



**DALHOUSIE  
UNIVERSITY**

**CSCI 5410**  
**Serverless Data Processing**

**Assignment 4**

**Name : Sagarkumar Pankajbhai Vaghasia**

**CSID : vaghasia**

**Banner ID : #B00878629**

**Gitlab Link :**

<https://git.cs.dal.ca/vaghasia/csci5410-f23-b00878629/-/tree/A4>

## PART-A: Use AWS Lambda-SQS-SNS.

- First and foremost, I logged in AWS account through AWS academy student login. After log in, I selected the dashboard option from the left-hand side panel where a course is available under the name of “ALLFv1-17043”. Selecting the course took me to AWS Academy Learner Lab – Foundation Services page from which I selected “Modules” section which displayed three options. Amongst those options, selected “Learner Lab – Foundational Services”. The screenshots from logging to the process of selecting modules are shown in Figure-1 to Figure-3.

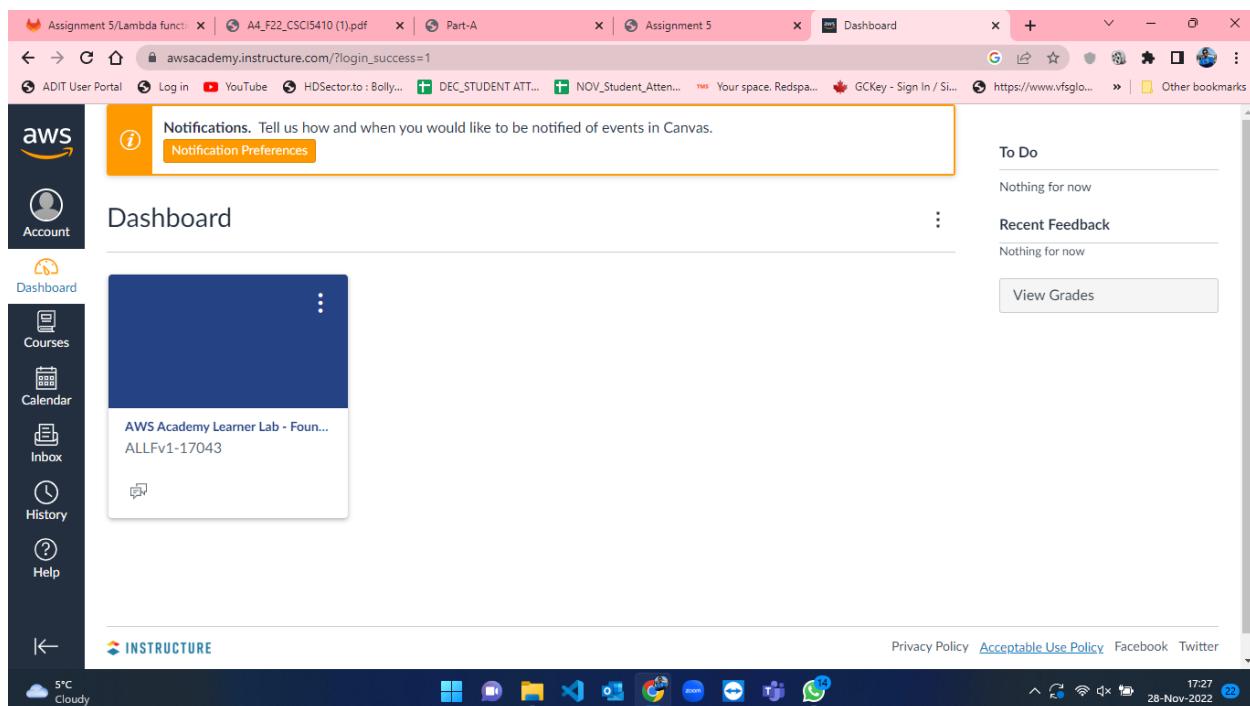


Figure 1: AWS Dashboard

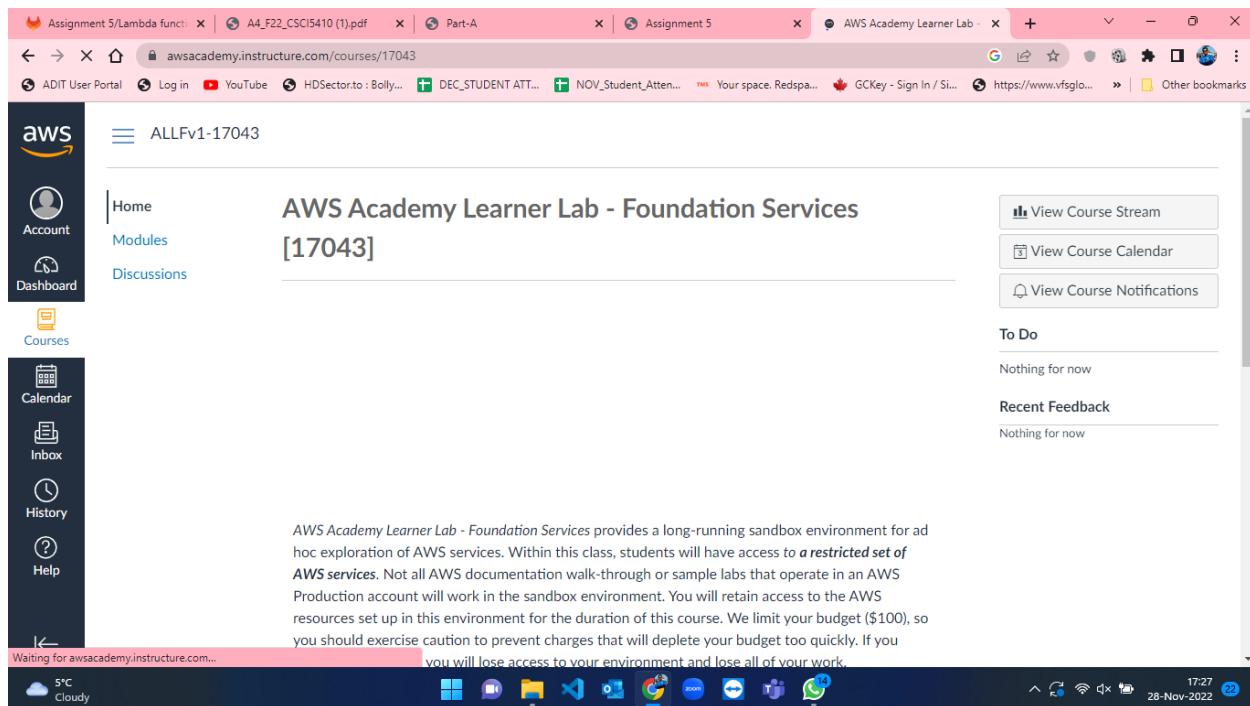


Figure 2: Homepage after course selection from dashboard in AWS.

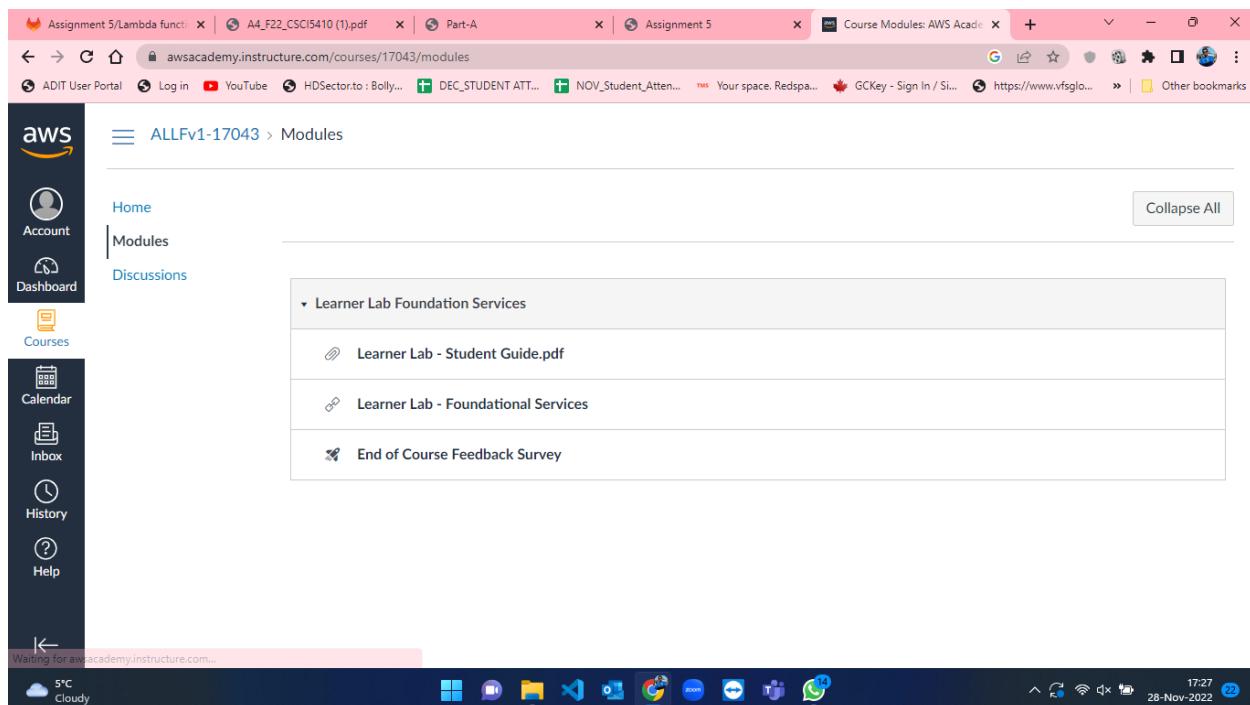


Figure 3: Modules page in AWS.

- After selecting “Learner Lab – Foundational Services”, the console page for the selected module is opened where the lab is currently not running. This can be seen in the below attached figure-4 where there is red mark besides AWS text. To start working on AWS and its services we must start a lab. So, click on “Start Lab” button to initiate lab. This can be seen in figure-5 where the lab is started, and the red mark turns to green besides AWS text.

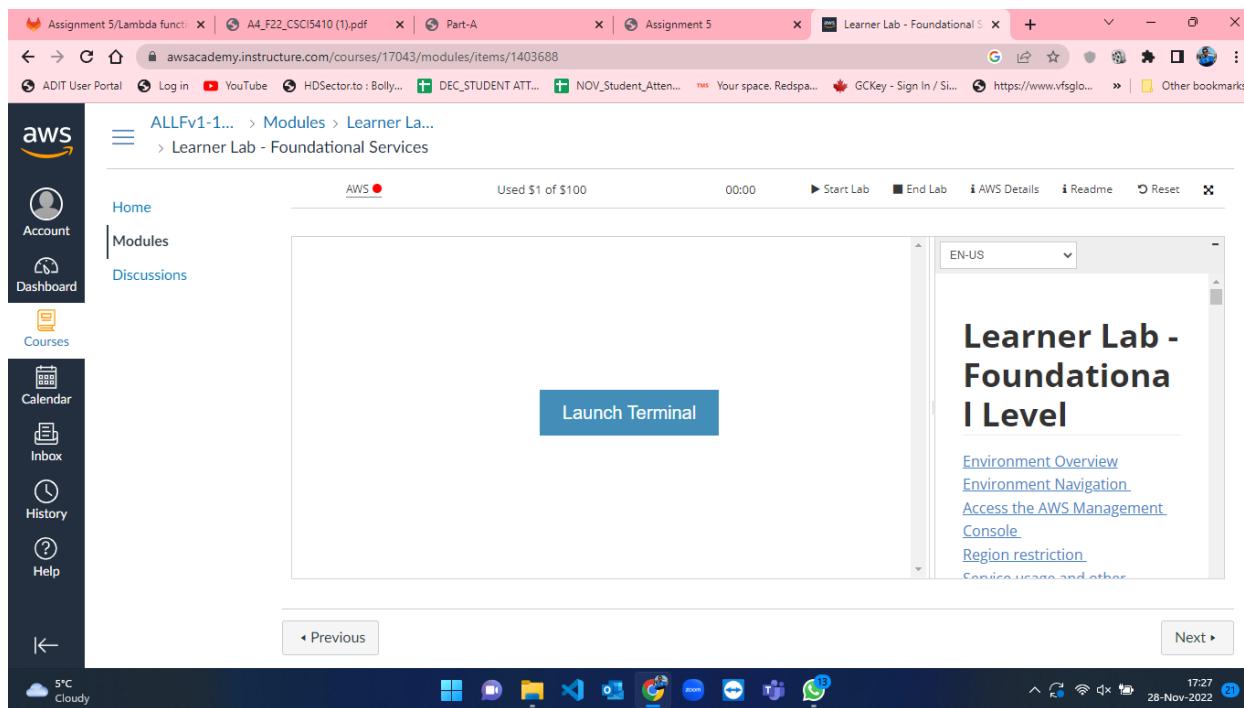


Figure 4: Lab not running in Modules page in AWS.

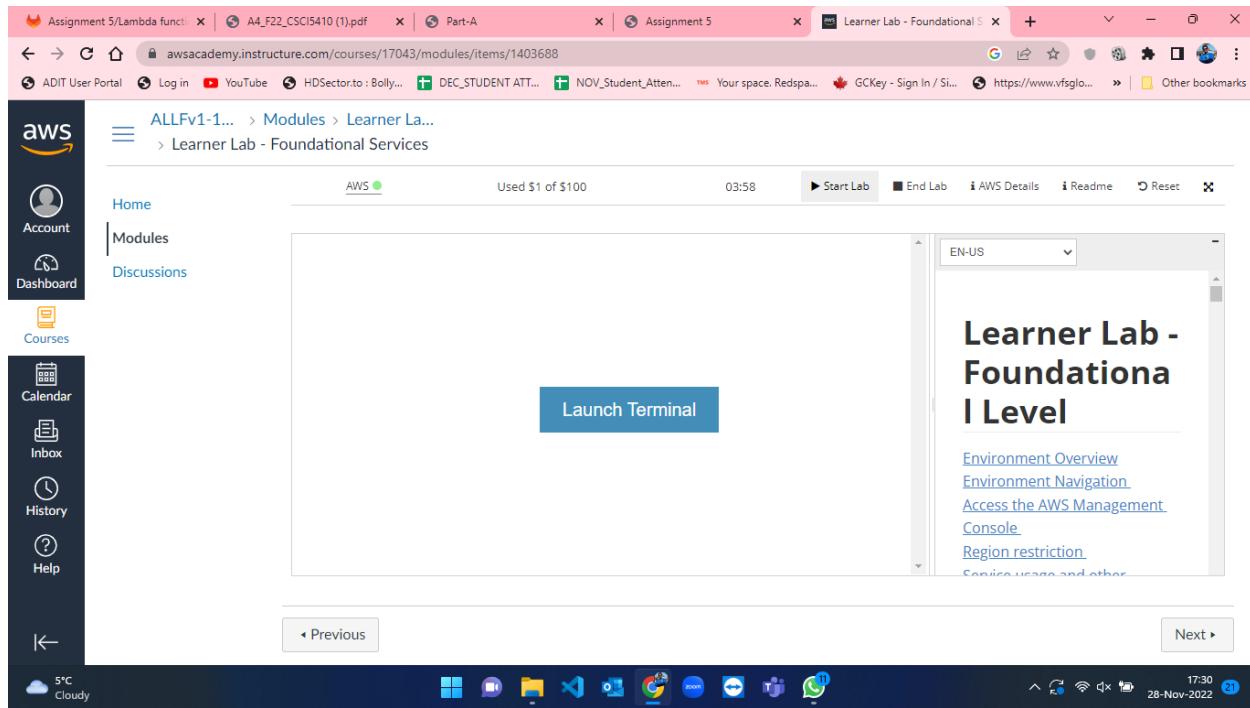


Figure 5: Lab started under Modules page in AWS.

- As soon as the lab started, click on AWS text beside the green mark which took me the AWS console home (Figure-6). From that now we can access the services of AWS.

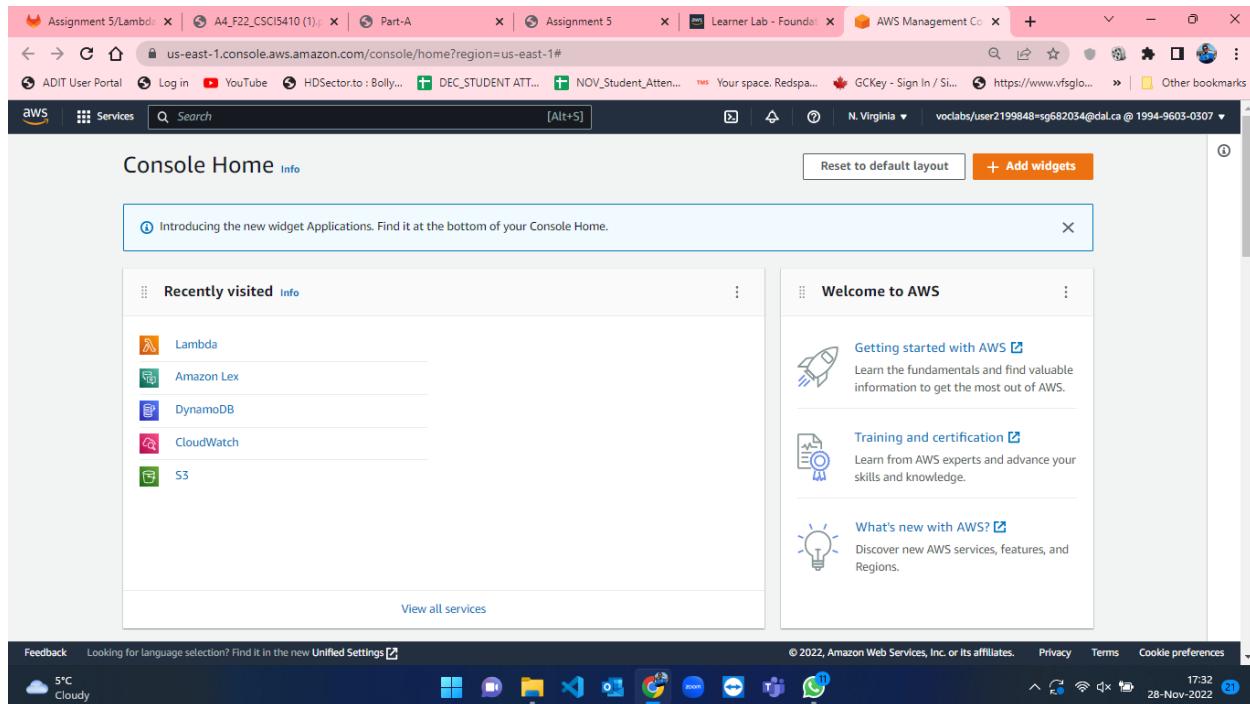


Figure 6: AWS Console Home.

- We have to create a Simple Queue Service so search SQS in the search bar at the top which will take us to SQS homepage. This is represented in figure-7 and figure-8.

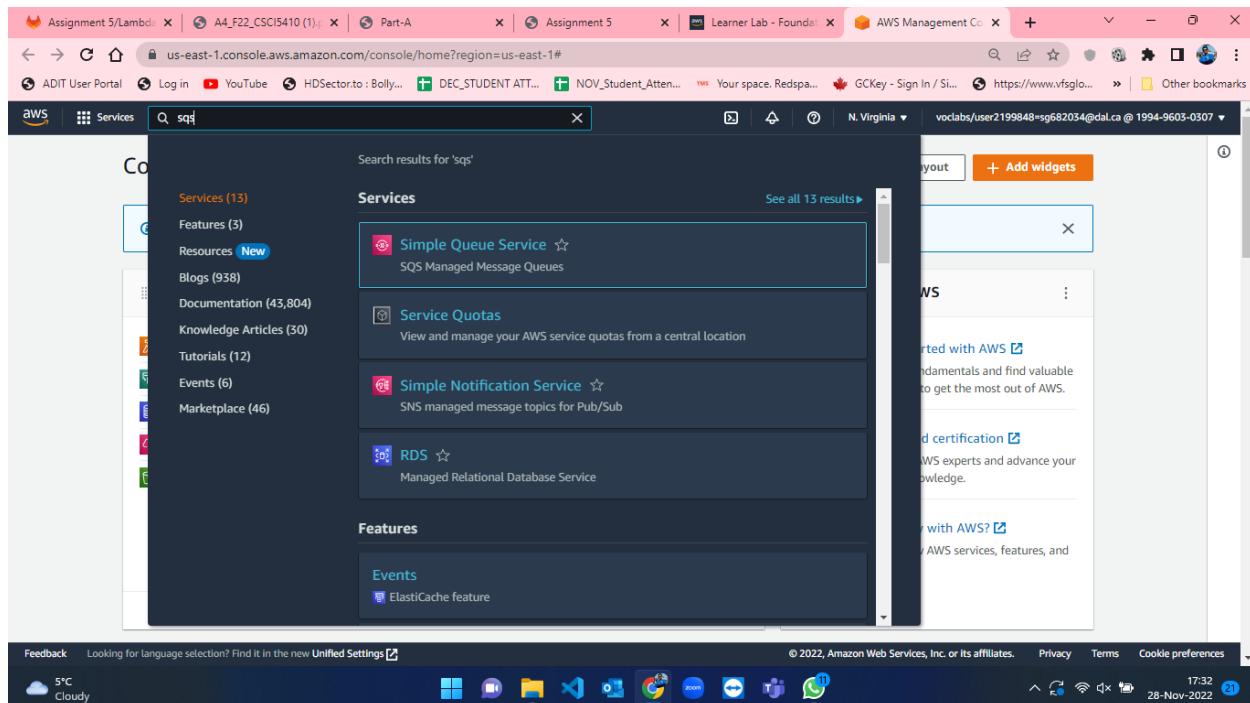


Figure 7: Searching for SQS in search bar from Console Home in AWS.

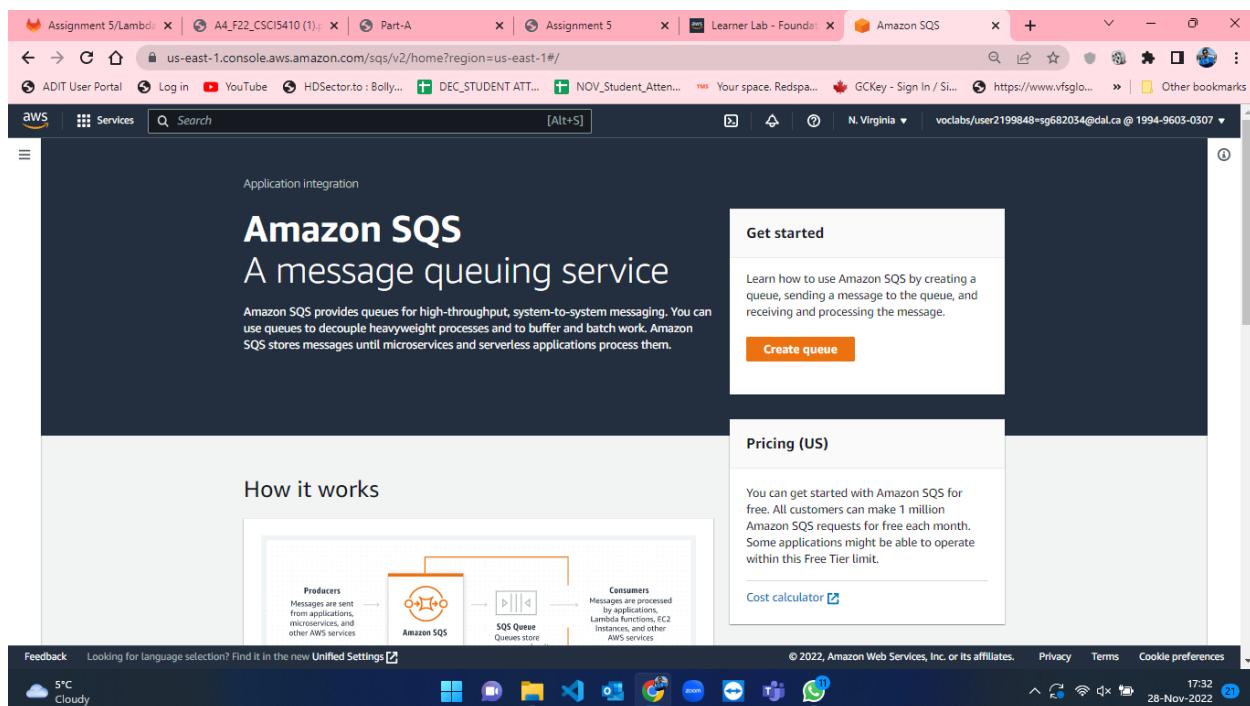


Figure 8: SQS homepage in AWS Console.

- From SQS homepage, we will click on “Create queue”[1] button to create a new queue which will take us to create queue page. The process for creating queue is not tedious where I have selected standard queue option and named it as “assignment4”[1]. Other parameters kept default and click on “Create queue” button at bottom of the page which will create a SQS queue for us. The process for creating queue is shown from figure-9 to figure-13.

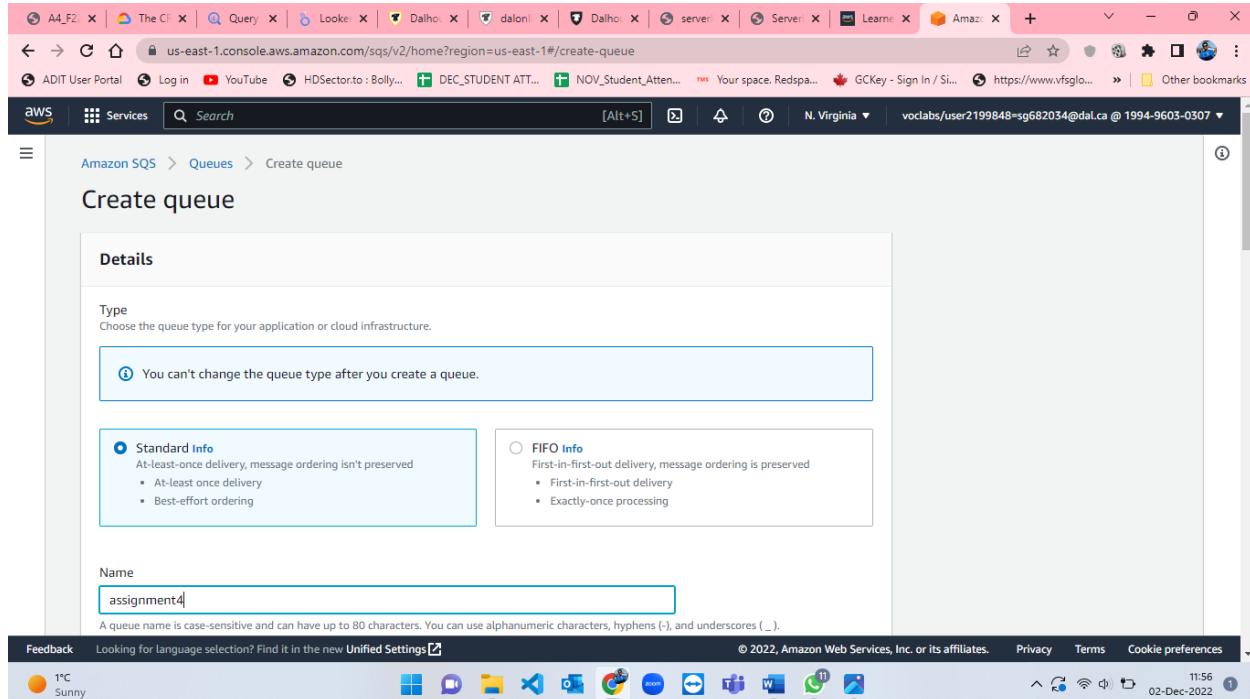


Figure 9: SQS create queue page – 1[1].

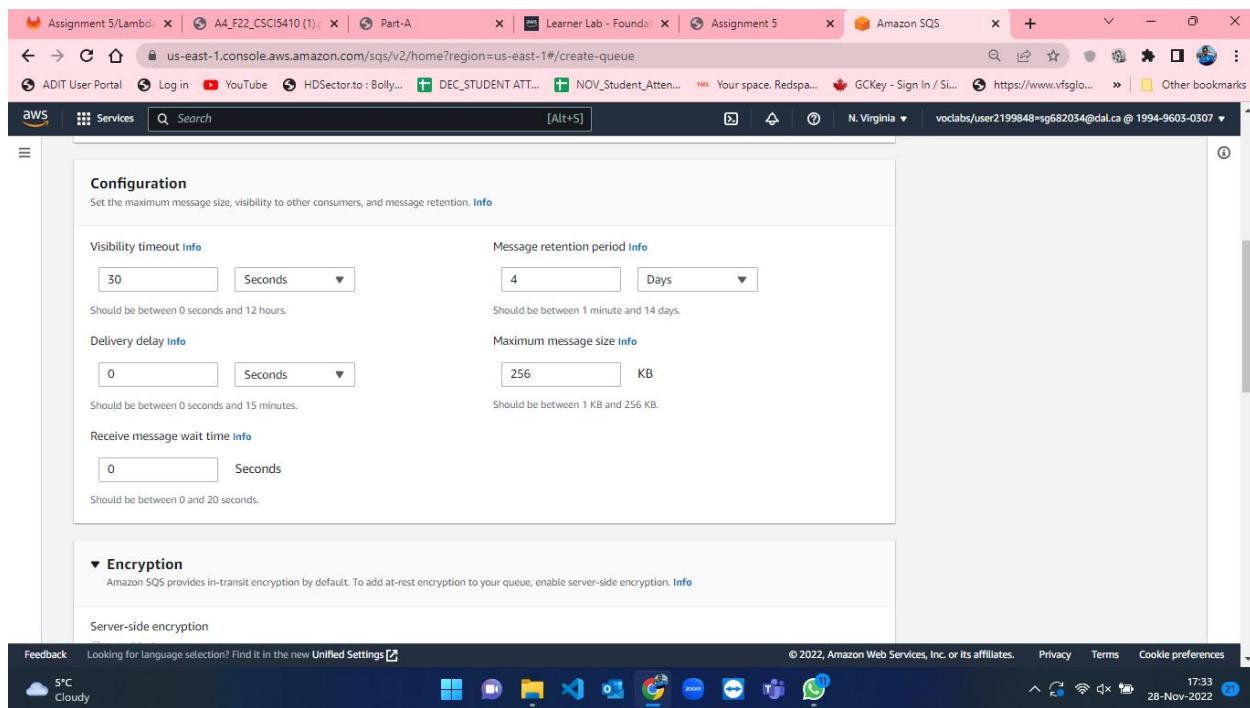


Figure 10: SQS create queue page – 2[1].

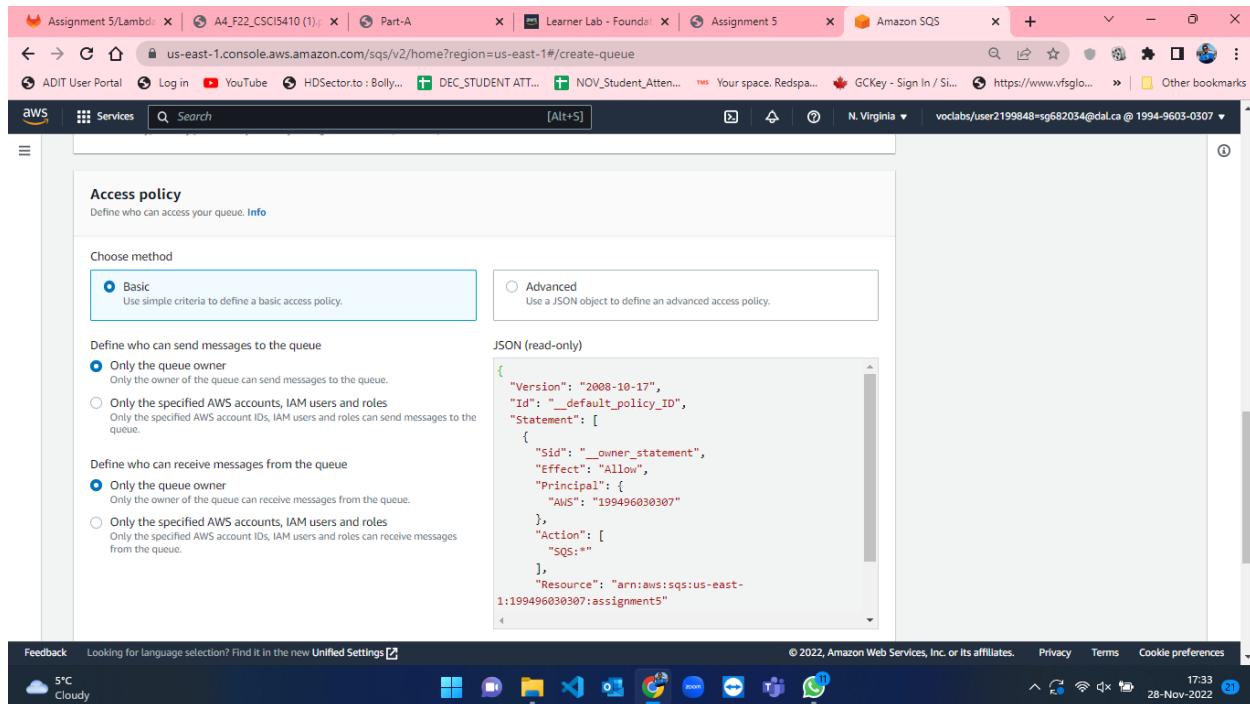


Figure 11: SQS create queue page – 3[1].

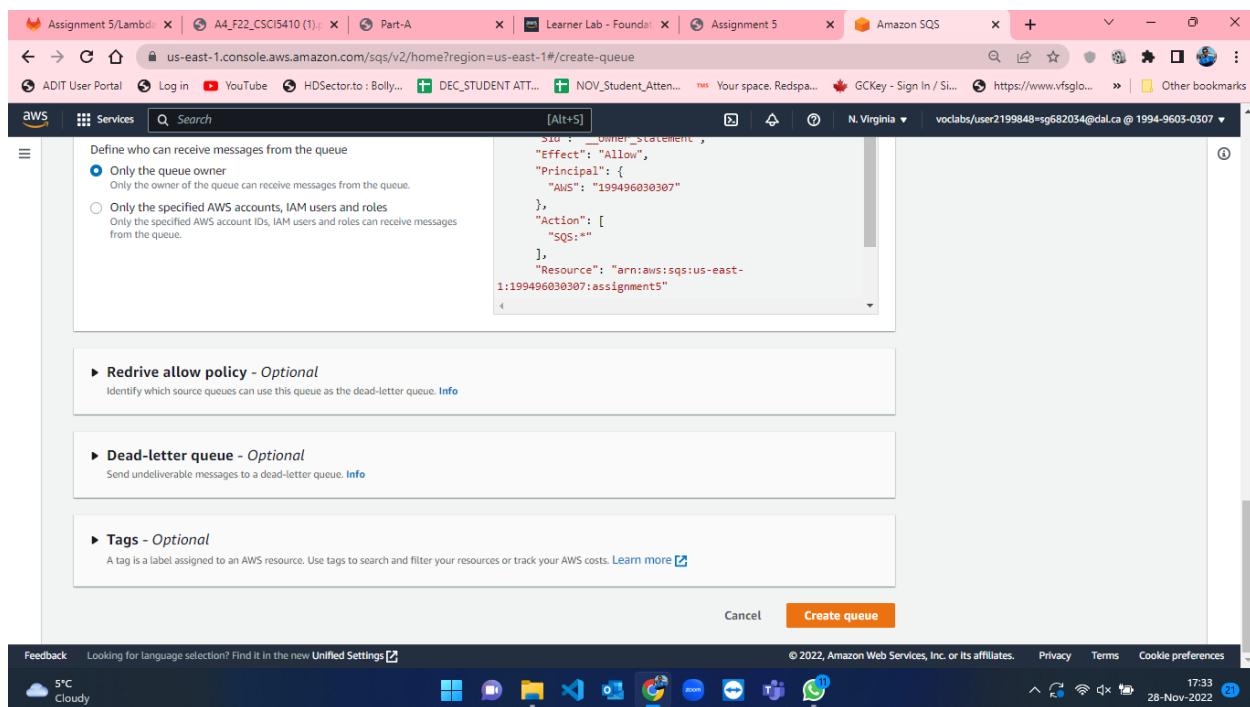


Figure 12: SQS create queue page – 4[1].

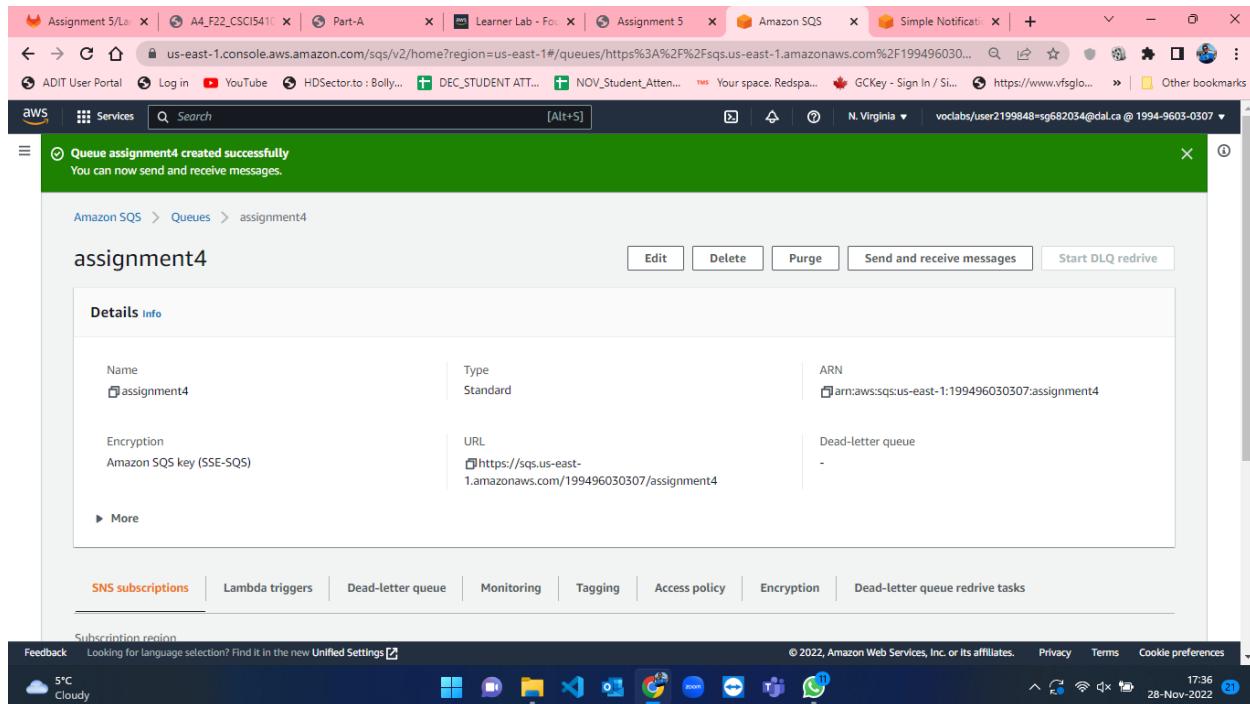
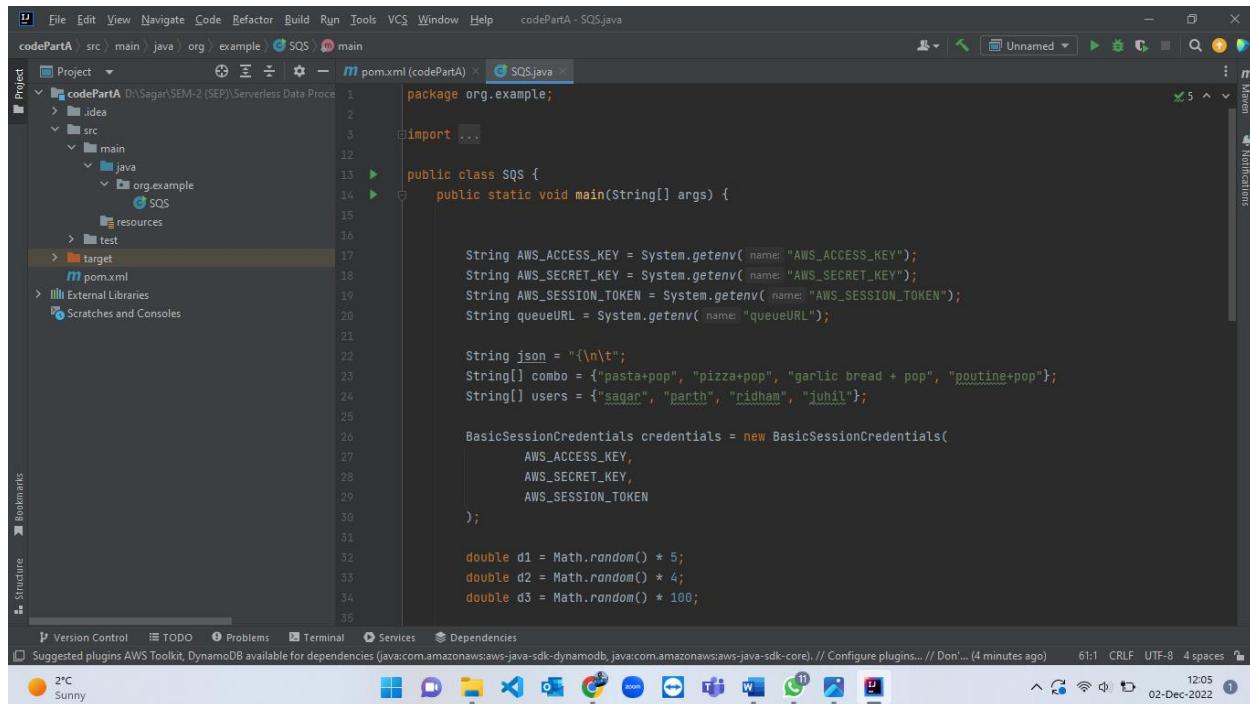


Figure 13: Successful creation of assignment4 queue.

