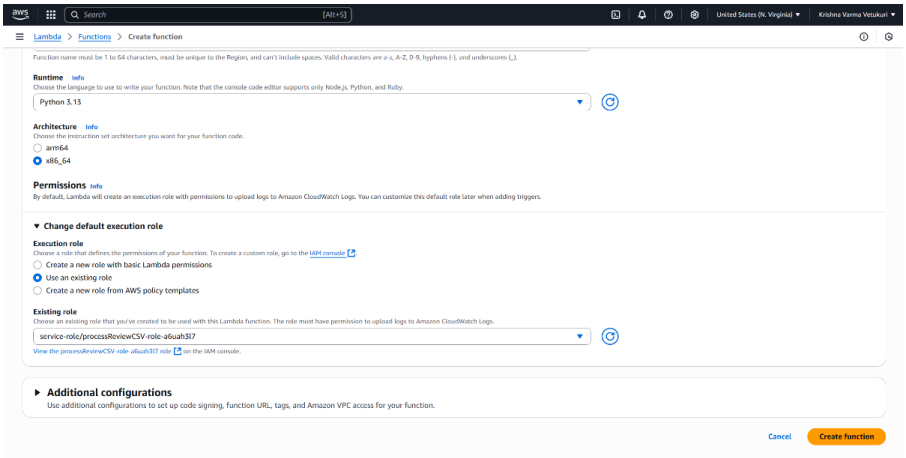
- Go to AWS Lambda > Create Function



- Enter:
  - Function name: GenerateCleanedCSVForQuickSight
  - Runtime: Python 3.13



- Under Execution Role:
    - Select Use an existing role
    - Choose: service-role/processReviewCSV-role-a6uah3l7

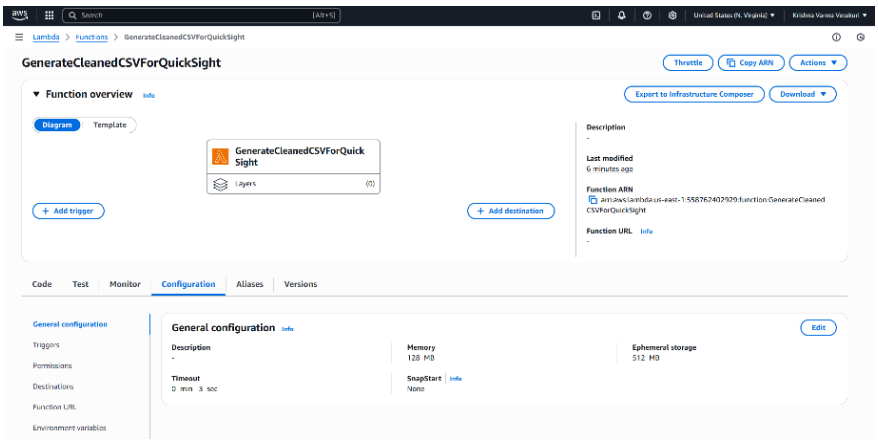


- Click Create Function

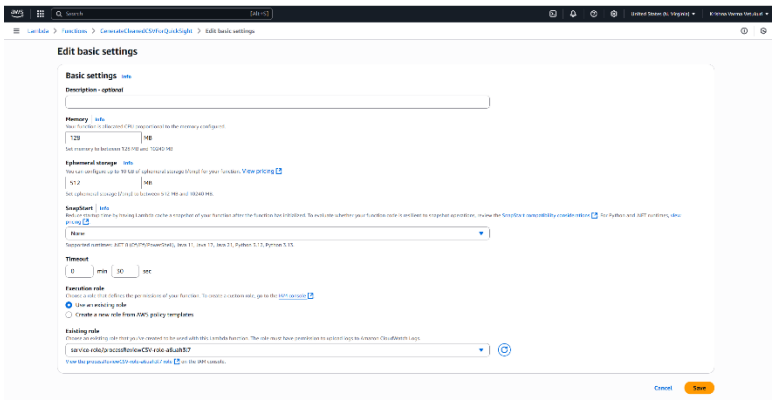
3.7.2 Increasing Function Timeout

Since cleaning and exporting can take more than a few seconds, update the function timeout.

- Go to the Configuration tab



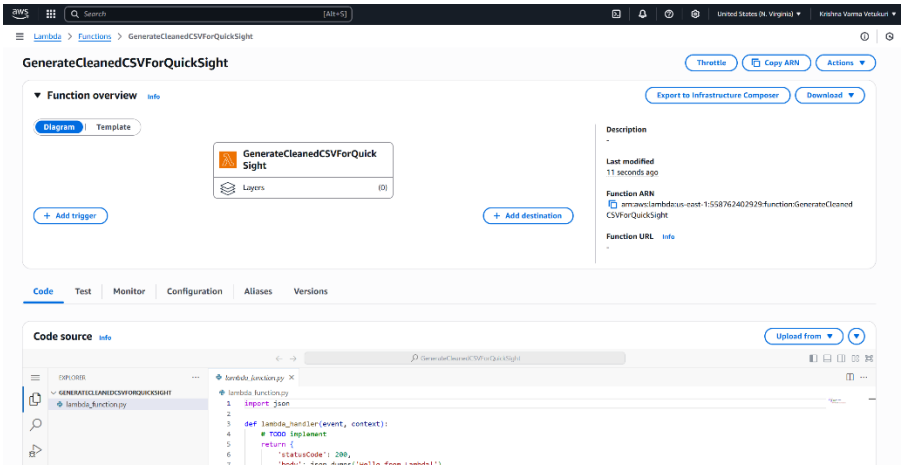
- Select General configuration → Click Edit
- Change timeout to 30 seconds



- Click Save

3.7.3 Writing and Deploying the Code

Navigate to the Code tab and paste the following code into lambda_function.py:

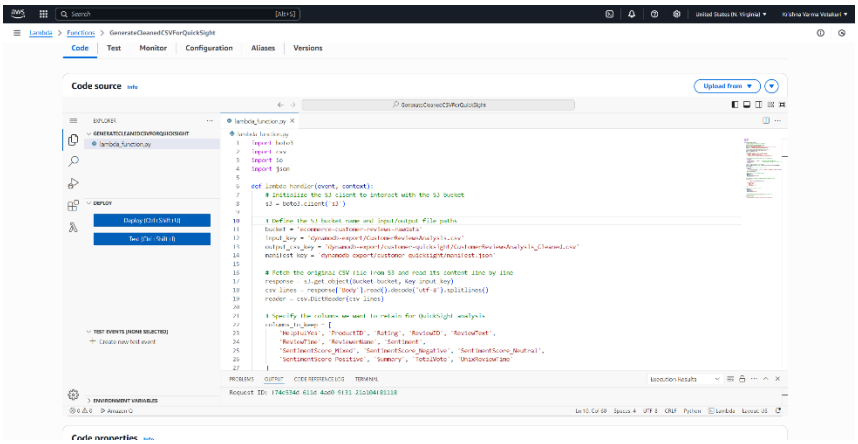


GenerateCleanedCSVForQuickSight lambda_function.py

```
import boto3
import csv
import io
import json

def lambda_handler(event, context):
    # Initialize the S3 client to interact with the S3 bucket
    s3 = boto3.client('s3')

    # Define the S3 bucket name and input/output file paths
    bucket = 'ecommerce-customer-reviews-rawdata'
    input_key = 'dynamodb-export/CustomerReviewsAnalysis.csv'
    output_csv_key = 'dynamodb-export/customer-quicksight/CustomerReviewsAnalysis_Cleaned.csv'
    manifest_key = 'dynamodb-export/customer-quicksight/manifest.json'
```

```
    # Fetch the original CSV file from S3 and read its content line by line
    response = s3.get_object(Bucket=bucket, Key=input_key)
    csv_lines = response['Body'].read().decode('utf-8').splitlines()
    reader = csv.DictReader(csv_lines)

    # Specify the columns we want to retain for QuickSight analysis
    columns_to_keep = [
        'HelpfulYes', 'ProductID', 'Rating', 'ReviewID', 'ReviewText',
        'ReviewTime', 'ReviewerName', 'Sentiment',
        'SentimentScore_Mixed', 'SentimentScore_Negative', 'SentimentScore_Neutral',
        'SentimentScore_Positive', 'Summary', 'TotalVote', 'UnixReviewTime'
    ]

    # Function to clean unwanted characters from text fields like quotes and commas
    def clean_text(text):
        if not text:
            return ""
        text = text.replace("", '').replace("'", '')  # remove both double and single quotes
        text = text.replace(", ", ")                  # remove comma followed by space
        text = text.replace(",", " ")                  # replace commas with space to avoid breaking CSV format
        return text

    # Prepare an in-memory CSV buffer to store cleaned data
    cleaned_csv_buffer = io.StringIO()
    writer = csv.DictWriter(cleaned_csv_buffer, fieldnames=columns_to_keep,
quoting=csv.QUOTE_NONNUMERIC)
    writer.writeheader()

    # Go through each row and clean only the necessary text fields
    for row in reader:
        cleaned_row = {
            col: clean_text(row.get(col, '')) if col in ['ReviewText', 'ReviewerName', 'Summary'] else row.get(col, '')
            for col in columns_to_keep
        }
        writer.writerow(cleaned_row)

    # Upload the cleaned CSV file to the specified S3 path
    s3.put_object(
        Bucket=bucket,
        Key=output_csv_key,
        Body=cleaned_csv_buffer.getvalue(),
        ContentType='text/csv'
    )

    # Build the manifest.json required for QuickSight to locate and interpret the CSV file
    manifest = {
        "fileLocations": [
            {
                "URIs": [f"s3://{bucket}/{output_csv_key}"]
            }
        ],
        "globalUploadSettings": {
            "format": "CSV",
            "delimiter": ",",
            "textqualifier": "'",
            "containsHeader": "true"
        }
    }

    # Upload the manifest file to the same folder in S3
```

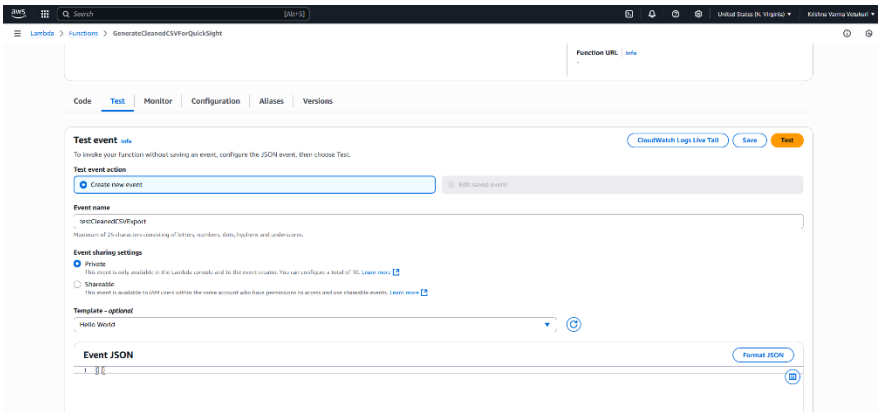
```
s3.put_object(
    Bucket=bucket,
    Key=manifest_key,
    Body=json.dumps(manifest),
    ContentType='application/json'
)

# Final message to indicate success when testing locally or via CloudWatch logs
print("Cleaned CSV and manifest.json successfully uploaded.")

# Return success response
return {
    'statusCode': 200,
    'body': json.dumps('CSV and manifest successfully created for QuickSight')
}
```



Click Deploy and wait for the success message.

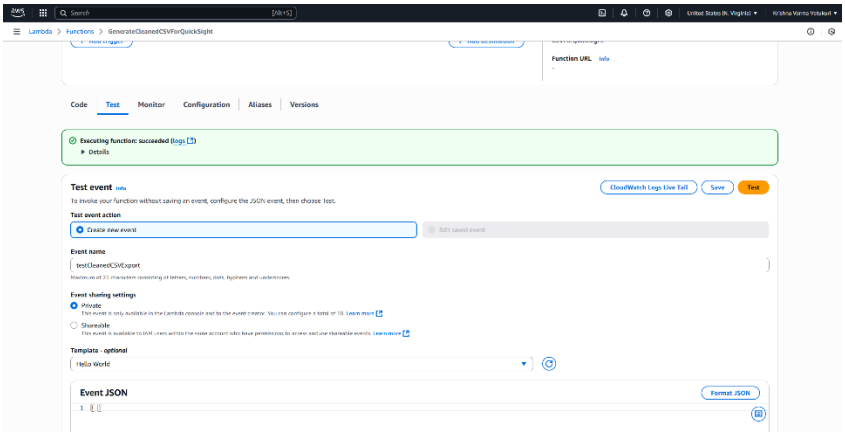
3.7.4 Testing the Function

- Go to the Test tab in Lambda
- Create a new test:
    - Name: testCleanedCSVExport
    - Event JSON: { }



- Click Test

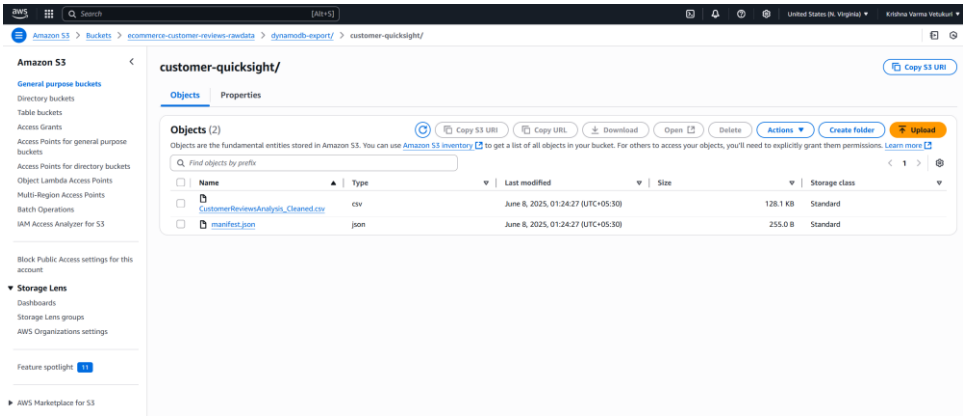
After a few seconds, the execution should show a success response.



3.7.5 Verifying in S3

Now go to the S3 bucket:

- Navigate to: ecommerce-customer-reviews-rawdata/dynamodb-export/customer-quicksight/



- We should see:
  - CustomerReviewsAnalysis_Cleaned.csv
  - manifest.json

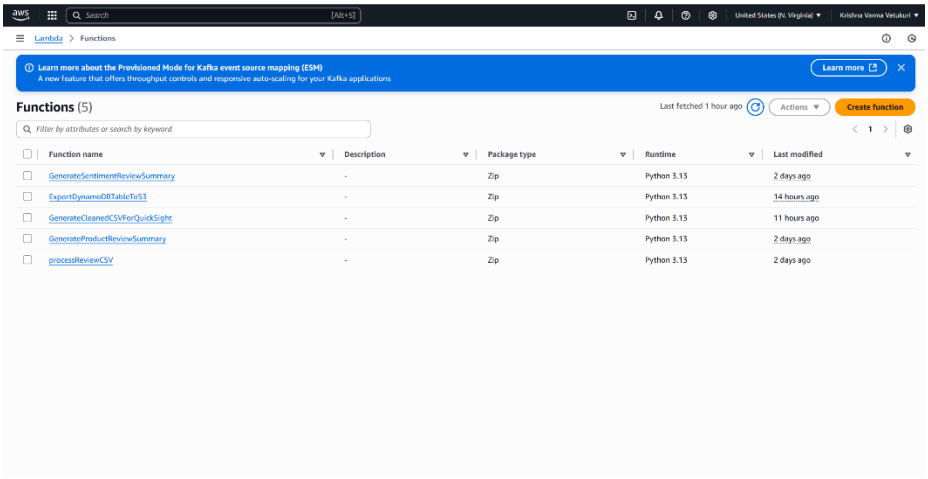
3.8. Generating Key Phrases CSV for Detailed Analysis

This step extracts individual key phrases from the customer reviews for more granular sentiment analysis in Amazon QuickSight. We achieve this by building a new Lambda function that flattens the KeyPhrases field from the original CSV into individual rows per phrase.

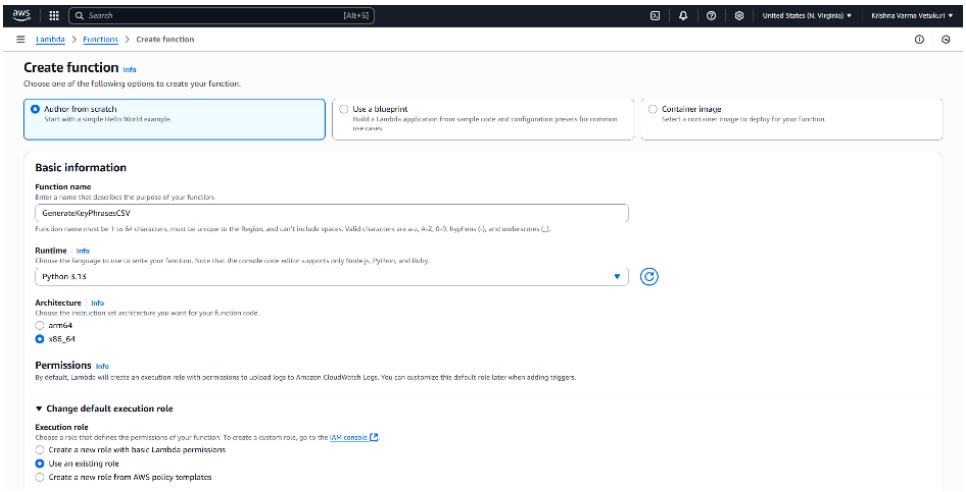
3.8.1 Creating the Lambda Function: GenerateKeyPhrasesCSV

To begin, we create a Lambda function that processes the cleaned CustomerReviewsAnalysis.csv and produces a normalized key phrases dataset.

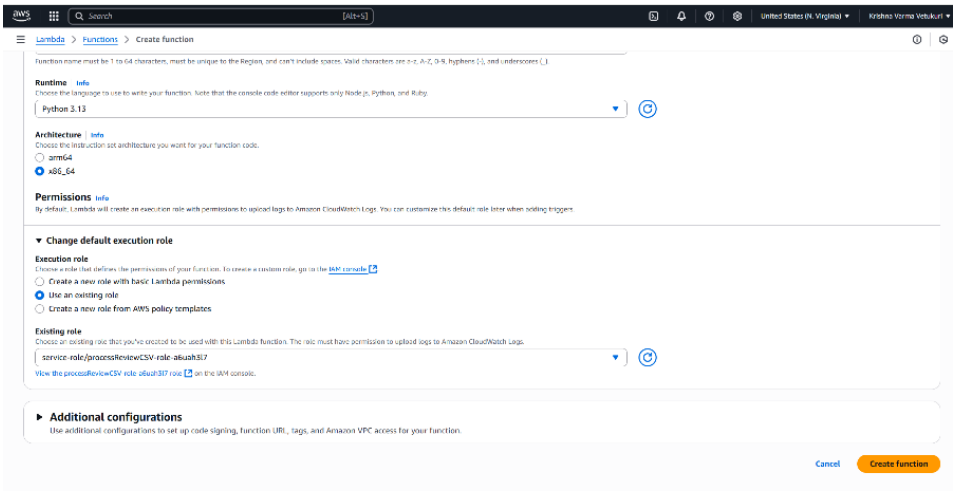
- Go to AWS Lambda and click Create Function



- Enter:
  - Function name: GenerateKeyPhrasesCSV
  - Runtime: Python 3.13



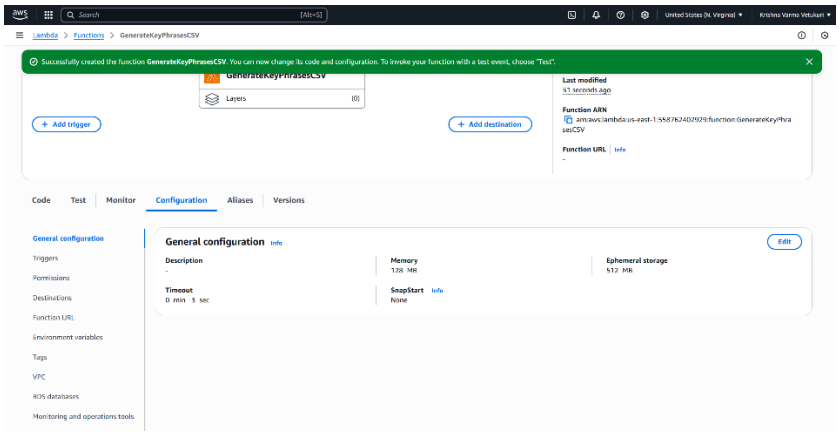
- For permissions:
  - Select Use an existing role
  - Choose processReviewCSV-role-a6uah3l7



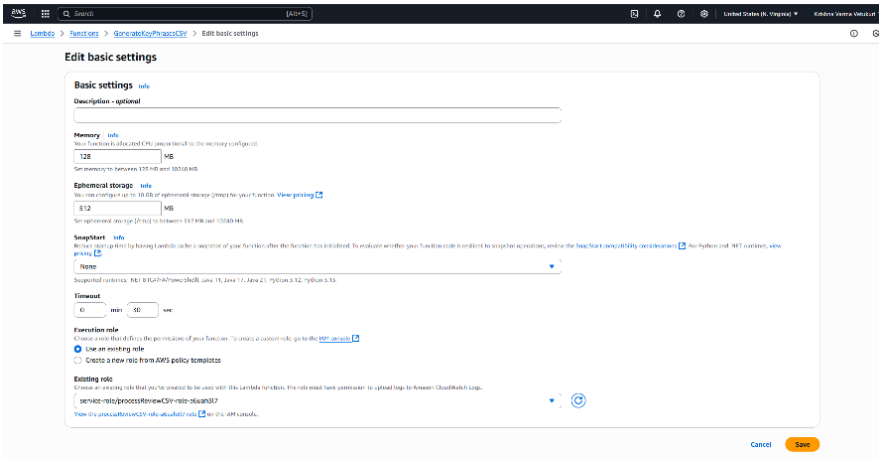
- Click Create Function

3.8.2 Setting Timeout to 30 Seconds

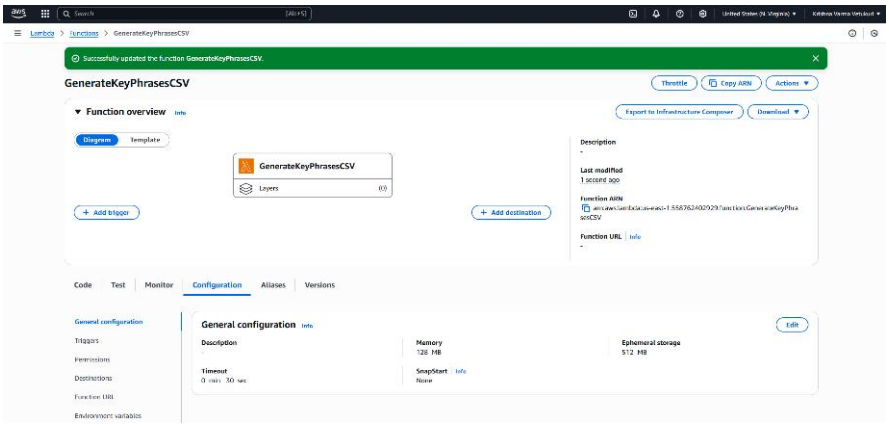
- Go to the Configuration tab



- Select General configuration and click Edit
- Update the timeout value from 3 seconds to 30 seconds

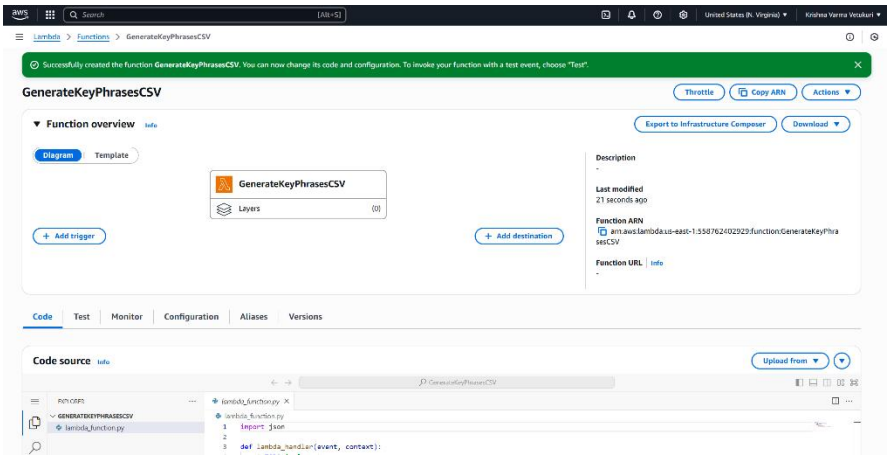


- Click Save



3.8.3 Adding the Lambda Code

Now go to the Code tab and replace the contents with the following code:

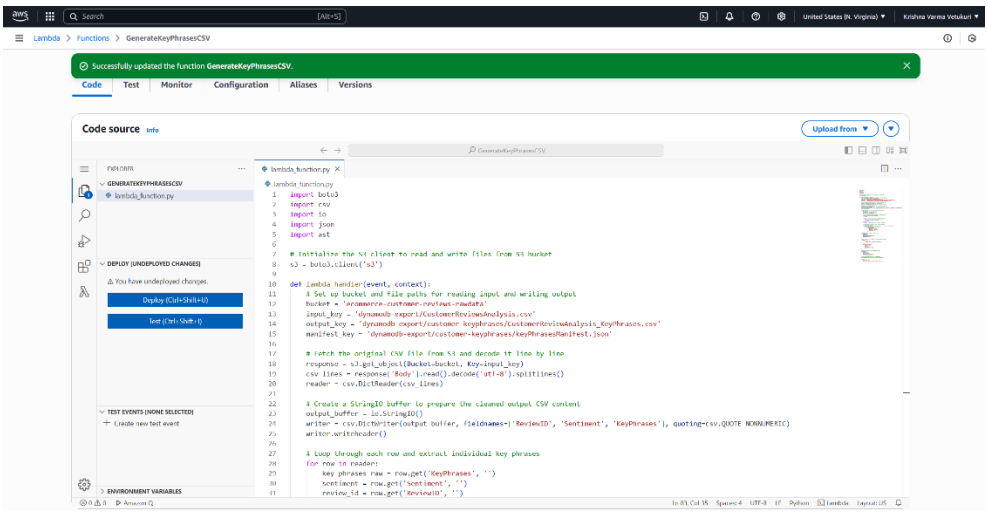


GenerateKeyPhrasesCSV lambda_function.py

```
import boto3
import csv
import io
import json
import ast

# Initialize the S3 client to read and write files from S3 bucket
s3 = boto3.client('s3')

def lambda_handler(event, context):
    # Set up bucket and file paths for reading input and writing output
    bucket = 'ecommerce-customer-reviews-rawdata'
    input_key = 'dynamodb-export/CustomerReviewsAnalysis.csv'
    output_key = 'dynamodb-export/customer-keyphrases/CustomerReviewAnalysis_KeyPhrases.csv'
    manifest_key = 'dynamodb-export/customer-keyphrases/keyPhrasesManifest.json'

    # Fetch the original CSV file from S3 and decode it line by line
    response = s3.get_object(Bucket=bucket, Key=input_key)
```

```
    csv_lines = response['Body'].read().decode('utf-8').splitlines()
    reader = csv.DictReader(csv_lines)

    # Create a StringIO buffer to prepare the cleaned output CSV content
    output_buffer = io.StringIO()
    writer = csv.DictWriter(output_buffer, fieldnames=['ReviewID', 'Sentiment', 'KeyPhrases'],
quoting=csv.QUOTE_NONNUMERIC)
    writer.writeheader()

    # Loop through each row and extract individual key phrases
    for row in reader:
        key_phrases_raw = row.get('KeyPhrases', '')
        sentiment = row.get('Sentiment', '')
        review_id = row.get('ReviewID', '')

        # Skip rows that are missing KeyPhrases or Sentiment values
        if not key_phrases_raw or not sentiment:
            continue

        # Attempt to convert the KeyPhrases string into a proper list
        try:
            parsed_list = ast.literal_eval(key_phrases_raw)
            if not isinstance(parsed_list, list):
                continue
        except:
            continue  # Ignore parsing errors and move to next row

        # Process each phrase and split it if it contains commas
        for phrase in parsed_list:
            split_phrases = [p.strip().lower().replace("'", "") for p in phrase.split(',')]
            for p in split_phrases:
                if p:  # Avoid writing empty strings
                    writer.writerow({
                        'ReviewID': review_id,
                        'Sentiment': sentiment,
                        'KeyPhrases': p
                    })

    # Save the cleaned key phrases CSV back to S3
    s3.put_object(
        Bucket=bucket,
        Key=output_key,
        Body=output_buffer.getvalue(),
        ContentType='text/csv'
    )

    # Build the manifest file required for loading data in QuickSight
    manifest = {
        "fileLocations": [
            {
                "URIs": [f"s3://{bucket}/{output_key}"]
            }
        ],
        "globalUploadSettings": {
            "format": "CSV",
            "delimiter": ",",
            "textqualifier": "'",
            "containsHeader": "true"
        }
    }
```