- After creation of queue, I created maven project and developed a JAVA code to send random values[2] of food combos, name of the user(here customers who ordered the food combo), time of order and order id to SQS queue[3]. For using AWS credentials in maven project, I configured the credentials in environment variables, and I have accessed those variables using System.getenv() [4]. The result from the code is sent in JSON format. This procedure can be seen in the attached figures below from 14 to 17.



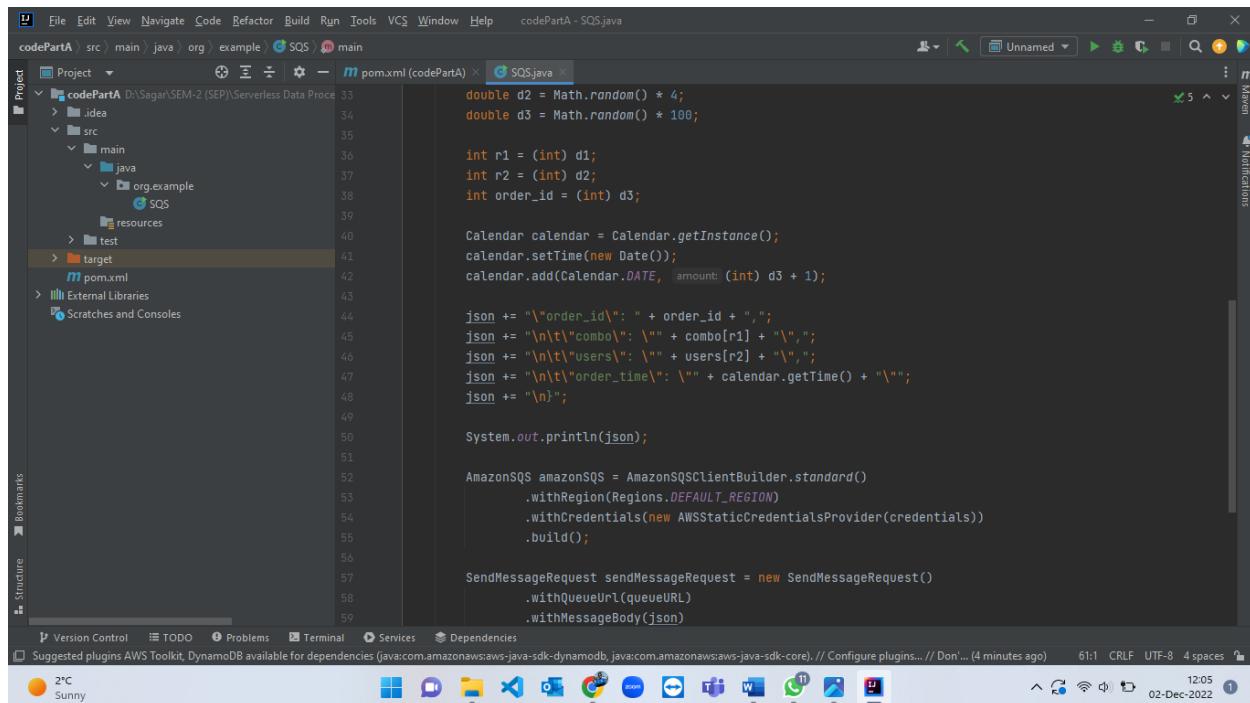
```

codePartA > src > main > java > org > example > SQS > main
Project pom.xml (codePartA) SQS.java

1 package org.example;
2
3 import ...
4
5 public class SQS {
6     public static void main(String[] args) {
7
8         String AWS_ACCESS_KEY = System.getenv("AWS_ACCESS_KEY");
9         String AWS_SECRET_KEY = System.getenv("AWS_SECRET_KEY");
10        String AWS_SESSION_TOKEN = System.getenv("AWS_SESSION_TOKEN");
11        String queueURL = System.getenv("queueURL");
12
13        String json = "\n\t";
14        String[] combo = {"pasta+pop", "pizza+pop", "garlic bread + pop", "poutine+pop"};
15        String[] users = {"sagar", "parth", "idham", "juhul"};
16
17        BasicSessionCredentials credentials = new BasicSessionCredentials(
18            AWS_ACCESS_KEY,
19            AWS_SECRET_KEY,
20            AWS_SESSION_TOKEN
21        );
22
23        double d1 = Math.random() * 5;
24        double d2 = Math.random() * 4;
25        double d3 = Math.random() * 100;
26
27        double d4 = Math.random() * 4;
28        double d5 = Math.random() * 100;
29
30        int r1 = (int) d1;
31        int r2 = (int) d2;
32        int order_id = (int) d3;
33
34        Calendar calendar = Calendar.getInstance();
35        calendar.setTime(new Date());
36        calendar.add(Calendar.DATE, amount((int) d3 + 1));
37
38        json += "{\"order_id\": " + order_id + ",";
39        json += "\n\t\"combo\": \"\" + combo[r1] + "\",";
40        json += "\n\t\"users\": \"\" + users[r2] + "\",";
41        json += "\n\t\"order_time\": \"\" + calendar.getTime() + \"\"}";
42        json += "\n";
43
44        System.out.println(json);
45
46        AmazonSQS amazonSQS = AmazonSQSClientBuilder.standard()
47            .withRegion(Regions.DEFAULT_REGION)
48            .withCredentials(new AWSStaticCredentialsProvider(credentials))
49            .build();
50
51
52        SendMessageRequest sendMessageRequest = new SendMessageRequest()
53            .withQueueUrl(queueURL)
54            .withMessageBody(json)
55
56
57
58
59

```

Figure 14: JAVA code for sending message in SQS queue part-1.



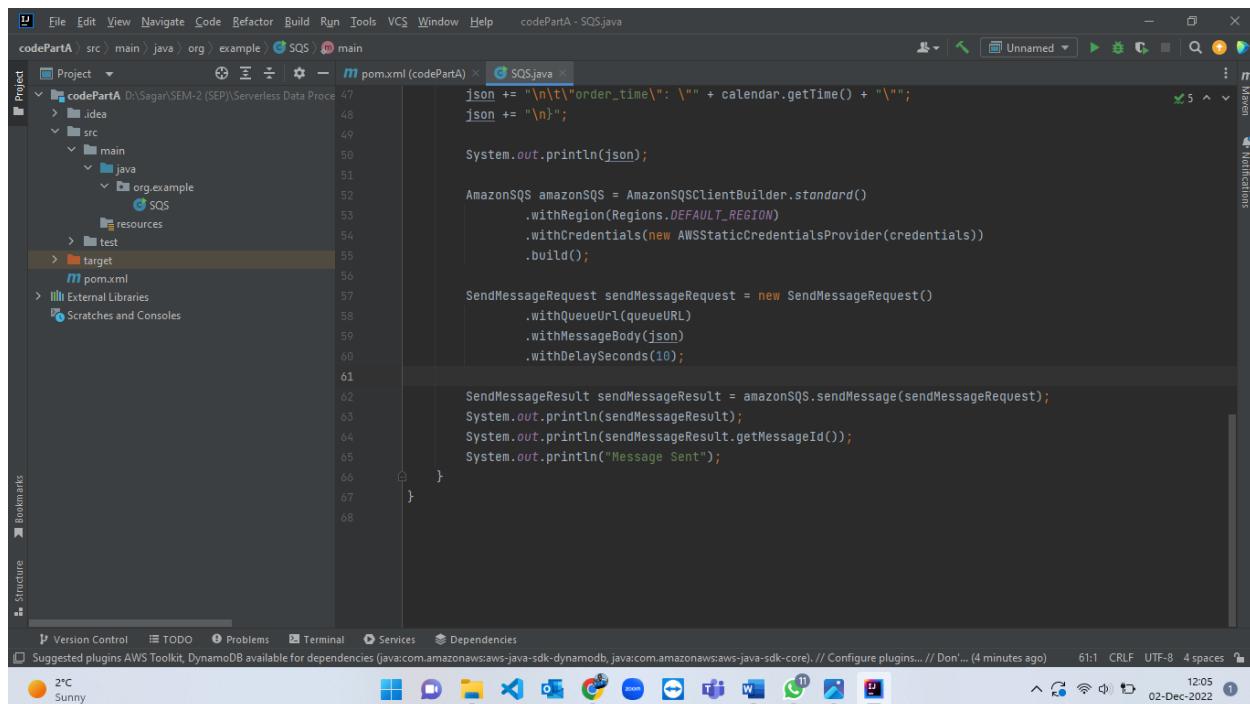
```

codePartA > src > main > java > org > example > SQS > main
Project pom.xml (codePartA) SQS.java

33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59

```

Figure 15: JAVA code for sending message in SQS queue part-2.



```

codePartA > src > main > java > org > example > SQS > main
Project pom.xml (codePartA) SQS.java pom.xml

    json += "\n\t\"order_time\": " + calendar.getTime() + "\n";
    json += "\n";

    System.out.println(json);

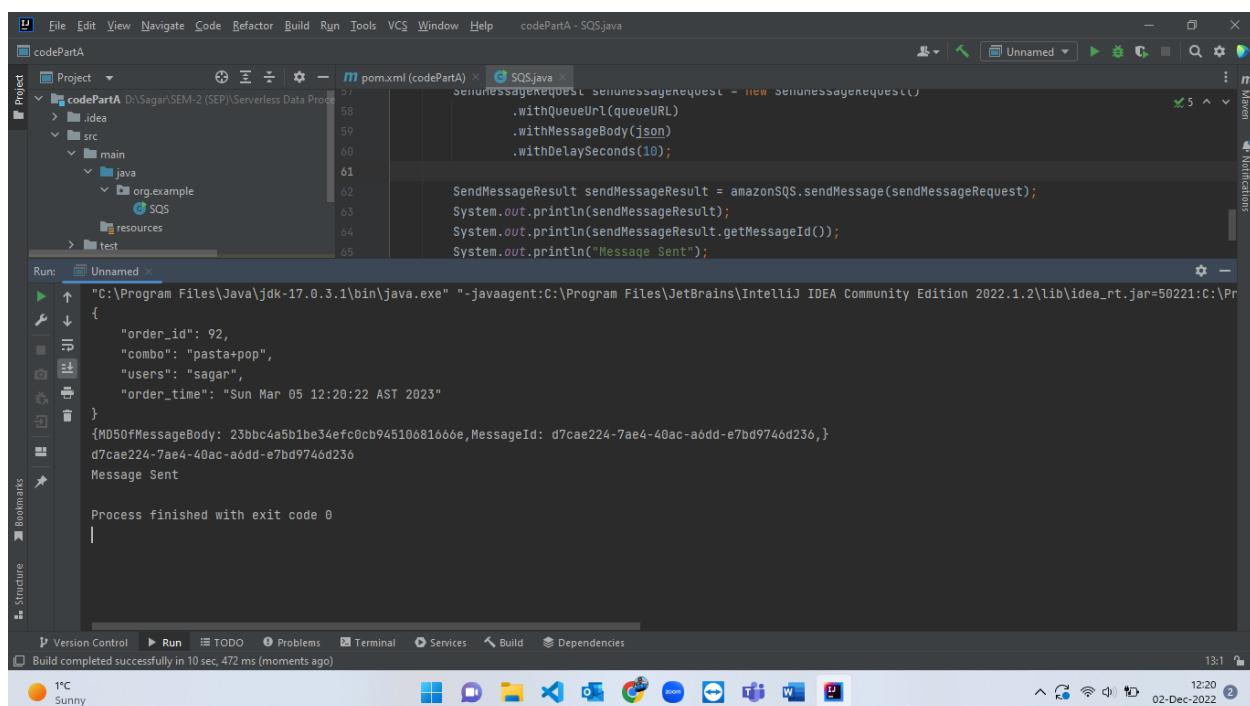
    AmazonSQS amazonSQS = AmazonSQSClientBuilder.standard()
        .withRegion(Regions.DEFAULT_REGION)
        .withCredentials(new AWSStaticCredentialsProvider(credentials))
        .build();

    SendMessageRequest sendMessageRequest = new SendMessageRequest()
        .withQueueUrl(queueURL)
        .withMessageBody(json)
        .withDelaySeconds(10);

    SendMessageResult sendMessageResult = amazonSQS.sendMessage(sendMessageRequest);
    System.out.println(sendMessageResult);
    System.out.println(sendMessageResult.getMessageId());
    System.out.println("Message Sent");
}
}

```

Figure 16: JAVA code for sending message in SQS queue part-3.



```

codePartA > src > main > java > org > example > SQS > main
Project pom.xml (codePartA) SQS.java pom.xml

    SendMessageRequest sendMessageRequest = new SendMessageRequest()
        .withQueueUrl(queueURL)
        .withMessageBody(json)
        .withDelaySeconds(10);

    SendMessageResult sendMessageResult = amazonSQS.sendMessage(sendMessageRequest);
    System.out.println(sendMessageResult);
    System.out.println(sendMessageResult.getMessageId());
    System.out.println("Message Sent");

Process finished with exit code 0

```

Run: Unnamed

```

" C:\Program Files\Java\jdk-17.0.3.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.1.2\lib\idea_rt.jar=50221:C:\Program Files\Java\jdk-17.0.3.1\bin" -Dfile.encoding=UTF-8
{
    "order_id": 92,
    "combo": "pasta+pop",
    "users": "sagan",
    "order_time": "Sun Mar 05 12:20:22 AST 2023"
}
{MD5ofMessageBody: 23bbc4a5b1be34efc0cb94510681666e, MessageId: d7cae224-7ae4-40ac-a6dd-e7bd9746d236}
d7cae224-7ae4-40ac-a6dd-e7bd9746d236
Message Sent

Process finished with exit code 0

```

Figure 17: Execution logs from JAVA code displaying that message has been successfully sent to SQS.

- When the message is successfully sent from JAVA code, now we have to verify that it is received in the SQS queue or not. For that, click on “Send and receive messages” option from SQS and then scroll to the bottom and click on ‘Poll for messages’ which will poll all the messages which are currently in a queue. The steps for viewing message in queue are shown from figure-18 to figure-21.

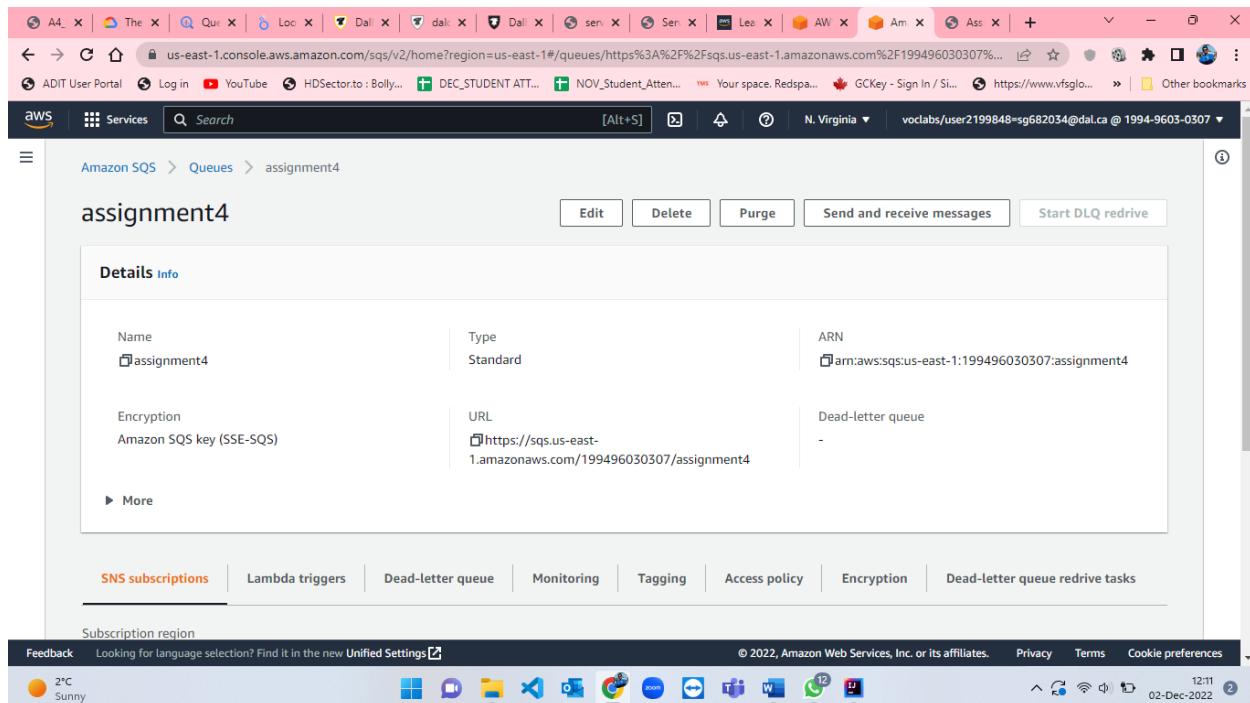


Figure 18: SQS page to select "Send and receive messages".

Should be between 0 seconds and 15 minutes.

► Message attributes - *Optional Info*

**Receive messages** Info

Messages available	Polling duration	Maximum message count	Polling progress
0	30	10	0 receives/second

**Messages (0)**

ID	Sent	Size	Receive count
No messages. To view messages in the queue, poll for messages.			

**Poll for messages**

Figure 19: "Poll for messages" option from SQS.

Should be between 0 seconds and 15 minutes.

► Message attributes - *Optional Info*

**Receive messages** Info

Messages available	Polling duration	Maximum message count	Polling progress
0	30	10	0.2 receives/second

**Messages (1)**

ID	Sent	Size	Receive count
<a href="#">d7cae224-7ae4-40ac-a6dd-g7bd9746d236</a>	12/2/2022, 12:20:26 AST	108 bytes	2

Figure 20: Message received from JAVA code to SQS.

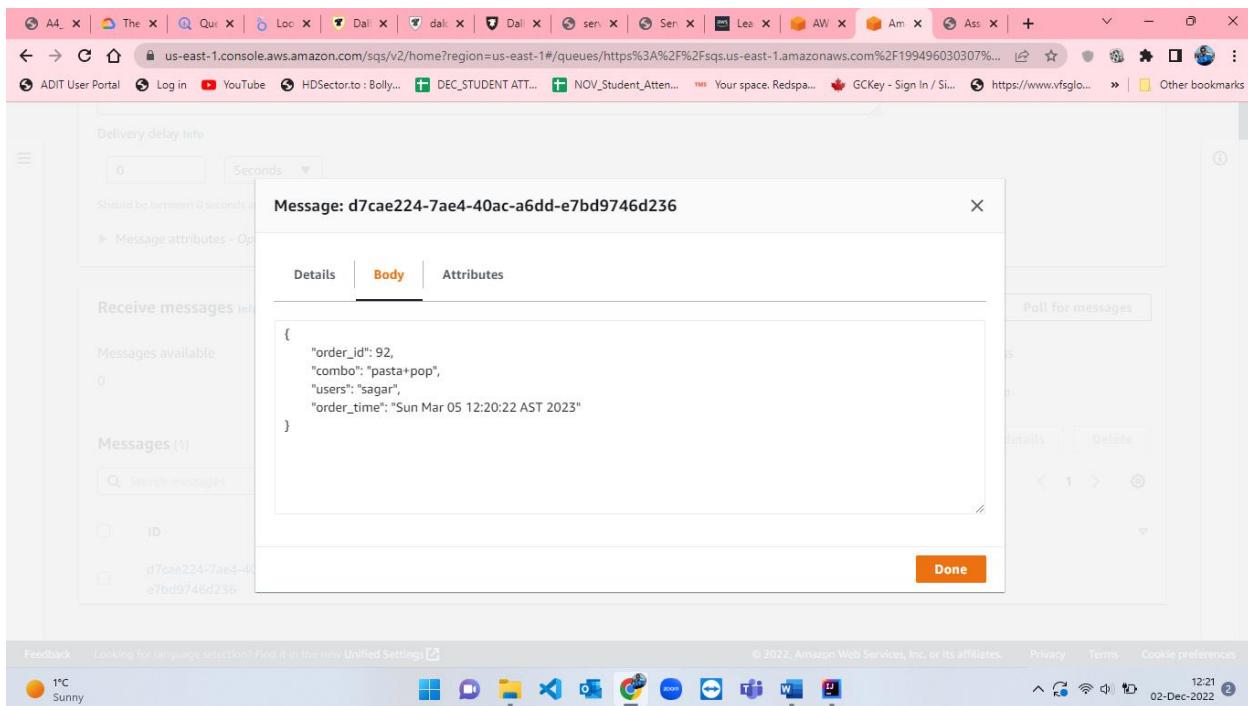


Figure 21: Displaying JSON format message received in SQS.

- After receiving message in SQS queue we have to create a SNS (simple notification service) subscription to notify the user by mail[5]. For creating SNS subscription, search for SNS in the search bar which will take us to SNS homepage where we click on “Create topic” button. I named SNS topic as “a4sns” and selected “standard” type for sns. The other parameters while creating sns are set to default[5]. The procedure for creating sns is shown from figure-22 to figure-26.

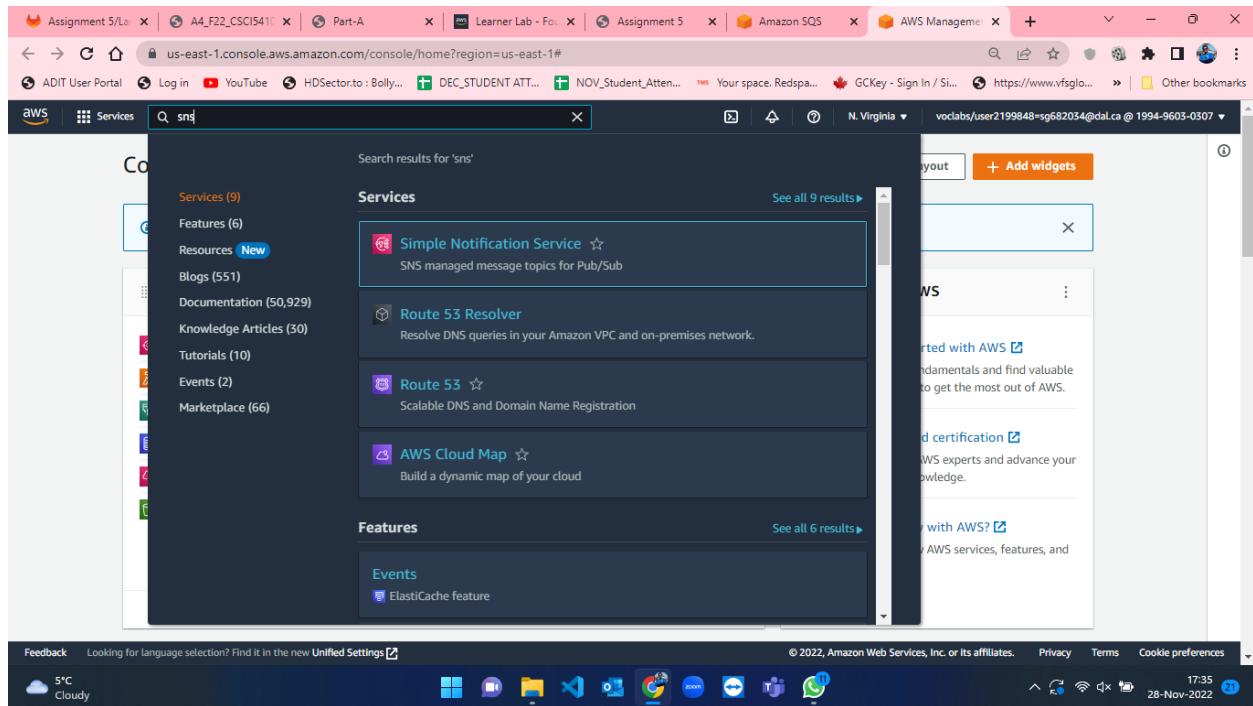


Figure 22: Searching for SNS in search bar from Console Home in AWS.

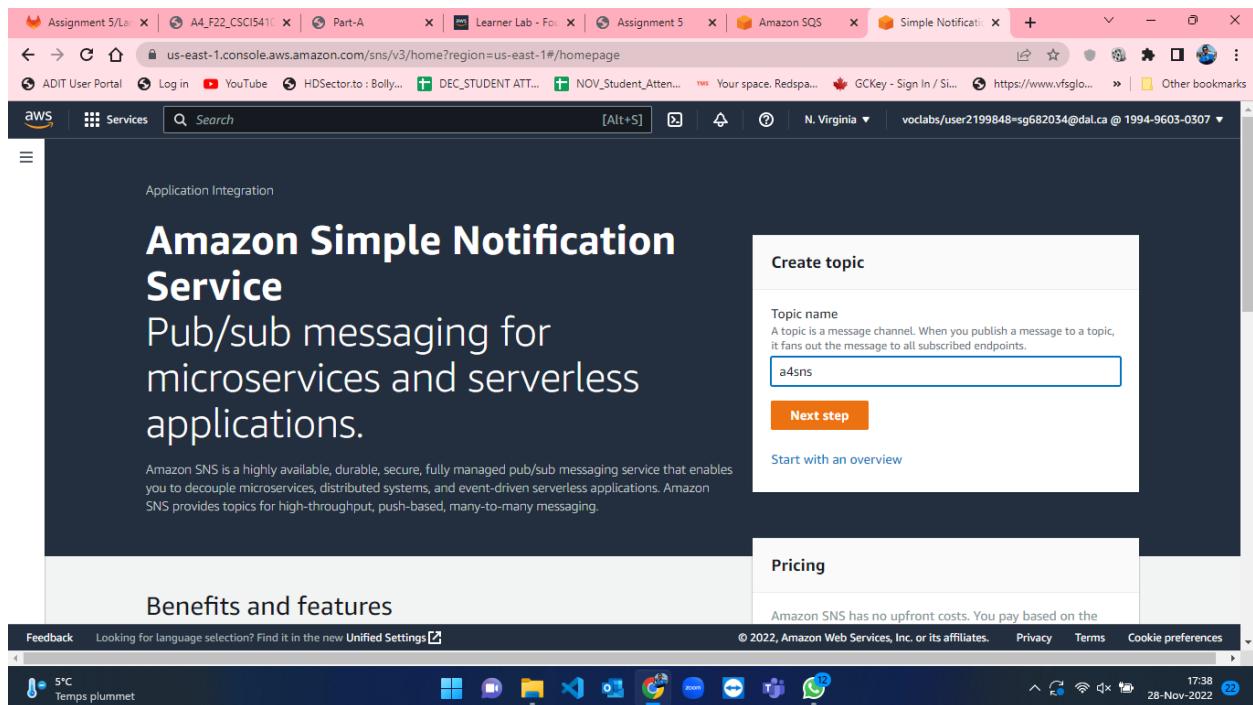


Figure 23: SNS homepage in AWS Console.

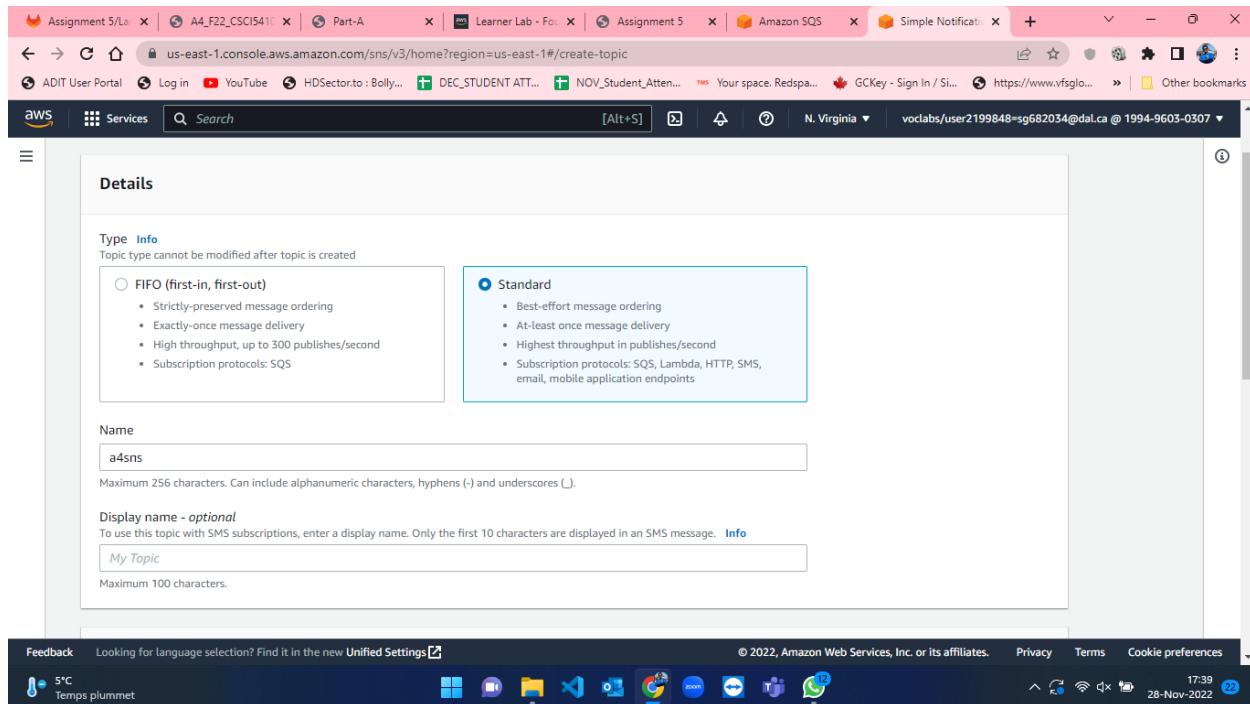


Figure 24: Create SNS part-1.

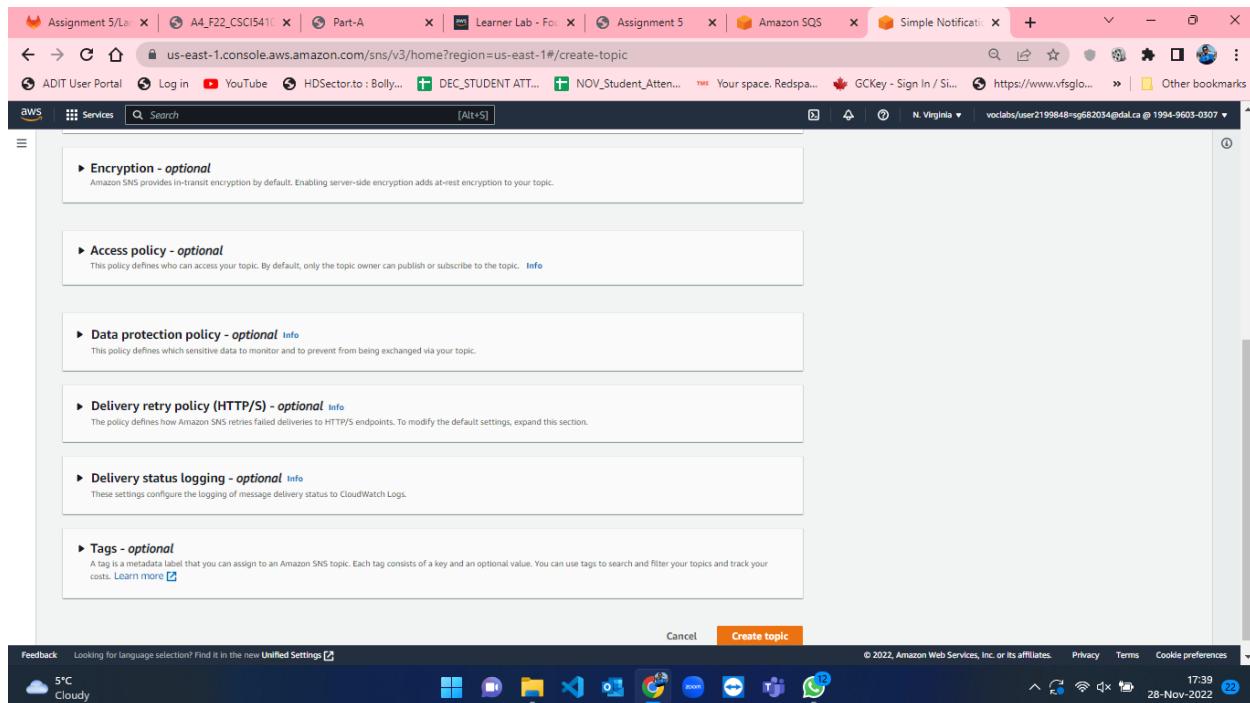


Figure 25: Create SNS part-2.

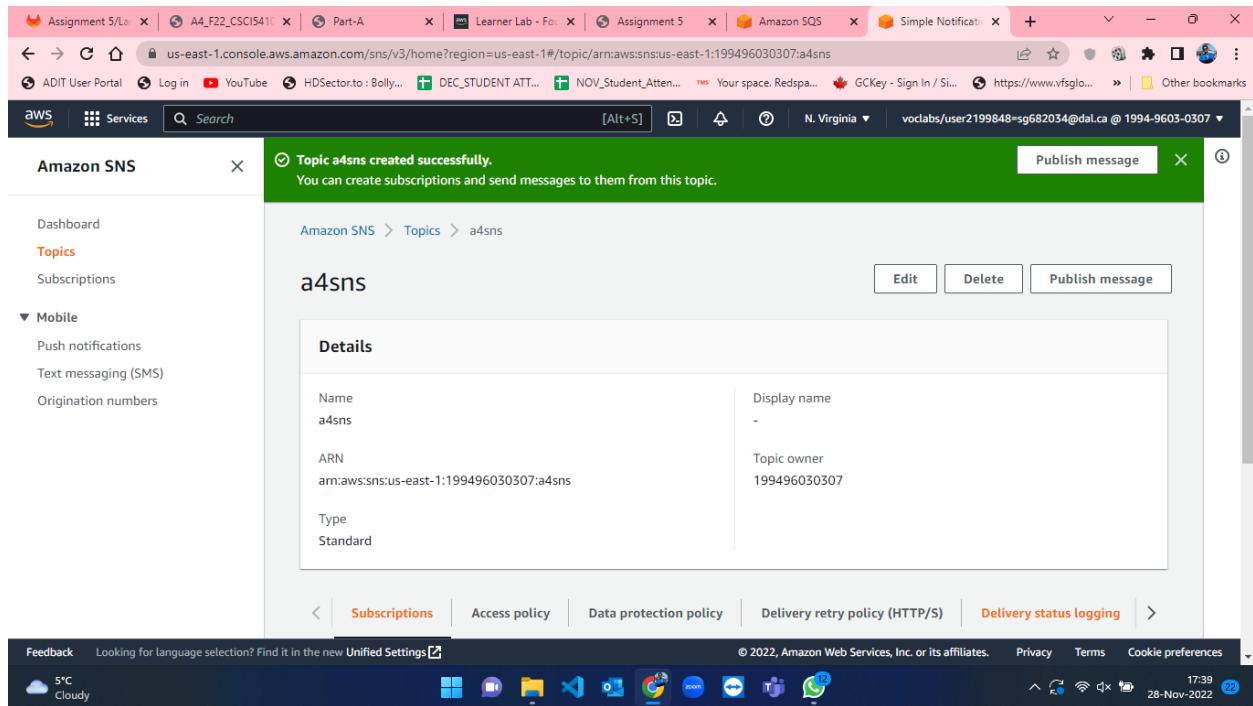


Figure 26: Successful creation of a4sns.

- As soon as SNS is created, we have to then create a subscription where we want to send the notification[5]. For that, we have to scroll down from the sns topic which is recently created and then we will click on “Create subscription” button which is shown in figure-27. After that, we have to select topic ARN, Protocol type which I have selected email for email notification and endpoint where I specified my email address for getting notification[5]. The other parameters are not changed and set to default only. The steps for subscription creation is shown from figure-27 to figure-30.

Amazon SNS

ARN: arn:aws:sns:us-east-1:199496030307:a4sns

Type: Standard

Subscriptions (0) Create subscription

No subscriptions found

Create subscription

Figure 27: Create SNS subscription step-1.

Details

Topic ARN: arn:aws:sns:us-east-1:199496030307:a4sns

Protocol: Email

Endpoint: sg682034@dal.ca

After your subscription is created, you must confirm it. [Info](#)

Figure 28: Create SNS subscription step-2.

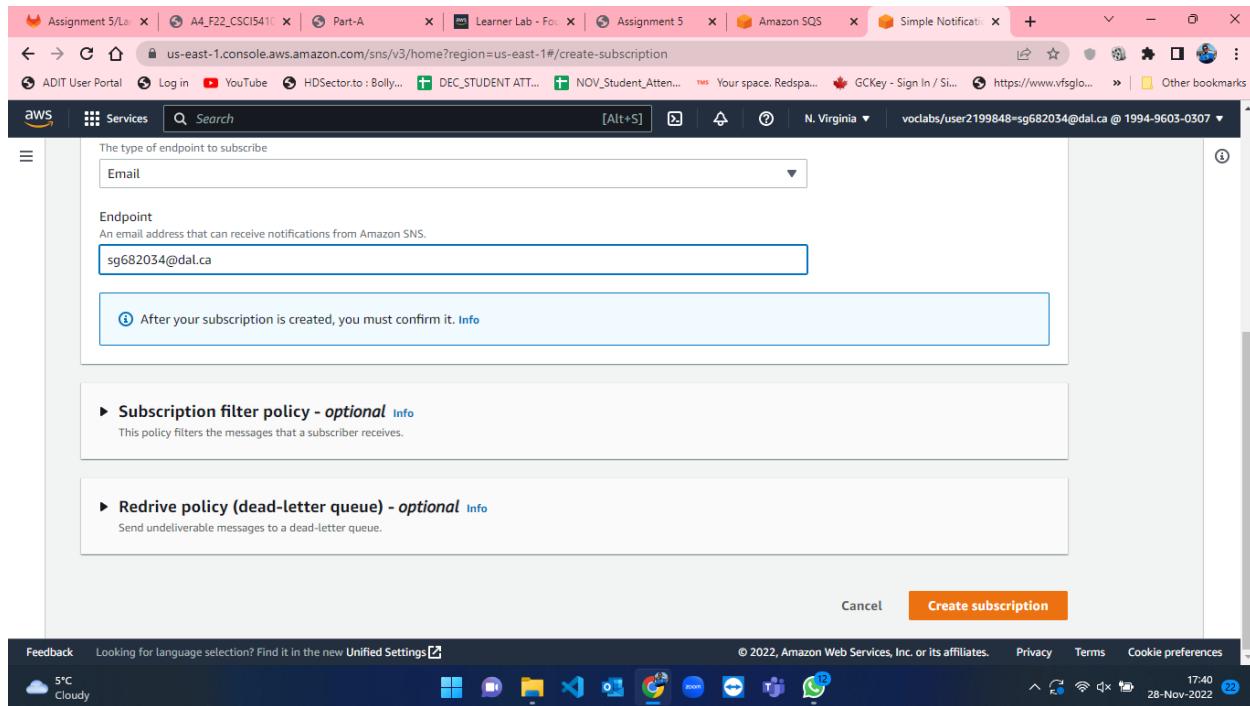


Figure 29: Create SNS subscription step-3.

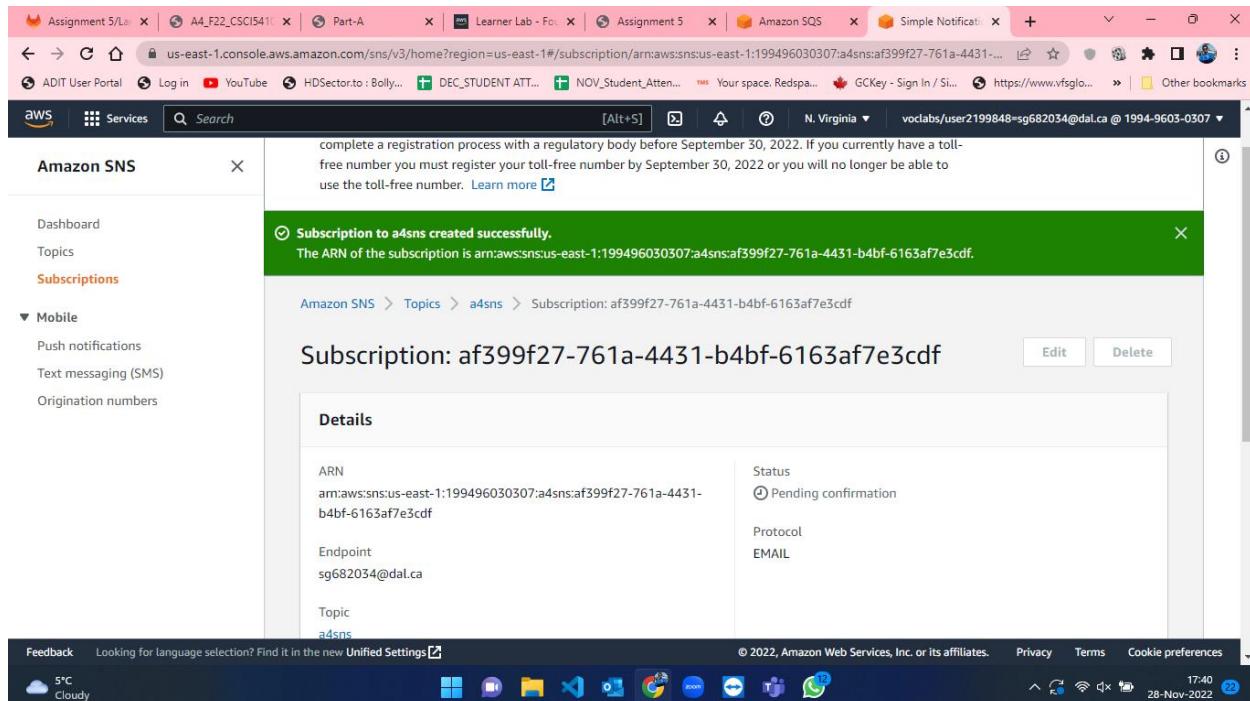


Figure 30: Successful creation of SNS subscription.

- Once the subscription is created, AWS will send the email to confirm the subscription for SNS. Click on “Confirm Subscription” link which is received in mail (figure-31 and figure-32).

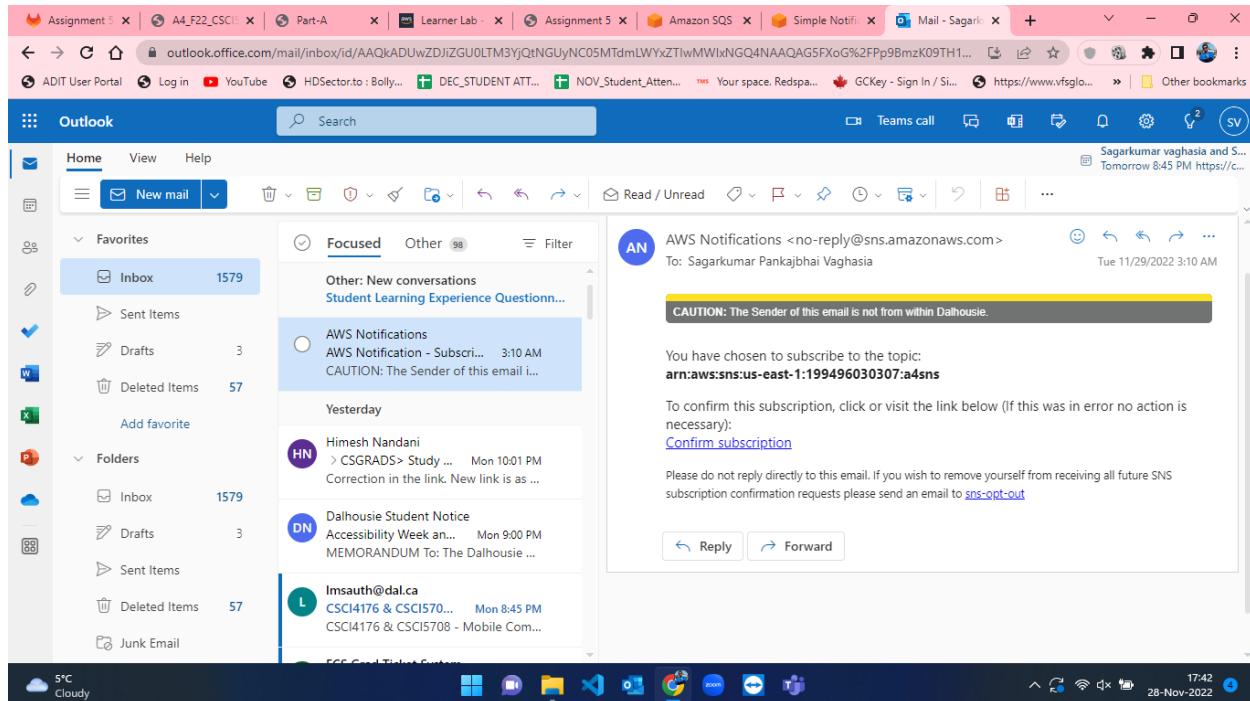


Figure 31: Confirm Subscription mail from AWS.

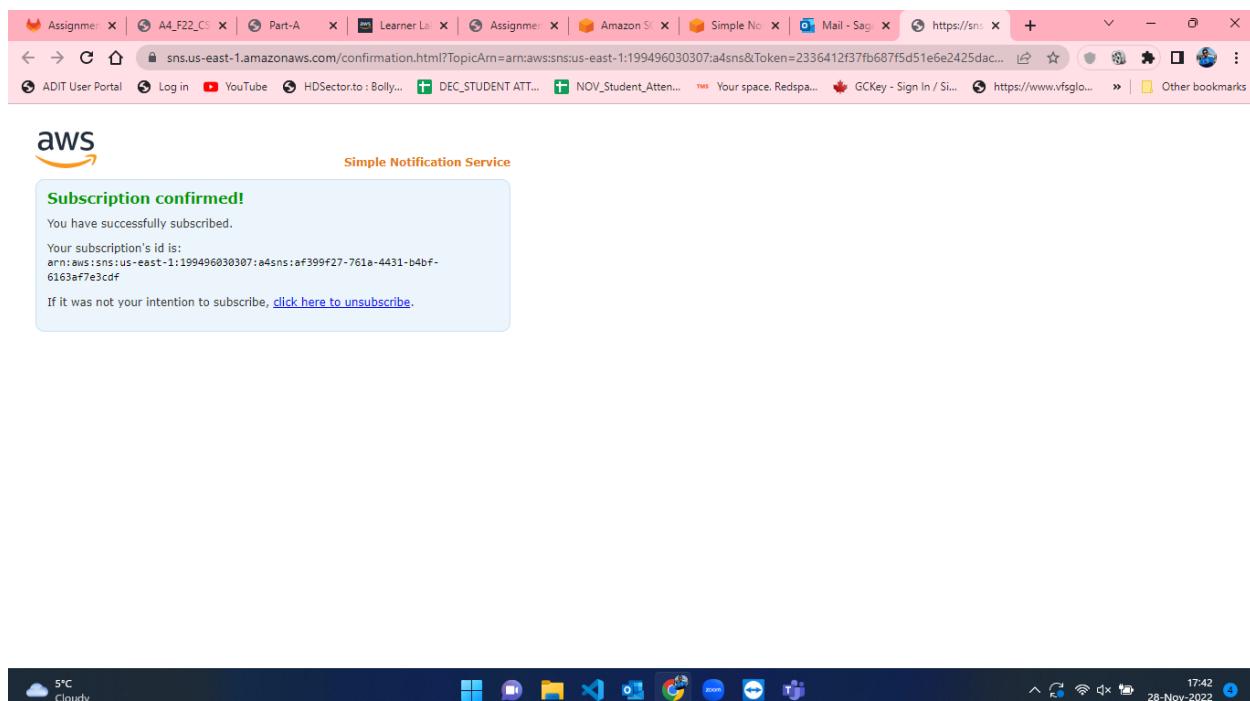


Figure 32: Subscription confirmation.

- Now, we have SQS queue where we can poll messages once it is sent also, we have SNS subscription ready with our mail. So, we have to create a lambda function which will invoke when SQS will receive a message and will send notification to the mail that we have subscribed to using SNS[6]. Therefore, I developed a lambda function and named it as “lambda-sqs” and selected “LabRole” when selecting execution role at the time of lambda function creation. The process of lambda function creation is visualized from figure-33 to figure-36.

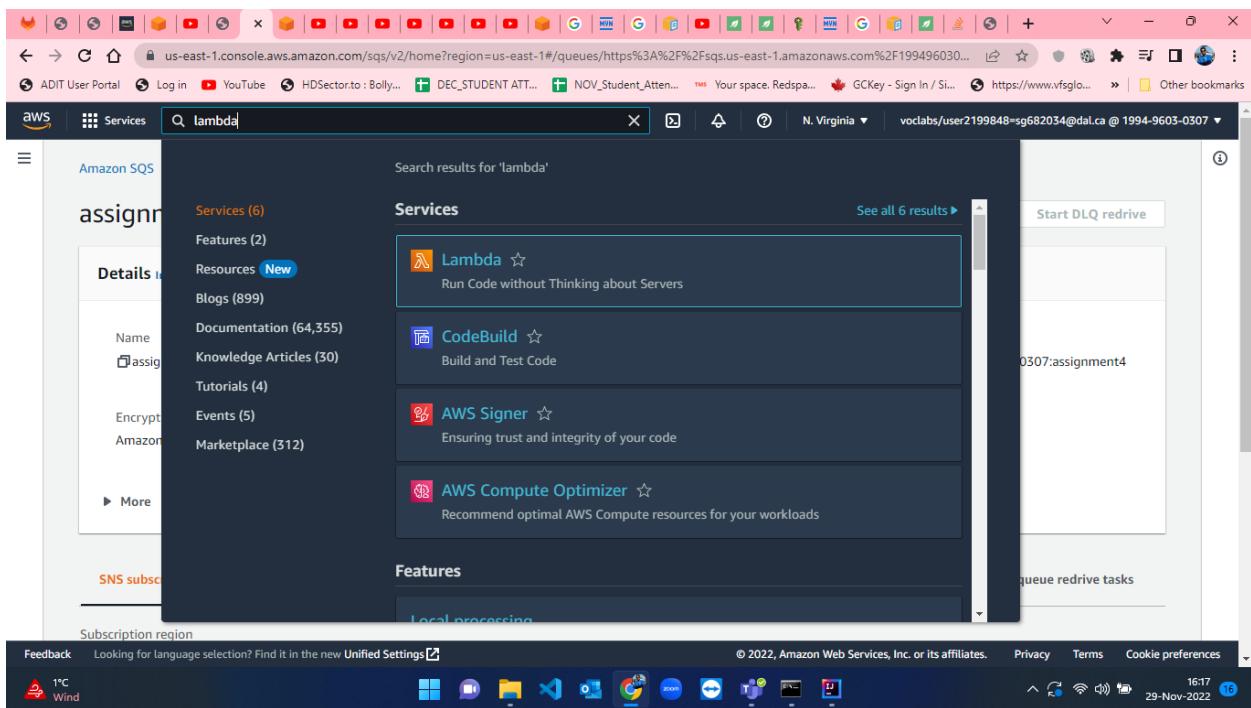


Figure 33: Lambda Function creation step-1.

AWS Lambda

Functions (3)

Function name	Description	Package type	Runtime	Last modified
LightsailMonitoringFunction	-	Zip	Python 3.8	2 months ago
MainMonitoringFunction	-	Zip	Python 3.8	2 months ago
StudentValidationLambda	-	Zip	Python 3.9	19 days ago

Feedback Looking for language selection? Find it in the new Unified Settings [\[?\]](#)

© 2022, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

16:17 29-Nov-2022

Figure 34: Lambda Function creation step-2.

Create function [\[?\]](#)

AWS Serverless Application Repository applications have moved to [Create application](#).

Author from scratch  Start with a simple Hello World example.

Use a blueprint  Build a Lambda application from sample code and configuration presets for common use cases.

Container image  Select a container image to deploy for your function.

Basic information

Function name [\[?\]](#)  
Enter a name that describes the purpose of your function.

Runtime [\[?\]](#)  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Feedback Looking for language selection? Find it in the new Unified Settings [\[?\]](#)

© 2022, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

16:17 29-Nov-2022

Figure 35: Lambda Function creation step-3.

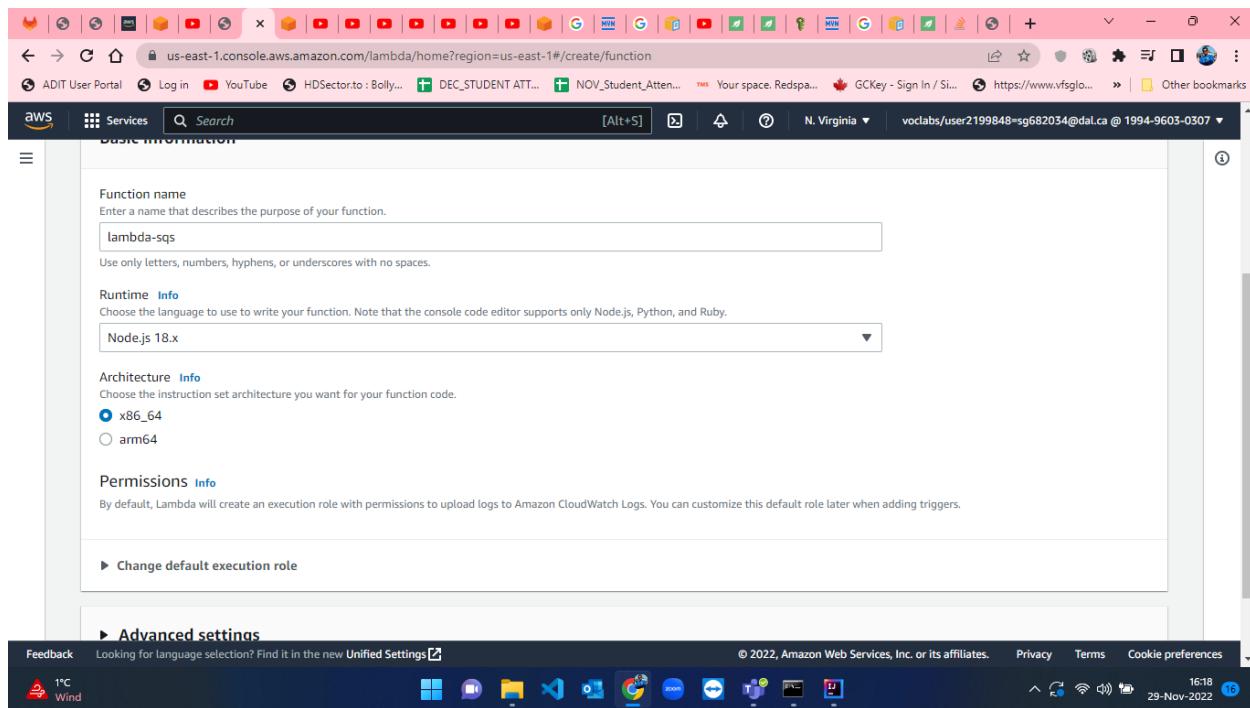


Figure 36: Lambda Function creation step-4.

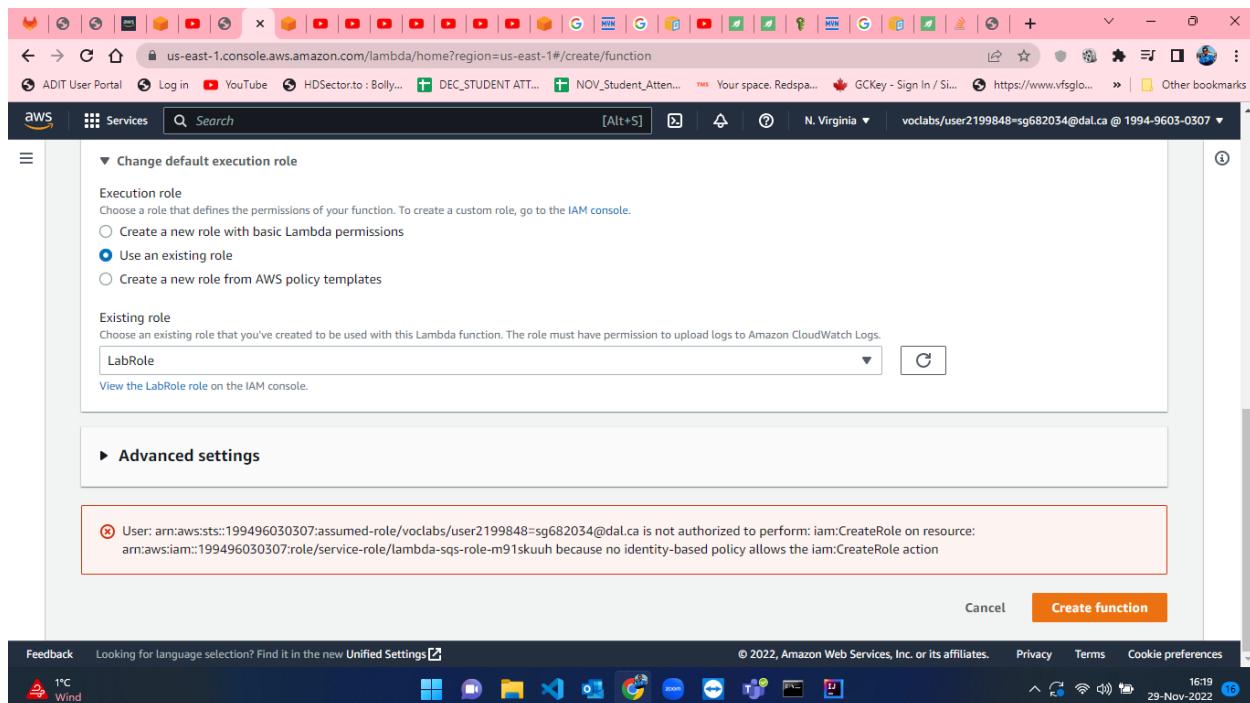


Figure 37: Lambda Function creation step-5.

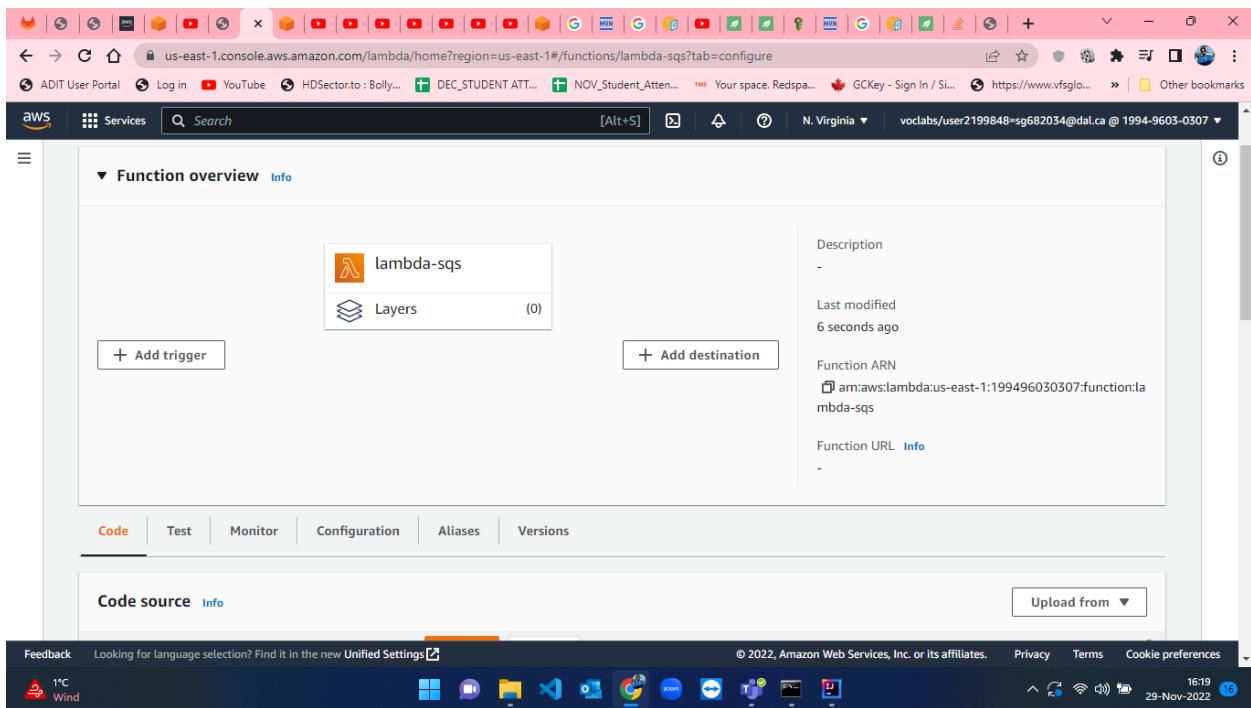


Figure 38: Lambda Function creation step-6.

- When the lambda function is created, we have to add trigger which can be done by clicking on “Add trigger” button from Function overview tab (figure-36). After that I searched for EventBridge (CloudWatch Events)[7] and I have created a trigger by setting “Create a new rule” and named that rule as “every5minutes” as we want notifications of new messages in queue after every 5 minutes. I selected “Schedule expression” and set the expression as “rate(5 minutes)”. Steps for adding trigger can be visualized in figure-37 and figure-38.

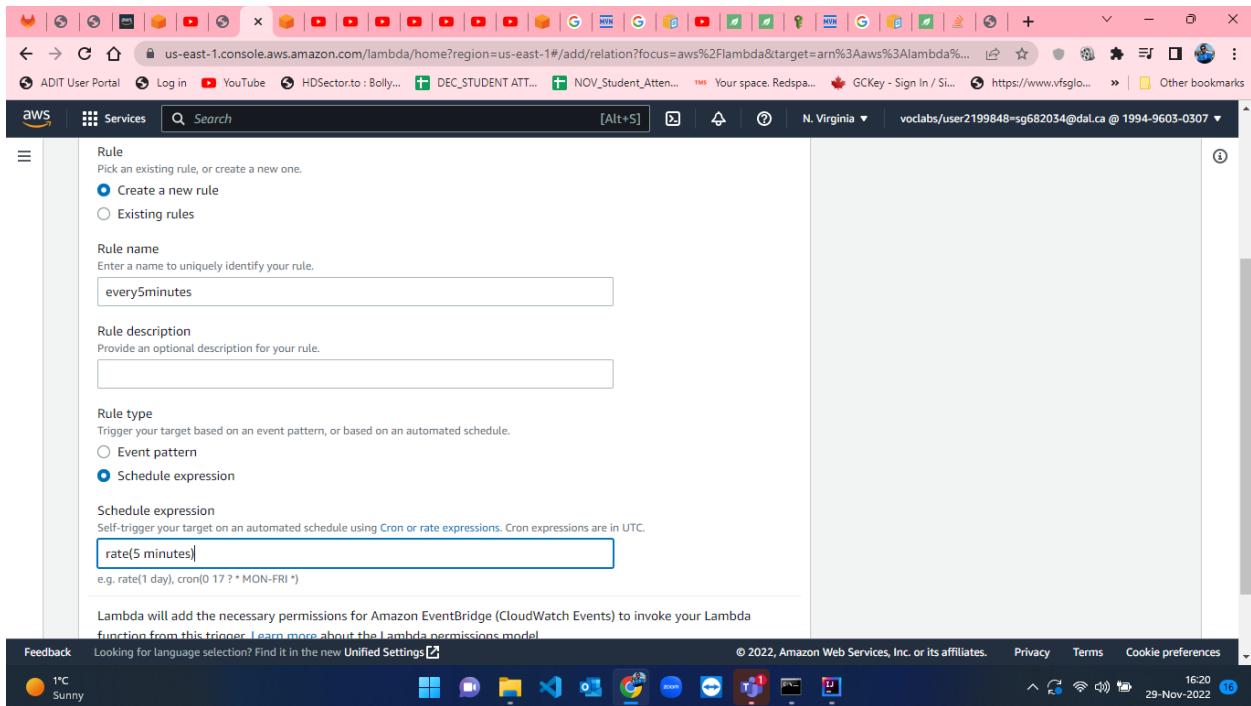


Figure 39: Adding trigger to lambda function.

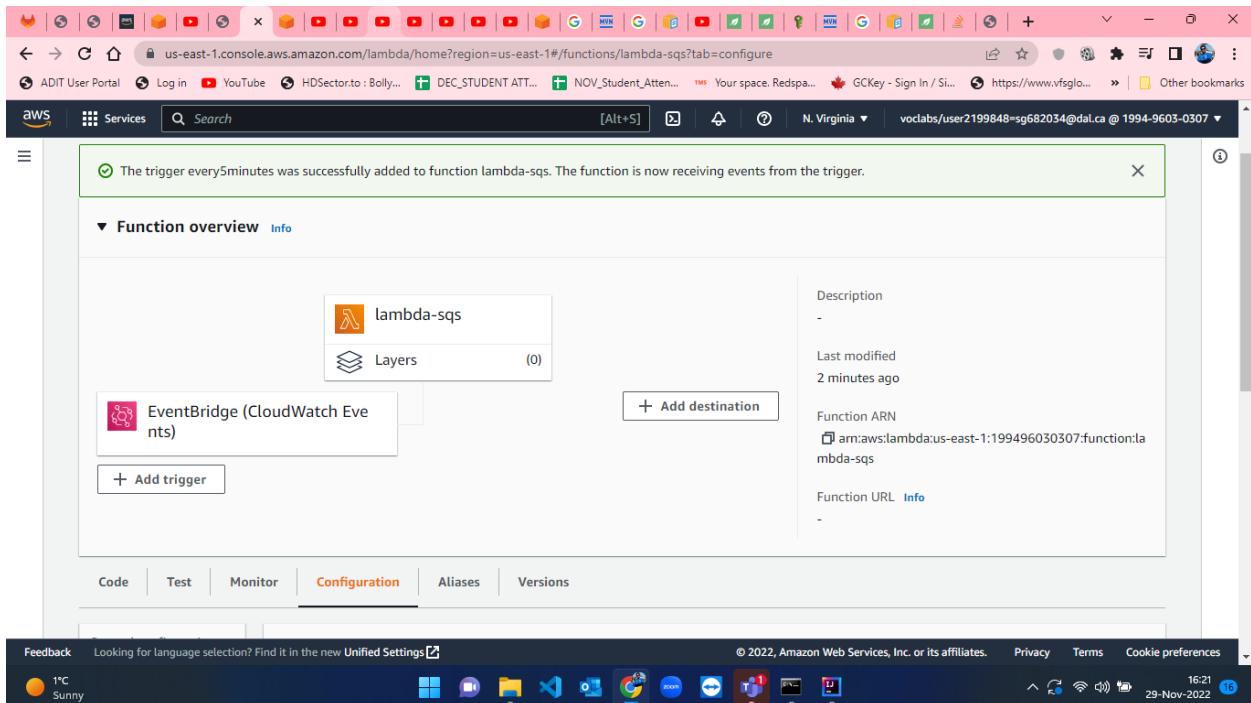
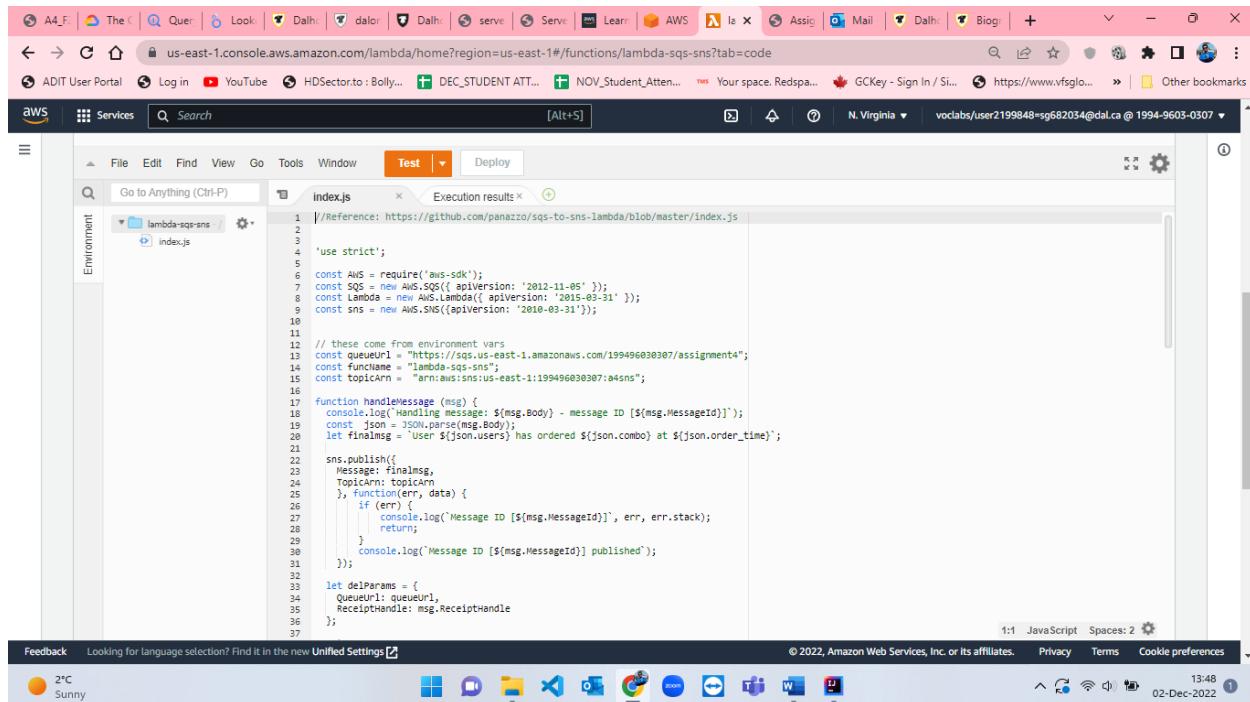


Figure 40: Successful trigger addition to lambda function.

- When the lambda function is ready with trigger, I developed a python script[8] for sending notification by SNS which is received in message inside SQS queue. The lambda code is shown from figure-41 to figure-43.

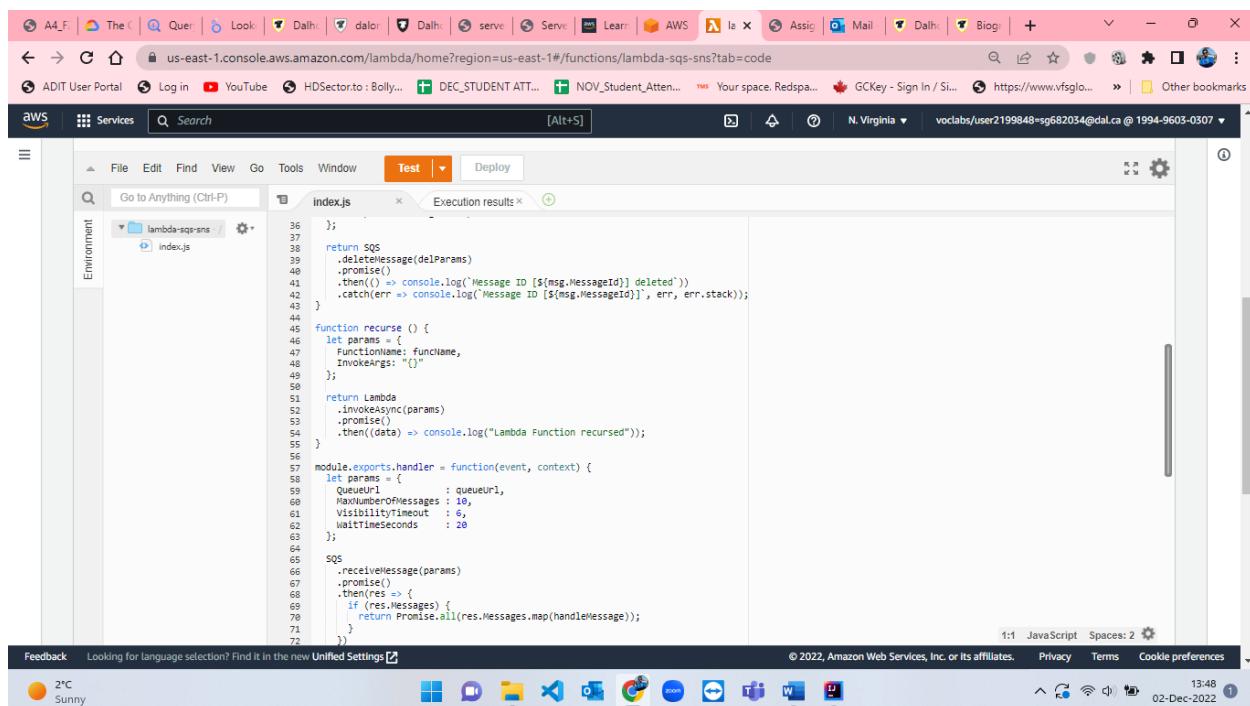


```

1 //Reference: https://github.com/panazzo/sqs-to-sns-lambda/blob/master/index.js
2
3 'use strict';
4
5 const AWS = require('aws-sdk');
6 const SQS = new AWS.SQS({ apiVersion: '2012-11-05' });
7 const Lambda = new AWS.Lambda({ apiVersion: '2015-03-31' });
8 const SNS = new AWS.SNS({ apiVersion: '2010-03-31' });
9
10
11 // these come from environment vars
12 const queueUrl = "https://sqs.us-east-1.amazonaws.com/199496030307/assignment4";
13 const funcName = "lambda-sqs-sns";
14 const topicArn = "arn:aws:sns:us-east-1:199496030307:84sns";
15
16
17 function handleMessage (msg) {
18     console.log(`Handling message: ${msg.Body} - message ID ${msg.MessageId}`);
19     const json = JSON.parse(msg.Body);
20     let finding = User.$(json.users).has_ordered ${json.combo} at ${json.order_time};
21
22     sns.publish({
23         Message: finalmsg,
24         TopicArn: topicArn
25     }, function(err, data) {
26         if (err) {
27             console.log(`Message ID ${msg.MessageId} published`);
28             return;
29         }
30         console.log(`Message ID ${msg.MessageId} published`);
31     });
32
33     let delParams = {
34         QueueUrl: queueUrl,
35         ReceiptHandle: msg.ReceiptHandle
36     };
37
38     return SQS
39     .deleteMessage(delParams)
40     .promise()
41     .then(() => console.log(`Message ID ${msg.MessageId} deleted`))
42     .catch(err => console.log(`Message ID ${msg.MessageId} , err, err.stack`));
43 }
44
45 function recurse () {
46     let params = {
47         FunctionName: funcName,
48         InvokeArgs: "{}"
49     };
50
51     return Lambda
52     .invoke(params)
53     .promise()
54     .then((data) => console.log("Lambda Function recursed"));
55 }
56
57 module.exports.handler = function(event, context) {
58     let params = {
59         QueueUrl: queueUrl,
60         MaxNumberOfMessages: 10,
61         VisibilityTimeout: 6,
62         WaitTimeSeconds: 20
63     };
64
65     SQS
66     .receiveMessage(params)
67     .promise()
68     .then(res => {
69         if (res.Messages) {
70             return Promise.all(res.Messages.map(handleMessage));
71         }
72     })
73 }

```

Figure 41: lambda function part-1.