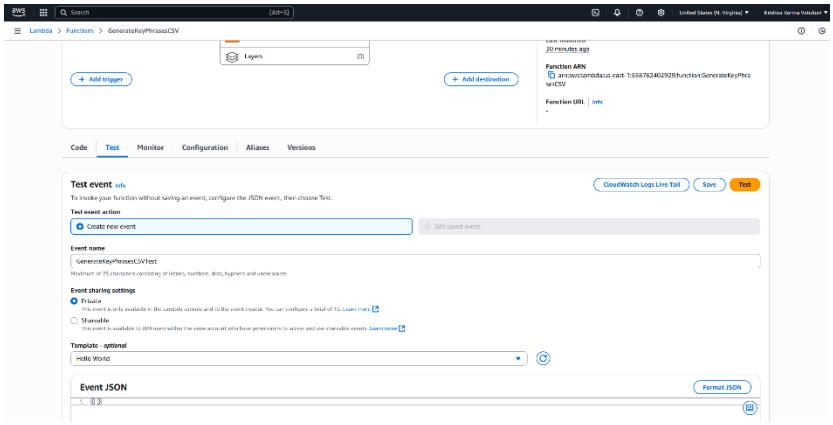
```
# Upload the manifest.json to the same S3 folder
s3.put_object(
    Bucket=bucket,
    Key=manifest_key,
    Body=json.dumps(manifest),
    ContentType='application/json'
)

# Final confirmation message for logs or debugging
print("KeyPhrases CSV and manifest uploaded successfully.")

return {
    'statusCode': 200,
    'body': json.dumps('KeyPhrases extraction complete.')
}
```



Click Deploy to update the function with the new code.

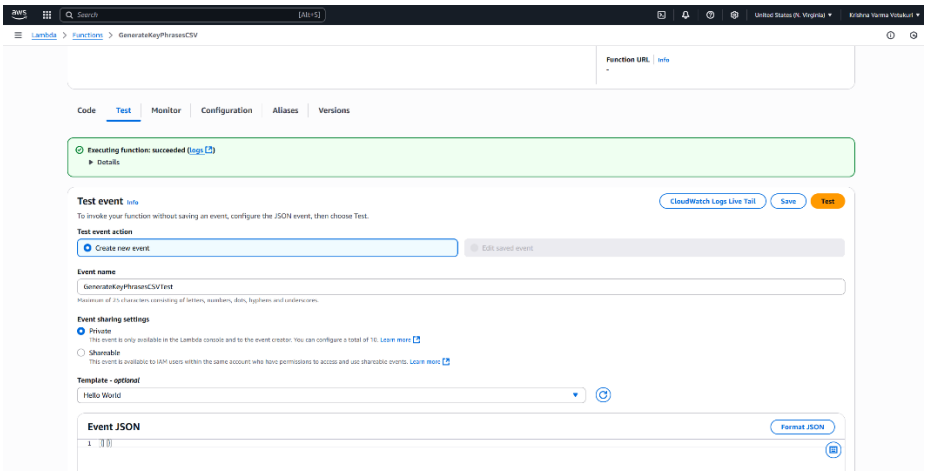
3.8.4 Testing the Lambda Function

- Go to the Test tab
- Create a test:
    - Event Name: GenerateKeyPhrasesCSVTest
    - Event JSON: { }



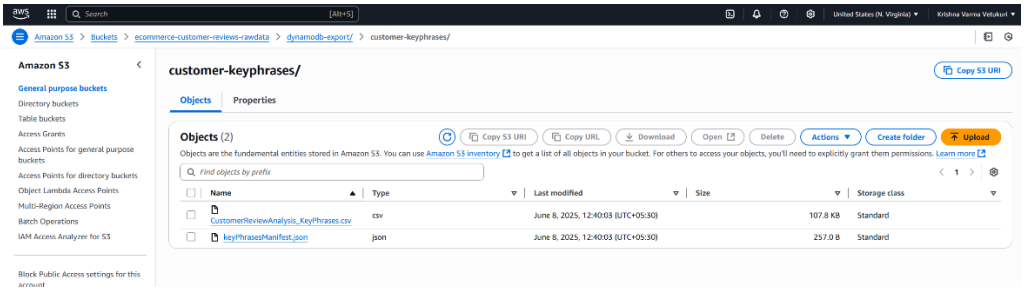
- Click Test

If the process is successful, we'll see a succeeded message in the console output.



3.8.5 Verifying Output in S3

- Navigate to the S3 bucket:
  - Bucket: ecommerce-customer-reviews-rawdata
  - Path: dynamodb-export/customer-keyphrases/



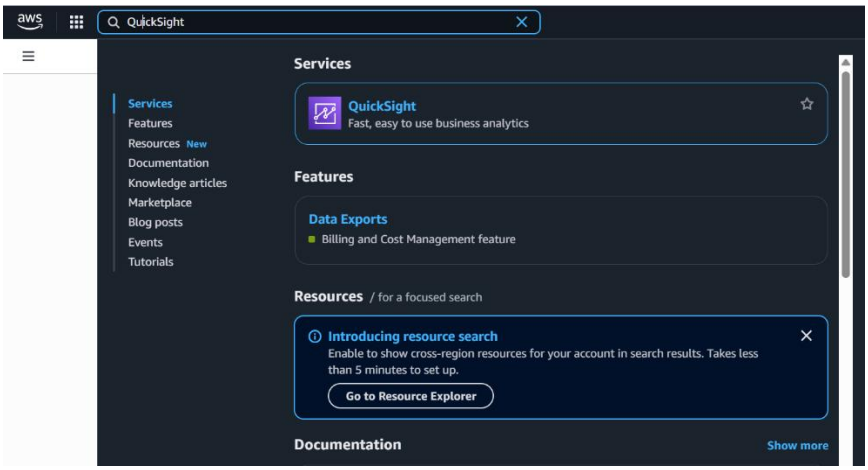
- We should see:
  - CustomerReviewAnalysis_KeyPhrases.csv
  - keyPhrasesManifest.json

3.9 QuickSight Dataset and Visualization Setup

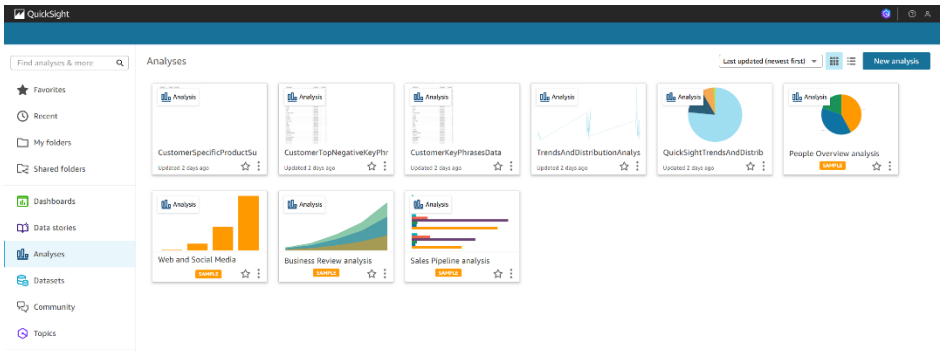
In this step, we configure Amazon QuickSight to visualize customer review trends using the cleaned and transformed dataset stored in S3. The data source is defined using a manifest file generated by the GenerateCleanedCSVForQuickSight Lambda function.

3.9.1 Accessing QuickSight and Adding a Dataset

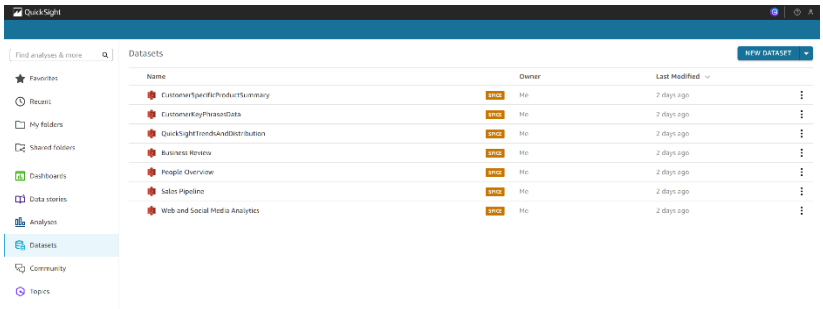
- Open Amazon QuickSight from the AWS Console search bar.



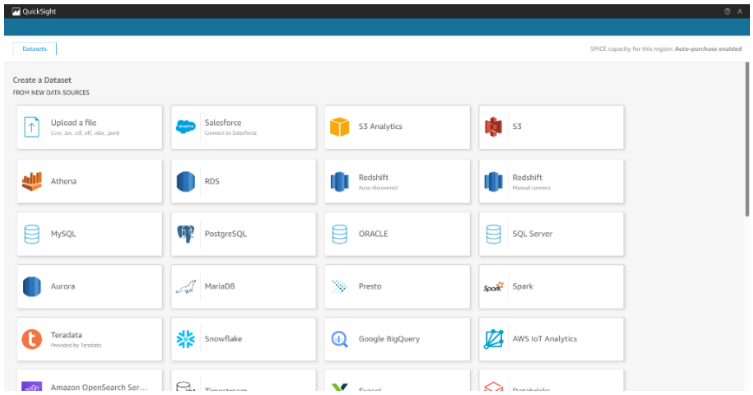
- We'll land on the Analysis homepage, which includes sample analyses.



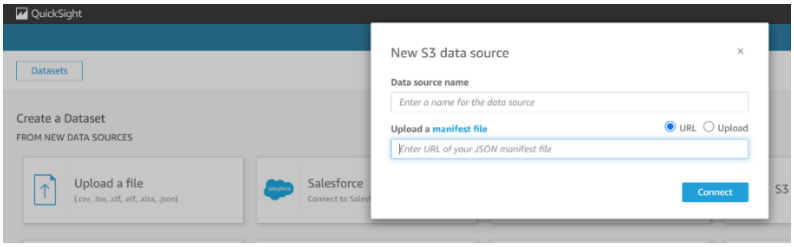
- On the left sidebar, click Datasets.



- Click the New dataset button at the top right corner.



- Choose S3 as the data source type.



It will ask for:

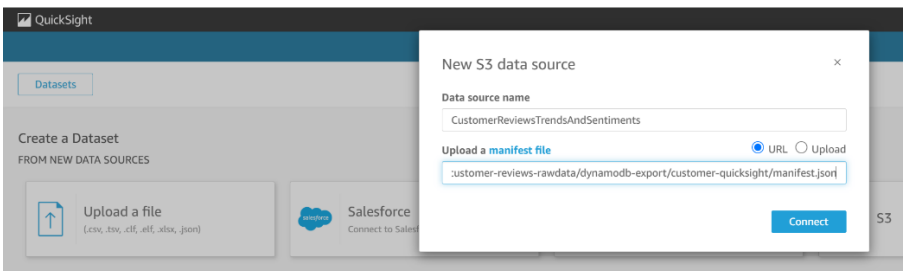
- Data source name — Enter: CustomerReviewsTrendsAndSentiments
- Upload a manifest file — Here, we provide the manifest.json from the cleaned CSV.