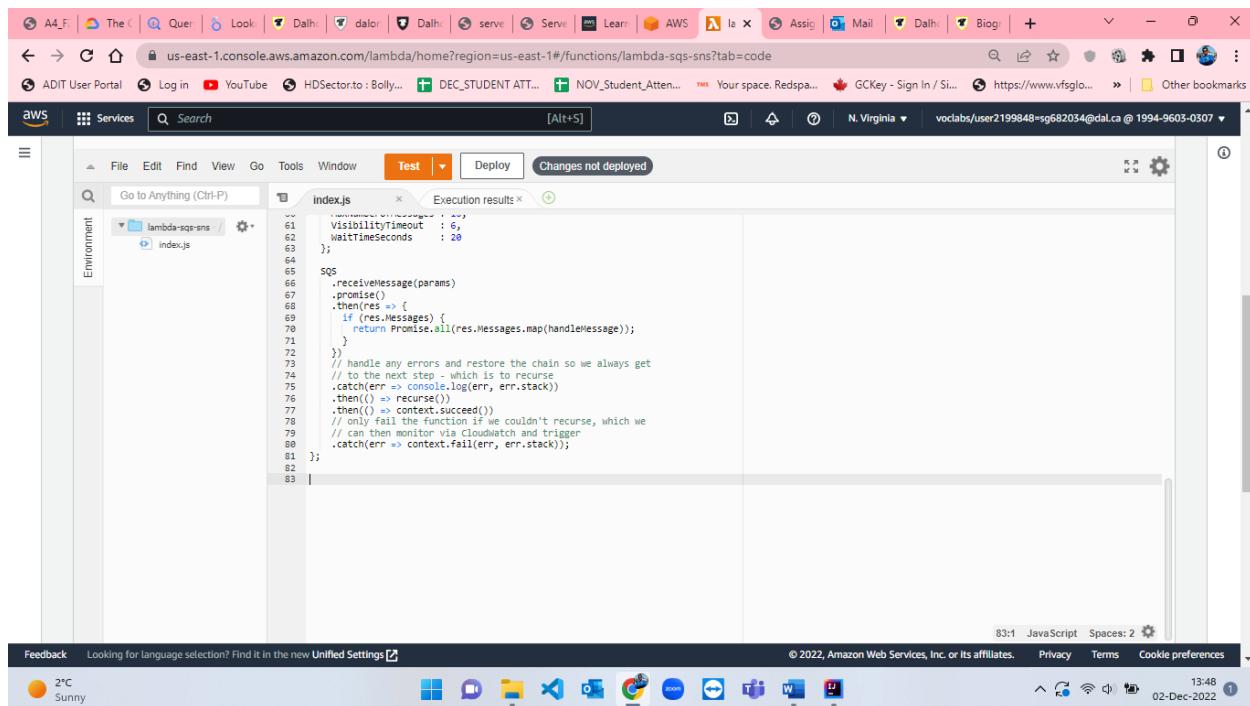


```

36     });
37
38     return SQS
39     .deleteMessage(delParams)
40     .promise()
41     .then(() => console.log(`Message ID ${msg.MessageId} deleted`))
42     .catch(err => console.log(`Message ID ${msg.MessageId} , err, err.stack`));
43 }
44
45 function recurse () {
46     let params = {
47         FunctionName: funcName,
48         InvokeArgs: "{}"
49     };
50
51     return Lambda
52     .invoke(params)
53     .promise()
54     .then((data) => console.log("Lambda Function recursed"));
55 }
56
57 module.exports.handler = function(event, context) {
58     let params = {
59         QueueUrl: queueUrl,
60         MaxNumberOfMessages: 10,
61         VisibilityTimeout: 6,
62         WaitTimeSeconds: 20
63     };
64
65     SQS
66     .receiveMessage(params)
67     .promise()
68     .then(res => {
69         if (res.Messages) {
70             return Promise.all(res.Messages.map(handleMessage));
71         }
72     })
73 }

```

Figure 42: lambda function part-2.



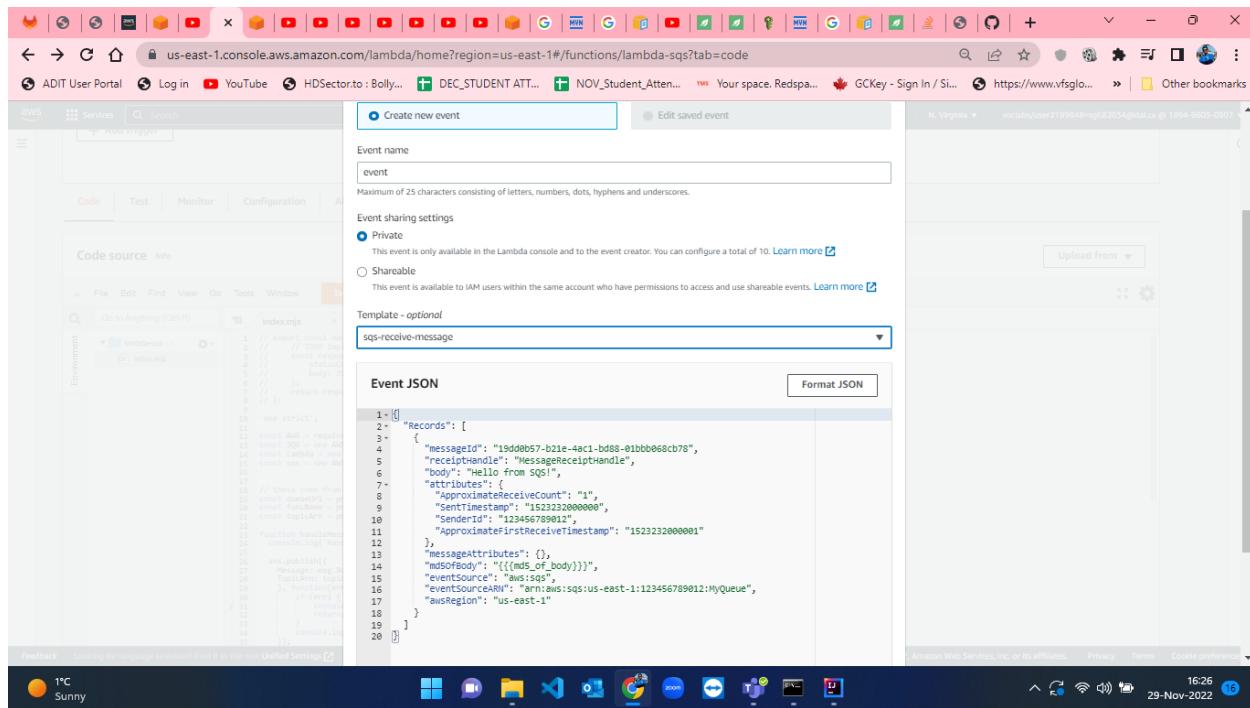
```

61  VisibilityTimeout : 6,
62  WaitTimeSeconds : 20
63  );
64
65  SQS
66  .receiveMessage(params)
67  .promise()
68  .then(res => {
69    if (res.Messages) {
70      return Promise.all(res.Messages.map(handleMessage));
71    }
72  })
73  // handle any errors and restore the chain so we always get
74  // to the next step - which is to recurse
75  .catch(err => console.log(err, err.stack))
76  .then(() => recursive())
77  .then(() => context.succeed())
78  // only fail the function if we couldn't recurse, which we
79  // can then monitor via CloudWatch and trigger
80  .catch(err => context.fail(err, err.stack));
81 }
82
83

```

Figure 43: lambda function part-3.

- After development of code, I have created a Test environment for testing my code by clicking on “Test” button besides Deploy button which can be seen in figure-44 and named the event as “event” where I set the event sharing settings to “Private” and opted for template as “sqS-receive-message”.



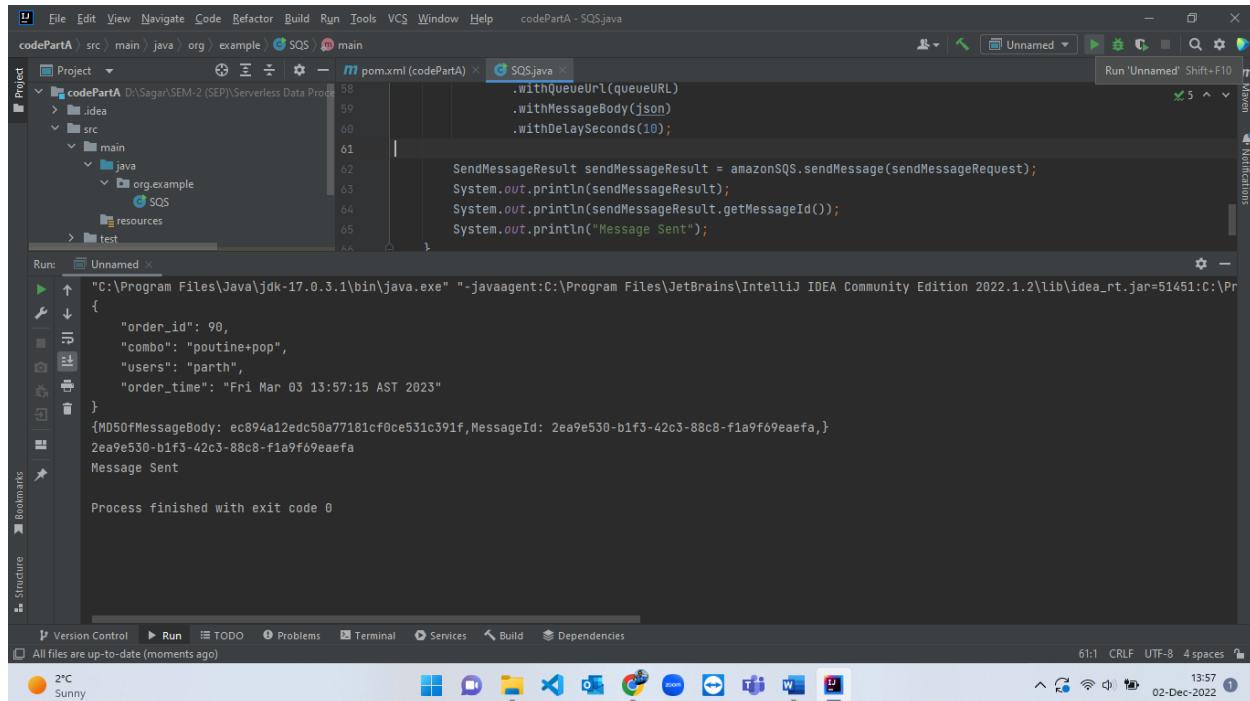
```

1: [
2:   {
3:     "Records": [
4:       {
5:         "messageId": "10dd08e7-b21e-4e11-bd88-01bbb068cb78",
6:         "receiptHandle": "MessageReceiptHandle",
7:         "body": "Hello from SQS",
8:         "attributes": {
9:             "approximateReceiveCount": "1",
10:             "sentTimestamp": "1523232000000",
11:             "SenderId": "123456789012",
12:             "approximateFirstReceiveTimestamp": "1523232000001"
13:         },
14:         "messageAttributes": {},
15:         "md5OfBody": "({md5_of_body})",
16:         "eventsSource": "aws:sqs",
17:         "eventsSourceARN": "arn:aws:sqs:us-east-1:123456789012:MyQueue",
18:         "awsRegion": "us-east-1"
19:     }
20:   ]
21: ]

```

Figure 44: Event creation to test the lambda function.

- Afterwards, we have to test the lambda function and deploy the lambda function. After deployment, I have created 3 orders from the JAVA code randomly and sent it to SQS queue which will send notification to email in every 5 minutes. This is represented from figure-45 to figure-50.



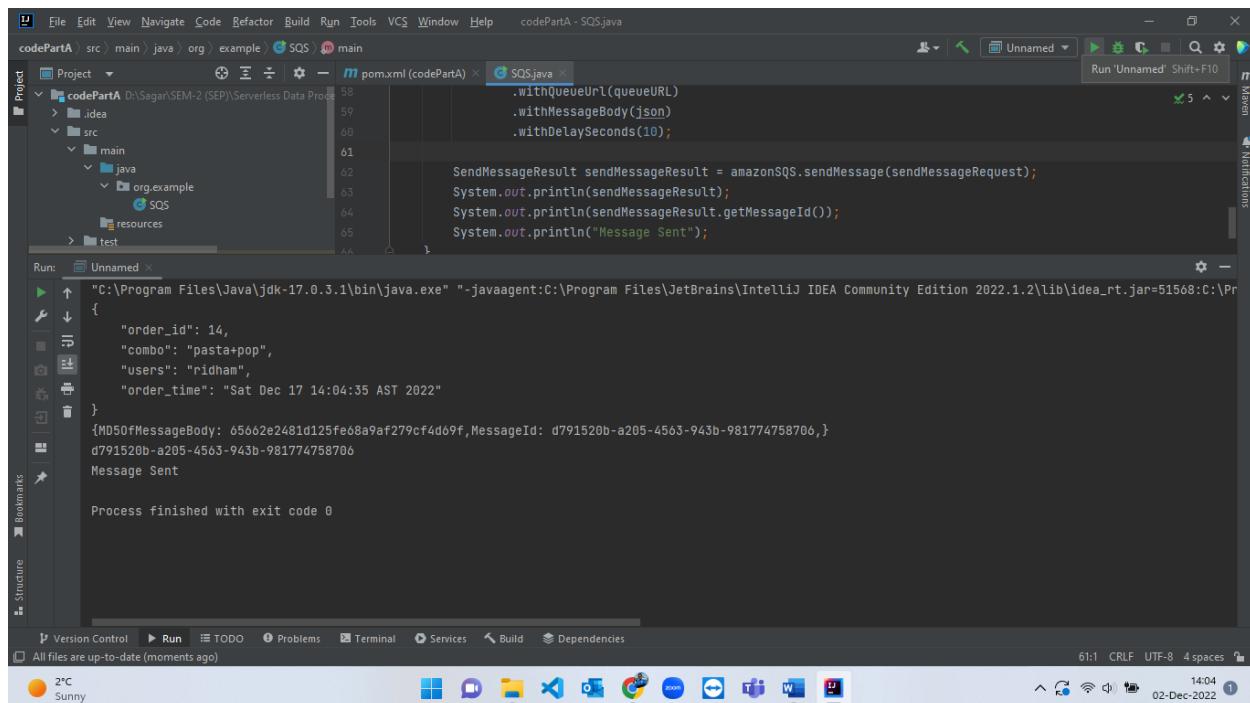
```

codePartA src main java org example SQS main
codePartA pom.xml (codePartA) SQS.java
  .withQueueUrl(queueURL)
  .withMessageBody(json)
  .withDelaySeconds(10);

SendMessageResult sendMessageResult = amazonSQS.sendMessage(sendMessageRequest);
System.out.println(sendMessageResult);
System.out.println(sendMessageResult.getMessageId());
System.out.println("Message Sent");

Process finished with exit code 0
  
```

Figure 45: Random order creation - 1.



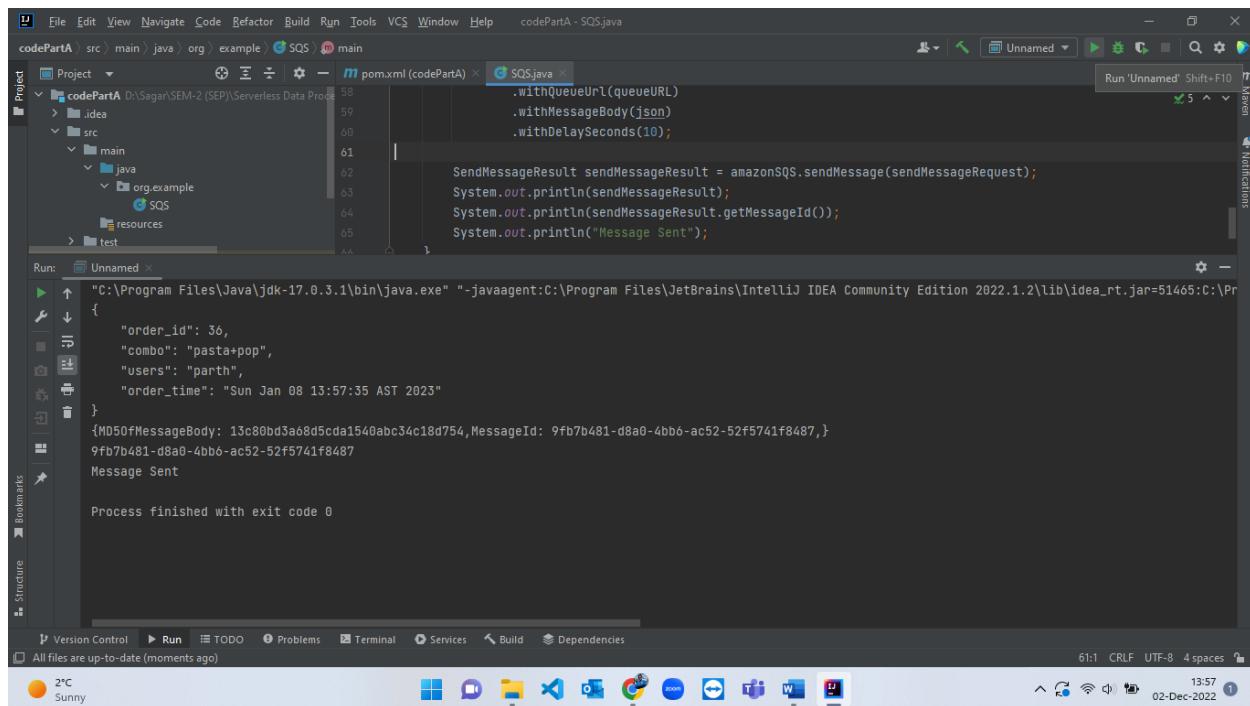
```

codePartA src main java org example SQS main
codePartA pom.xml (codePartA) SQS.java
  .withQueueUrl(queueURL)
  .withMessageBody(json)
  .withDelaySeconds(10);

SendMessageResult sendMessageResult = amazonSQS.sendMessage(sendMessageRequest);
System.out.println(sendMessageResult);
System.out.println(sendMessageResult.getMessageId());
System.out.println("Message Sent");

Process finished with exit code 0
  
```

Figure 46: Random order creation - 2.



```

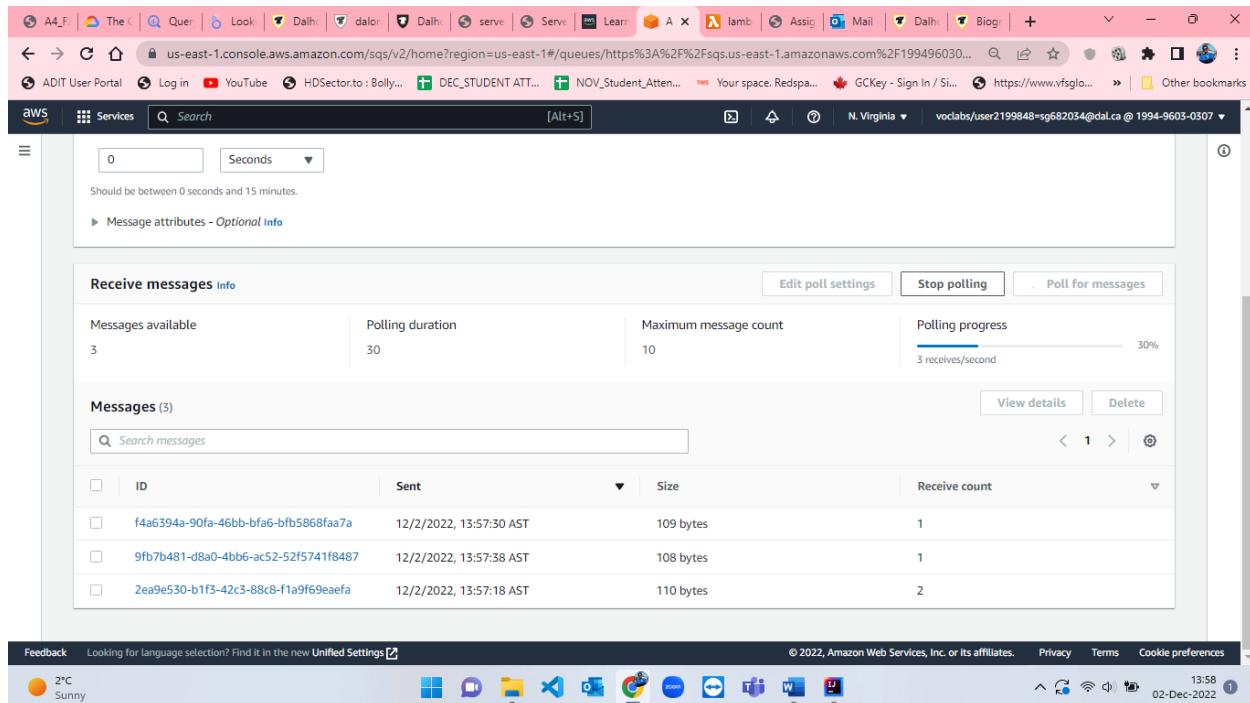
codePartA > src > main > java > org > example > SQS > main
Project pom.xml (codePartA) SQS.java
  .withQueueUrl(queueURL)
  .withMessageBody(json)
  .withDelaySeconds(10);

SendMessageResult sendMessageResult = amazonSQS.sendMessage(sendMessageRequest);
System.out.println(sendMessageResult);
System.out.println(sendMessageResult.getMessageId());
System.out.println("Message Sent");

Run Unnamed
" C:\Program Files\Java\jdk-17.0.3.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.1.2\lib\idea_rt.jar=51465:C:\Program Files\Java\jdk-17.0.3.1\bin" -Dfile.encoding=UTF-8 -classpath "C:\Program Files\Java\jdk-17.0.3.1\lib\charsets.jar;C:\Program Files\Java\jdk-17.0.3.1\lib\rt.jar;C:\Program Files\Java\jdk-17.0.3.1\lib\java.base.jar;C:\Program Files\Java\jdk-17.0.3.1\lib\java.logging.jar;C:\Program Files\Java\jdk-17.0.3.1\lib\java.prefs.jar;C:\Program Files\Java\jdk-17.0.3.1\lib\java.xml.jar;C:\Program Files\Java\jdk-17.0.3.1\lib\management-agent.jar" org.example.SQS
{
    "order_id": 36,
    "combo": "pasta+pop",
    "users": "parth",
    "order_time": "Sun Jan 08 13:57:35 AST 2023"
}
{MD50fMessageBody: 13c80bd3a68d5cda1540abc34c18d754,MessageId: 9fb7b481-d8a0-4bb6-ac52-52f5741f8487,}
9fb7b481-d8a0-4bb6-ac52-52f5741f8487
Message Sent

Process finished with exit code 0
  
```

Figure 47: Random order creation - 3.



ID	Sent	Size	Receive count
f4a6394a-90fa-46bb-bfa6-bfb5868faa7a	12/2/2022, 13:57:30 AST	109 bytes	1
9fb7b481-d8a0-4bb6-ac52-52f5741f8487	12/2/2022, 13:57:38 AST	108 bytes	1
2ea9e530-b1f3-42c3-88c8-f1a9f69eaefa	12/2/2022, 13:57:18 AST	110 bytes	2

Figure 48: 3 orders messages polled in SQS queue.

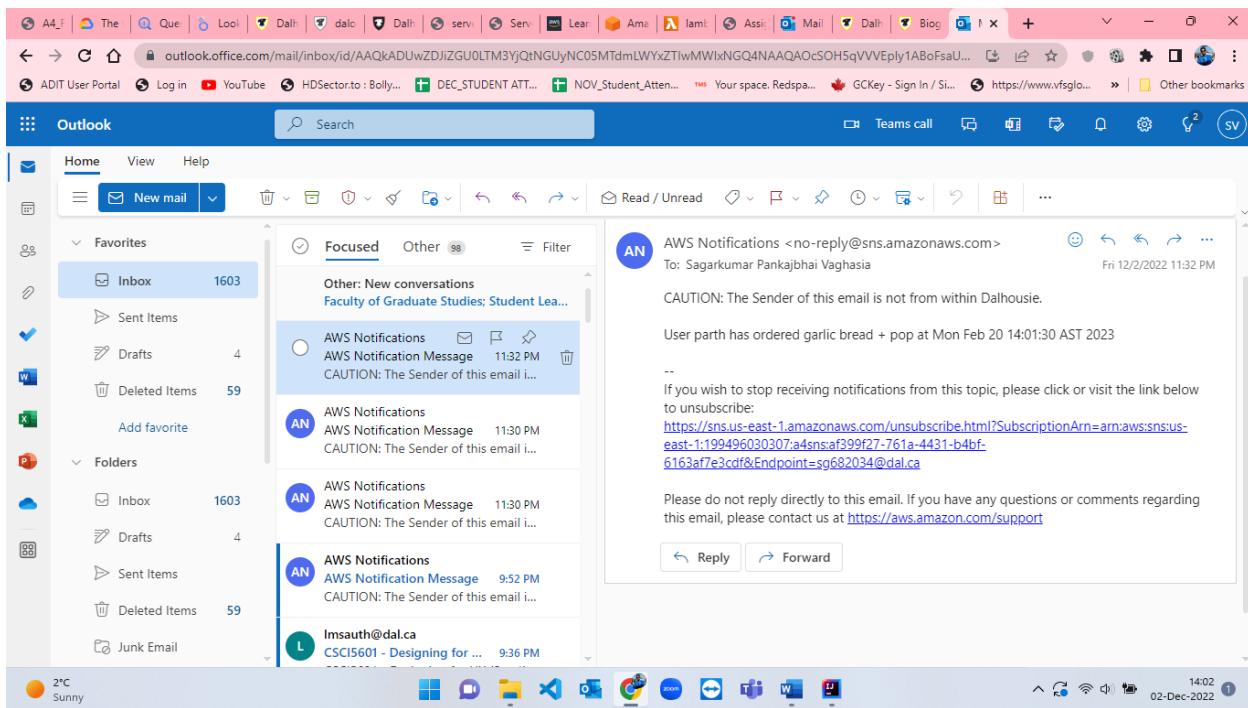


Figure 49: Email notification from SNS - 1.

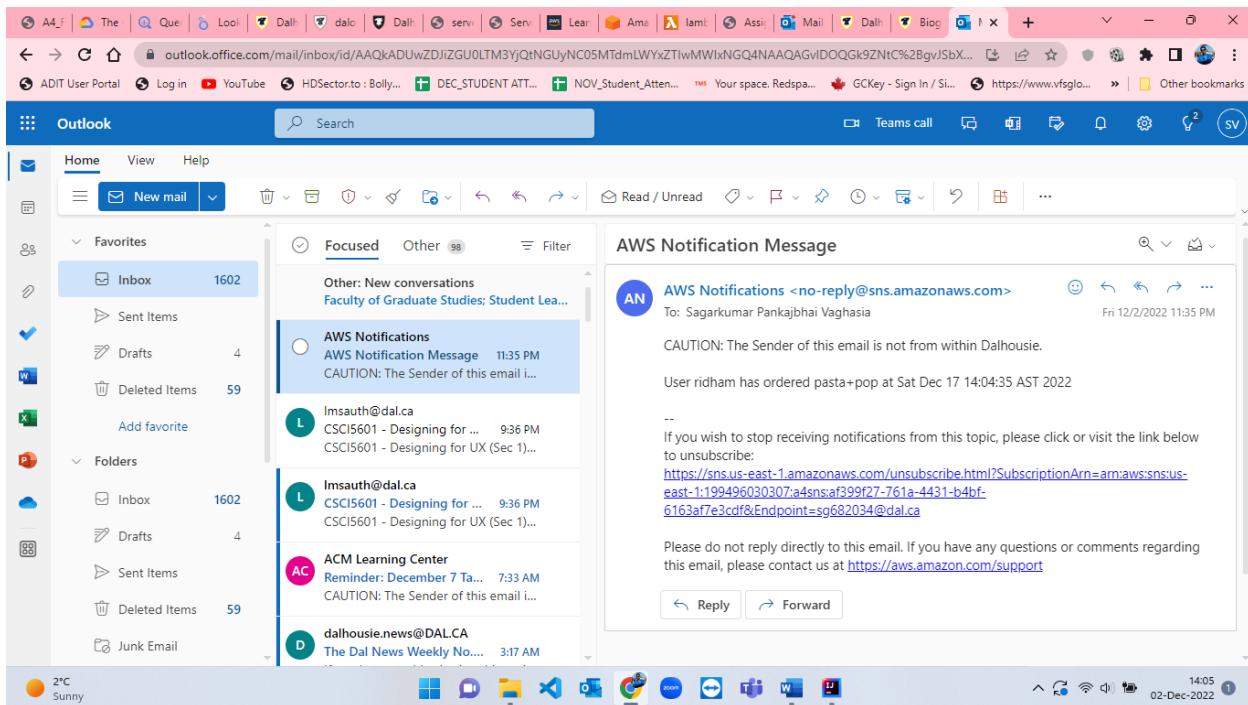


Figure 50: Email notification from SNS - 2.

## Code:

### SQS.java

```

package org.example;

import com.amazonaws.auth.*;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.SendMessageRequest;
import com.amazonaws.services.sqs.model.SendMessageResult;
import java.lang.*;
import java.util.Calendar;
import java.util.Date;

//Reference: https://docs.aws.amazon.com/sdk-for-java/v1/developer-
guide/examples-sqs-messages.html
//Reference: https://www.youtube.com/watch?v=_C_SykraCVk
public class SQS {
    public static void main(String[] args) {

        //Reference: https://www.baeldung.com/java-system-get-property-vs-
        system.getenv
        String AWS_ACCESS_KEY = System.getenv("AWS_ACCESS_KEY");
        String AWS_SECRET_KEY = System.getenv("AWS_SECRET_KEY");
        String AWS_SESSION_TOKEN = System.getenv("AWS_SESSION_TOKEN");
        String queueURL = System.getenv("queueURL");

        String json = "{\n\t";
        String[] combo = {"pasta+pop", "pizza+pop", "garlic bread + pop",
        "poutine+pop"};
        String[] users = {"sagar", "parth", "ridham", "juhil"};

        //Reference: https://docs.aws.amazon.com/sdk-for-java/v1/developer-
        guide/credentials.html
        //Reference:
        https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/auth/Basi
        cSessionCredentials.html
        BasicSessionCredentials credentials = new BasicSessionCredentials(
            AWS_ACCESS_KEY,
            AWS_SECRET_KEY,
            AWS_SESSION_TOKEN
        );

        //Reference: https://www.educative.io/answers/how-to-generate-random-
        numbers-in-java
        double d1 = Math.random() * 5;
        double d2 = Math.random() * 4;
        double d3 = Math.random() * 100;

        int r1 = (int) d1;
        int r2 = (int) d2;
        int order_id = (int) d3;
    }
}

```

```

//Reference: https://www.geeksforgeeks.org/calendar-settime-method-in-java-with-examples/
Calendar calendar = Calendar.getInstance();
calendar.setTime(new Date());
calendar.add(Calendar.DATE, (int) d3 + 1);

json += "\"order_id\": " + order_id + ",";
json += "\n\t\"combo\": \"\" + combo[r1] + "\",";
json += "\n\t\"users\": \"\" + users[r2] + "\",";
json += "\n\t\"order_time\": \"\" + calendar.getTime() + "\"";
json += "\n}";

System.out.println(json);

//Reference:
https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/services/sqs/AmazonSQS.html#deleteQueue-
com.amazonaws.services.sqs.model.DeleteQueueRequest-
AmazonSQS amazonSQS = AmazonSQSClientBuilder.standard()
    .withRegion(Regions.DEFAULT_REGION)
    .withCredentials(new
AWSStaticCredentialsProvider(credentials))
    .build();

//Refernece:
https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/services/sqs/model/SendMessageRequest.html
SendMessageRequest sendMessageRequest = new SendMessageRequest()
    .withQueueUrl(queueURL)
    .withMessageBody(json)
    .withDelaySeconds(10);

//Reference:
https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/services/sqs/model/SendMessageResult.html
SendMessageResult sendMessageResult =
amazonSQS.sendMessage(sendMessageRequest);
System.out.println(sendMessageResult);
System.out.println(sendMessageResult.getMessageId());
System.out.println("Message Sent");
}
}

```

## pom.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>codePartA</artifactId>
    <version>1.0-SNAPSHOT</version>

```

```

<properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
</properties>

<dependencies>
    <dependency>
        <groupId>com.googlecode.json-simple</groupId>
        <artifactId>json-simple</artifactId>
        <version>1.1.1</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk -->
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk</artifactId>
        <version>1.12.350</version>
    </dependency>
</dependencies>

</project>

```

### Lambda function : index.js

```

//Reference: https://github.com/panazzo/sqs-to-sns-
lambda/blob/master/index.js

'use strict';

const AWS = require('aws-sdk');
const SQS = new AWS.SQS({ apiVersion: '2012-11-05' });
const Lambda = new AWS.Lambda({ apiVersion: '2015-03-31' });
const sns = new AWS.SNS({apiVersion: '2010-03-31'});

const queueUrl = "https://sqs.us-east-
1.amazonaws.com/199496030307/assignment4";
const funcName = "lambda-sqs-sns";
const topicArn = "arn:aws:sns:us-east-1:199496030307:a4sns";

function handleMessage (msg) {
    console.log(`Handling message: ${msg.Body} - message ID
[${msg.MessageId}]`);
    const json = JSON.parse(msg.Body);
    let finalmsg = `User ${json.users} has ordered ${json.combo} at
${json.order_time}`;

    sns.publish({
        Message: finalmsg,
        TopicArn: topicArn
    }, function(err, data) {
        if (err) {
            console.log(`Message ID [${msg.MessageId}]`, err, err.stack);
            return;
        }
        console.log(`Message ID [${msg.MessageId}] published`);
    });
}

```

```

});;

let delParams = {
  QueueUrl: queueUrl,
  ReceiptHandle: msg.ReceiptHandle
};

return SQS
  .deleteMessage(delParams)
  .promise()
  .then(() => console.log(`Message ID [${msg.MessageId}] deleted`))
  .catch(err => console.log(`Message ID [${msg.MessageId}]`, err,
err.stack));
}

function recurse () {
let params = {
  FunctionName: funcName,
  InvokeArgs: "{ }"
};

return Lambda
  .invokeAsync(params)
  .promise()
  .then((data) => console.log("Lambda Function recursed"));
}

module.exports.handler = function(event, context) {
let params = {
  QueueUrl : queueUrl,
  MaxNumberOfMessages : 10,
  VisibilityTimeout : 6,
  WaitTimeSeconds : 20
};

SQS
  .receiveMessage(params)
  .promise()
  .then(res => {
    if (res.Messages) {
      return Promise.all(res.Messages.map(handleMessage));
    }
  })
  // handle any errors and restore the chain so we always get
  // to the next step - which is to recurse
  .catch(err => console.log(err, err.stack))
  .then(() => recurse())
  .then(() => context.succeed())
  // only fail the function if we couldn't recurse, which we
  // can then monitor via CloudWatch and trigger
  .catch(err => context.fail(err, err.stack));
};
}

```

## PART-B: Use GCP BigQueryML.

- Firstly, I have read and understood the document of GCP BigQueryML [9] provided in the assignment which helped to create the KMeans model and using the information from that document I learned to develop the model and trained it for the given dataset. Figure 1 shows the document on GCP BigQueryML[9].

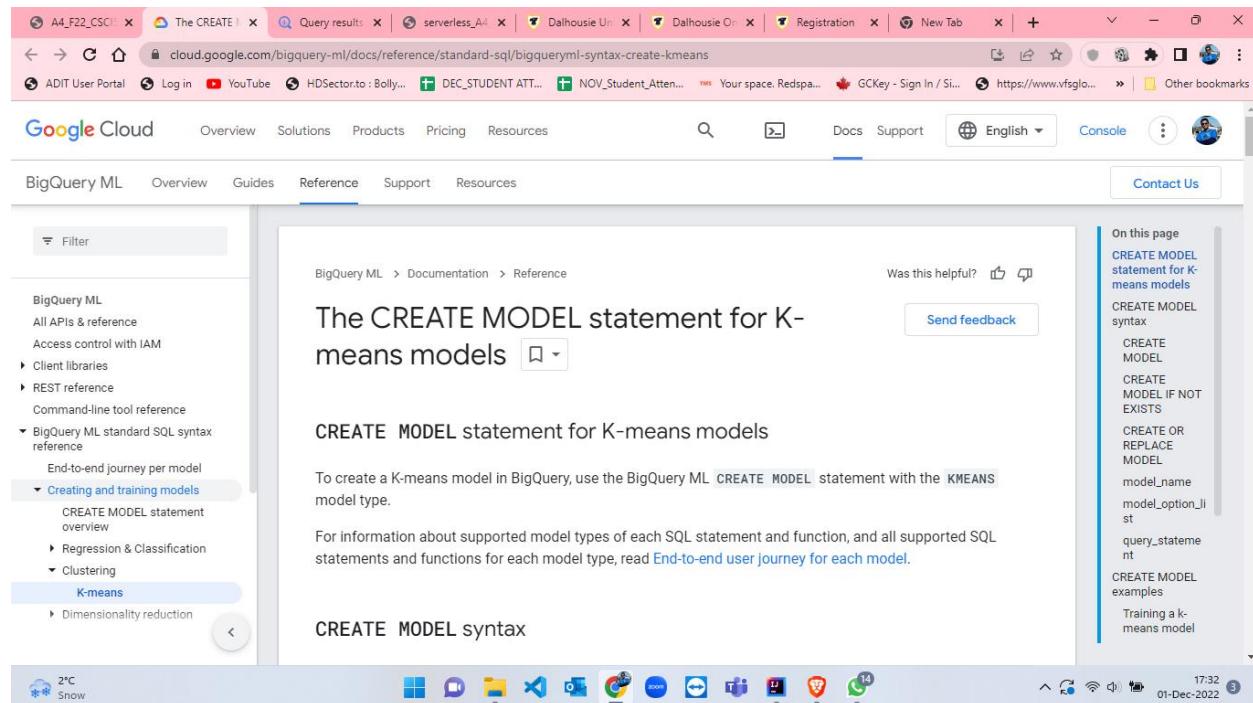


Figure 51: GCP BigQueryML documentation.

- After reading the documentation[9], I went to GCP and from that I clicked on “Go to console”. In that, I have created a new project and named as “serverless-assignment-4”. After that I searched for BigQuery from the search bar which took me to the homepage of BigQuery and selected the newly created project. This is shown from figures 52 to 59.

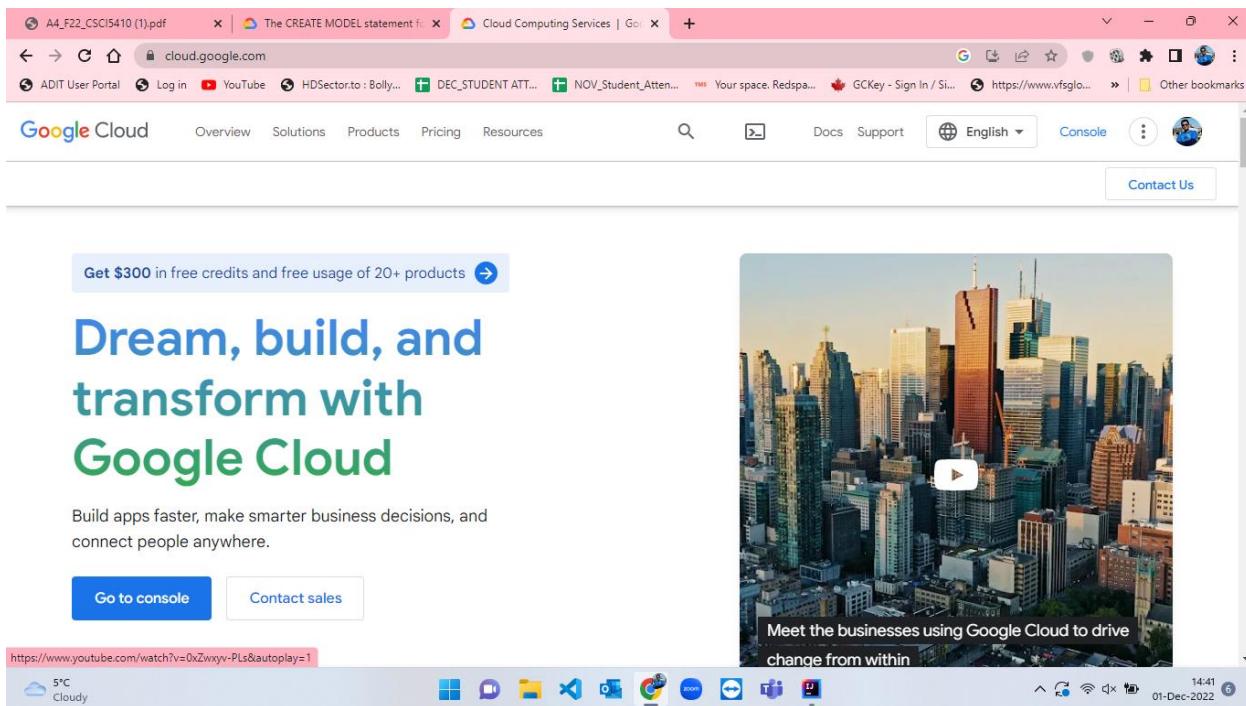


Figure 52: GCP homepage.

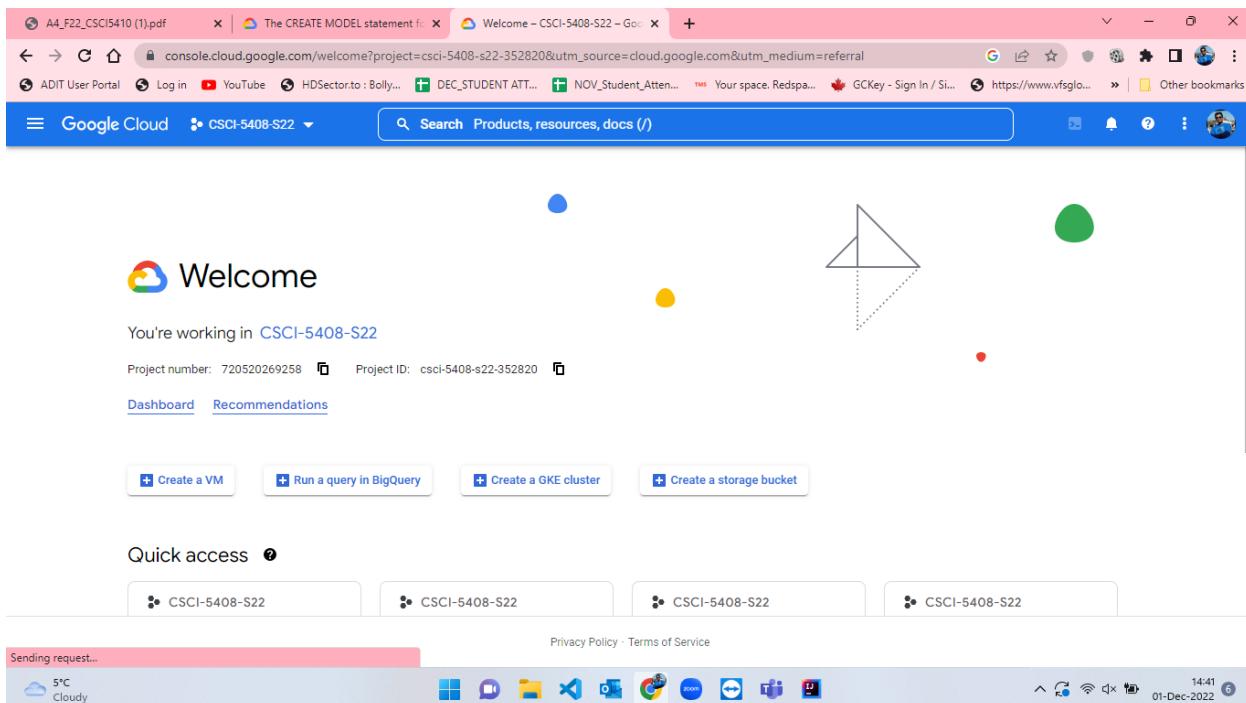


Figure 53: GCP console page.

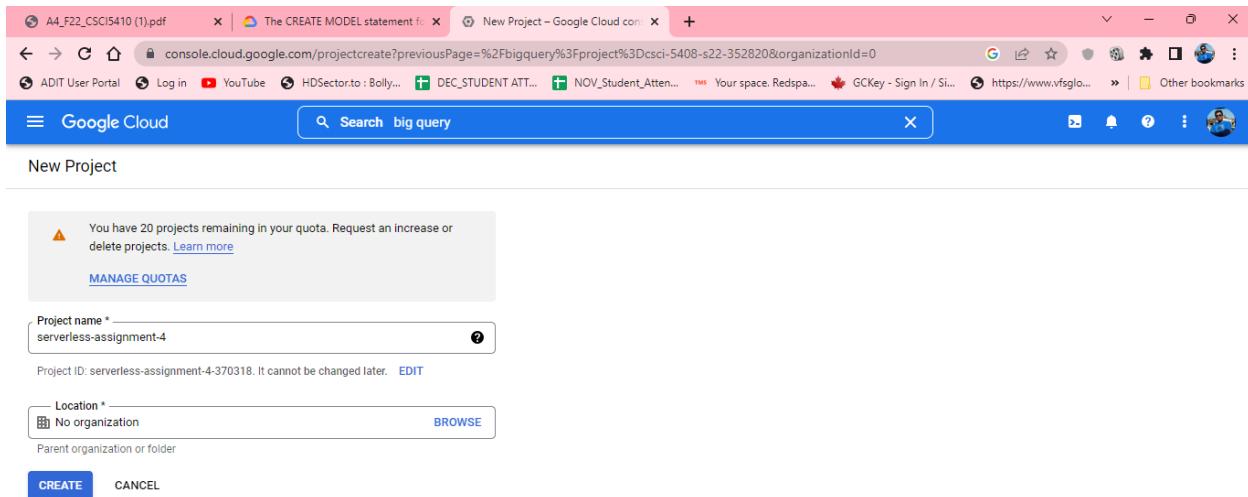


Figure 54: Project creation in GCP.

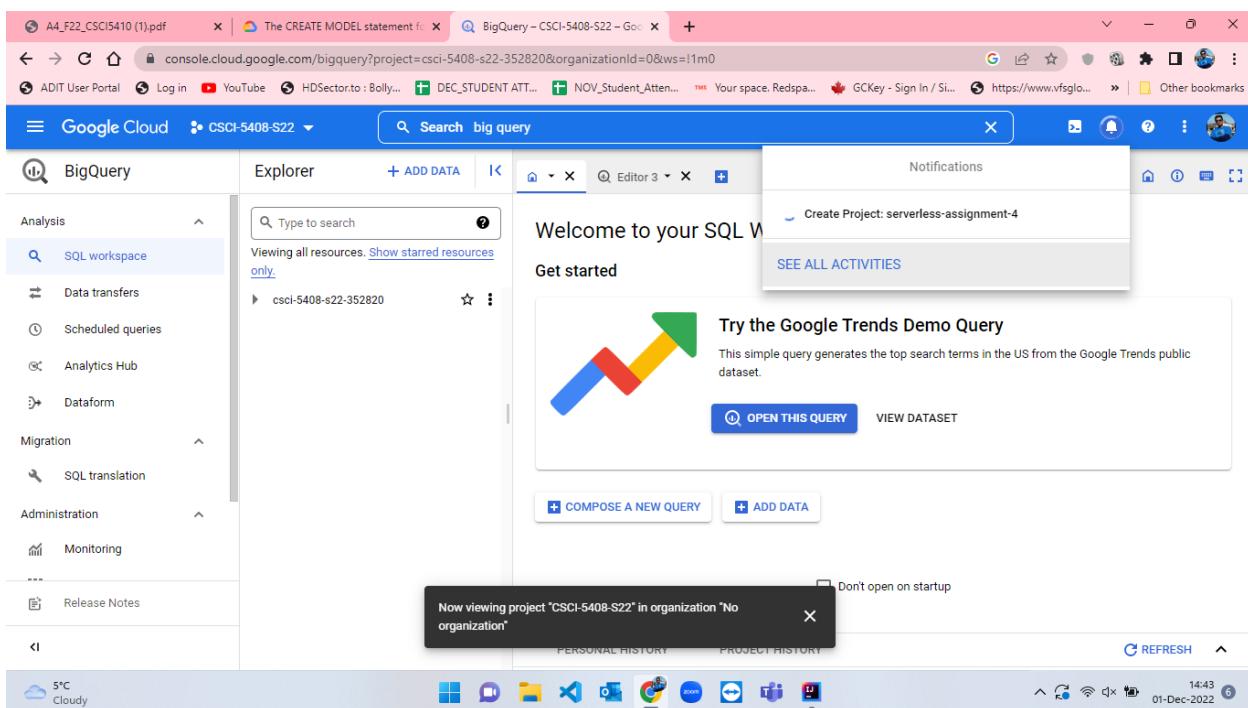


Figure 55: Project creation process taking place.

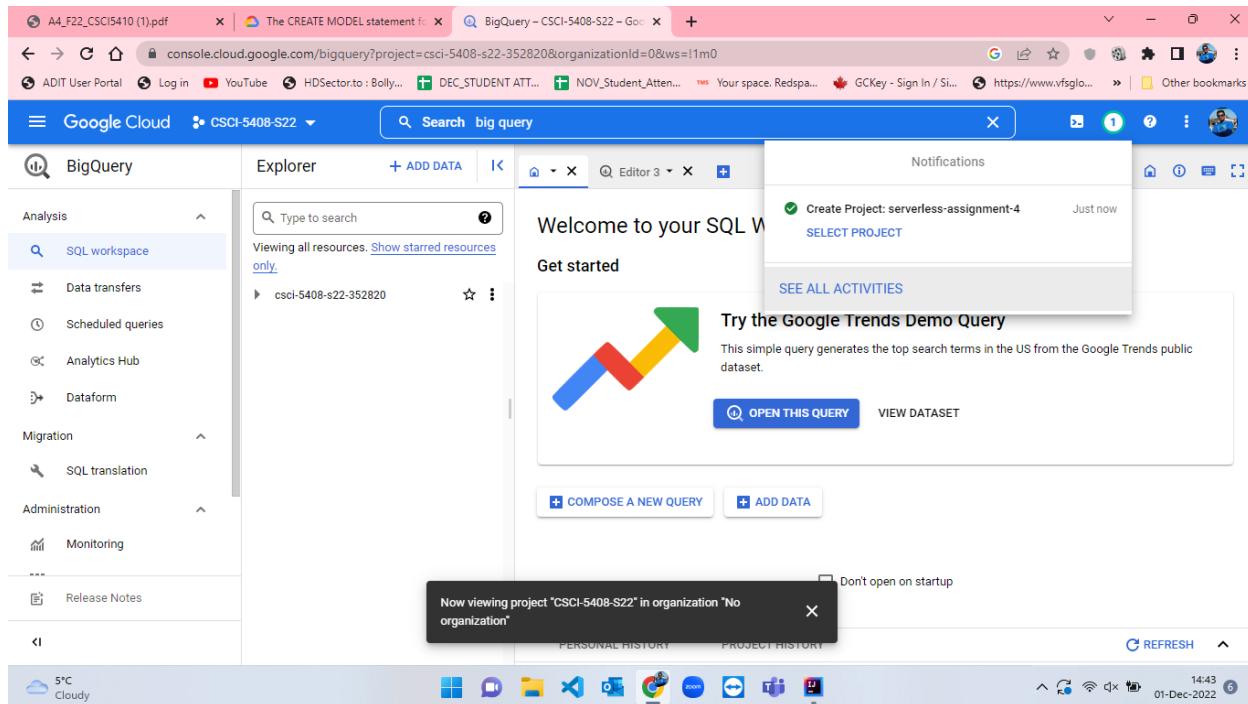


Figure 56: Successfully creation of project.

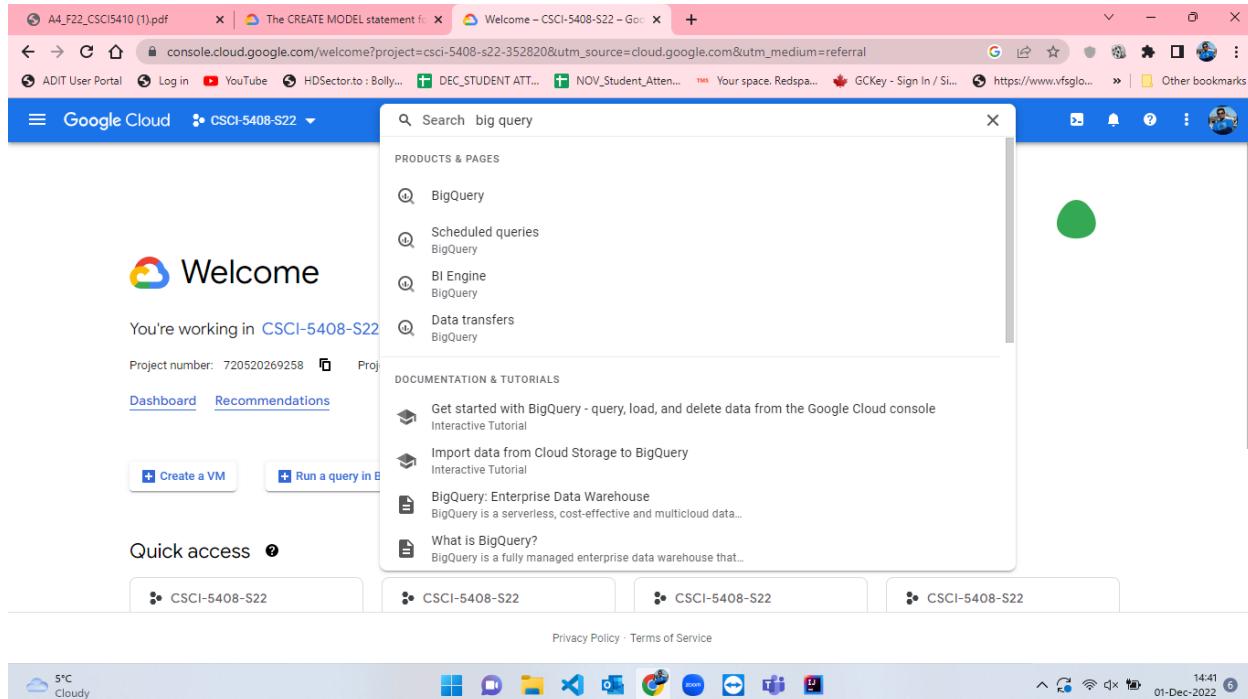


Figure 57: Searching for big query in search bar in GCP.

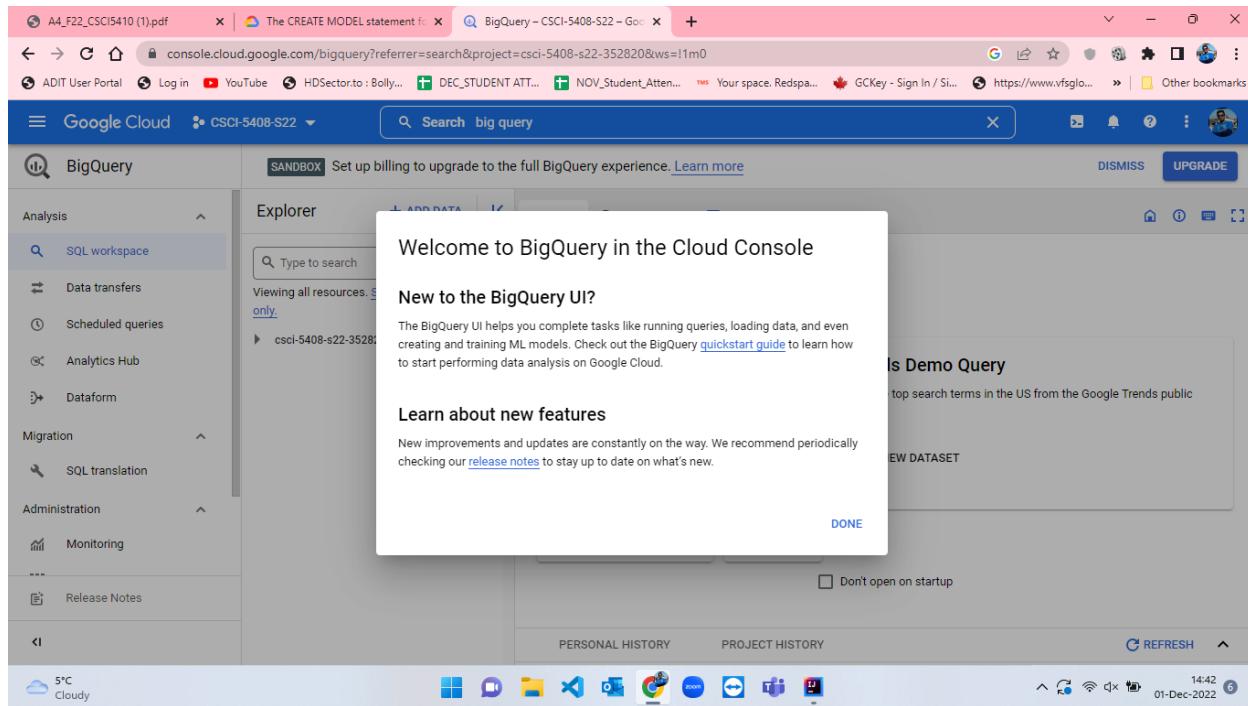


Figure 58: BigQuery welcome message.

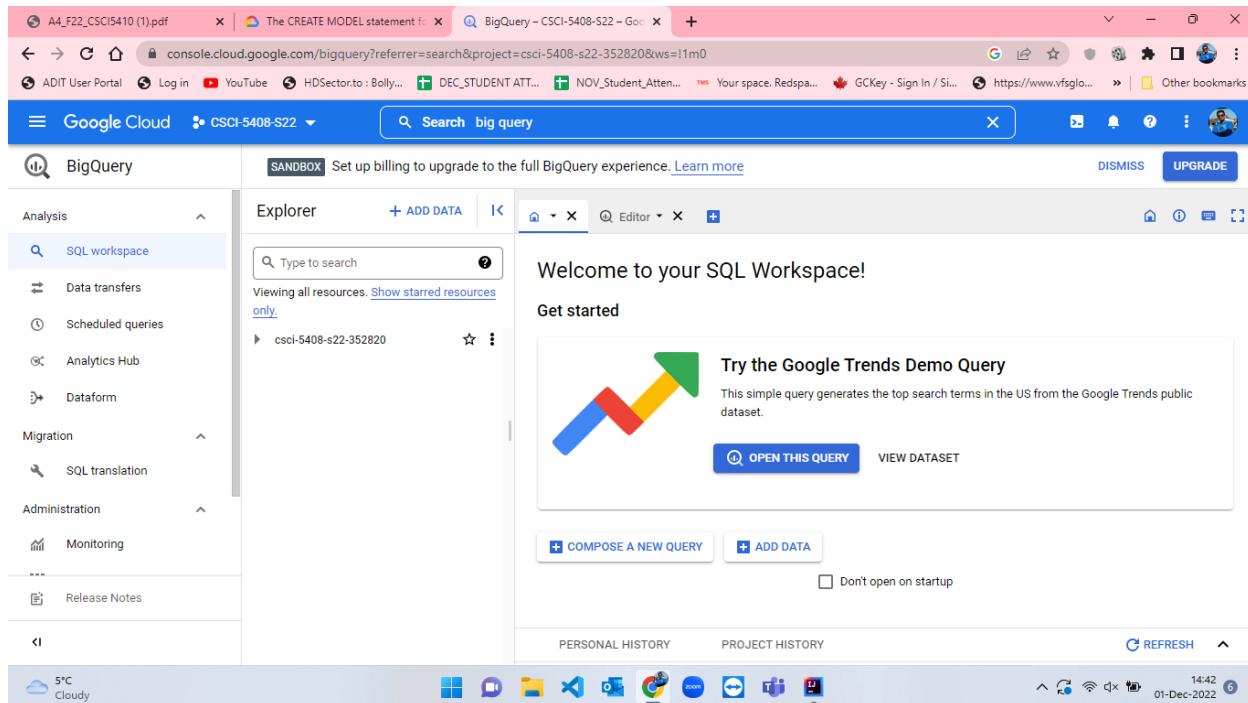


Figure 59: BigQuery Homepage.

- I have to select current project which is “serverless-assignment-4” and the editor to develop the query which is seen in figure-60. After selecting project I will create a dataset (figure-61) and name the dataset as assignment\_4\_dataset. I have selected Data location as “us (multiple regions in United States)” and then clicked on “CREATE DATASET” button.(figure-62 and figure-63).

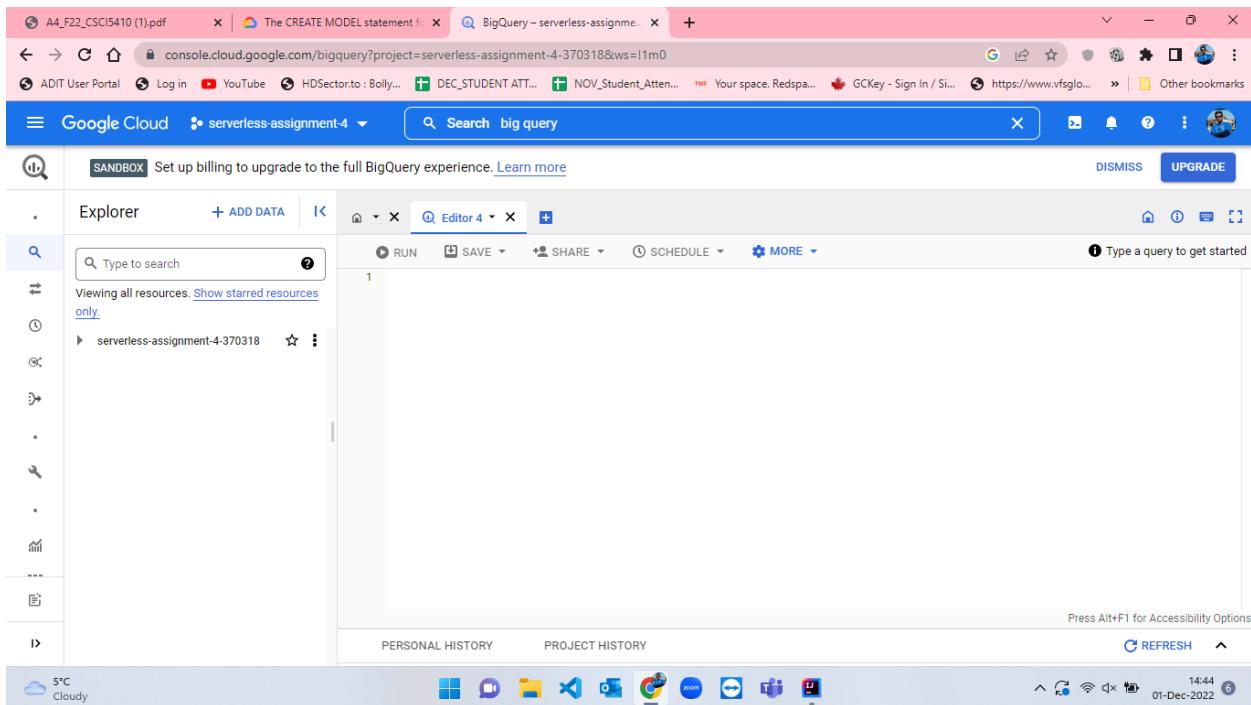


Figure 60: Selecting current project to work and opening editor to write query in GCP BigQuery.

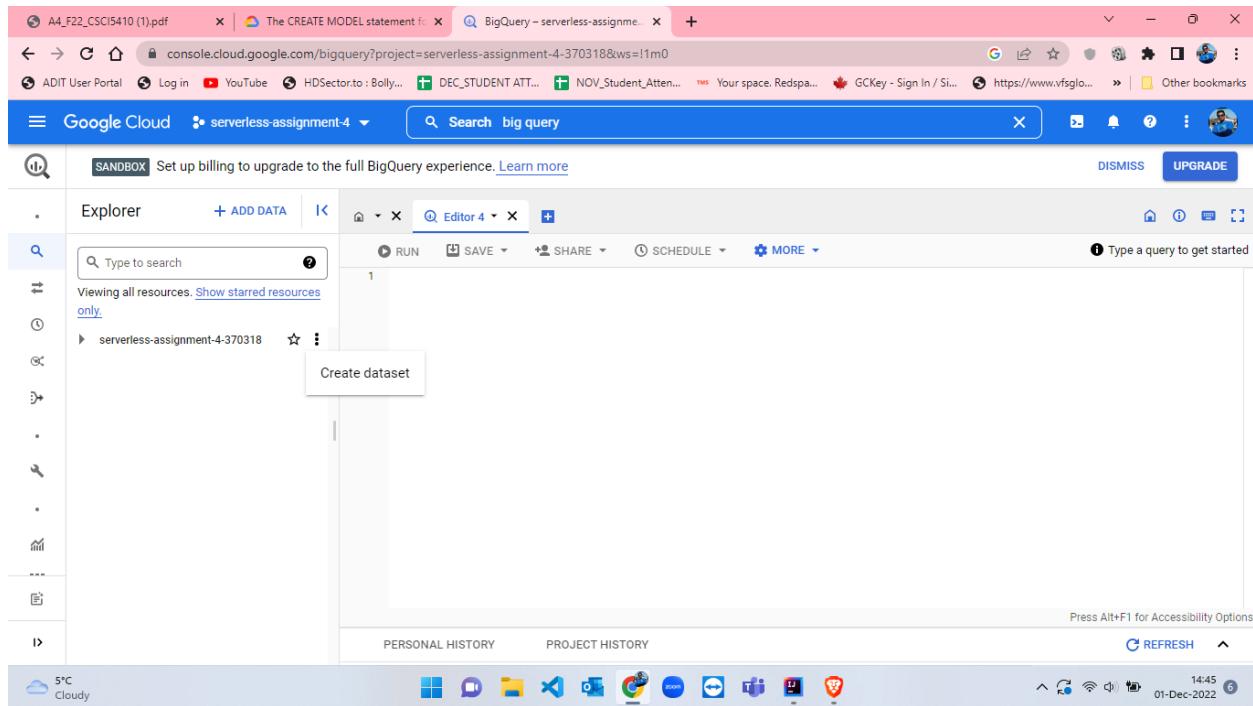


Figure 61: Create dataset in BigQuery.

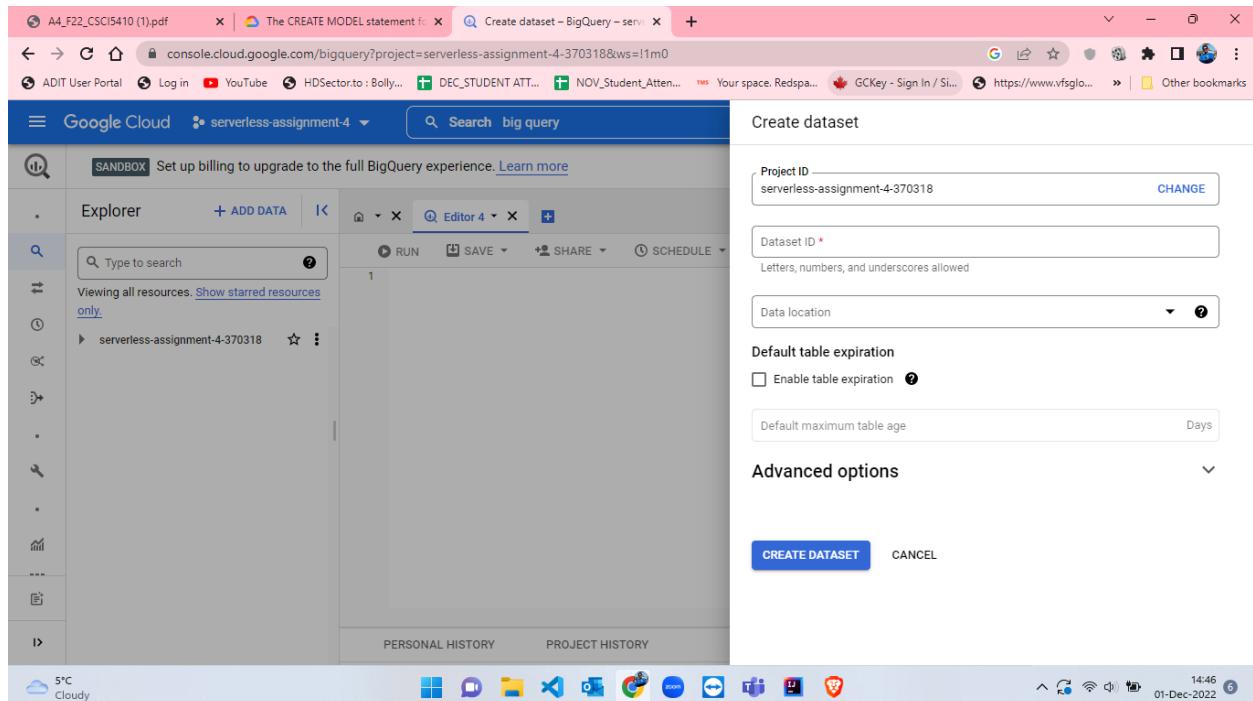


Figure 62: Create dataset blank page in BigQuery.

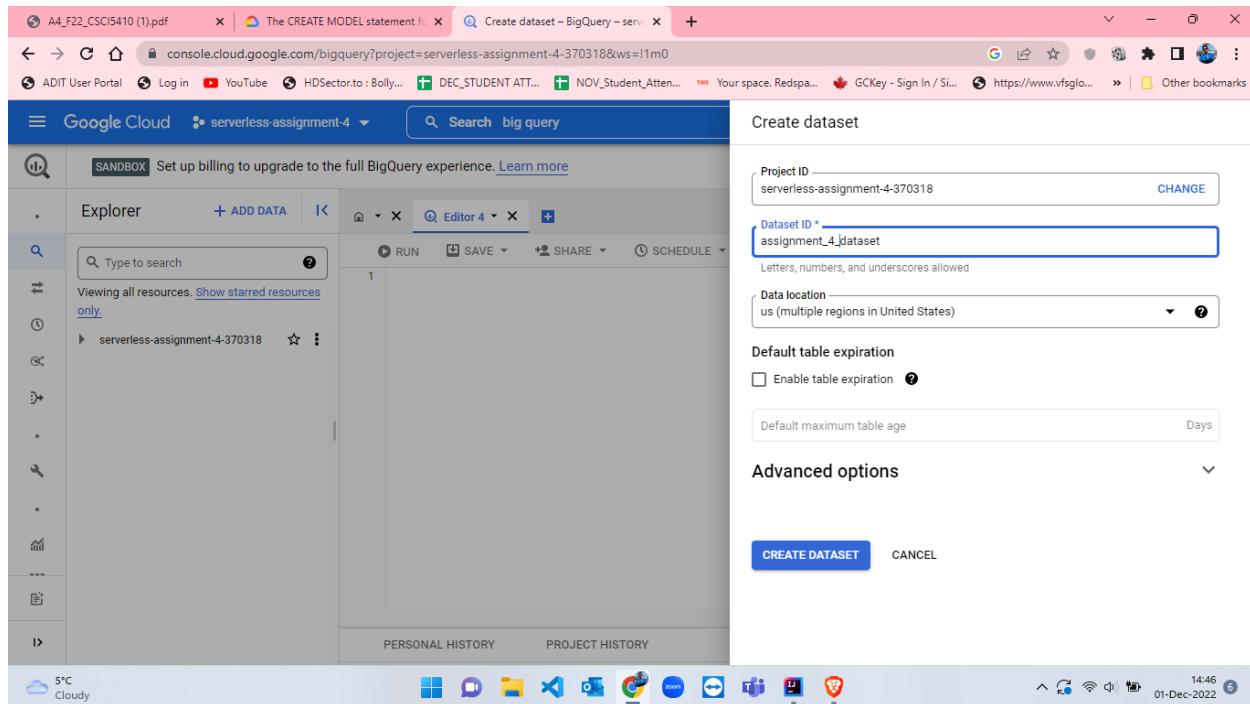


Figure 63: Filling details to create dataset.

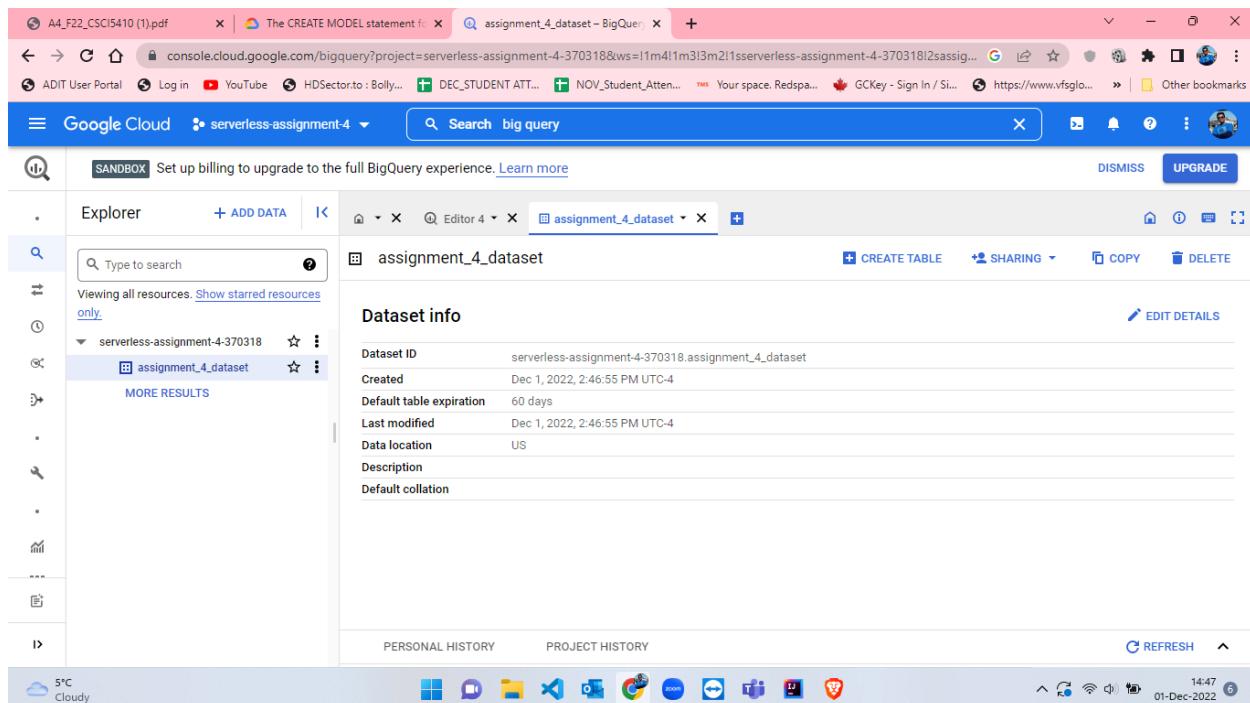


Figure 64: Successful creation of dataset

- From dataset, click on create table to create new table in which select “Upload” option as we have to create that table from the given csv file by professor. I named the table as “Table-1” and selected “Auto-detect” for detecting schema and at the end click on “Create Table”. The table creation process is presented from figures 65 to 68.

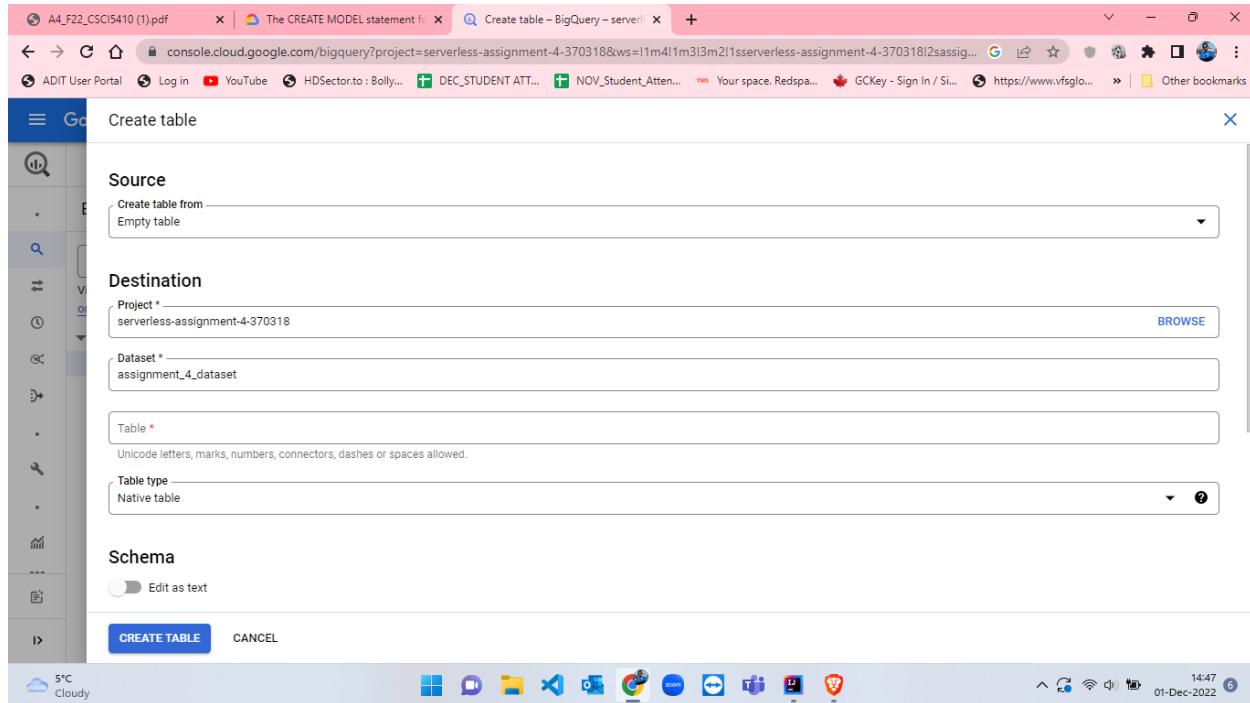


Figure 65: Create table empty form in BigQuery.

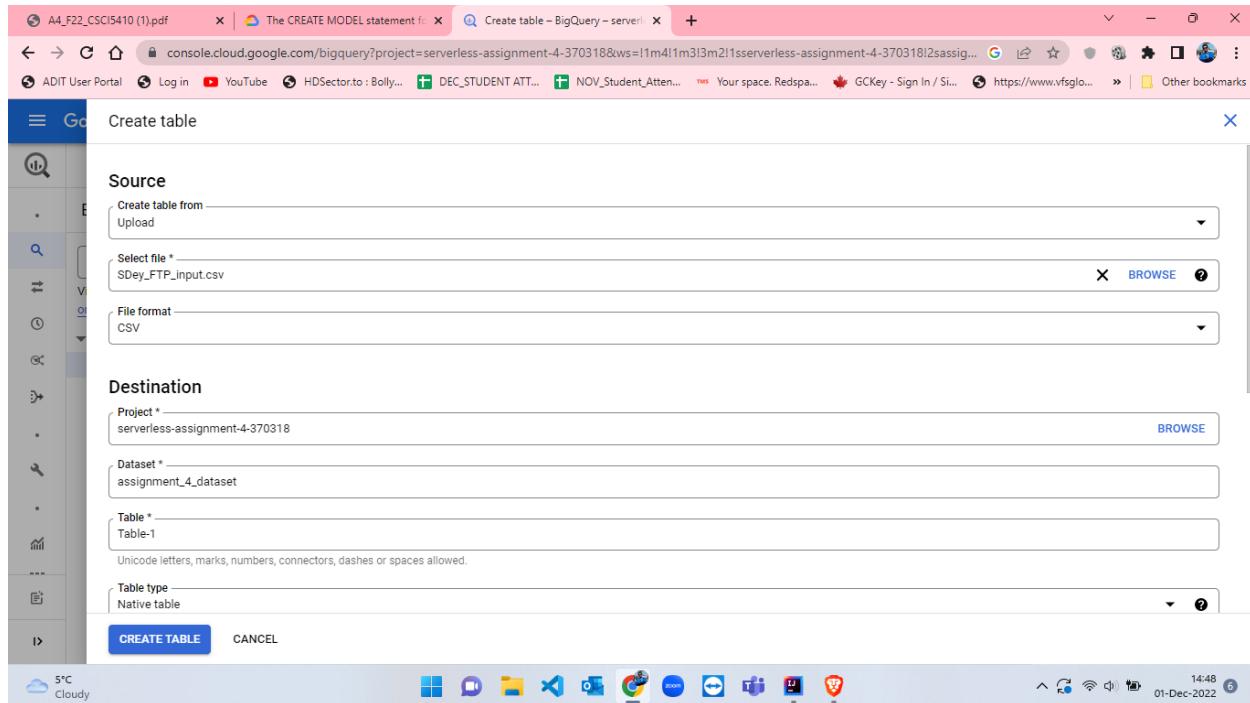


Figure 66: Filling details in create table form part-1.

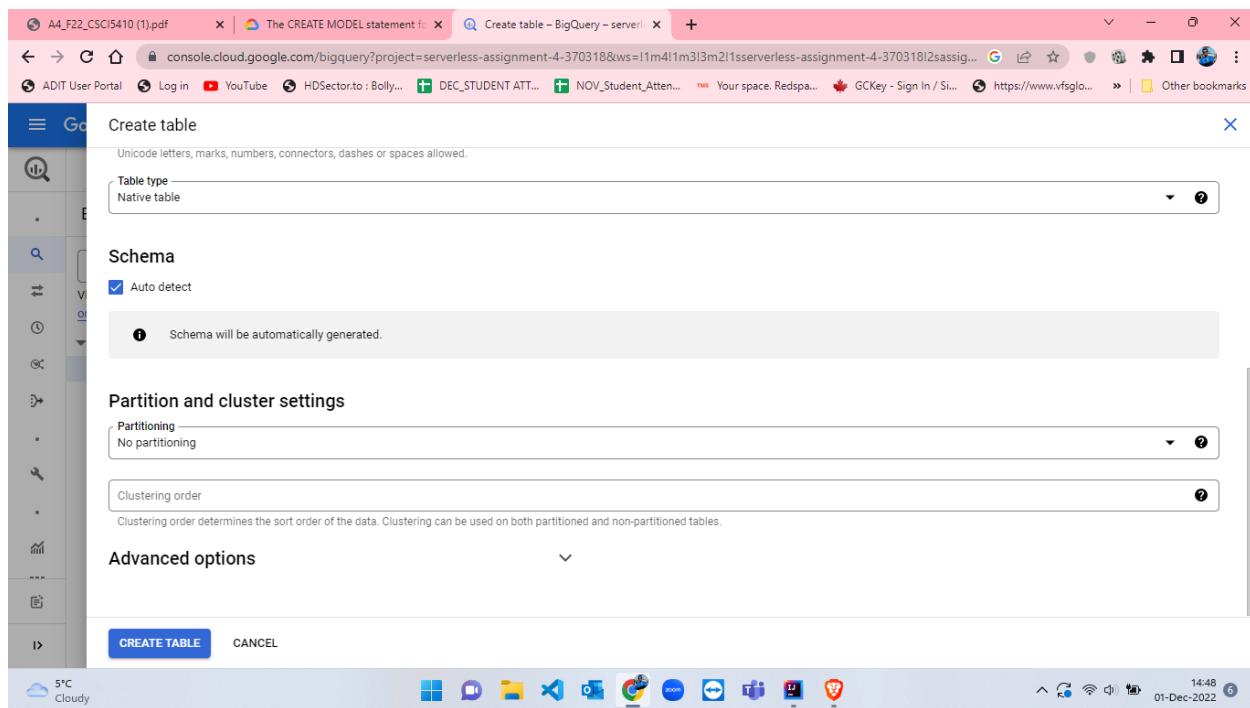


Figure 67: Filling details in create table form part-2.

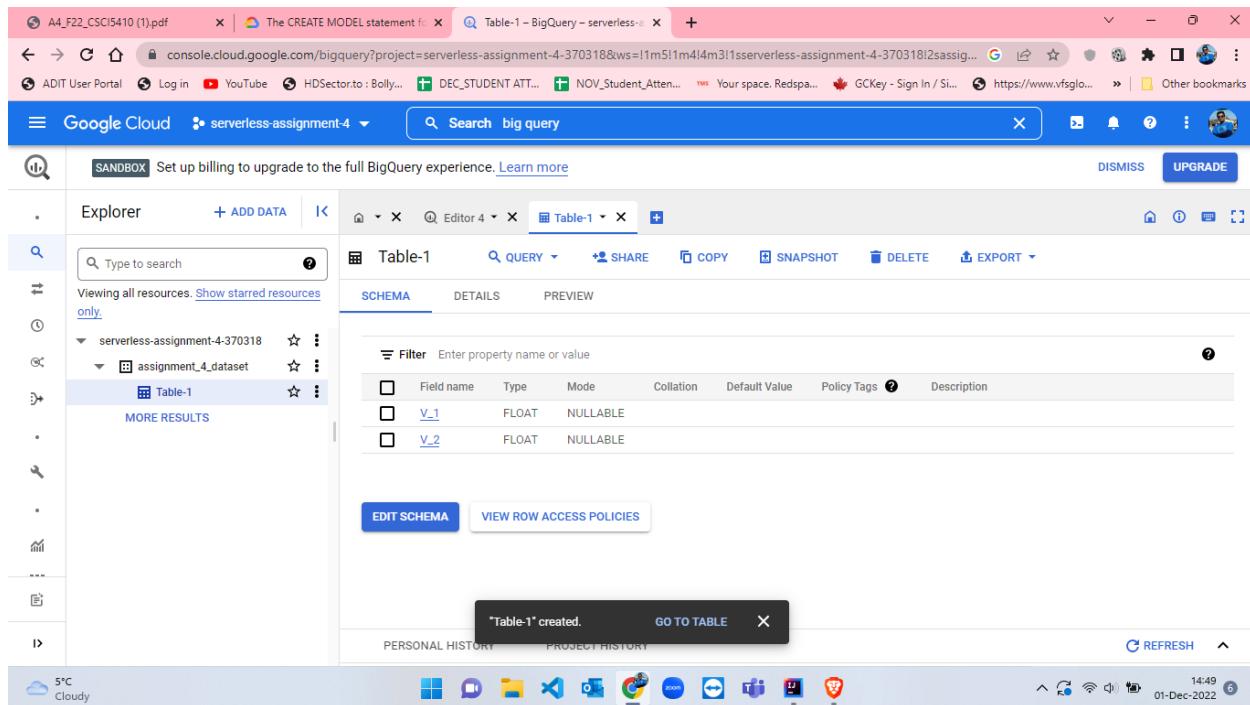
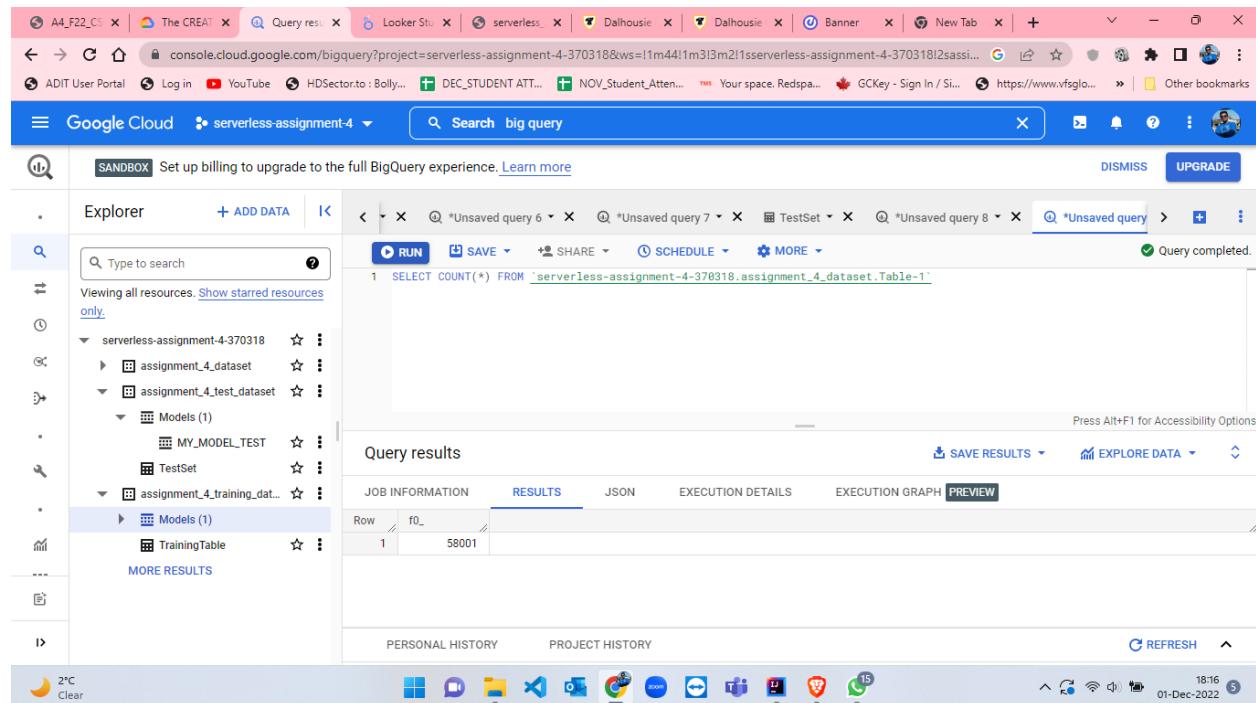


Figure 68: Successful creation of Table-.

- I have created Table-1 with 100% of the data which is shown in the below figure-69 where total number of rows are 58001.



The screenshot shows the Google Cloud BigQuery interface. The left sidebar displays the project structure, including datasets like 'assignment\_4\_dataset' and 'assignment\_4\_test\_dataset'. The main area shows a query editor with the following SQL query:

```
1 SELECT COUNT(*) FROM `serverless-assignment-4-370318.assignment_4_dataset.Table-1`
```

The 'Query results' section shows the output of the query:

Row	f0_
1	58001

Figure 69: Table-1 number of rows.

- After creating Table-1, I have to change the query settings to execute the query. So, I have clicked on “More” option (figure-70) from the editor and from that I have selected “Query Settings” where I have changed the Data Location to “us (multiple regions in United States)” (figure-72). By doing this, the location of the dataset and queries will become the same which will help us to execute queries.

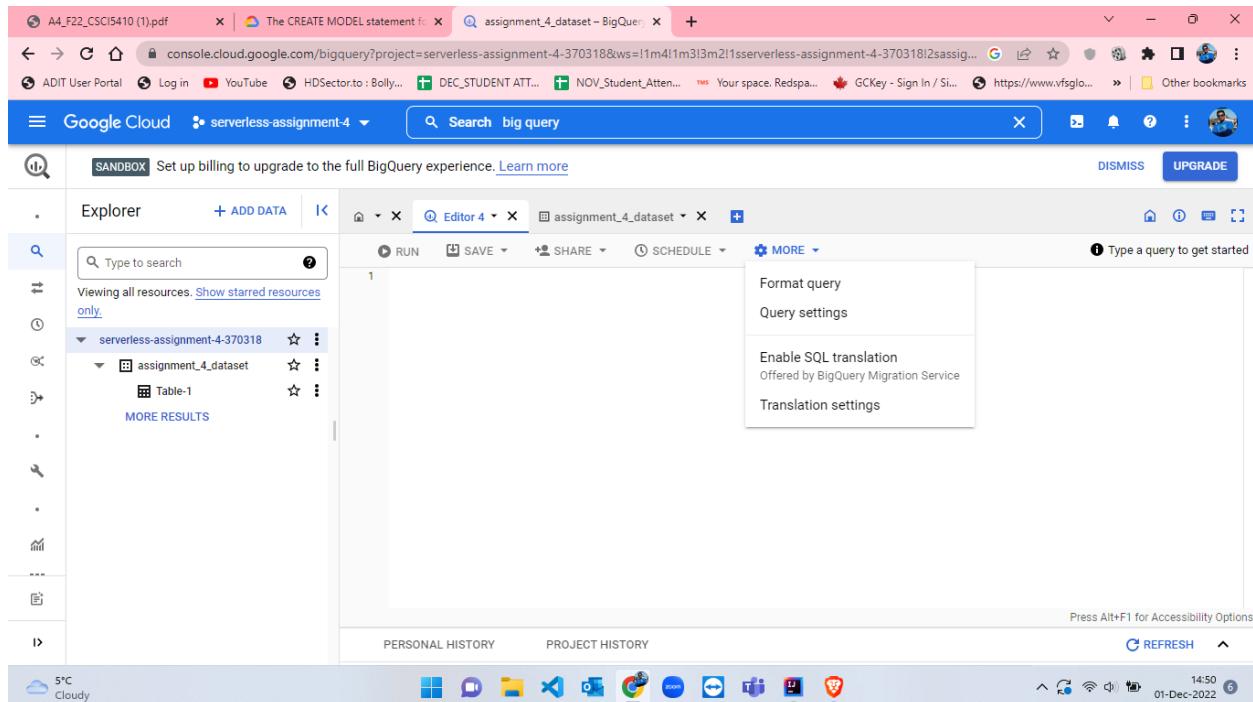


Figure 70: More options to change Query Settings.

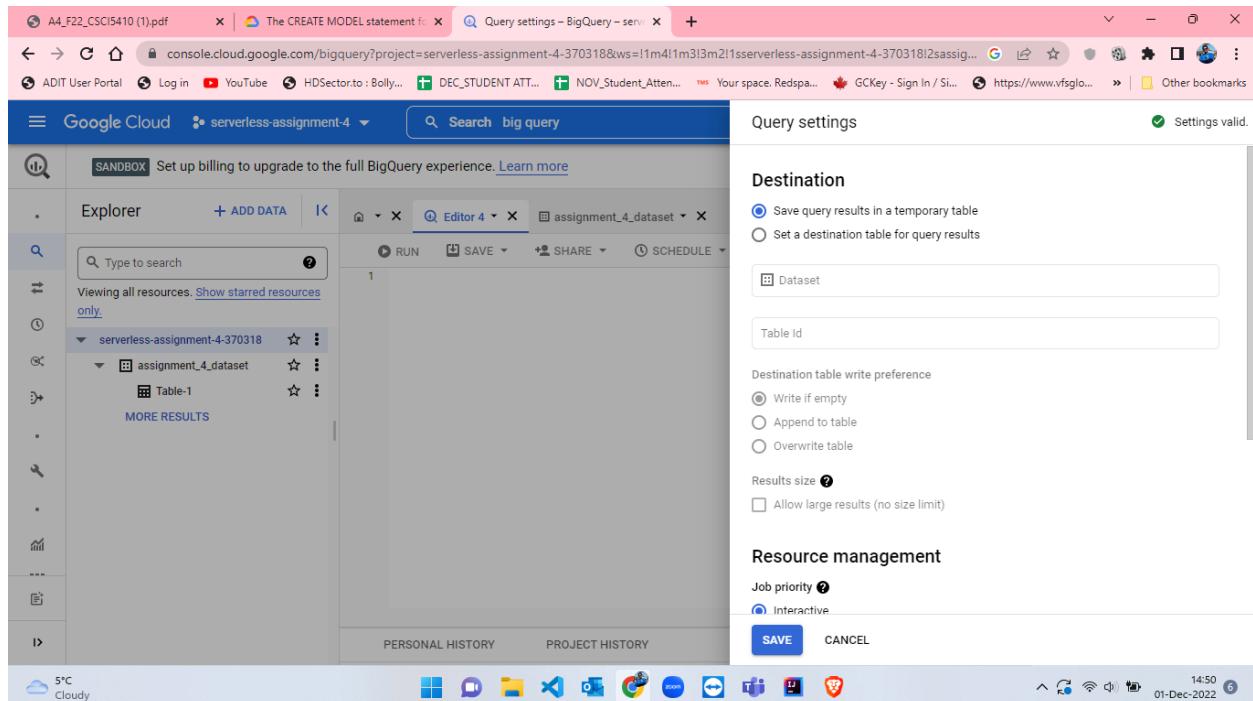


Figure 71: Query Settings in BigQuery step-1.

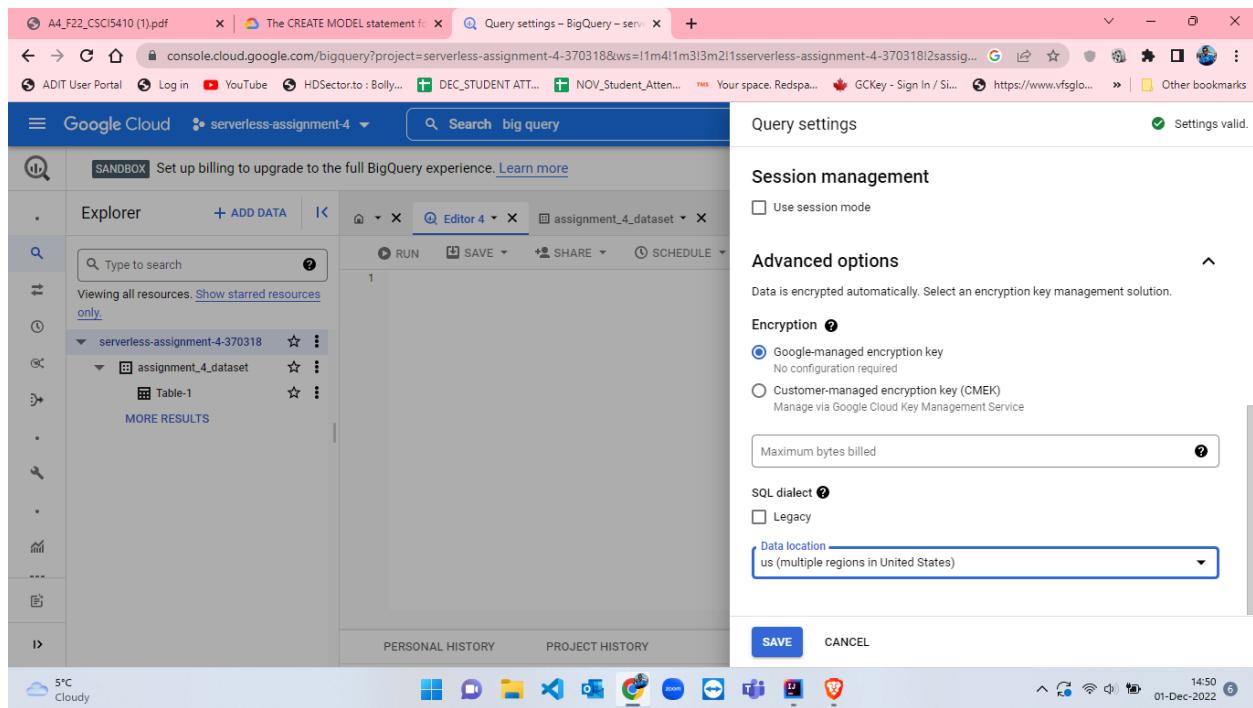
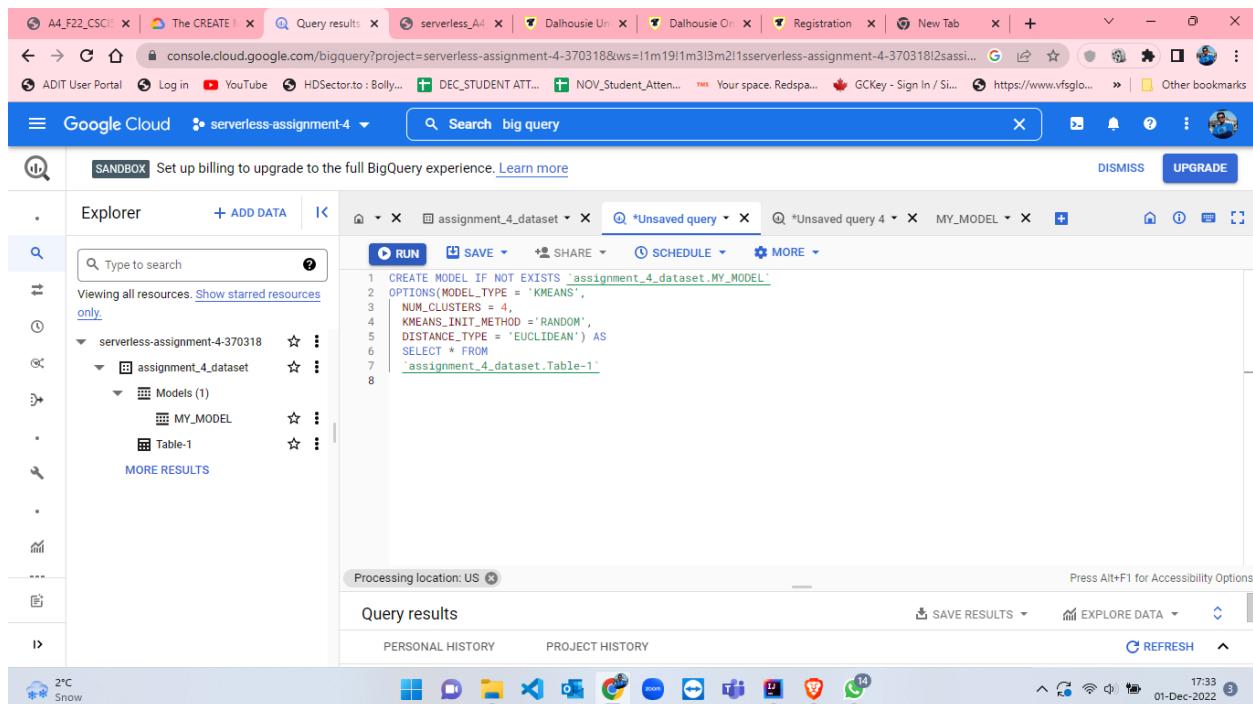


Figure 72: Query Settings in BigQuery step-2.

- Now, I have to create a model to train the data[10]. So, I have created a KMeans model by using the below query:

```
CREATE MODEL IF NOT EXISTS `assignment_4_dataset.MY_MODEL`
OPTIONS(MODEL_TYPE = 'KMEANS',
        NUM_CLUSTERS = 4,
        KMEANS_INIT_METHOD ='RANDOM',
        DISTANCE_TYPE = 'EUCLIDEAN') AS
SELECT * FROM
`assignment_4_dataset.Table-1`
```

- I have set the NUM\_CLUSTERS as 4, KMEANS\_INIT\_METHOD as RANDOM and DISTANCE\_TYPE as EUCLIDEAN. The results from the above query where the model is trained and developed is shown from figure-73 to figure-77.



The screenshot shows the Google Cloud BigQuery interface. The left sidebar shows the 'Explorer' with a dataset named 'assignment\_4\_dataset' containing a 'Models' folder with 'MY\_MODEL' and 'Table-1'. The main area displays an 'Unsaved query' with the following SQL code:

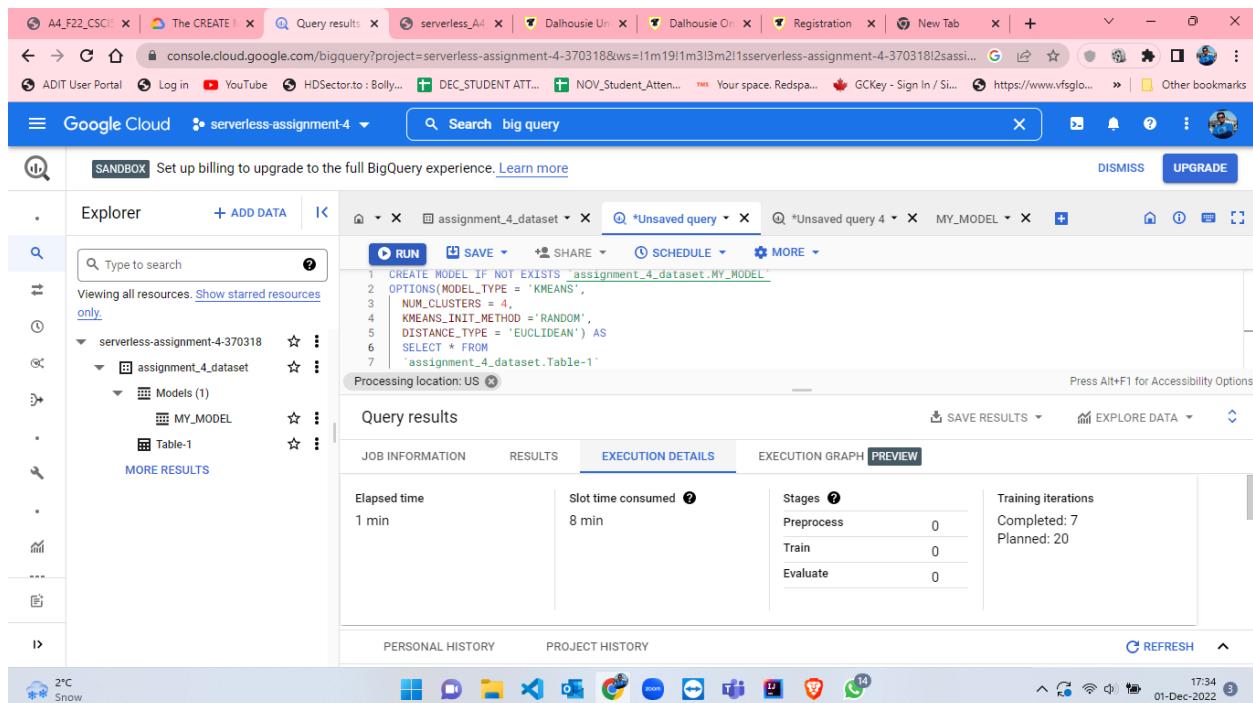
```

1 CREATE MODEL IF NOT EXISTS `assignment_4_dataset.MY_MODEL`
2 OPTIONS(MODEL_TYPE = 'KMEANS',
3         NUM_CLUSTERS = 4,
4         KMEANS_INIT_METHOD = 'RANDOM',
5         DISTANCE_TYPE = 'EUCLIDEAN') AS
6 SELECT * FROM
7   `assignment_4_dataset.Table-1`
8

```

The 'Query results' section is empty. The status bar at the bottom right shows '17:33 01-Dec-2022'.

Figure 73: Query to create model in BigQuery[10].



The screenshot shows the Google Cloud BigQuery interface. The left sidebar shows the 'Explorer' with a dataset named 'assignment\_4\_dataset' containing a 'Models' folder with 'MY\_MODEL' and 'Table-1'. The main area displays an 'Unsaved query' with the same SQL code as Figure 73. Below the code, the 'Execution Details' tab is selected in the 'Query results' section, showing the following data:

Elapsed time	Slot time consumed	Stages	Training iterations
1 min	8 min	Preprocess: 0 Train: 0 Evaluate: 0	Completed: 7 Planned: 20

The status bar at the bottom right shows '17:34 01-Dec-2022'.

Figure 74: Query Execution details when KMeans model is ready part-1.

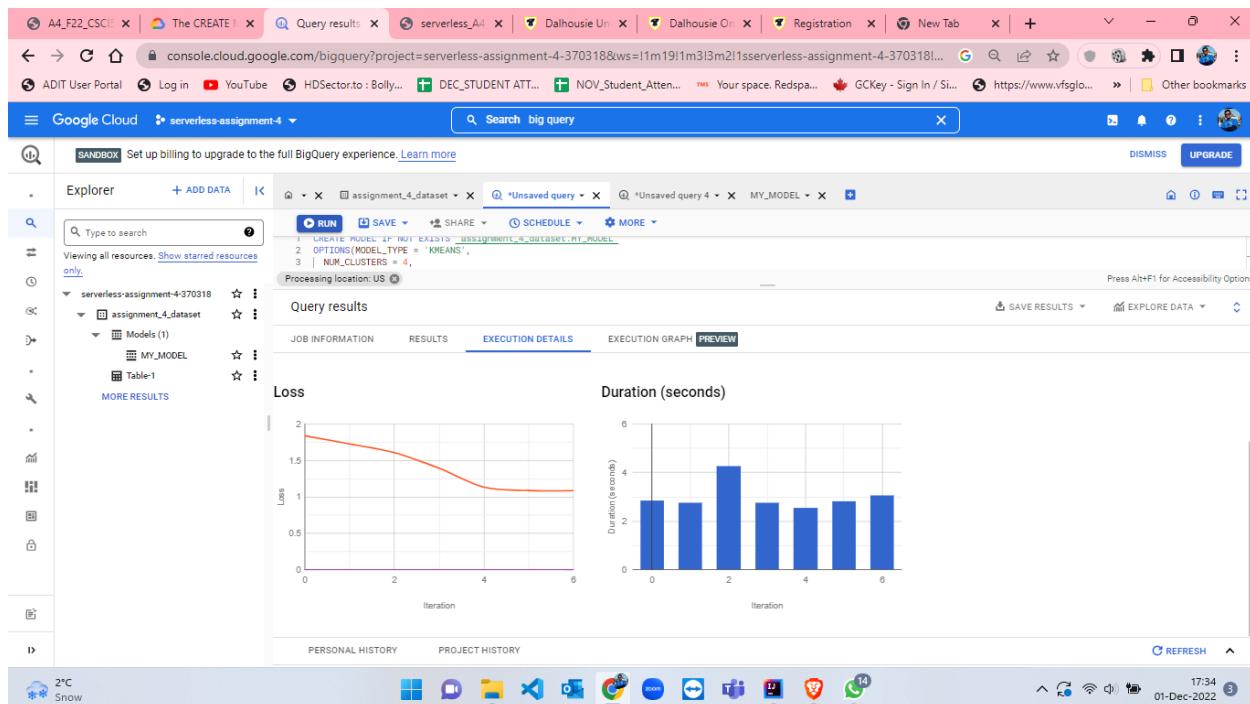


Figure 75: Query Execution details when KMeans model is ready part-2.

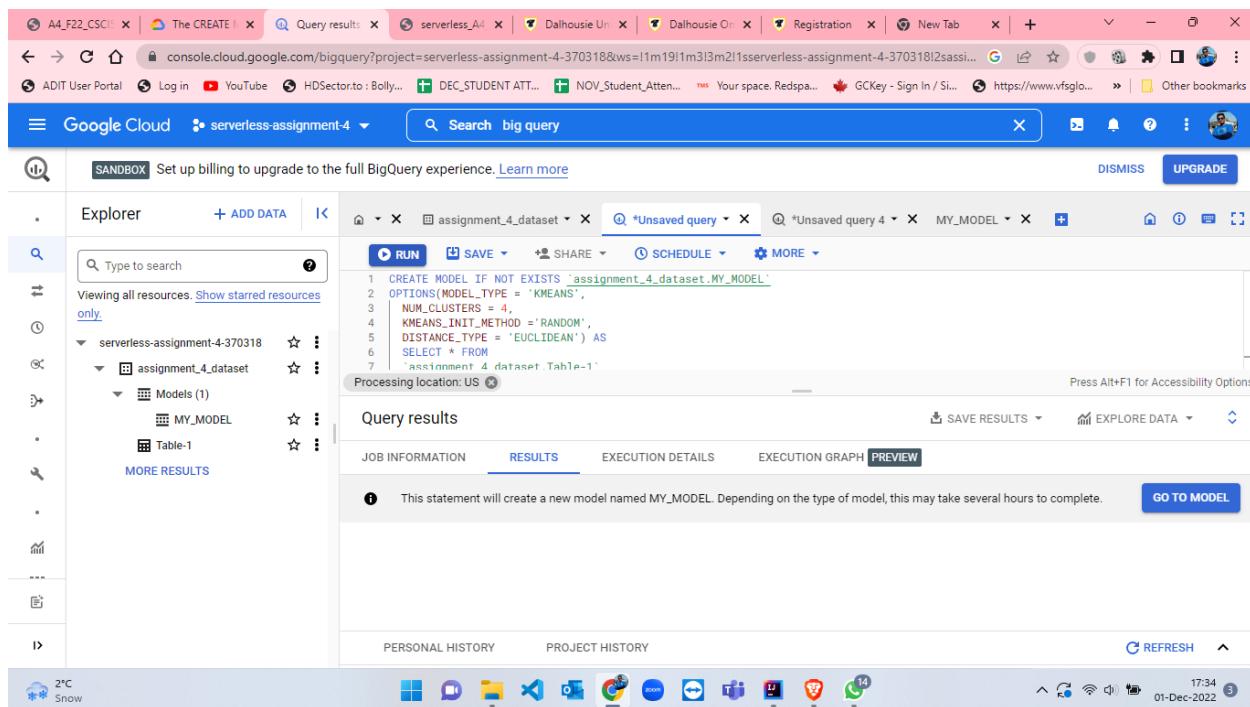


Figure 76: Query results when KMeans model is ready.

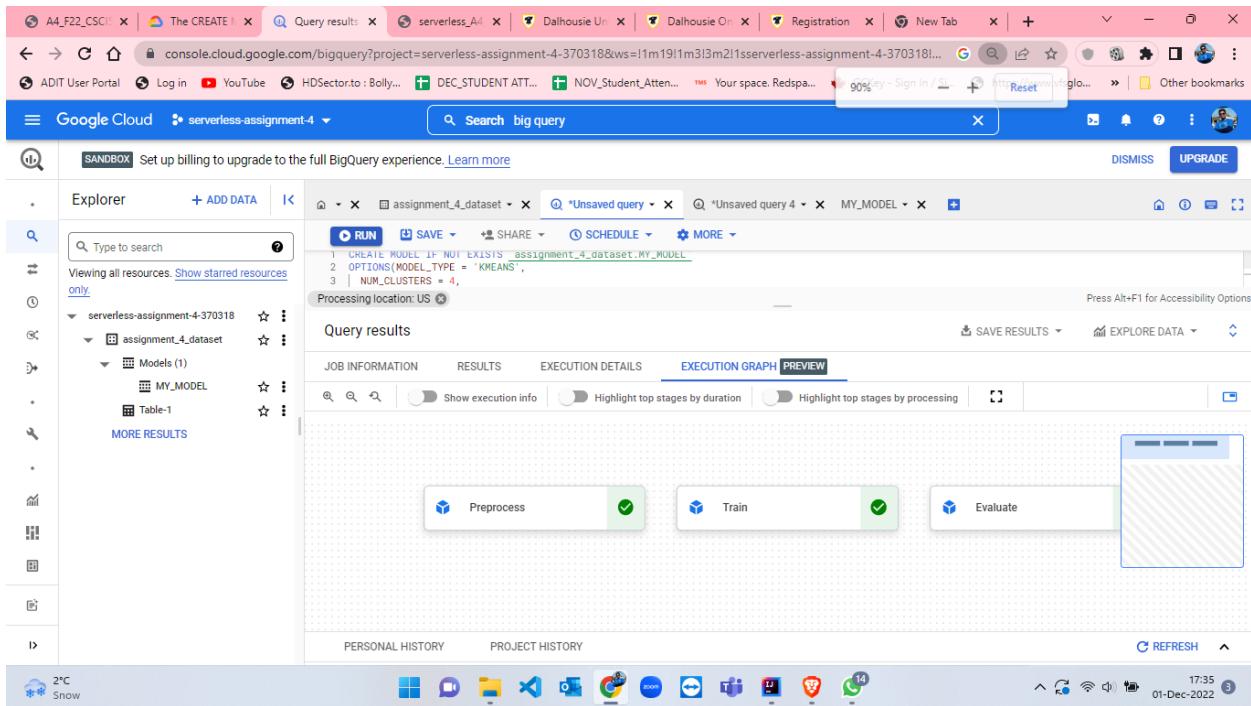


Figure 77: Query Execution Graph when KMeans model is ready.

- When the model is ready, we have to click on “GO TO MODEL” button (figure-76) from results section of query results to view the model. Figure-78 to figure-83 displays the model details, training details, evaluation details and schema details for the first model having 100% data of Table-1 which was MY\_MODEL.

MY\_MODEL

Model type: KMEANS

Data location: US

Model ID: serverless-assignment-4-370318.assignment\_4\_dataset.MY\_MODEL

Model Details

Model ID: serverless-assignment-4-370318.assignment\_4\_dataset.MY\_MODEL

Description:

Labels:

Date created: Dec 1, 2022, 5:29:37 PM UTC-4

Model expiration: Jan 30, 2023, 5:29:37 PM UTC-4

Date modified: Dec 1, 2022, 5:29:37 PM UTC-4

Data location: US

Model type: KMEANS

Training Options

Max allowed iterations: 20

Actual iterations: 7

Early stop: true

Min relative progress: 0.01

Distance type: Euclidean

Number of clusters: 4

Centroids initialization: Random

method

Figure 78: Model details of MY\_MODEL part-1.

MY\_MODEL

Data location: US

Model type: KMEANS

Training Options

Training options are the optional parameters that were added in the script to create this model.

Max allowed iterations: 20

Actual iterations: 7

Early stop: true

Min relative progress: 0.01

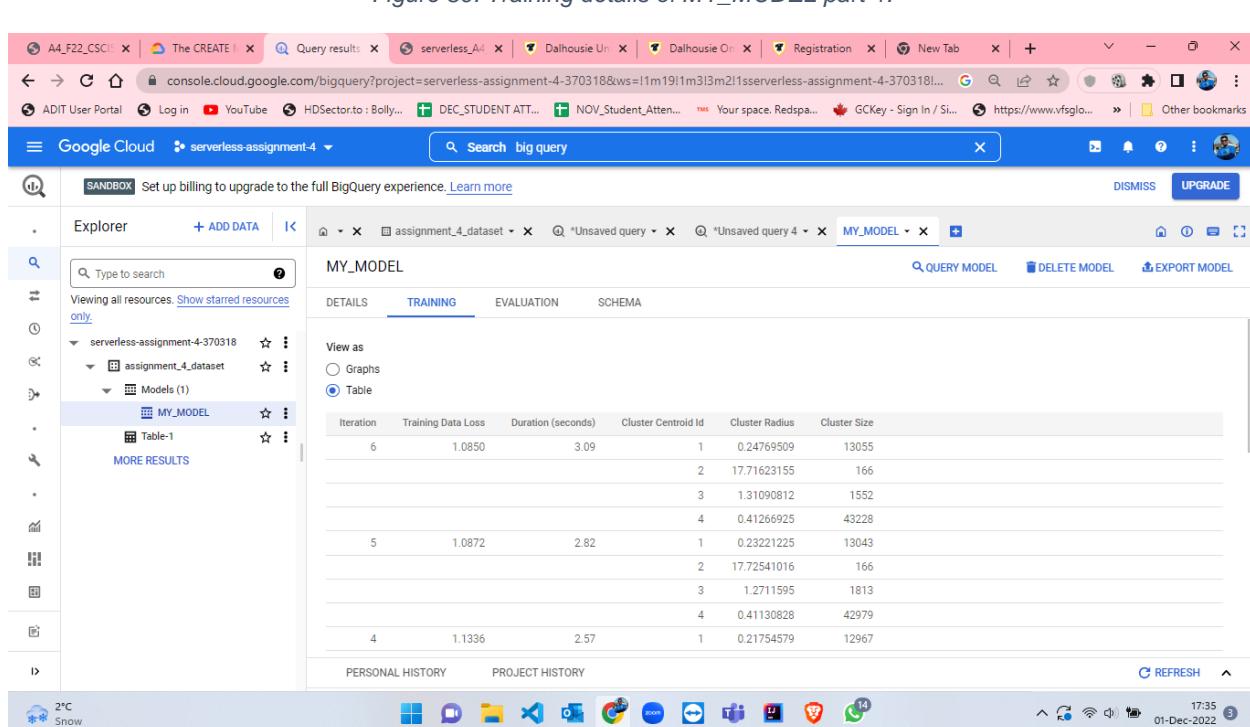
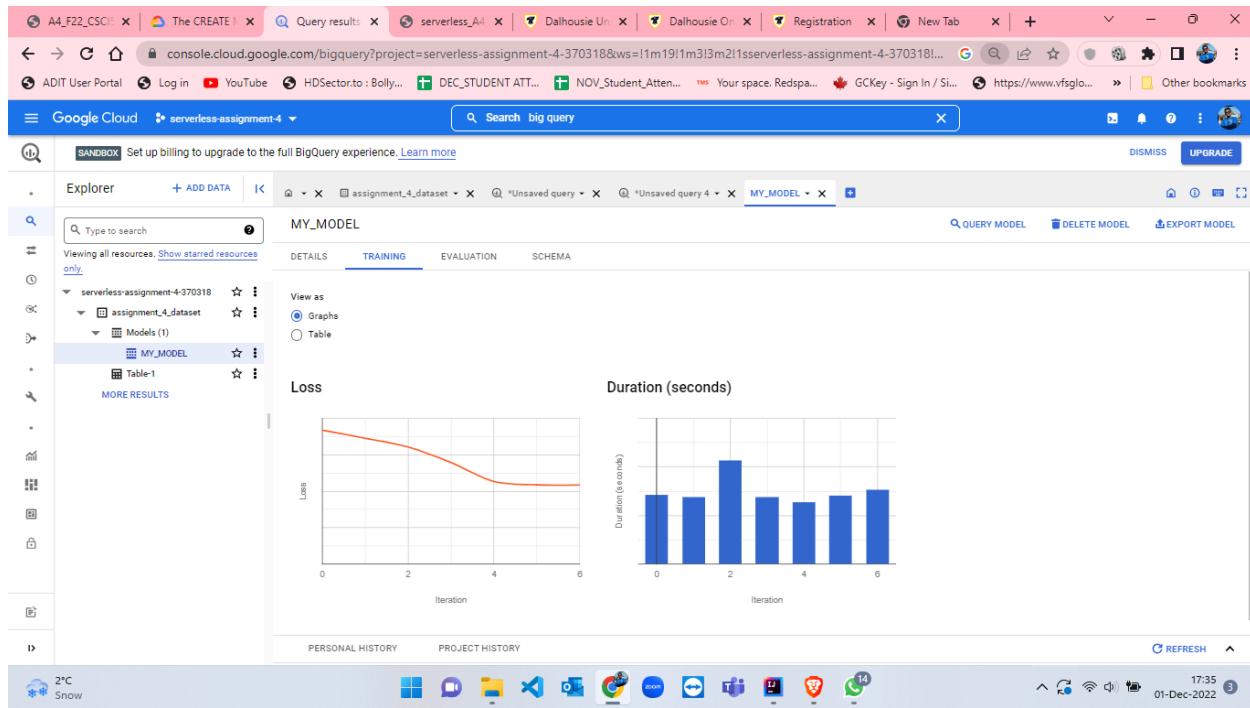
Distance type: Euclidean

Number of clusters: 4

Centroids initialization: Random

method

Figure 79: Model details of MY\_MODEL part-2.



The screenshot shows the Google Cloud BigQuery interface. The main content area is titled 'MY\_MODEL' and is currently displaying the 'EVALUATION' tab. It shows the following metrics:

- Davies-Bouldin index: 1.4361
- Mean squared distance: 1.085

Below the metrics, there is a section for 'Numeric features' which displays a table of centroid values for features V\_1 and V\_2. The table is as follows:

Centroid Id	Count	V_1	V_2
1	13,055	0.0003	0.0023
2	166	0.0723	0.0718
3	1,552	0.0016	0.0101
4	43,228	0.0010	0.0001

At the bottom of the interface, there are tabs for 'PERSONAL HISTORY' and 'PROJECT HISTORY', and a 'REFRESH' button.

Figure 82: Evaluation details of MY\_MODEL.

The screenshot shows the Google Cloud BigQuery interface. The main content area is titled 'MY\_MODEL' and is currently displaying the 'SCHEMA' tab. It shows the following table:

Field name	Type	Mode	Description
V_1	FLOAT64	NULLABLE	
V_2	FLOAT64	NULLABLE	

At the bottom of the interface, there are tabs for 'PERSONAL HISTORY' and 'PROJECT HISTORY', and a 'REFRESH' button.

Figure 83: Schema details of MY\_MODEL.

- After model creation and looking into the details I have developed a new query for ML prediction. The query is attached below, and the process and execution are also attached from figure-84 to figure-87.