3.9.2 Copying the Manifest File URI from S3

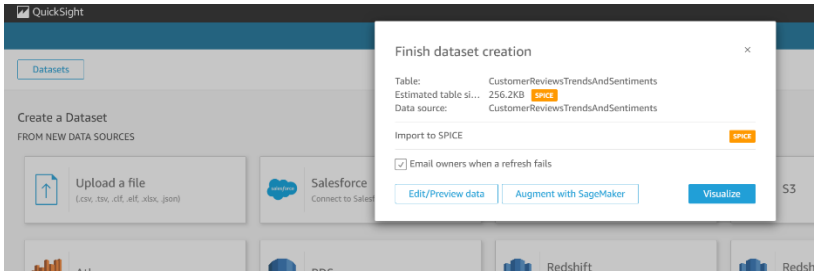
- Go to the S3 bucket: ecommerce-customer-reviews-rawdata
- Navigate to: dynamodb-export/customer-quicksight/
- Select the manifest.json file.
- Click Copy S3 URI

3.9.3 Connecting the Dataset in QuickSight

- Paste the copied S3 URI in the manifest file input.
    - Example URI: s3://ecommerce-customer-reviews-rawdata/dynamodb-export/customer-quicksight/manifest.json

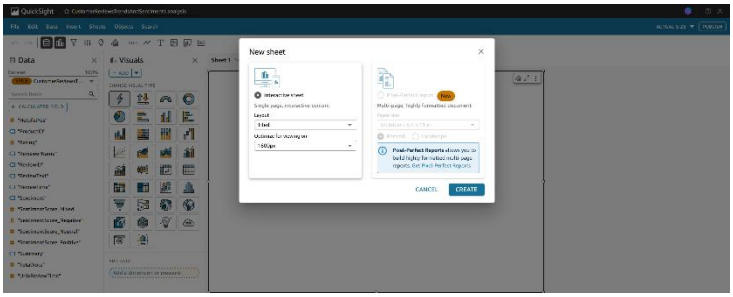


- Click Connect
  Once connected, QuickSight will finish loading the dataset.



- Click Visualize

3.9.4 Creating an Interactive Sheet

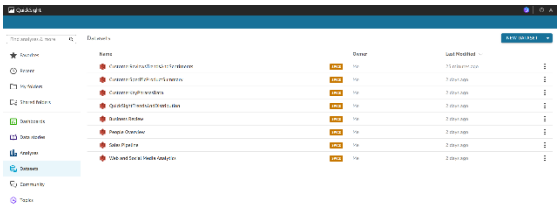


- When prompted:
  - Select Interactive Sheet
  - Click Create
- This opens the new analysis workspace with the dataset loaded.

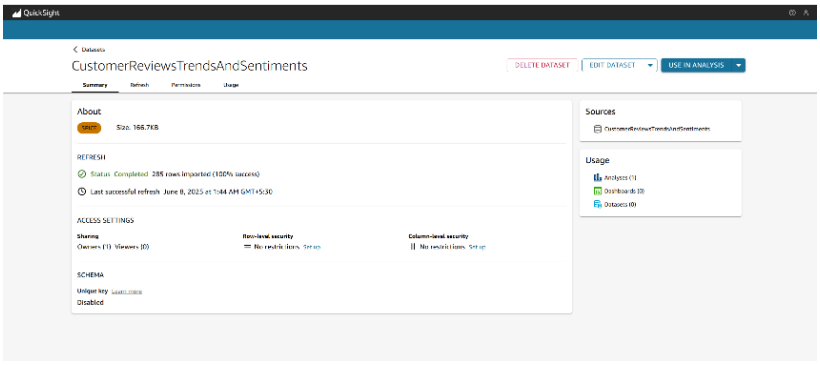
3.9.5. Fixing the ReviewTime Column Type

By default, QuickSight may recognize the ReviewTime column as a string. To analyze trends over time, we must convert it to a date type.

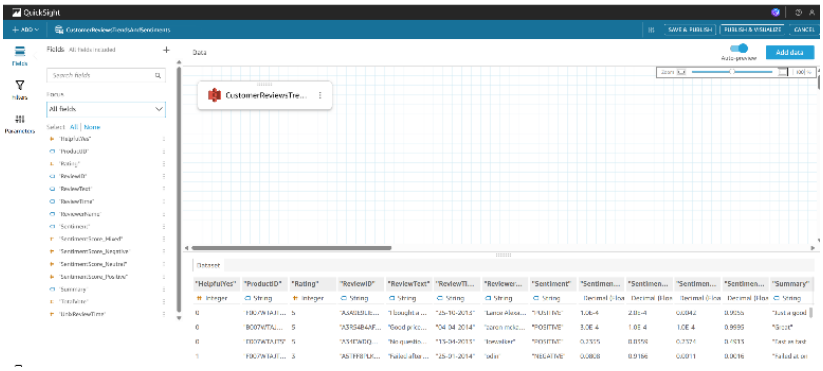
- Go back to the Datasets section in QuickSight.



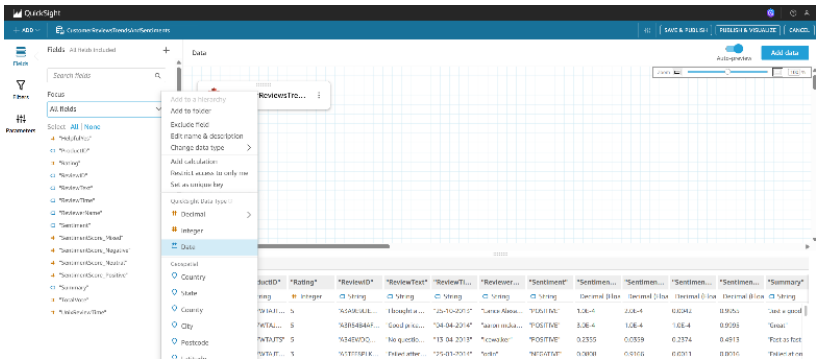
- Click CustomerReviewsTrendsAndSentiments



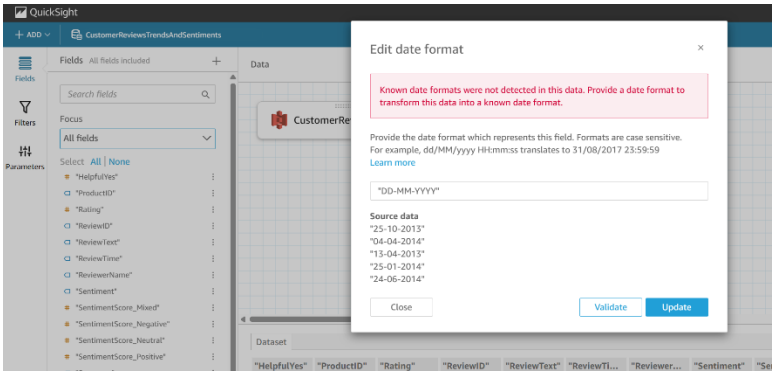
- Click Edit Dataset and wait for the editor to load.



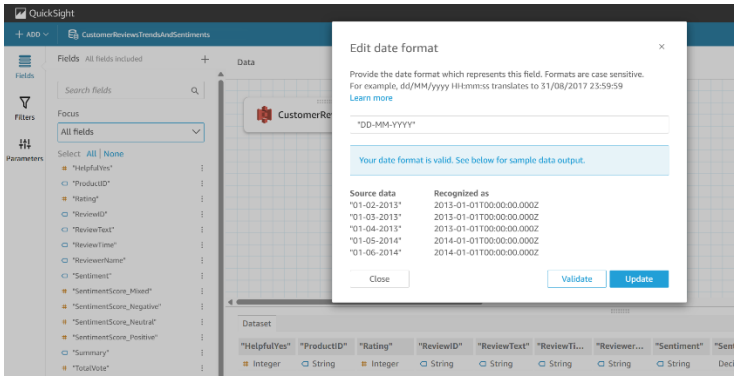
- In the left column list, find ReviewTime
- Click the three dots next to it → select Change data type → choose Date



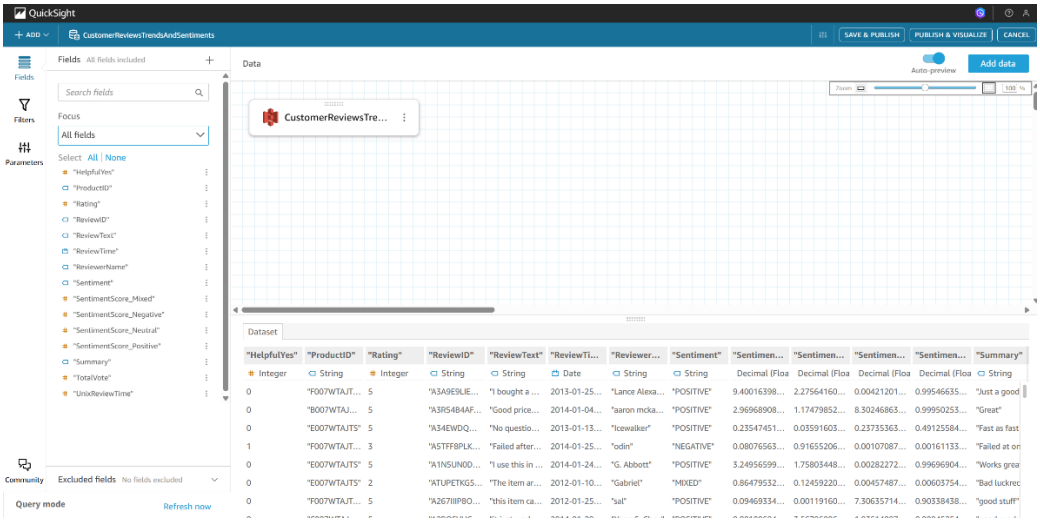
- In the format dialog, enter "DD-MM-YYYY" (since our data uses this format)



- Click Validate – it should show a success message



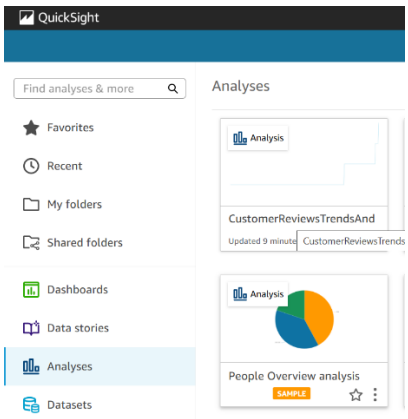
- Click Update



Now, we'll notice a calendar icon next to ReviewTime, confirming the type conversion.

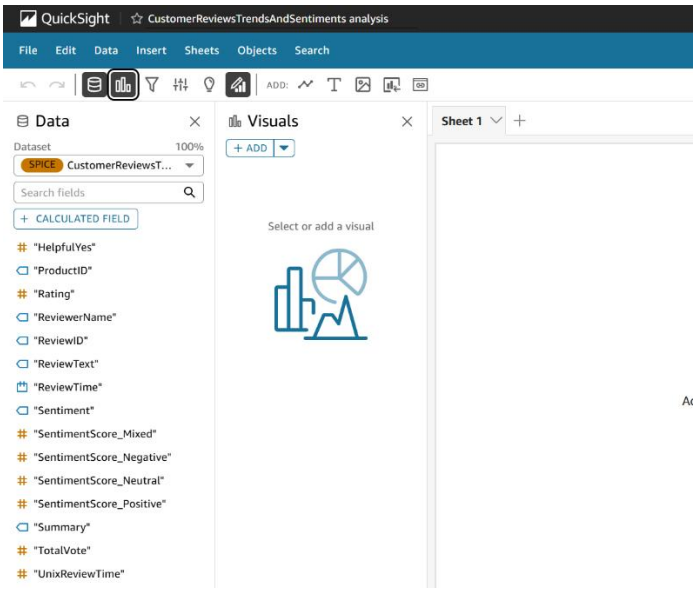
- Click Save and Publish

3.9.6. Returning to the Analysis Tab



Now return to the CustomerReviewsTrendsAndSentiments analysis.

- We'll see that ReviewTime now shows a **date icon**
- It can now be used for timeline visualizations and trend analysis in charts

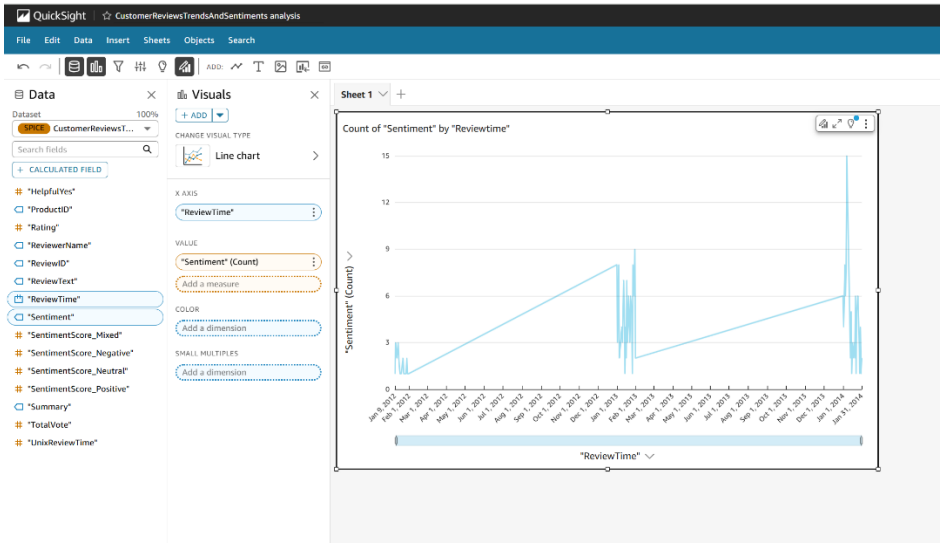


3.10. Creating Visuals in QuickSight Analysis

Now that the dataset is ready, we begin building visuals inside the CustomerReviewsTrendsAndSentiments analysis to gain insights.