```
SELECT * FROM ML.PREDICT(MODEL `assignment_4_dataset.MY_MODEL`, (SELECT * FROM `assignment_4_dataset.Table-1`))
```

Row	CENTROID_ID	CENTROID_ID	N_DISTANCE	V_1	V_2
1	4	4	6.73465472...	0.041282	4e-06
		3	6.85244734...		
		1	6.86983111...		
		2	13.0905063...		
2	2	2	16.2416539...	0.003368	0.140052
		3	21.7487640...		
		1	23.0566663...		
		4	23.4219473		

Figure 84: Results from ML.PREDICT query.

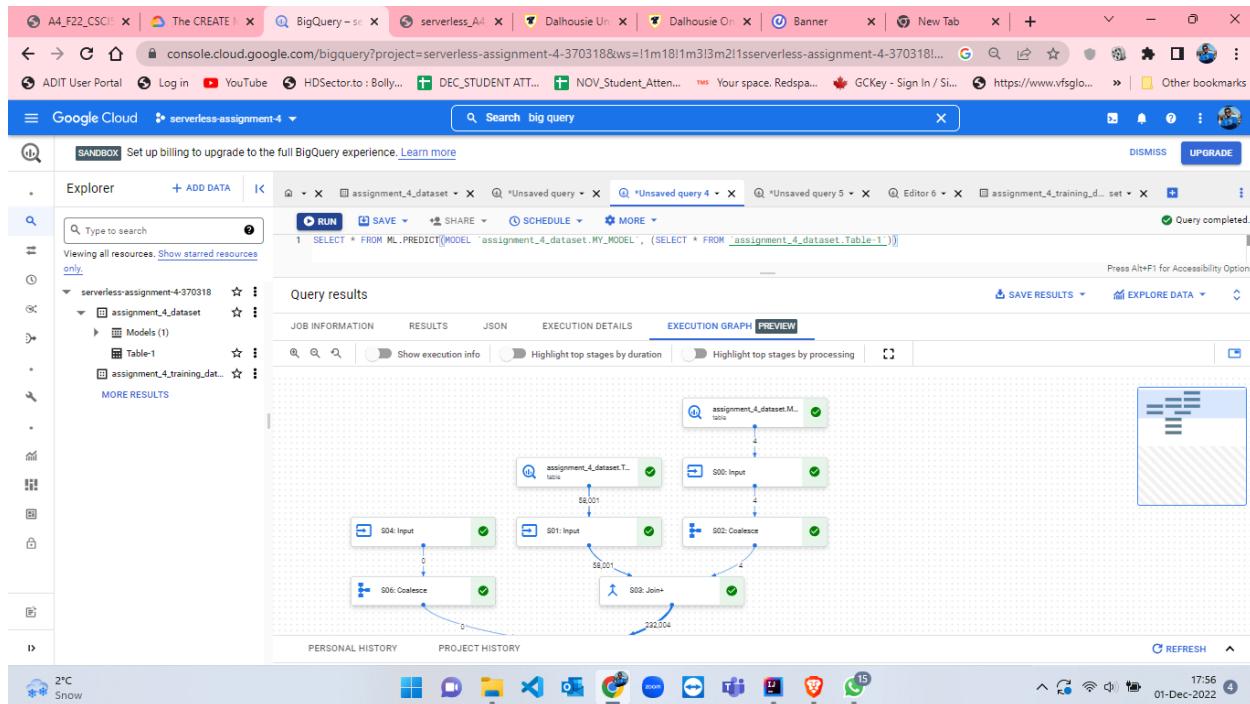


Figure 85: Execution Graph from ML.PREDICT query.

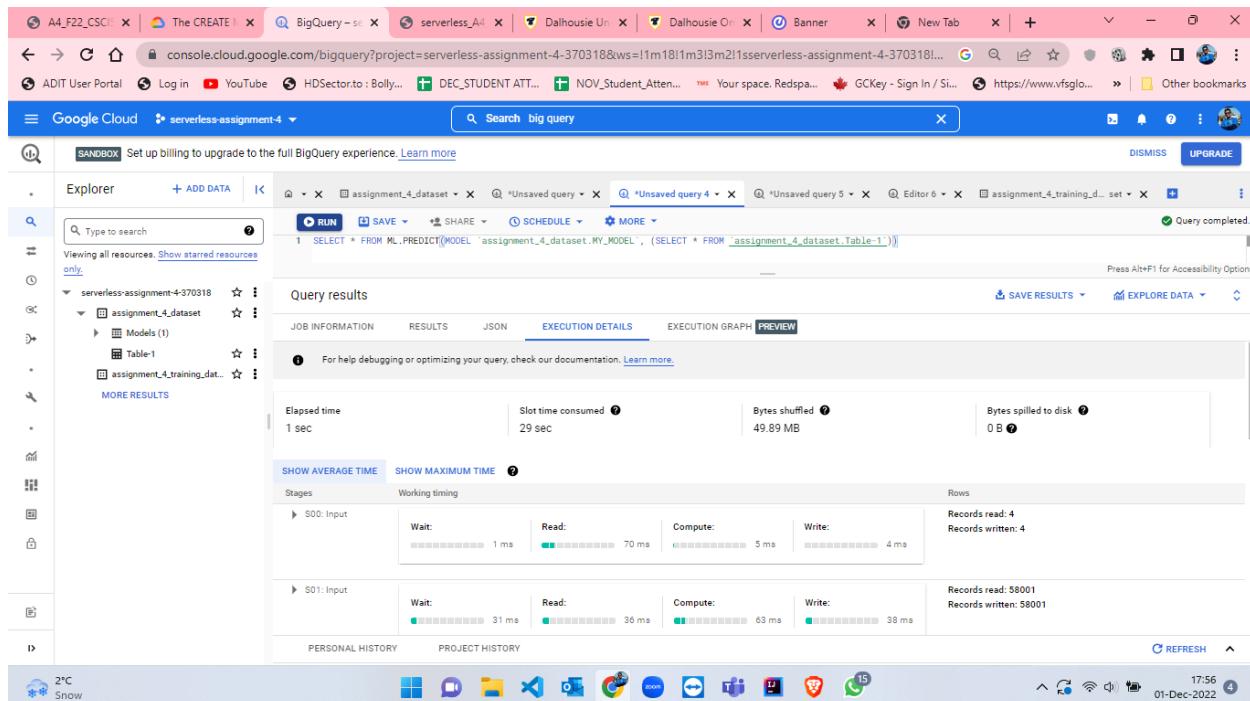


Figure 86: Execution Details from ML.PREDICT query part-1.

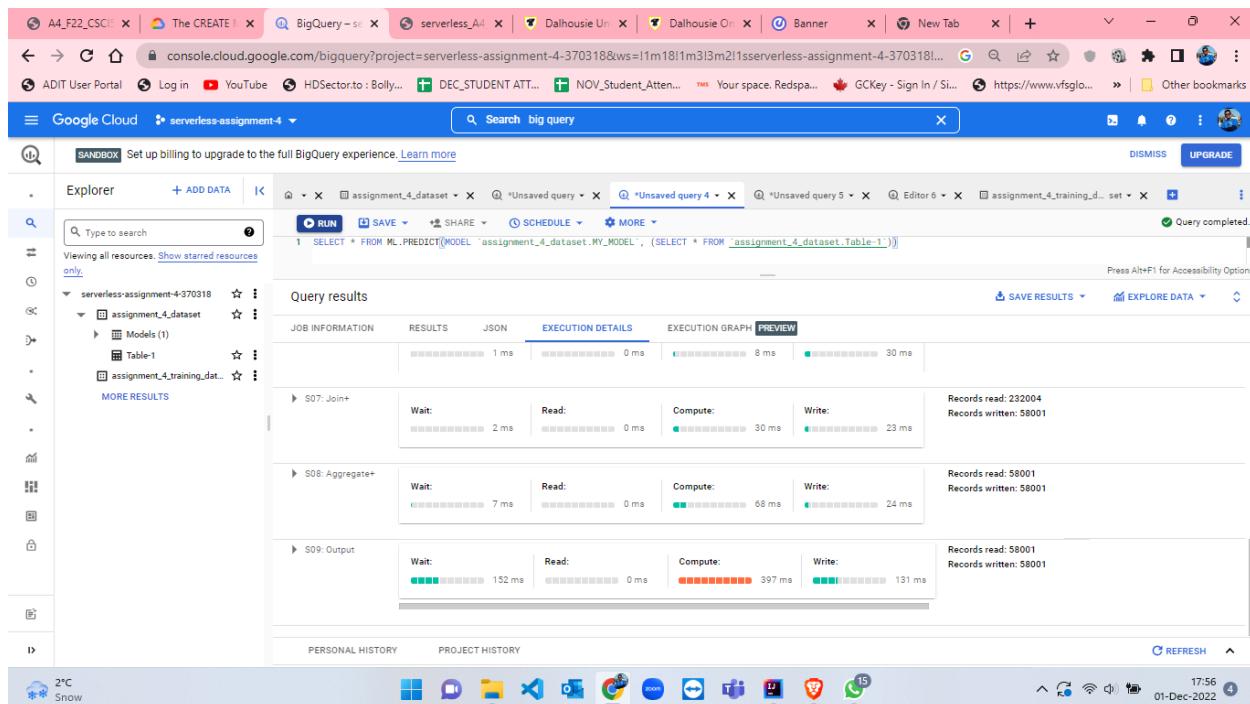


Figure 87: Execution Graph from ML.PREDICT query part-2.

- After getting ML Prediction if we want to explore more in data then click on “EXPLORE DATA” on the right hand side (figure-87) which will give us options for visualization from which I selected Looker Studio (figure-88).

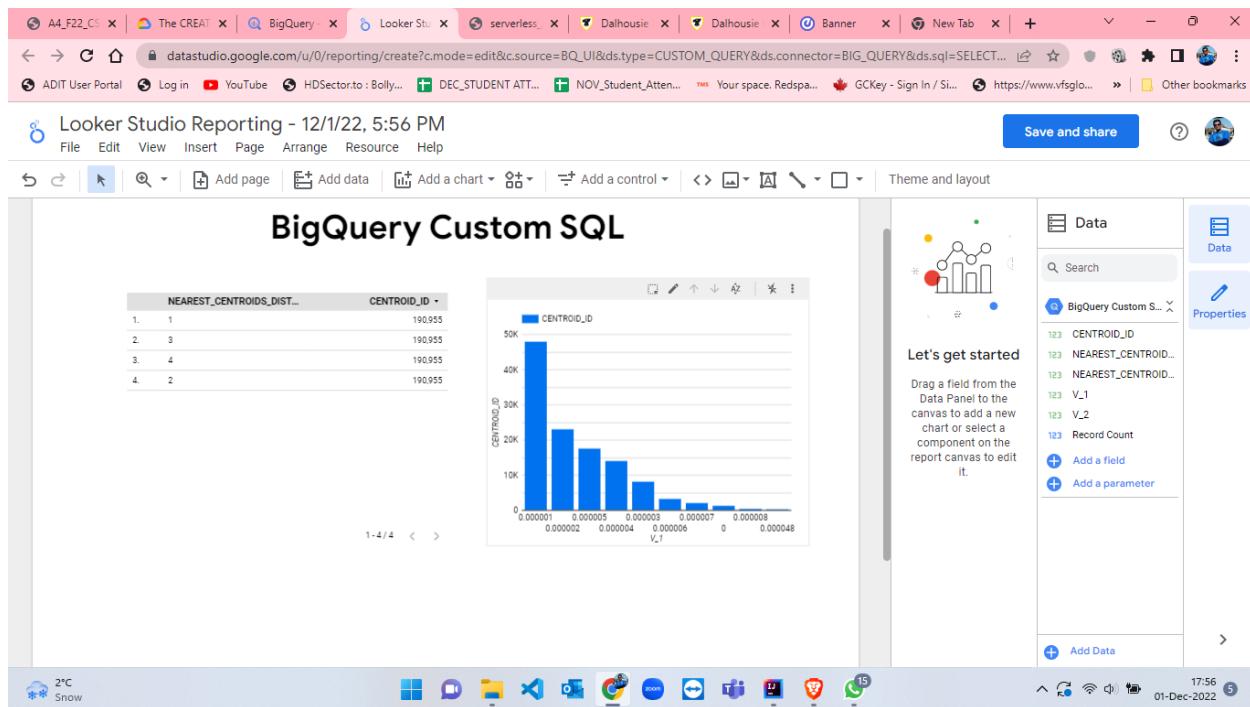


Figure 88: Looker Studio Reporting for visualization.

- We have trained and test the dataset where we are having all records i.e. we have 58001 records. Now, we have to create a random sampling with 75% of the records from the given csv file as training set. I have created this sampling by using excel and by manual sampling. Dataset creation, table creation, model training[10] process and testing alongwith evaluation of model and ML prediction is attached below from figure-89 to figure-103. This is the similar process as we have done for 100% of the data.

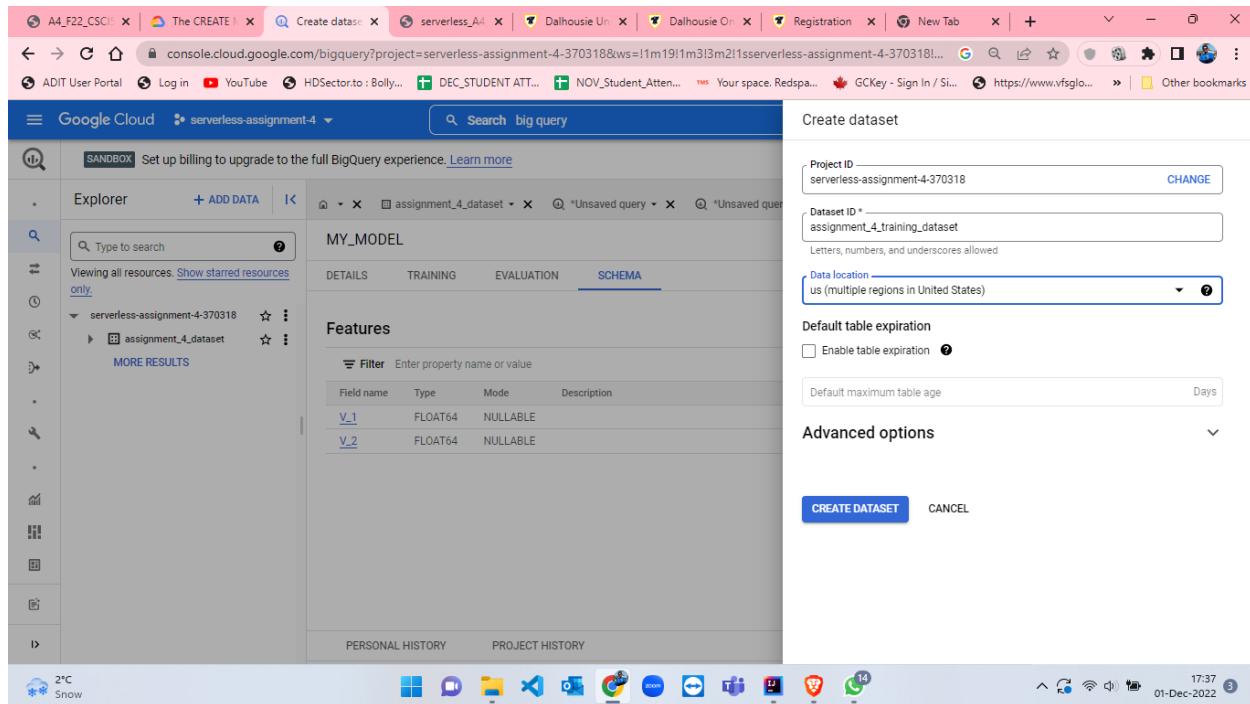


Figure 89: Create dataset named as `training_dataset`.

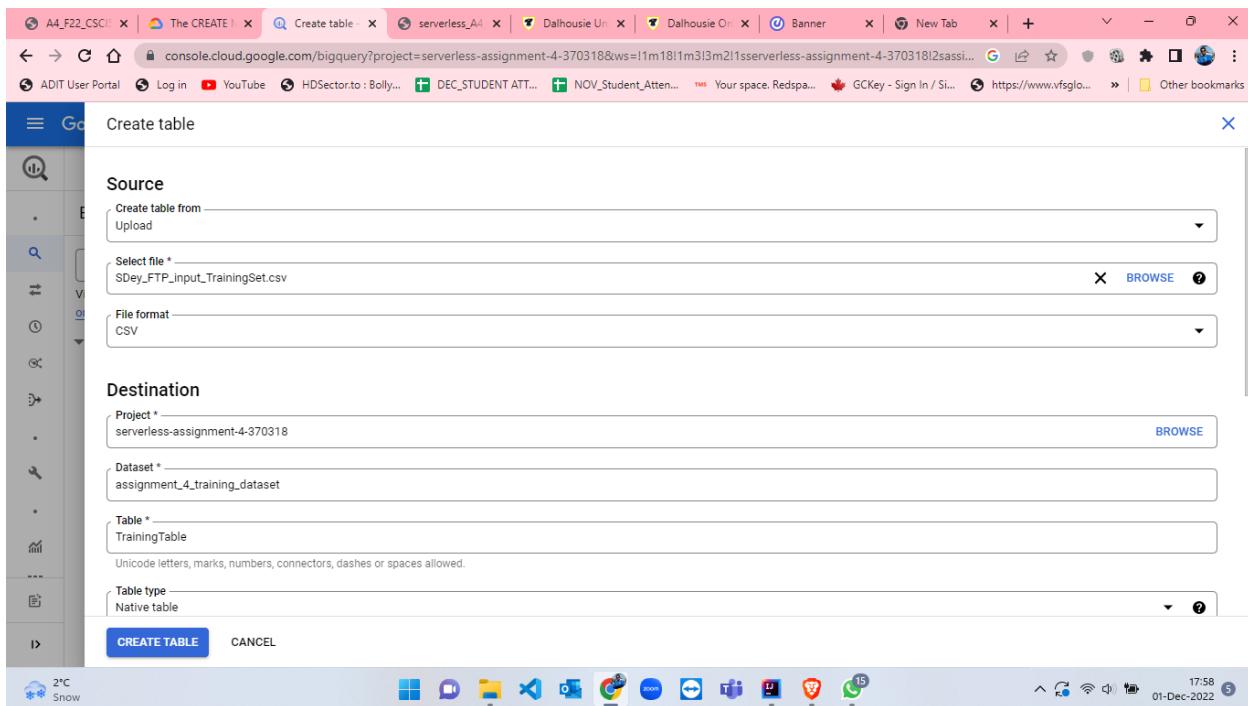


Figure 90: Create table for training named as *TrainingTable* having 75% of records.

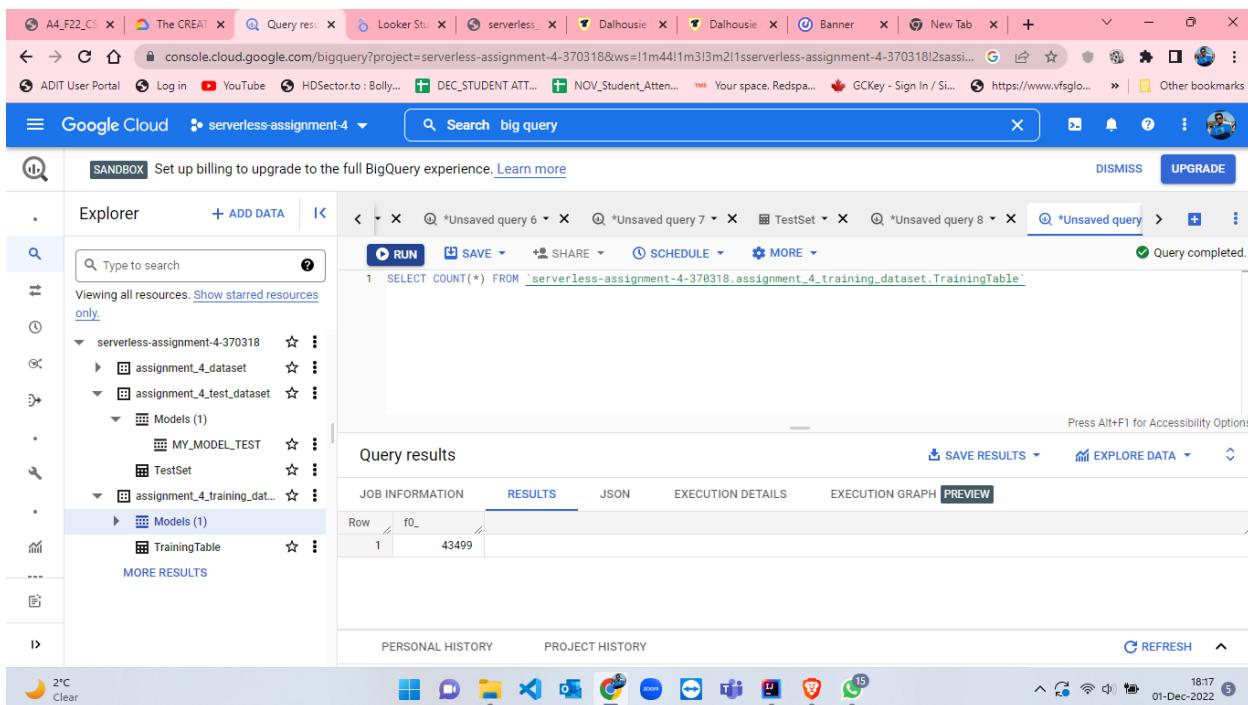


Figure 91: *TrainingTable* number of rows.

```

CREATE MODEL IF NOT EXISTS `assignment_4_training_dataset.MY_MODEL_TRAINING`

OPTIONS(
  MODEL_TYPE = 'KMEANS',
  NUM_CLUSTERS = 10,
  KMEANS_INIT_METHOD = 'RANDOM',
  DISTANCE_TYPE = 'EUCLIDEAN') AS
SELECT * FROM
`assignment_4_training_dataset.TrainingTable`
```

- Here, for training model I have made some changes while creating model[10]. I have set the NUM\_CLUSTERS as 10 which was 4 in previous model, KMEANS\_INIT\_METHOD as RANDOM and DISTANCE\_TYPE as EUCLIDEAN.

The screenshot shows the Google Cloud BigQuery interface. The left sidebar displays the project structure, including the 'assignment\_4\_training\_dataset' dataset which contains a 'TrainingTable'. The main area shows a query editor with the following SQL code:

```

CREATE MODEL IF NOT EXISTS `assignment_4_training_dataset.MY_MODEL_TRAINING`

OPTIONS(
  MODEL_TYPE = 'KMEANS',
  NUM_CLUSTERS = 10,
  KMEANS_INIT_METHOD = 'RANDOM',
  DISTANCE_TYPE = 'EUCLIDEAN') AS
SELECT * FROM
`assignment_4_training_dataset.TrainingTable`
```

The status bar indicates the query is running (3.9 sec - Stage: Preprocess). Below the query editor, the 'Query results' section shows job information: Elapsed time 3 sec, Slot time consumed 0 sec, Stages 0 (Preprocess), and Training Iterations Completed: 0, Planned: 0.

Figure 92: Query Execution details when KMeans model is ready for TrainingTable[10].

The screenshot shows the Google Cloud BigQuery interface. On the left, the Explorer sidebar displays a project structure with a dataset named 'assignment\_4\_dataset' containing a table 'Table-1' and a model 'MY\_MODEL\_TRAINING'. The main area shows a query editor with the following code:

```

1 CREATE MODEL IF NOT EXISTS `assignment_4_training_dataset.MY_MODEL_TRAINING`
2 OPTIONS(
3   MODEL_TYPE = 'KMEANS',

```

The 'Execution Graph' tab is selected, showing a flowchart with three stages: 'Preprocess', 'Train', and 'Evaluate', each with a green checkmark indicating completion. The 'RESULTS' tab is also visible in the navigation bar.

Figure 93: Execution Graph details when KMeans model is ready for TrainingTable.

The screenshot shows the Google Cloud BigQuery interface. The Explorer sidebar is identical to Figure 93. The main area shows the query results with the following message:

This statement will create a new model named MY\_MODEL\_TRAINING. Depending on the type of model, this may take several hours to complete.

The 'RESULTS' tab is selected in the navigation bar.

Figure 94: Query results when KMeans model is ready for TrainingTable.

MY\_MODEL\_TRAINING

Model Details

Model ID	serverless-assignment-4-370318.assignment_4_training_dataset.MY_MODEL_TRAINING
Description	
Labels	
Date created	Dec 1, 2022, 6:01:05 PM UTC-4
Model expiration	Jan 30, 2023, 6:01:05 PM UTC-4
Date modified	Dec 1, 2022, 6:01:05 PM UTC-4
Data location	US

Training Options

Max allowed iterations	20
Actual iterations	14
Early stop	true
Min relative progress	0.01
Distance type	Euclidean
Number of clusters	10
Centroids initialization method	Random

Figure 95: Model details of MY\_MODEL\_TRAINING part-1.

MY\_MODEL\_TRAINING

Training Options

Max allowed iterations	20
Actual iterations	14
Early stop	true
Min relative progress	0.01
Distance type	Euclidean
Number of clusters	10
Centroids initialization method	Random

Figure 96: Model details of MY\_MODEL\_TRAINING part-2.

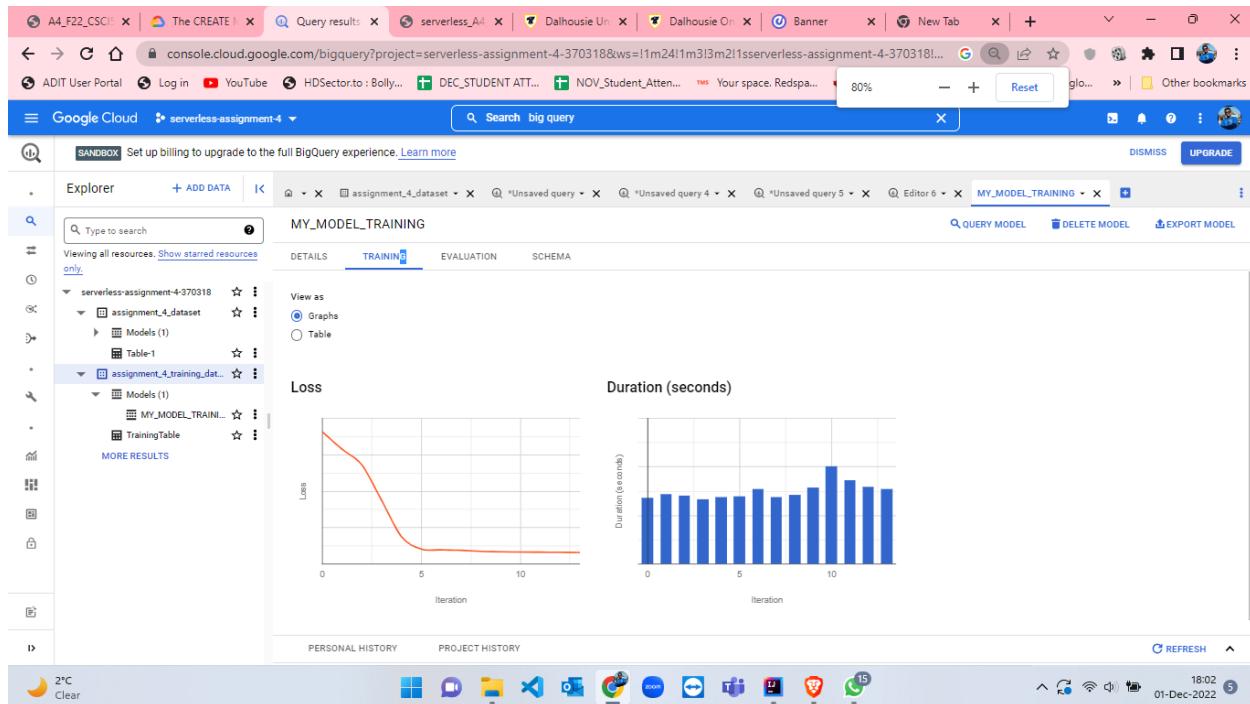


Figure 97: Model details of MY\_MODEL\_TRAINING part-3.

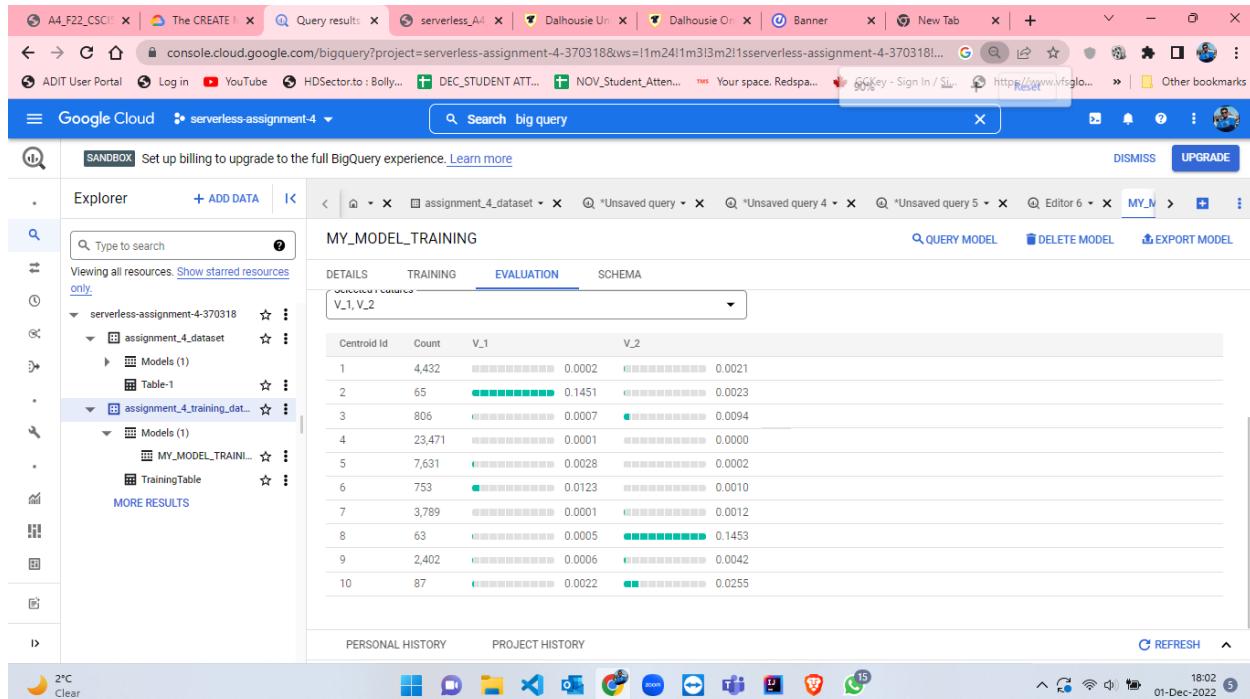
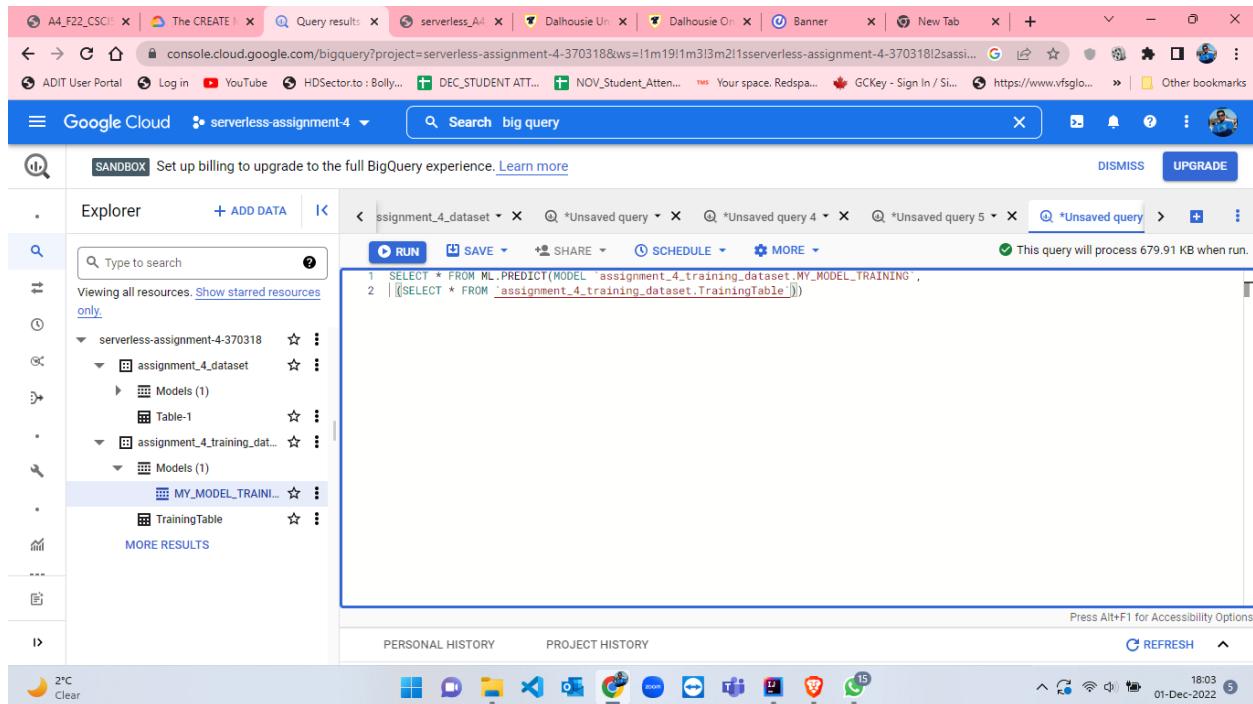


Figure 98: Model details of MY\_MODEL\_TRAINING part-4.



The screenshot shows the Google Cloud BigQuery interface. The left sidebar displays the project structure, including the 'assignment\_4\_dataset' and its 'Models' and 'Table-1'. The main query editor window contains the following SQL code:

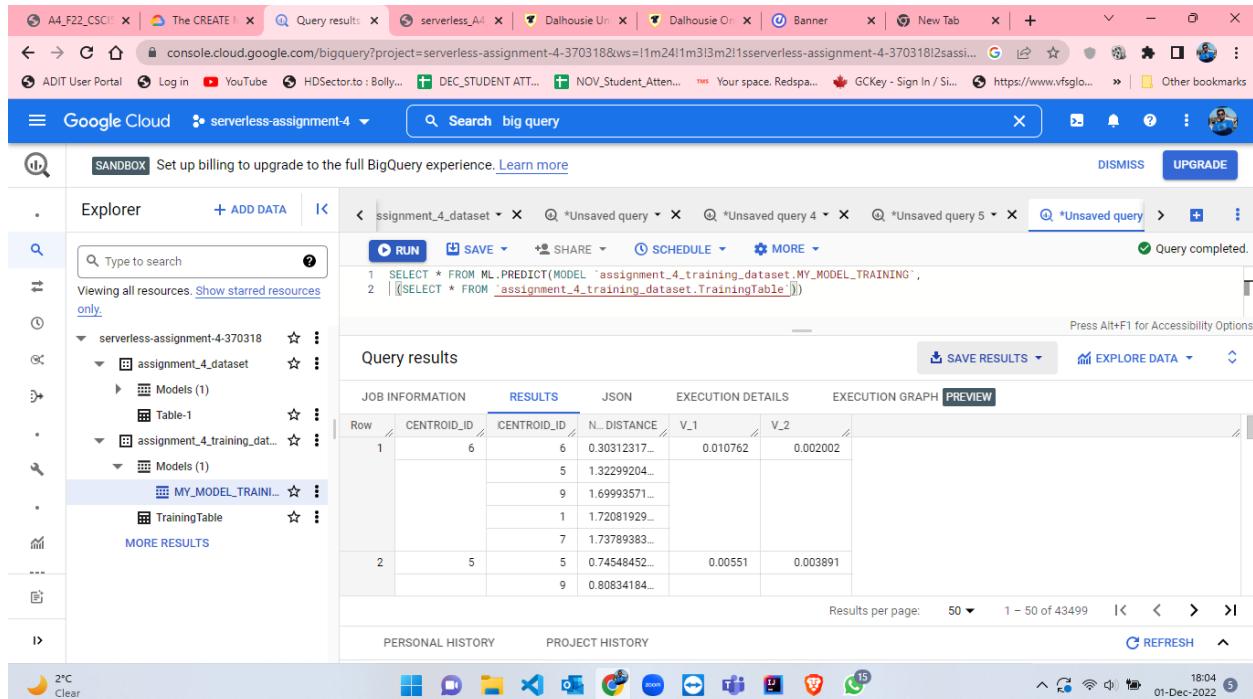
```