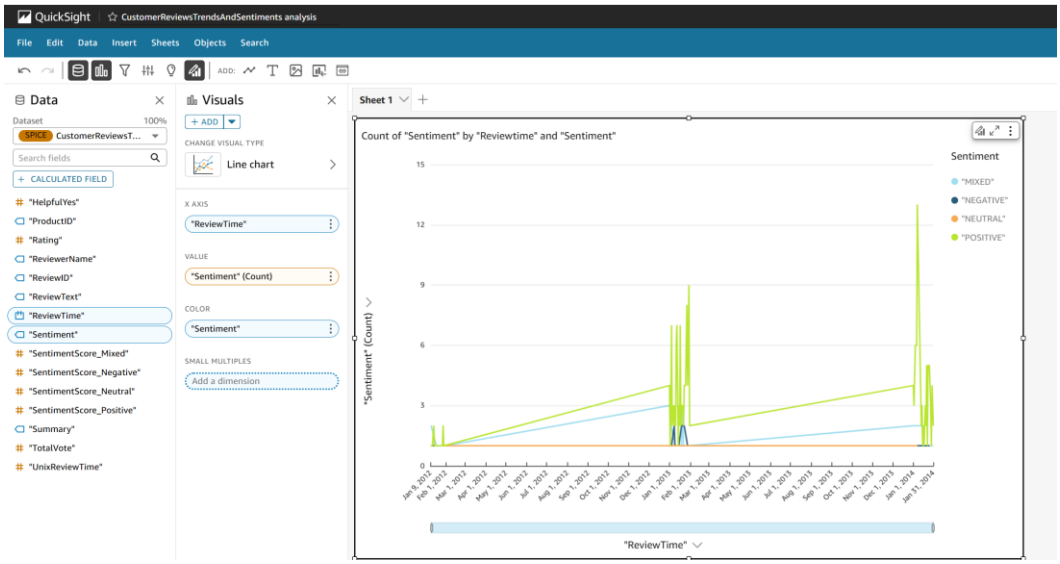
3.10.1.1 Overall Sentiment Trends Over Time

- Select Line Chart from the Visual types panel.
- Set X-axis as ReviewTime.
- Set Value as Sentiment → use Count aggregation.



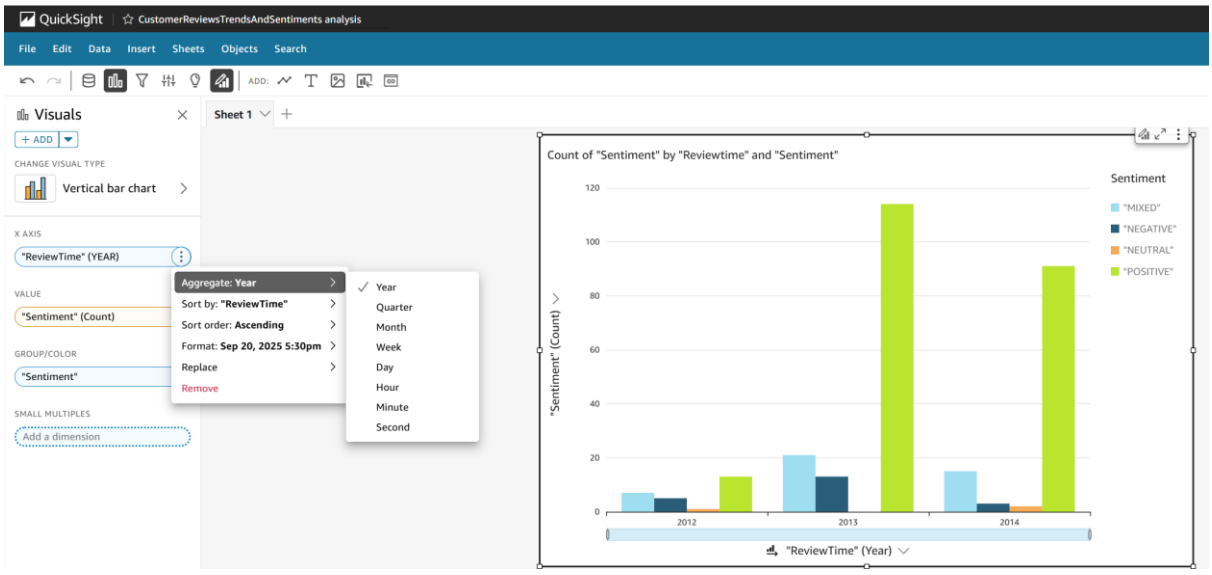
3.10.1.2 Overall Sentiment Trends Over Time based on Sentiments

- Select Line Chart from the Visual types panel.
- Set X-axis as ReviewTime.
- Set Value as Sentiment → use Count aggregation.
- Set Group/Color: Sentiment



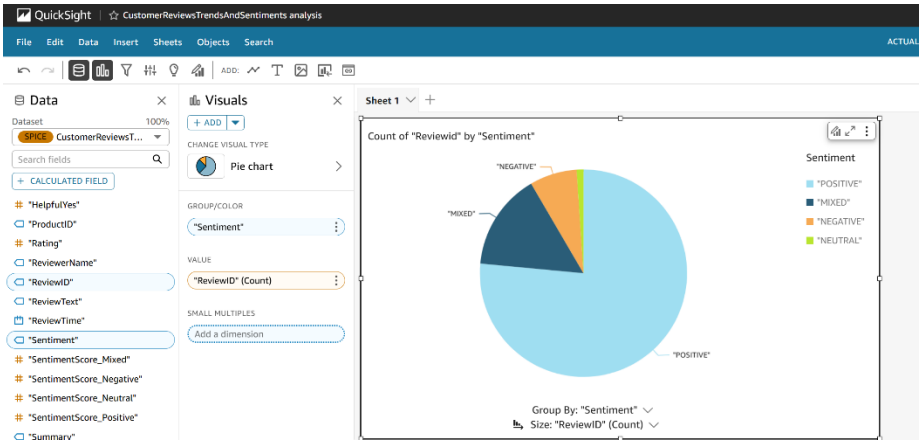
3.10.1.3 Overall Sentiment Trends Over Time based on Year

- Select Vertical Bar Chart from the Visual types panel.
- Set X-axis as ReviewTime. (Aggregate as Year)
- Set Value as Sentiment → use Count aggregation.
- Set Group/Color: Sentiment



3.10.2. Distribution of Sentiments (Positive, Negative, Neutral)

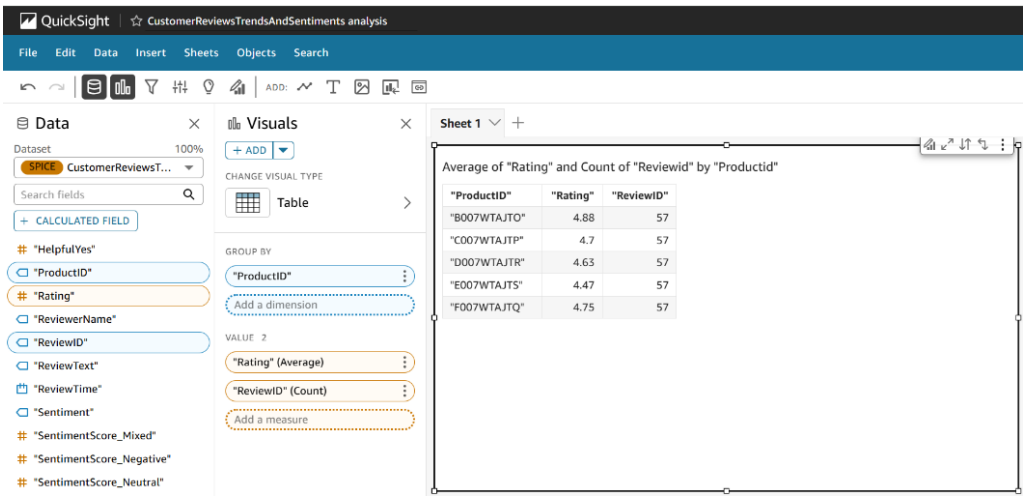
- Select Pie Chart as visual.
- For Group/Color, choose Sentiment.
- For Value, use Sentiment → Count.



3.11. Product Feedback Summaries

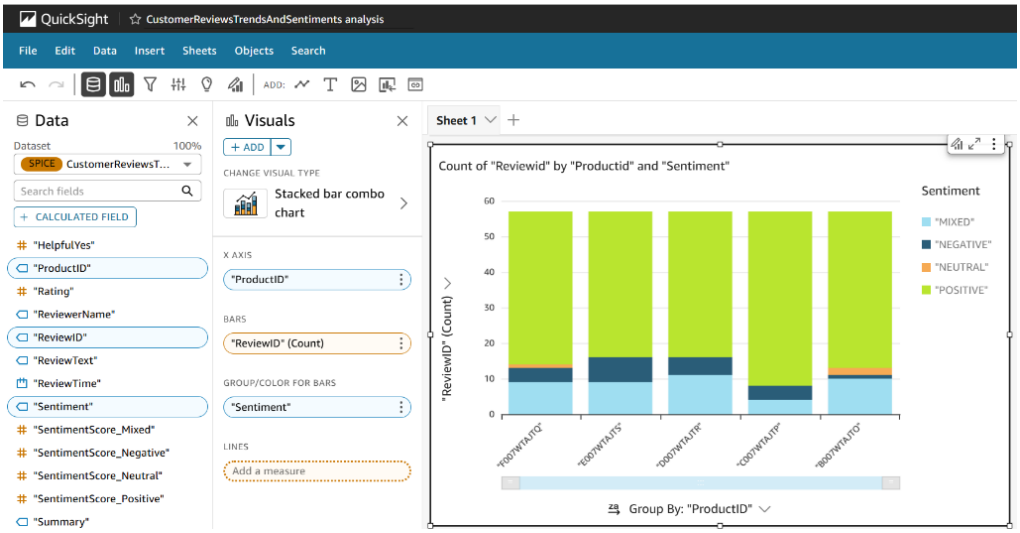
3.11.1. Overall Feedback Summary per Product

- Select Table.
- Set Group by → ProductID
- Add Values:
  - Rating → Average
  - ReviewID → Count



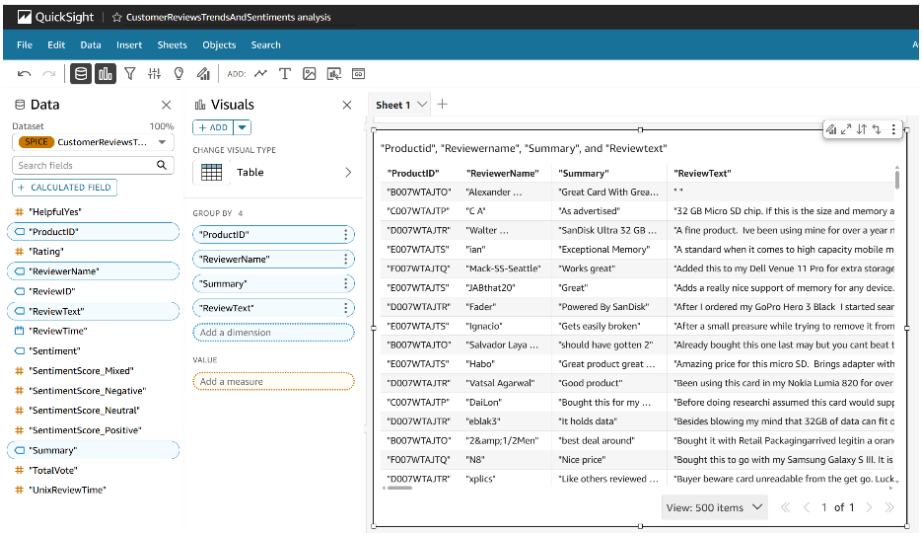
3.11.2. Sentiment Distribution per Product

- Select Stacked Bar Chart.
- Set X-axis → ProductID
- Set Bars (Value) → ReviewID → Count
- Set Group/Color → Sentiment



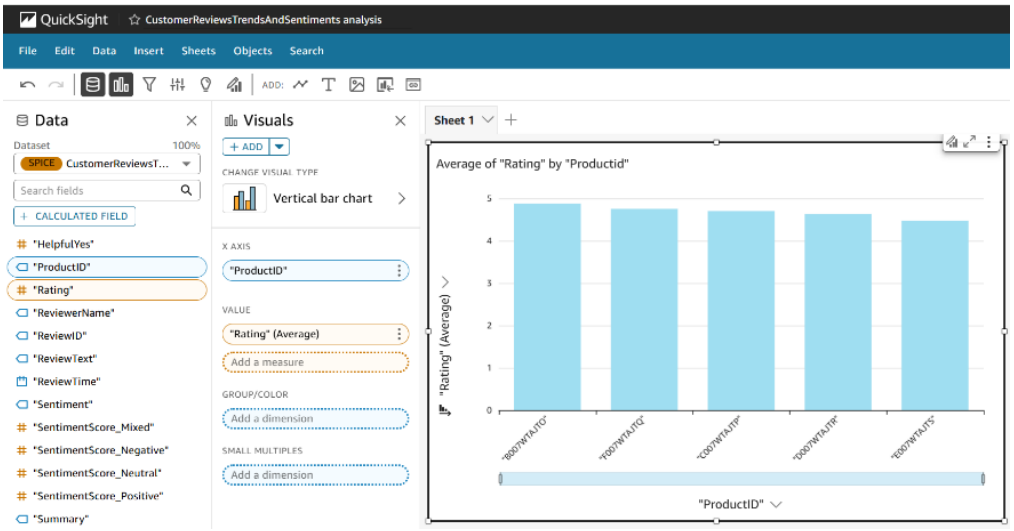
3.11.3. Sample Feedback Table

- Select Table
- Set Group by fields:
  - ProductID
  - ReviewerName
  - Summary
  - ReviewText



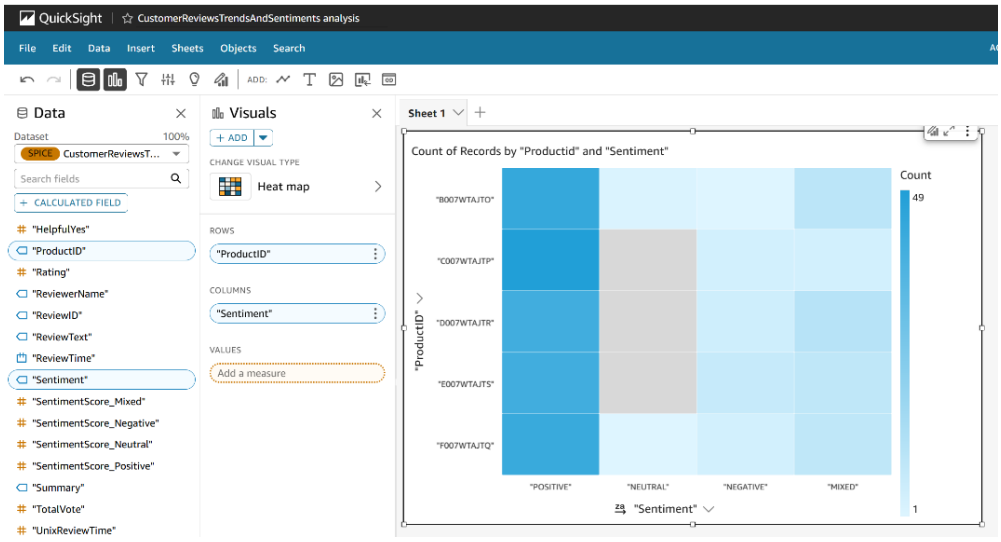
3.11.4. Average Rating per Product

- Select Vertical Bar Chart
- Set X-axis → ProductID
- Set Value → Rating → Average



3.11.5. Review Count per Product (Heatmap)

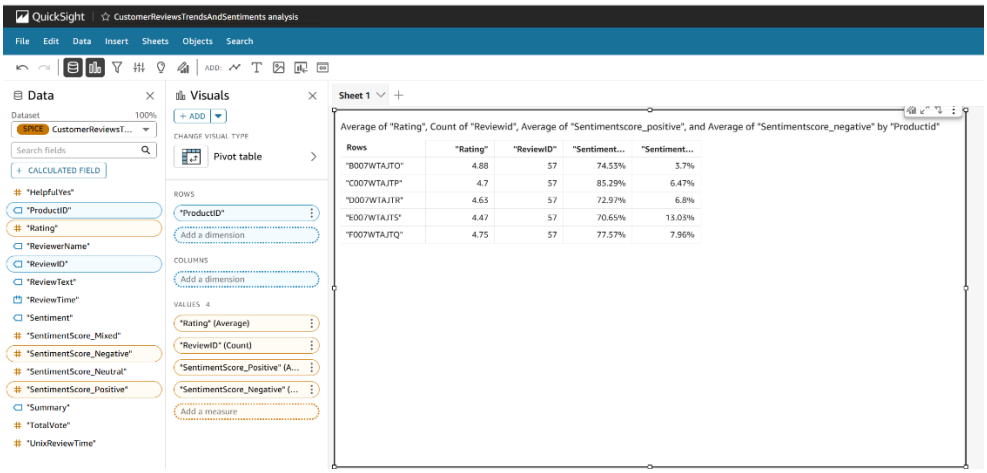
- Select Heatmap
- Set Rows → ProductID
- Set Columns → Sentiment



3.11.6. Product Feedback Summary (Pivot Table)

- Select Pivot Table
- Set Rows → ProductID

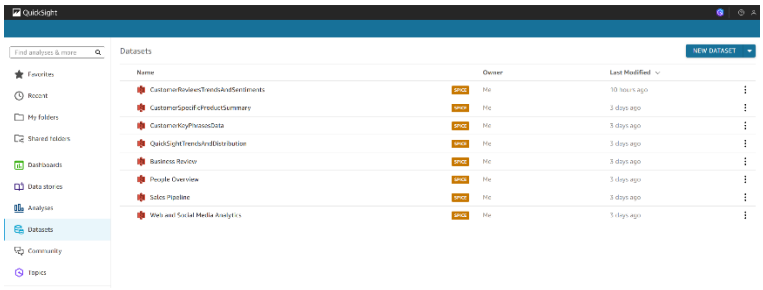
- Columns can be left empty
- Add Values:
    - Rating → Average
    - ReviewID → Count
    - SentimentScore_Positive → Average (show as percentage)
    - SentimentScore_Negative → Average (show as percentage)



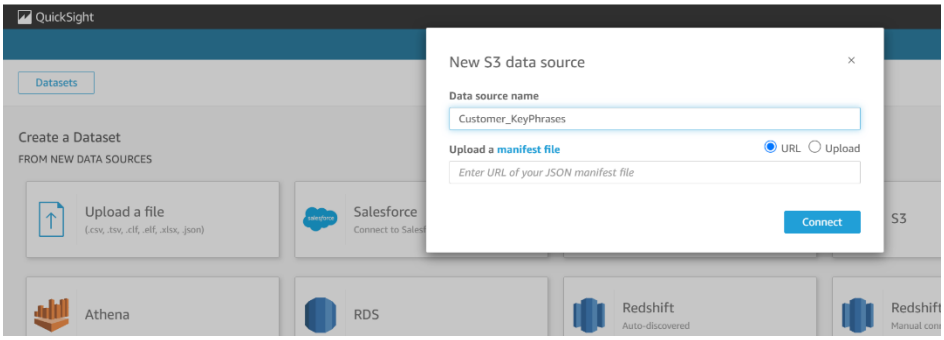
3.12 Create Dataset for Key Phrases Analysis

To analyze key phrases extracted earlier:

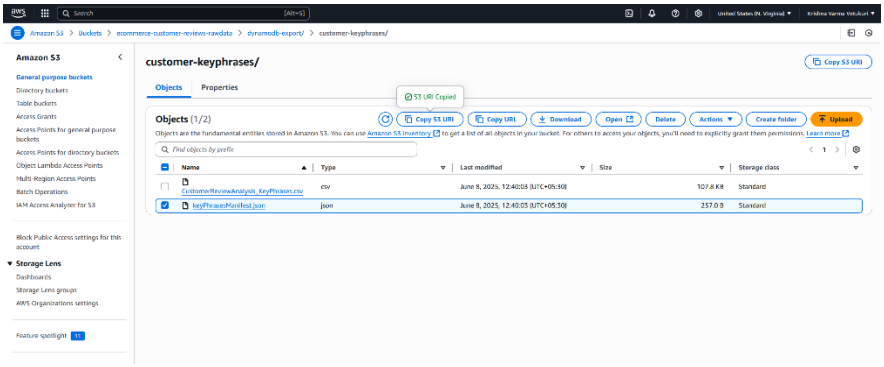
- Go to QuickSight Home → Datasets



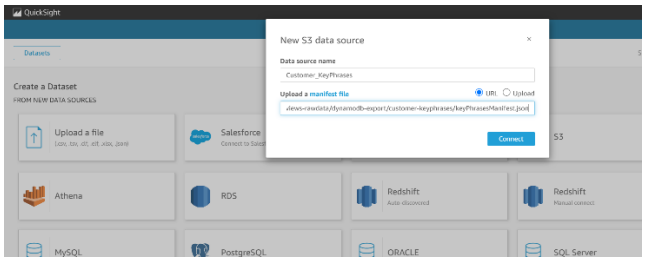
- Click New Dataset
- Select S3 as the data source.



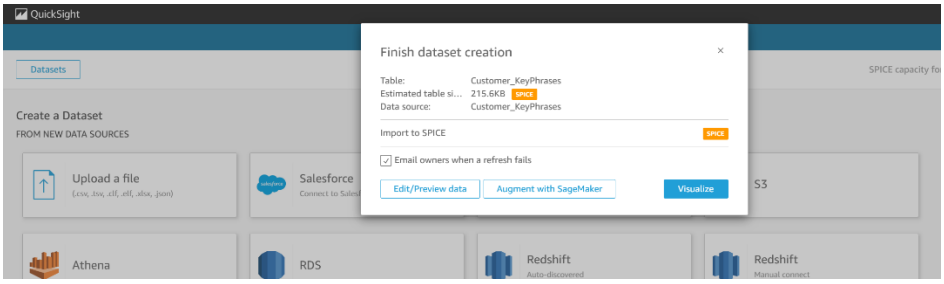
- Open S3 → navigate to:
  ecommerce-customer-reviews-rawdata/dynamodb-export/customer-keyphrases/



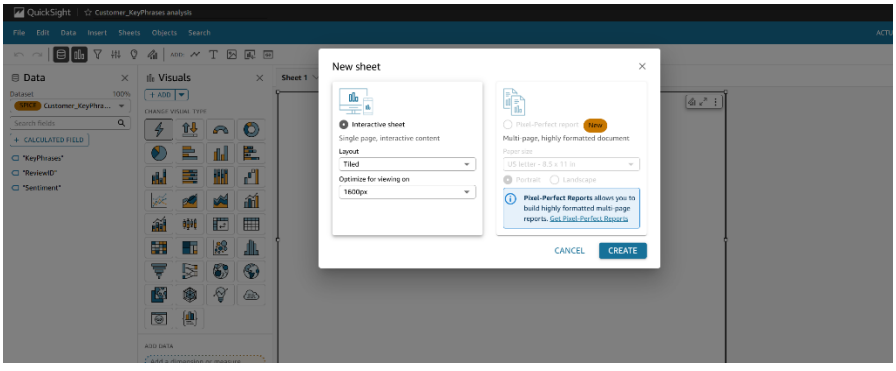
- Select keyPhrasesManifest.json and click Copy S3 URI
- In QuickSight, paste that URI in the Upload a manifest file box.



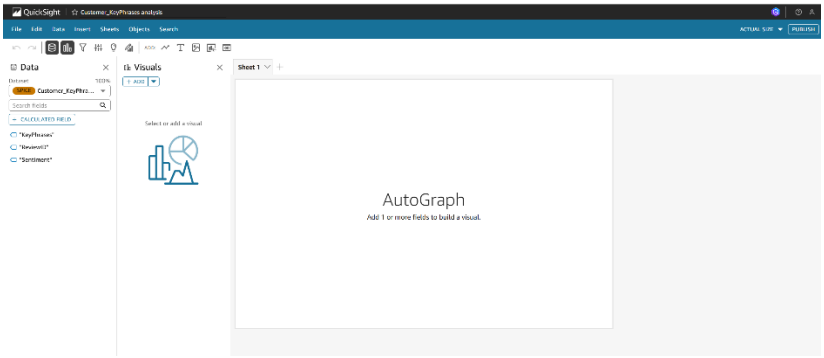
- Click Connect



- Once dataset creation is successful, click Visualize



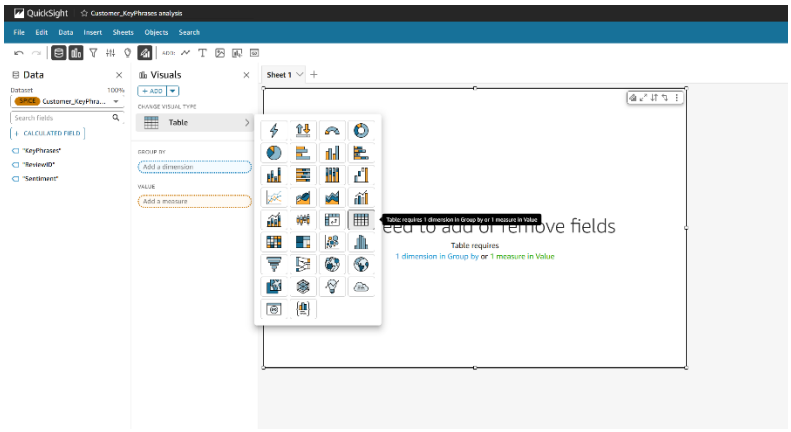
- When prompted, choose Interactive Sheet → Click Create



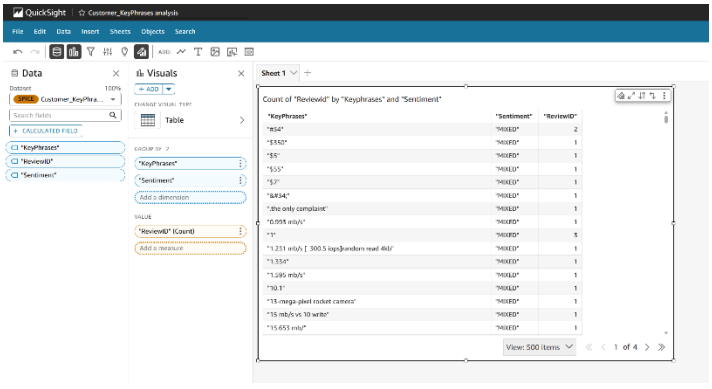
3.13 Visualize Top Negative Key Phrases

To identify the most frequent negative phrases from reviews:

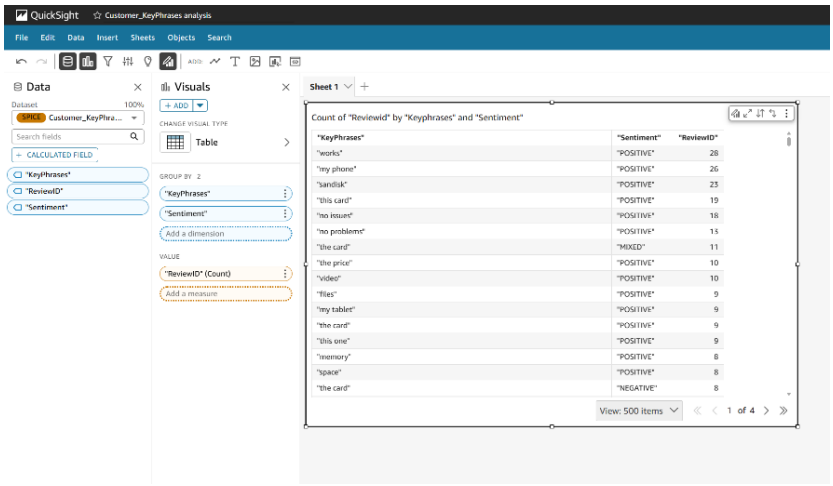
- In the Key Phrases dataset analysis, choose Table from visual types.



- Set Group by:
  - KeyPhrases
  - Sentiment
- Set Value → ReviewID → Count

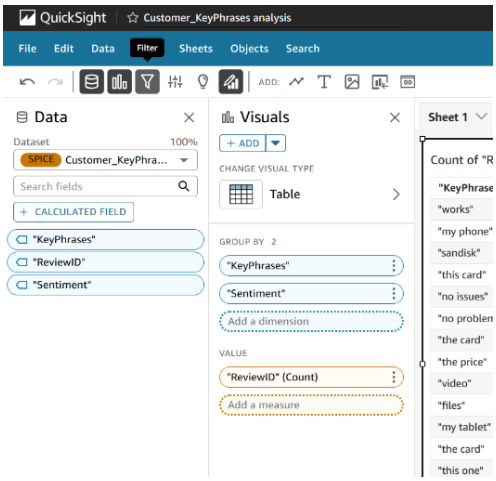


- Click the three dots on ReviewID column → choose Set Order → Descending

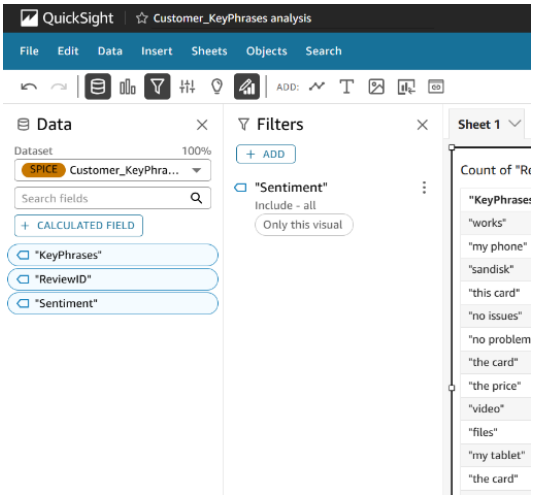


Adding Filter for Negative Sentiment

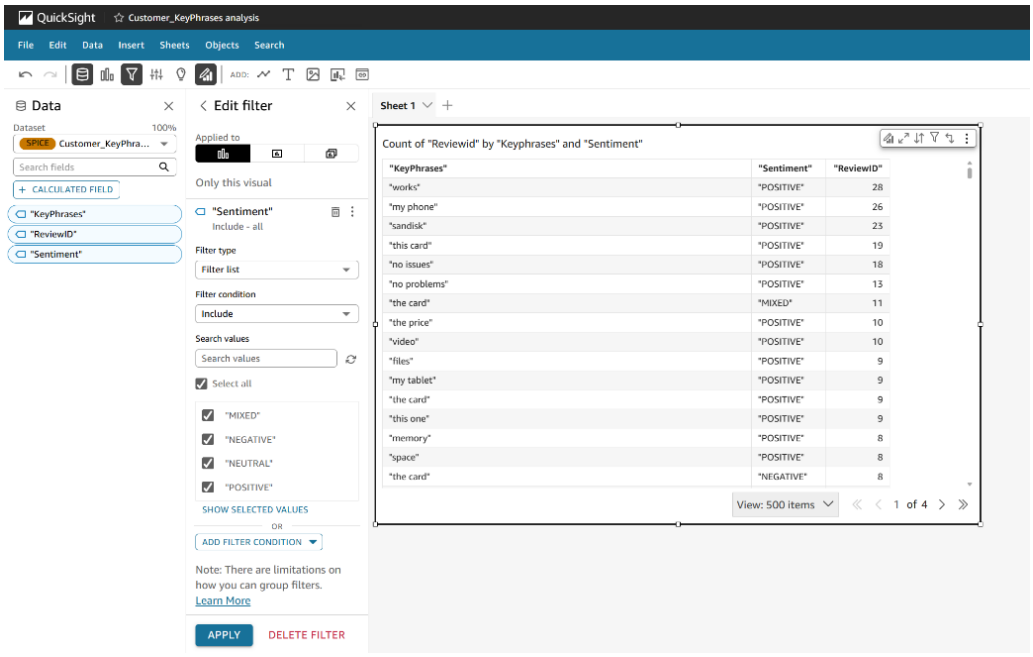
- Go to the Filters pane next to the visual.



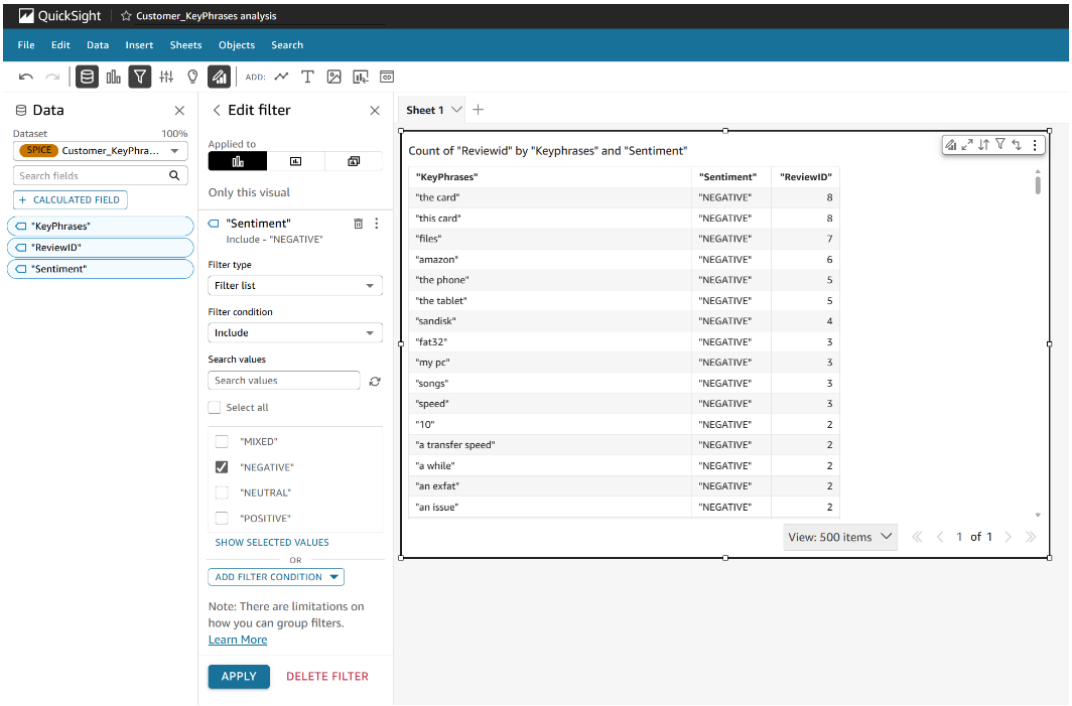
- Click Add → Select Sentiment



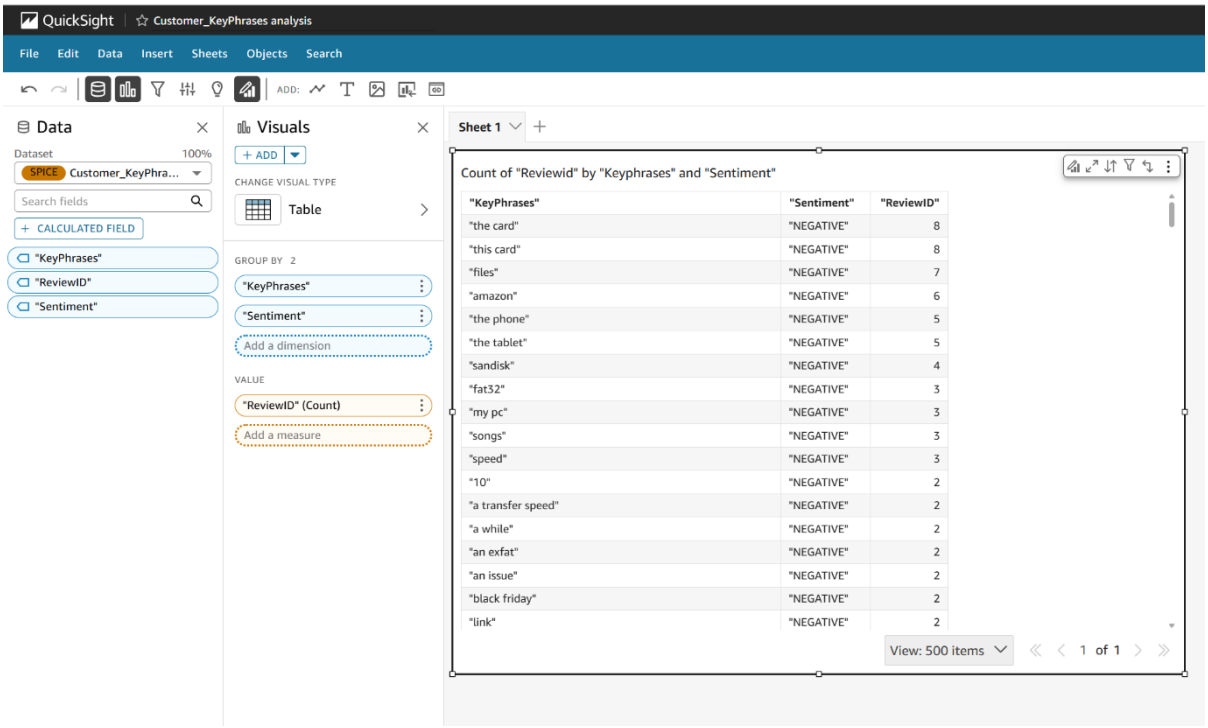
- Select the Filter



- In filter settings:
  - Deselect all
  - Select only Negative



- Click Apply



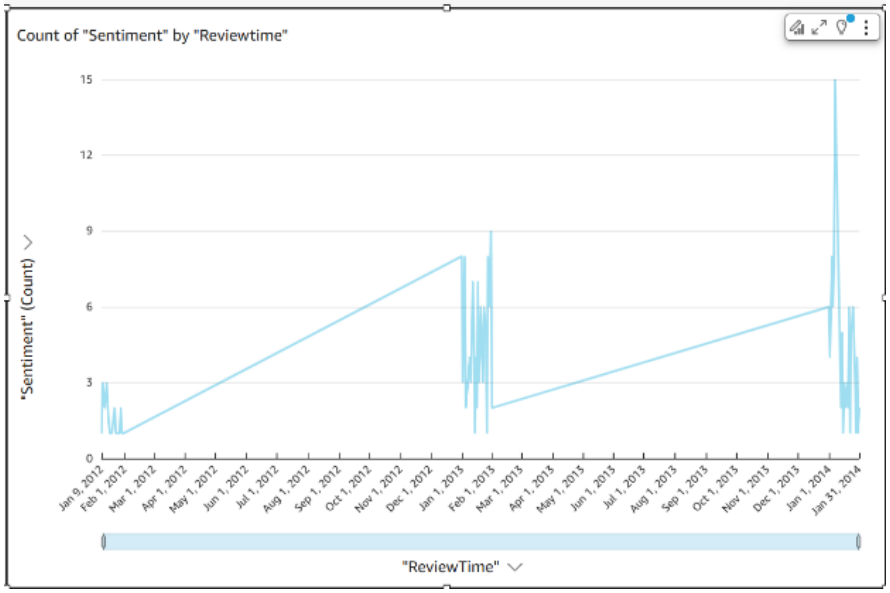
This final table gives a clear view of the most common problems or issues raised by customers.

4. Results and Insights

This section highlights key insights discovered through the visualizations created in Amazon QuickSight, using the processed outputs from Amazon Comprehend and other AWS services. Each subsection corresponds to a specific aspect of customer sentiment, behavior trends, or product-level insights.

4.1 Sentiment Trends Over Time

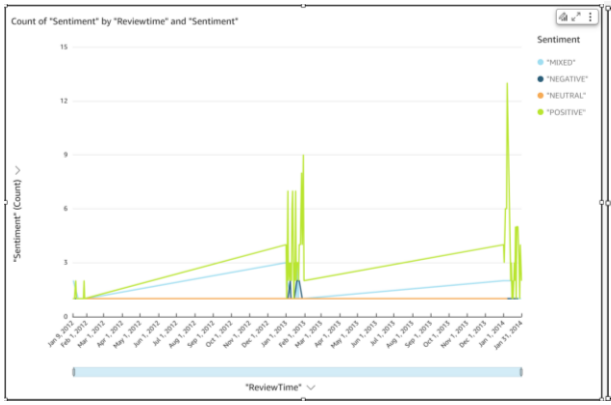
4.1.1 Daily Sentiment Volume

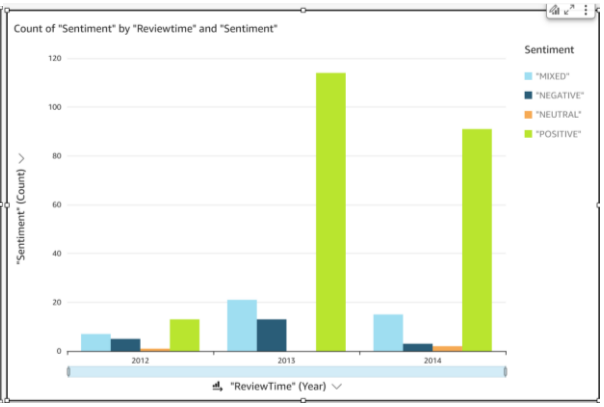


(Figure 4.1: Daily Sentiment Volume)

- Reviews were recorded only during January and February across all three years: 2012, 2013, and 2014.
- There is a clear year-over-year increase in the total number of reviews:
  - 2012: 26 reviews
  - 2013: 148 reviews
  - 2014: 111 reviews
- The highest number of reviews submitted in a single day occurred on January 7, 2014.

4.1.2 Positive Sentiment Growth



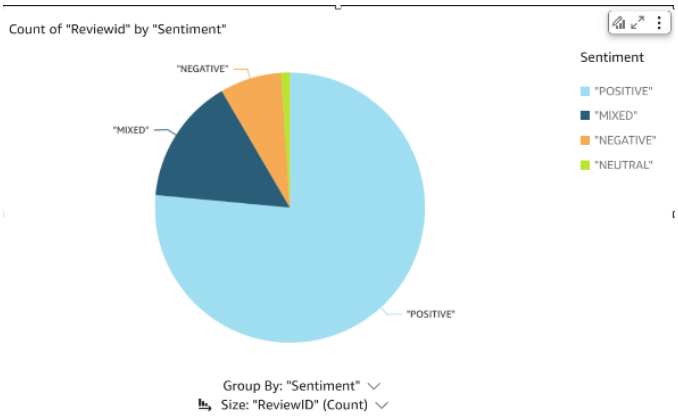


(Figure 4.2: Positive Sentiment Growth Line) (Figure 4.3: Positive Sentiment Growth Bar)

- There is a consistent increase in the percentage of positive reviews each year, reflecting improved customer satisfaction:
    - 2012: 50% (13/26)
    - 2013: 77% (114/148)
    - 2014: 82% (91/111)
- This shows growing customer approval and potentially better product performance or service.

4.2 Overall Sentiment Distribution

4.2.1 Pie Chart Analysis



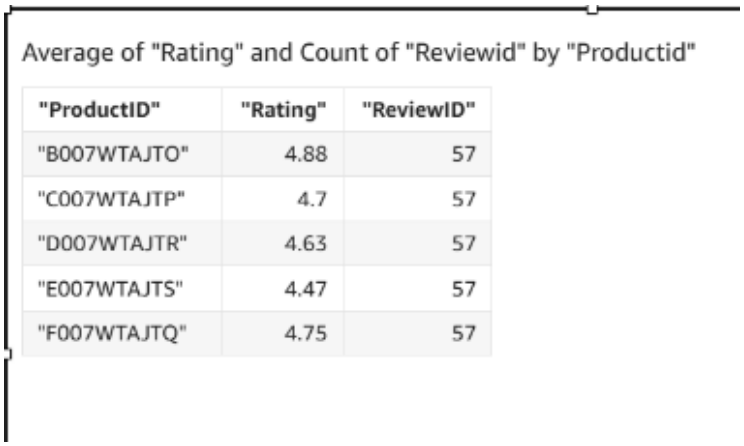
(Figure 4.4: Pie Chart Analysis)

- Out of 285 total reviews, the sentiment distribution is:
    - Positive: 76%
    - Mixed: 15%
    - Negative: 7%
    - Neutral: 1%

- A large majority of feedback is positive, indicating generally high customer satisfaction.

4.3 Product-Level Review Analysis

4.3.1 Average Ratings and Review Counts per Product



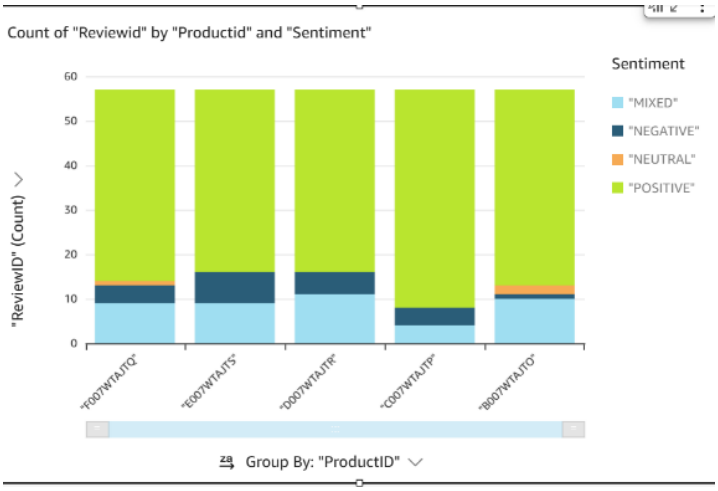
Average of "Rating" and Count of "Reviewid" by "Productid"

"ProductID"	"Rating"	"ReviewID"
"B007WTAJTO"	4.88	57
"C007WTAJTP"	4.7	57
"D007WTAJTR"	4.63	57
"E007WTAJTS"	4.47	57
"F007WTAJTQ"	4.75	57

(Figure 4.5: Average Ratings and Review Counts per Product)

- All five products received more than 50 reviews each, showing good customer engagement.
- Product B007WTAJTO has the highest average rating at 4.88.

4.3.2 Sentiment Distribution per Product



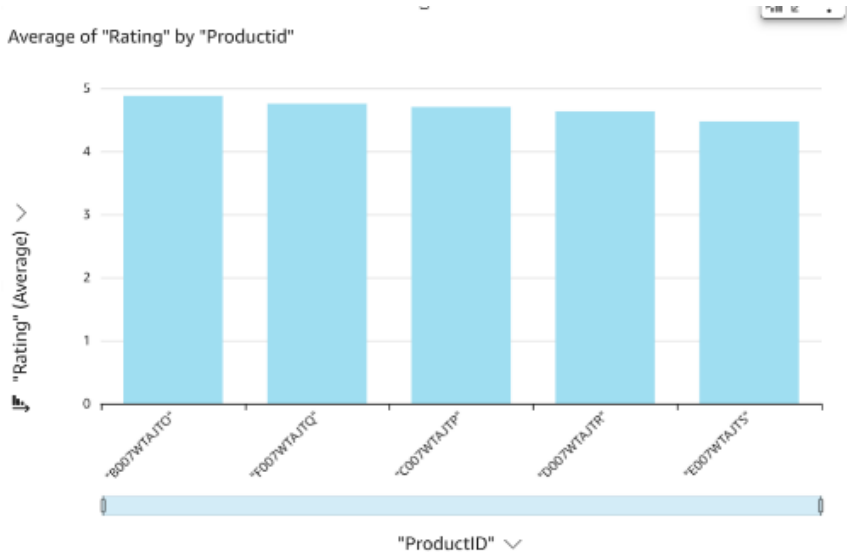
(Figure 4.6: Sentiment Distribution per Product)

Product-wise breakdown of sentiments reveals:

- C007WTAJTP received the highest count of positive reviews (49).

- D007WTAJTR and E007WTAJTS had a notable number of Positive reviews (41), but E007WTAJTS had little more negative reviews when compared with D007WTAJTR for quality issues.
- Mixed reviews were present across all products, suggesting customers experienced both pros and cons.

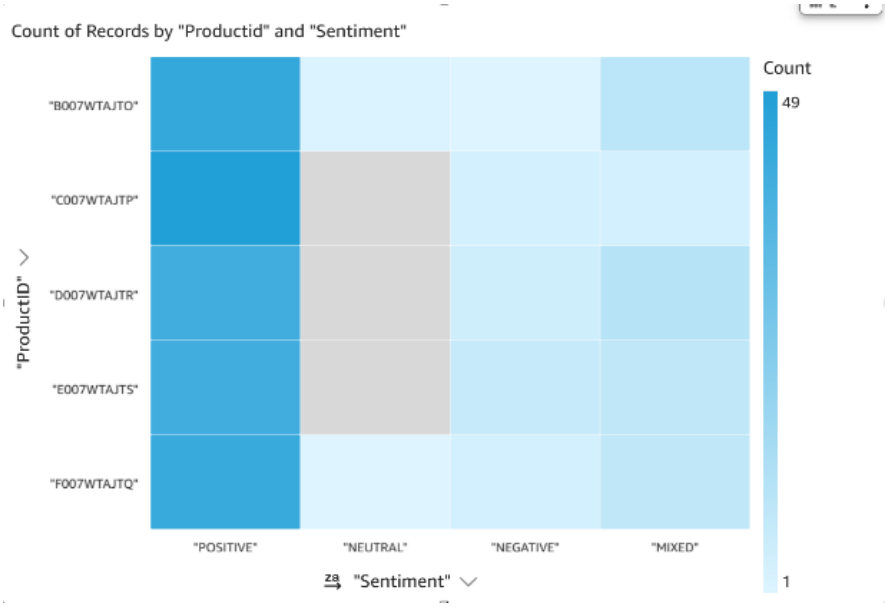
4.3.3 Average Rating Visualization



(Figure 4.7: Average Rating Visualization)

- Vertical bar charts confirm consistent high average ratings across products, reinforcing the reliability and satisfaction of the product lineup.

4.3.4 Review Heatmap



(Figure 4.8: Review Heatmap)

The heatmap visually confirms sentiment concentration for each product, making it easier to identify:

- Which product received the most negative or positive sentiments.
- Products with higher “Mixed” feedback, indicating possible inconsistencies.

4.3.5 Feedback Summary (Pivot Table)

Average of "Rating", Count of "Reviewid", Average of "Sentimentscore_positive", and Average of "Sentimentscore_negative" by "Productid"

Rows	"Rating"	"ReviewID"	"Sentiment...	"Sentiment...
"B007WTAJTO"	4.88	57	74.53%	3.7%
"C007WTAJTP"	4.7	57	85.29%	6.47%
"D007WTAJTR"	4.63	57	72.97%	6.8%
"E007WTAJTS"	4.47	57	70.65%	13.03%
"F007WTAJTQ"	4.75	57	77.57%	7.96%

(Figure 4.9: Feedback Summary (Pivot Table))

- Average positive sentiment scores:
  - Highest: C007WTAJTP – 85.29%
  - Lowest: E007WTAJTS – 70.65%
- Products with lower positive scores and higher negative scores might require attention for quality improvement or customer support enhancements.

4.4 Key Phrase Analysis – Negative Feedback

Count of "Reviewid" by "Keyphrases" and "Sentiment"

"KeyPhrases"	"Sentiment"	"ReviewID"
"the card"	"NEGATIVE"	8
"this card"	"NEGATIVE"	8
"files"	"NEGATIVE"	7
"amazon"	"NEGATIVE"	6
"the phone"	"NEGATIVE"	5
"the tablet"	"NEGATIVE"	5
"sandisk"	"NEGATIVE"	4
"fat32"	"NEGATIVE"	3
"my pc"	"NEGATIVE"	3
"songs"	"NEGATIVE"	3
"speed"	"NEGATIVE"	3
"10"	"NEGATIVE"	2
"a transfer speed"	"NEGATIVE"	2
"a while"	"NEGATIVE"	2
"an exfat"	"NEGATIVE"	2
"an issue"	"NEGATIVE"	2
"black friday"	"NEGATIVE"	2
"link"	"NEGATIVE"	2

View: 500 items 1 of 1

(Figure 4.10: Key Phrase Analysis – Negative Feedback)

- Top recurring negative key phrases include:
    - “the card”, “this card”, “files”, “amazon”, “the phone”
- These suggest product compatibility, performance, or delivery issues.
- Phrases like “fat32”, “exfat”, “transfer speed” point to technical shortcomings or customer confusion about storage formats.

5. Challenges faced and solutions

1. Lambda Function Timeout Issue

- Challenge: The Lambda function initially exceeded the 30-second execution limit due to the size of data (over 3000 records) and inefficient handling.
- Cause: Processing all data at once without batching caused timeout.
- Solution Implemented: Reduced the dataset size to 285 records, implemented batch-wise processing, and increased the Lambda timeout duration from 3 seconds to 30 seconds. As a result, the function executed successfully within 13 seconds.

2. Daily Trigger of Product Summary Lambda

- Challenge: Needed to automate the execution of the product review summary generation every day.
- Cause: Manual execution was inefficient and error-prone.
- Solution Implemented: Configured Amazon EventBridge to schedule the Lambda function to run automatically once per day.

3. Exporting DynamoDB Data to QuickSight Without Paid Services

- Challenge: Connecting DynamoDB directly to QuickSight required AWS Glue and Athena, but Athena was not available in the Free Tier. Additionally, the DynamoDB → Export to S3 feature also had paid limitations.
- Cause: AWS architecture limitations and service tier restrictions.
- Solution Implemented: Used AWS CloudShell and Python scripts to export data from DynamoDB as JSON into S3. Then, a separate Python-based Lambda function was used to convert the JSON into a cleaned CSV file suitable for QuickSight.

4. Data Cleaning and Formatting for QuickSight

- Challenge: When visualizing data in QuickSight, many columns showed incorrect values due to improper parsing—especially from fields like Entities,

KeyPhrases, and ReviewText that contained commas or unnecessary quotation marks.

- Cause: Unescaped punctuation caused column misalignment during CSV interpretation in QuickSight.
- Solution Implemented: Wrote a Lambda function to clean the data by removing commas and quotation marks from text fields during the CSV conversion. Avoided removing periods (.) to preserve decimal numbers such as ratings or sentiment scores. This cleanup ensured proper column mapping in QuickSight.

6. Future enhancements

- **Reduce Comprehend API calls by checking for duplicates**
  I want to make the whole system more efficient in terms of time and API usage. One idea I have is to avoid reprocessing the same review again and again. So maybe I can keep a hash or ID check if the same review is already analyzed in a previous run, the Lambda function can skip it instead of calling Comprehend again. That way, I can reduce the number of API calls and save time and cost.

- **Add product category detection**
  In the future, I might try to classify reviews based on product categories like electronics, fashion, etc. This can help generate more focused summaries per product type and give better insights.

- **Improve summary quality using Bedrock or SageMaker**
  If time permits and it fits the free tier, I may look into using Amazon Bedrock or SageMaker JumpStart to generate better summaries. The current ones are working, but something more natural sounding would be nice to explore.

- **Language detection for multi-language support**
  I might add language detection to the system using Comprehend. Later, I can use Amazon Translate if needed. This would help support reviews in different languages and expand the scope.

- **Live dashboard auto-refresh in QuickSight**
  Right now, I'm manually updating the visuals in QuickSight. Later, I want to set up auto-refresh so that whenever new data is added, the dashboard reflects it automatically—without needing to reload manually.

- **Backup DynamoDB data automatically**
  I'm thinking of setting up an automated daily or weekly backup of DynamoDB data to S3, just to keep everything safe and versioned.