1 SELECT * FROM ML.PREDICT(MODEL `assignment_4_training_dataset.MY_MODEL_TRAINING` ,
2 | (SELECT * FROM `assignment_4_training_dataset.TrainingTable`))

```

The status bar at the bottom right indicates '18:03 01-Dec-2022'.

Figure 99:ML.PREDICT query.



The screenshot shows the Google Cloud BigQuery interface after the query has completed. The status bar at the bottom right indicates '18:04 01-Dec-2022'. The results table is titled 'Query results' and contains the following data:

Row	CENTROID_ID	CENTROID_ID	N_DISTANCE	V_1	V_2
1	6	6	0.30312317...	0.010762	0.002002
		5	1.32299204...		
		9	1.69993571...		
		1	1.72081929...		
		7	1.73789383...		
2	5	5	0.74548452...	0.00551	0.003891
		9	0.80834184...		

Figure 100: Results from ML.PREDICT query.

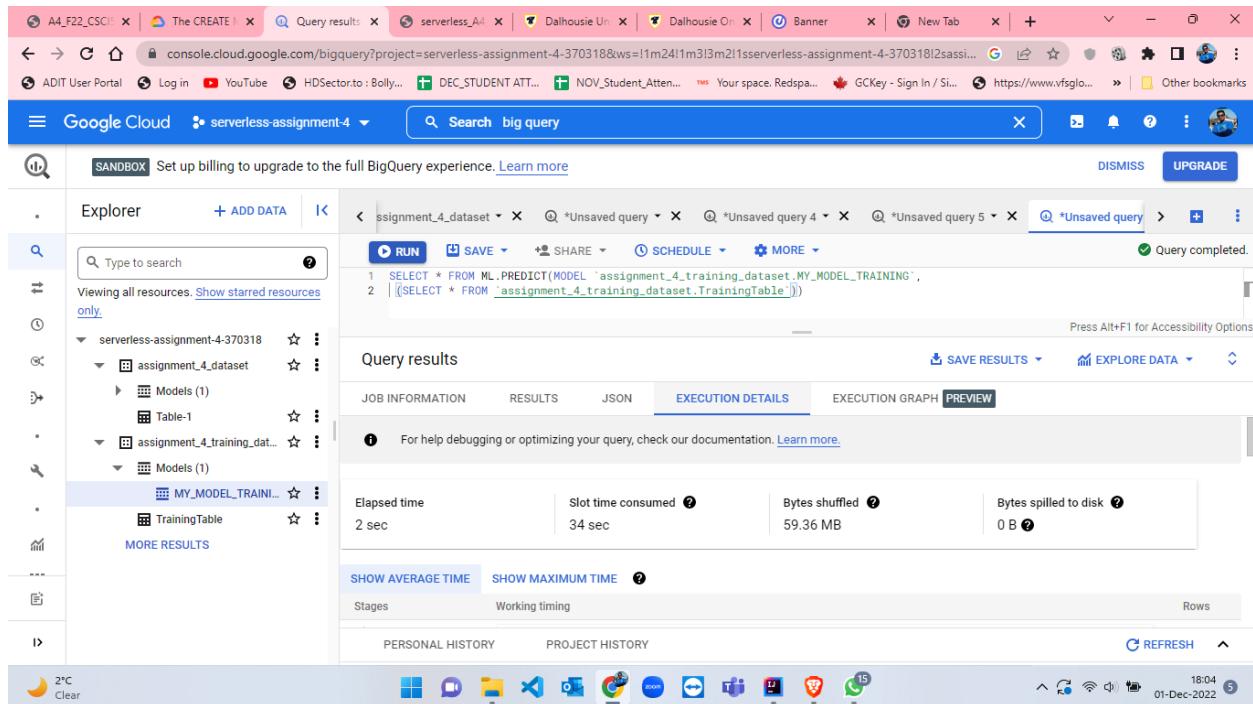


Figure 101: Execution Details from ML.PREDICT query part-1.

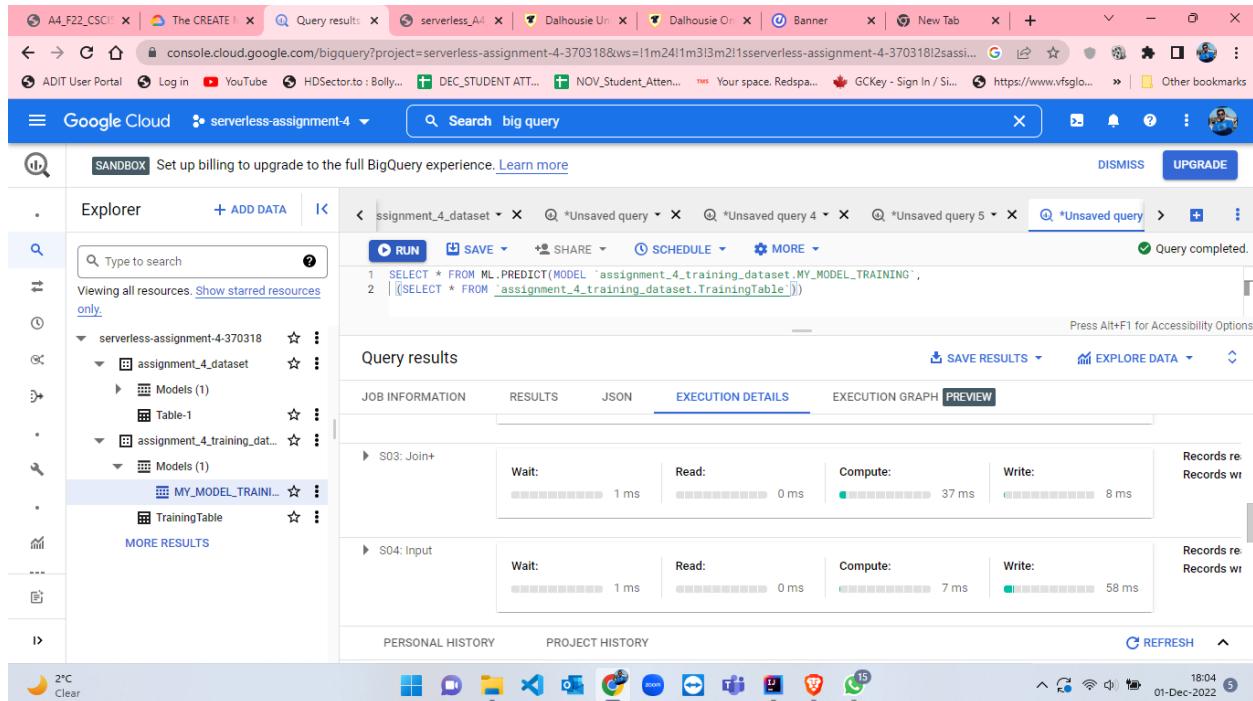
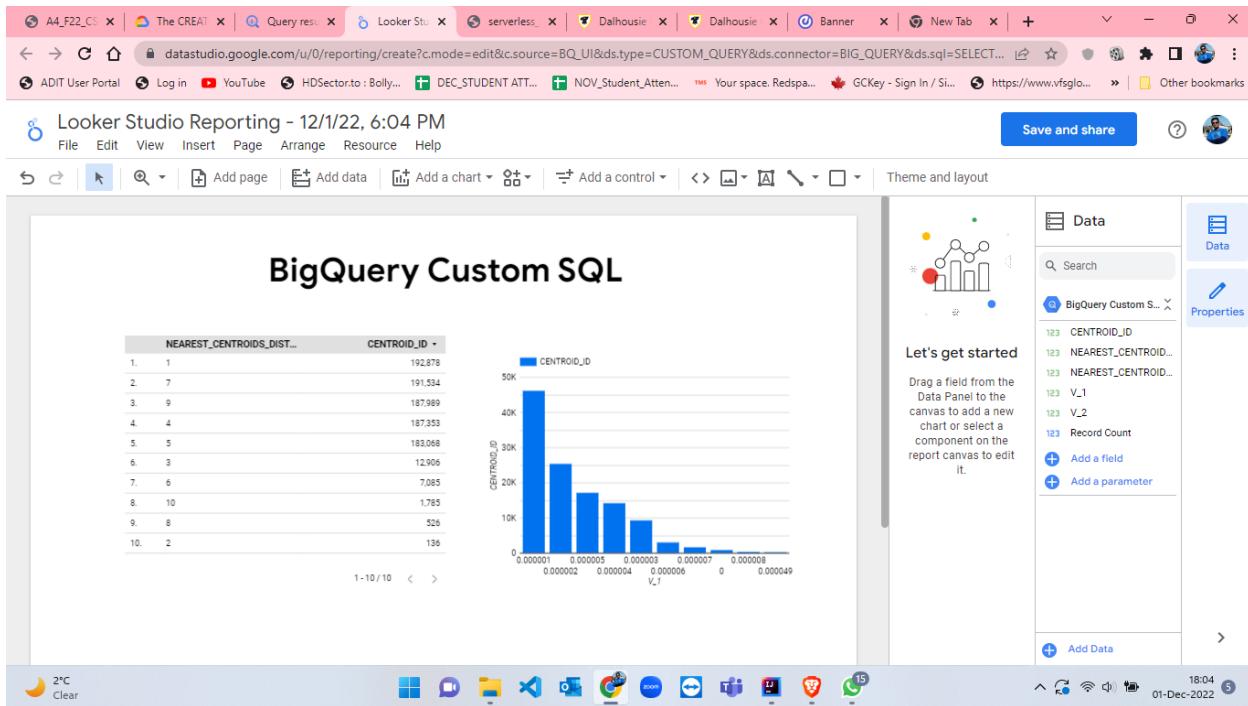


Figure 102: Execution Details from ML.PREDICT query part-2.



The screenshot shows the Looker Studio Reporting interface. The title bar reads "Looker Studio Reporting - 12/1/22, 6:04 PM". The menu bar includes File, Edit, View, Insert, Page, Arrange, Resource, and Help. The toolbar below the menu bar includes "Save and share", "Theme and layout", and various navigation and search icons. The main canvas displays a title "BigQuery Custom SQL" and a bar chart titled "CENTROID\_ID". The chart shows the distribution of CENTROID\_ID values, with the y-axis labeled "CENTROID\_ID" and the x-axis labeled "V\_1". The data table below the chart lists "NEAREST\_CENTROIDS\_DIST..." and "CENTROID\_ID" for 10 rows. The data panel on the right shows fields like CENTROID\_ID, NEAREST\_CENTROID\_ID, V\_1, V\_2, and Record Count, along with buttons for "Add a field" and "Add a parameter". The properties panel on the far right is titled "Properties". The bottom of the screen shows a taskbar with various icons and the system clock "18:04 01-Dec-2022".

NEAREST_CENTROIDS_DIST...	CENTROID_ID
1.	192,878
2.	191,534
3.	187,989
4.	187,353
5.	183,068
6.	12,906
7.	7,085
8.	1,785
9.	526
10.	136

Figure 103: Looker Studio Reporting for visualization.

- Now, we have to create a random sampling with 25% of the records from the given csv file as training set. I have created this sampling by using excel and by manual sampling. Dataset creation, table creation, model training process and testing along with evaluation of model and ML prediction is attached below from figure-104 to figure-119. This is a similar process to what we have done for 100% and 75% of the data.

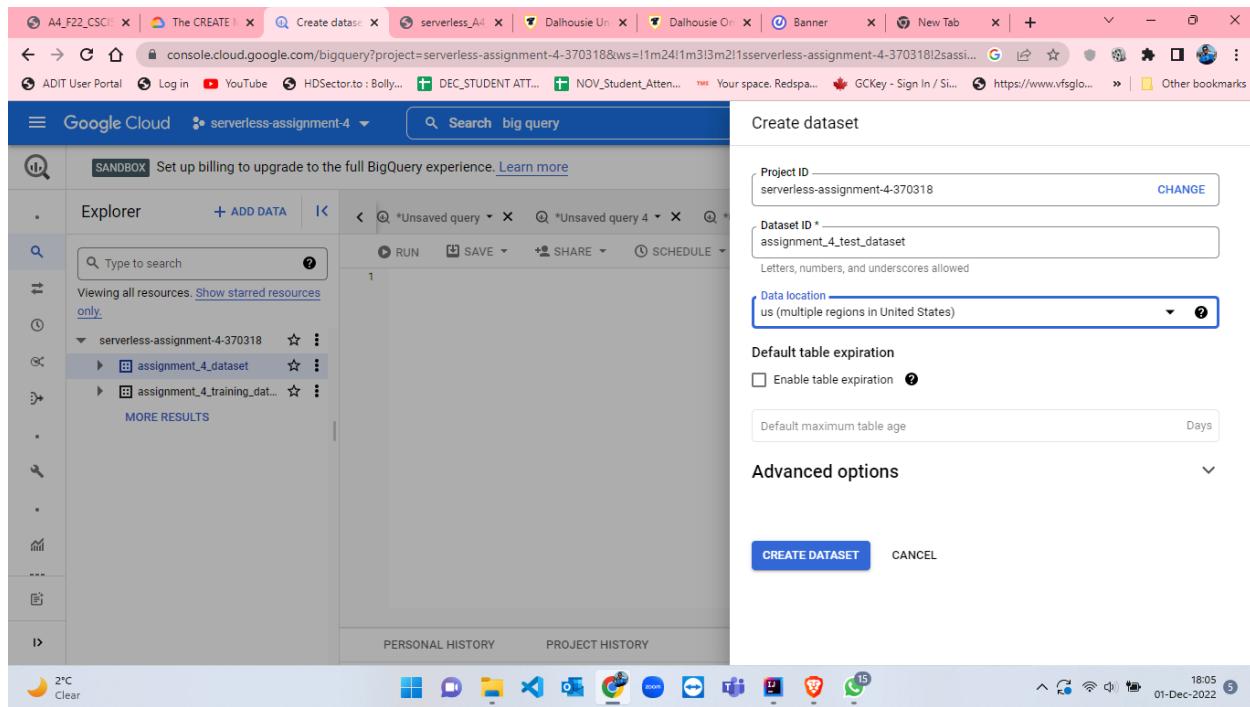


Figure 104: Create dataset named as *test\_dataset*.

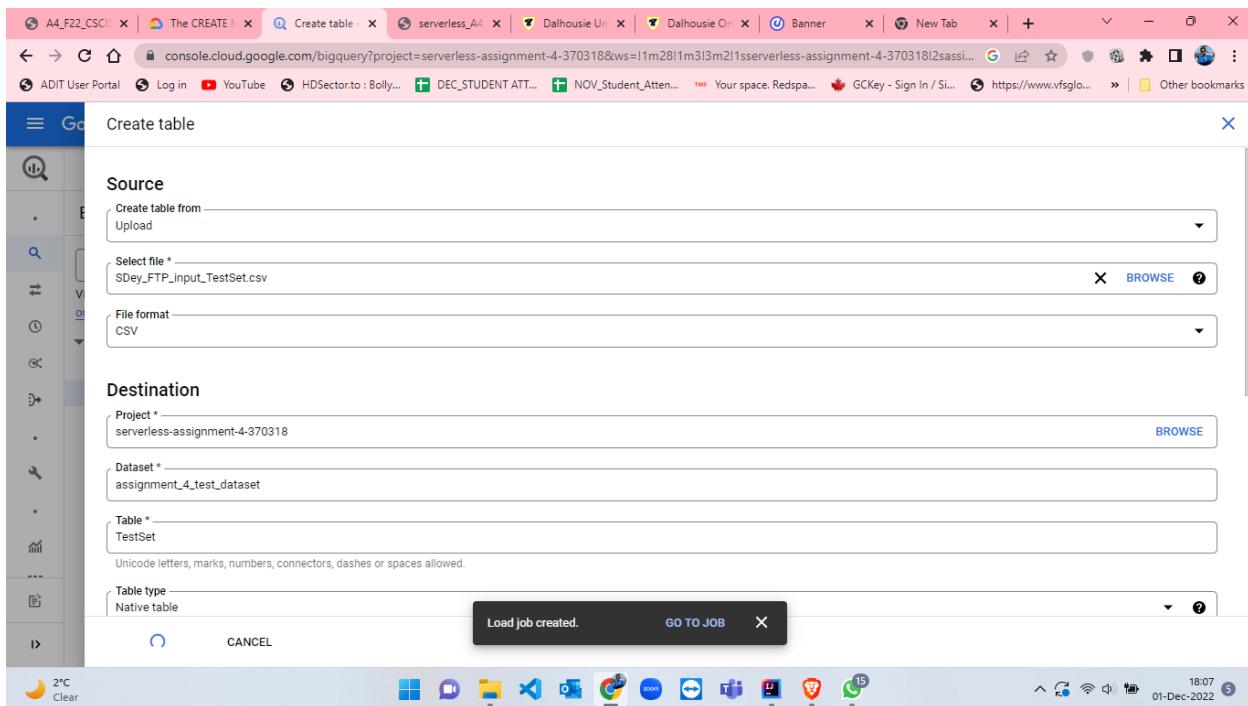


Figure 105: Create table for test named as TestSet having 25% of records.

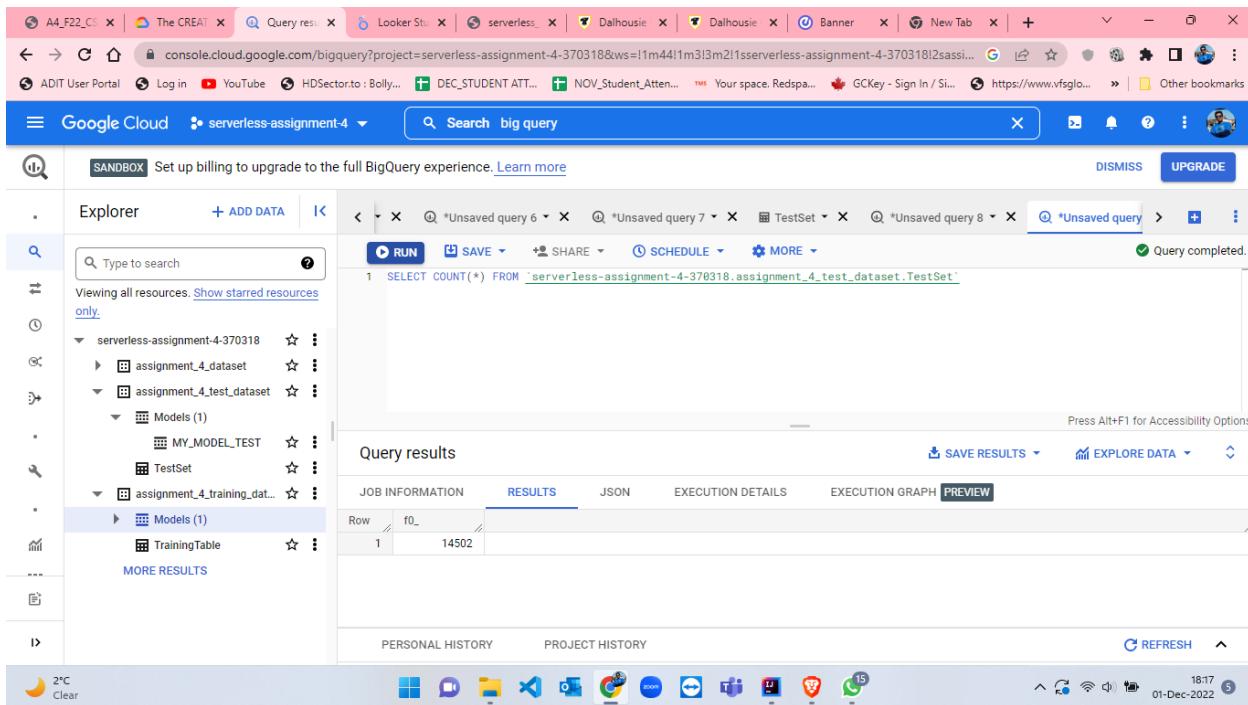


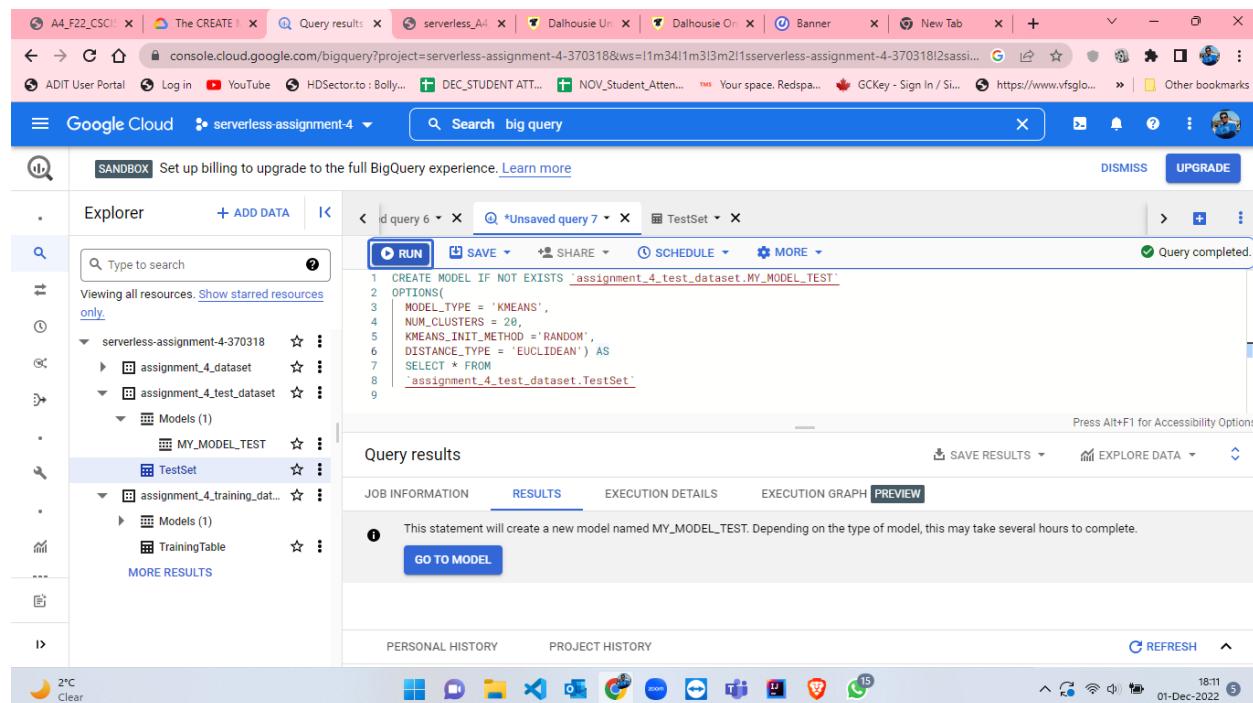
Figure 106: TestSet table number of rows.

```

CREATE MODEL IF NOT EXISTS `assignment_4_test_dataset.MY_MODEL_TEST`
OPTIONS(
  MODEL_TYPE = 'KMEANS',
  NUM_CLUSTERS = 20,
  KMEANS_INIT_METHOD = 'RANDOM',
  DISTANCE_TYPE = 'EUCLIDEAN') AS
SELECT * FROM
`assignment_4_test_dataset.TestSet`

```

- Here, for test model, I have made some changes while creating model[10]. I have set the NUM\_CLUSTERS as 20 which was 10 in training model, KMEANS\_INIT\_METHOD as RANDOM and DISTANCE\_TYPE as EUCLIDEAN.



The screenshot shows the Google Cloud BigQuery interface. The left sidebar displays a project structure with datasets like 'serverless-assignment-4-370318', 'assignment\_4\_dataset', and 'assignment\_4\_test\_dataset'. The 'TestSet' table is selected. The main area shows a query editor with the following SQL code:

```

CREATE MODEL IF NOT EXISTS `assignment_4_test_dataset.MY_MODEL_TEST`
OPTIONS(
  MODEL_TYPE = 'KMEANS',
  NUM_CLUSTERS = 20,
  KMEANS_INIT_METHOD = 'RANDOM',
  DISTANCE_TYPE = 'EUCLIDEAN') AS
SELECT * FROM
`assignment_4_test_dataset.TestSet`

```

The 'RUN' button is highlighted, and a message 'Query completed.' is visible. Below the query editor, the 'Query results' section shows a message: 'This statement will create a new model named MY\_MODEL\_TEST. Depending on the type of model, this may take several hours to complete.' A 'GO TO MODEL' button is present. The bottom of the interface shows a toolbar with various icons and a status bar indicating the date and time.

Figure 107: Query results when KMeans model is ready for TestSet table[10].

The screenshot shows the Google Cloud BigQuery interface. The Explorer sidebar on the left lists datasets and models. The main area shows a query editor with the following code:

```

1 CREATE MODEL IF NOT EXISTS `assignment_4_test_dataset.MY_MODEL_TEST`;
2 OPTIONS(
3   MODEL_TYPE = 'KMEANS',
4   NUM_CLUSTERS = 28,
5   KMFANS_TNTT_METHOD = 'RANDOM'

```

The results table shows the following data:

Elapsed time	Slot time consumed	Stages	Training Iterations
2 min 27 sec	19 min 8 sec	Preprocess: 0 Train: 0 Evaluate: 0	Completed: 20 Planned: 20

Figure 108: Execution details when KMeans model is ready for TestSet table.

The screenshot shows the Google Cloud BigQuery interface with the execution graph tab selected. The graph visualizes the workflow with three stages: Preprocess, Train, and Evaluate, each marked with a green checkmark.

Figure 109: Execution Graph when KMeans model is ready for TestSet table.

MY\_MODEL\_TEST

**Model Details**

Model ID: serverless-assignment-4-370318.assignment\_4\_test\_dataset.MY\_MODEL\_TEST

Description:

Labels:

Date created: Dec 1, 2022, 6:10:55 PM UTC-4

Model expiration: Jan 30, 2023, 6:10:55 PM UTC-4

Date modified: Dec 1, 2022, 6:10:55 PM UTC-4

Data location: US

Model type: KMEANS

Figure 110: Model details of MY\_MODEL\_TEST part-1.

**Training Options**

Training options are the optional parameters that were added in the script to create this model.

Max allowed iterations: 20

Actual iterations: 20

Early stop: true

Min relative progress: 0.01

Distance type: Euclidean

Number of clusters: 20

Centroids initialization method: Random

Figure 111: Model details of MY\_MODEL\_TEST part-2.

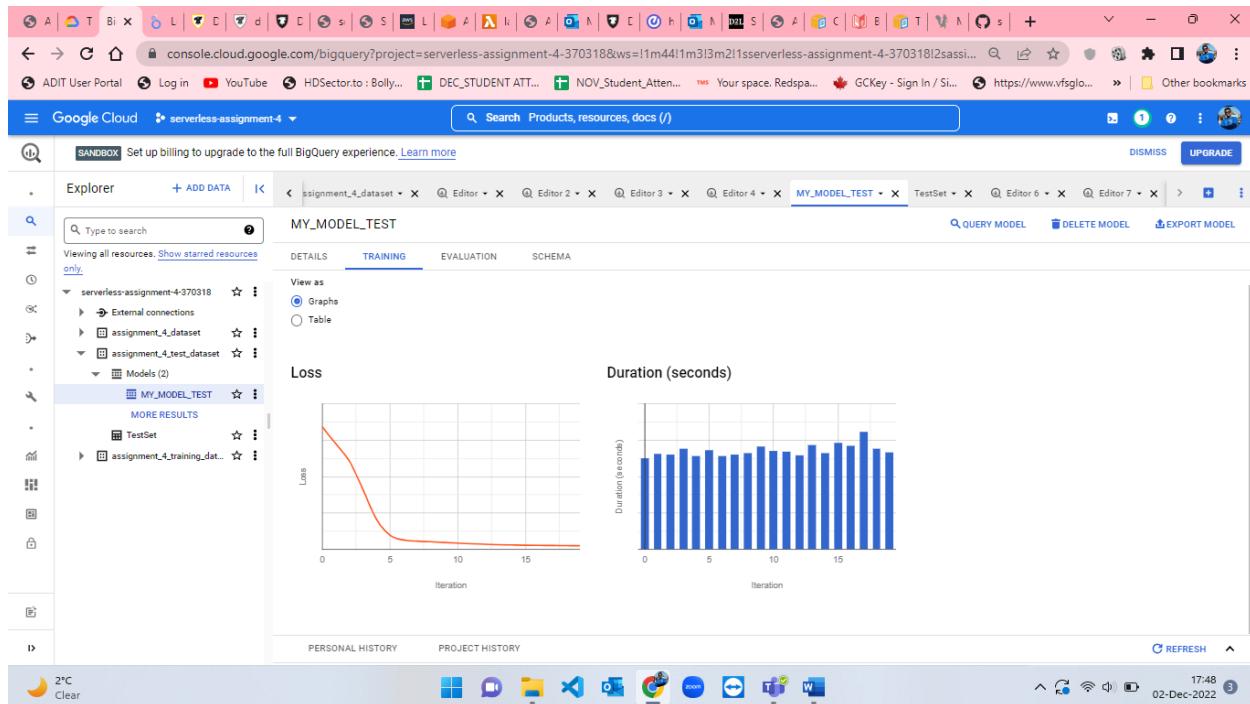


Figure 112: Model details of MY\_MODEL\_TEST part-3.

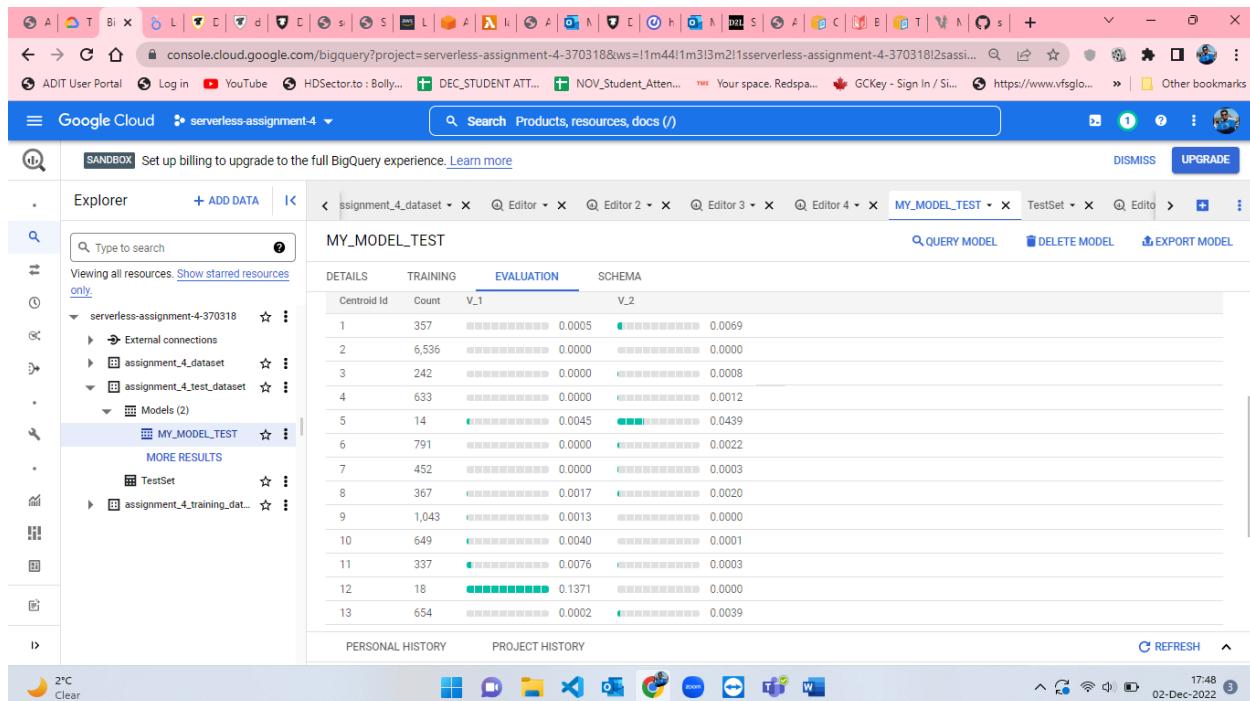


Figure 113: Model details of MY\_MODEL\_TEST part-4.

DETAILS	TRAINING	EVALUATION	SCHEMA
6	791	0.0000	0.0022
7	452	0.0000	0.0003
8	367	0.0017	0.0020
9	1,043	0.0013	0.0000
10	649	0.0040	0.0001
11	337	0.0076	0.0003
12	18	0.1371	0.0000
13	654	0.0002	0.0039
14	157	0.0008	0.0130
15	18	0.0004	0.1371
16	118	0.0142	0.0014
17	14	0.0439	0.0010
18	715	0.0000	0.0017
19	160	0.0046	0.0034

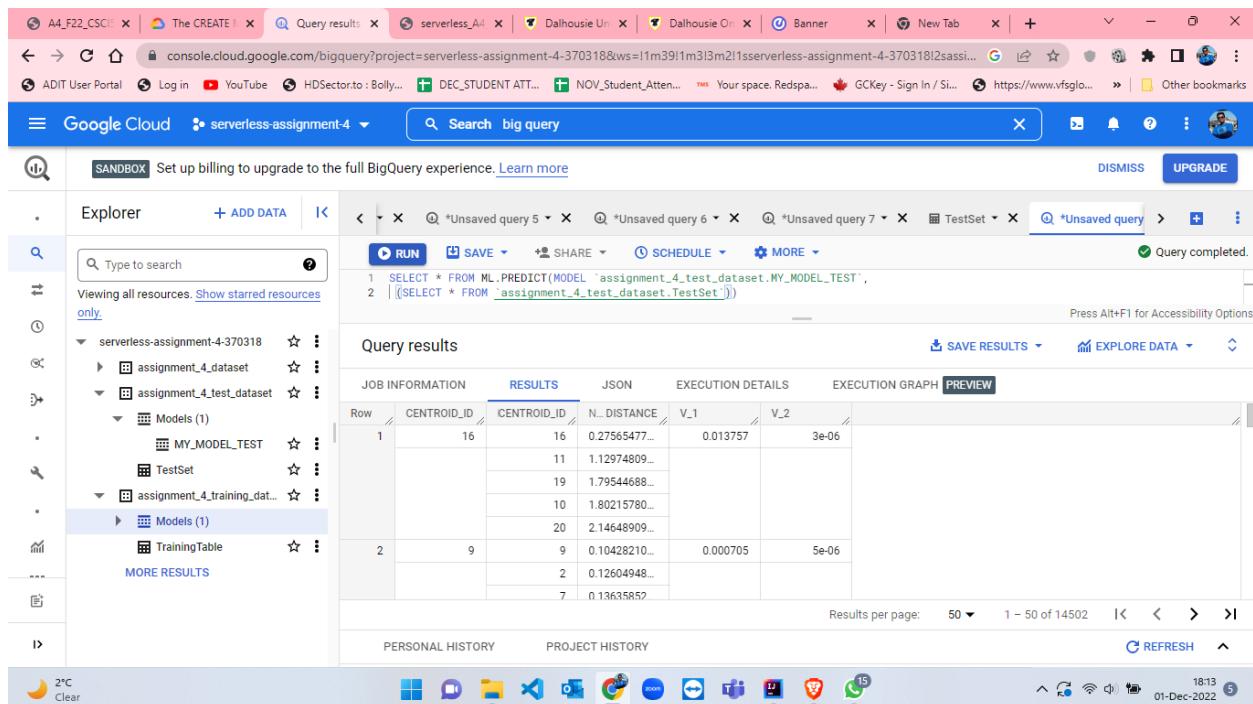
Figure 114: Model details of MY\_MODEL\_TEST part-5.

```

1 SELECT * FROM ML.PREDICT(MODEL assignment_4_test_dataset.MY_MODEL_TEST,
2 | (SELECT * FROM assignment_4_test_dataset.TestSet))

```

Figure 115:ML.PREDICT query for TestSet Table.



The screenshot shows the Google BigQuery interface. The left sidebar displays the project structure, including datasets like assignment\_4\_dataset and assignment\_4\_test\_dataset, and models like MY\_MODEL\_TEST and TestSet. The main area shows the results of a query. The query is:

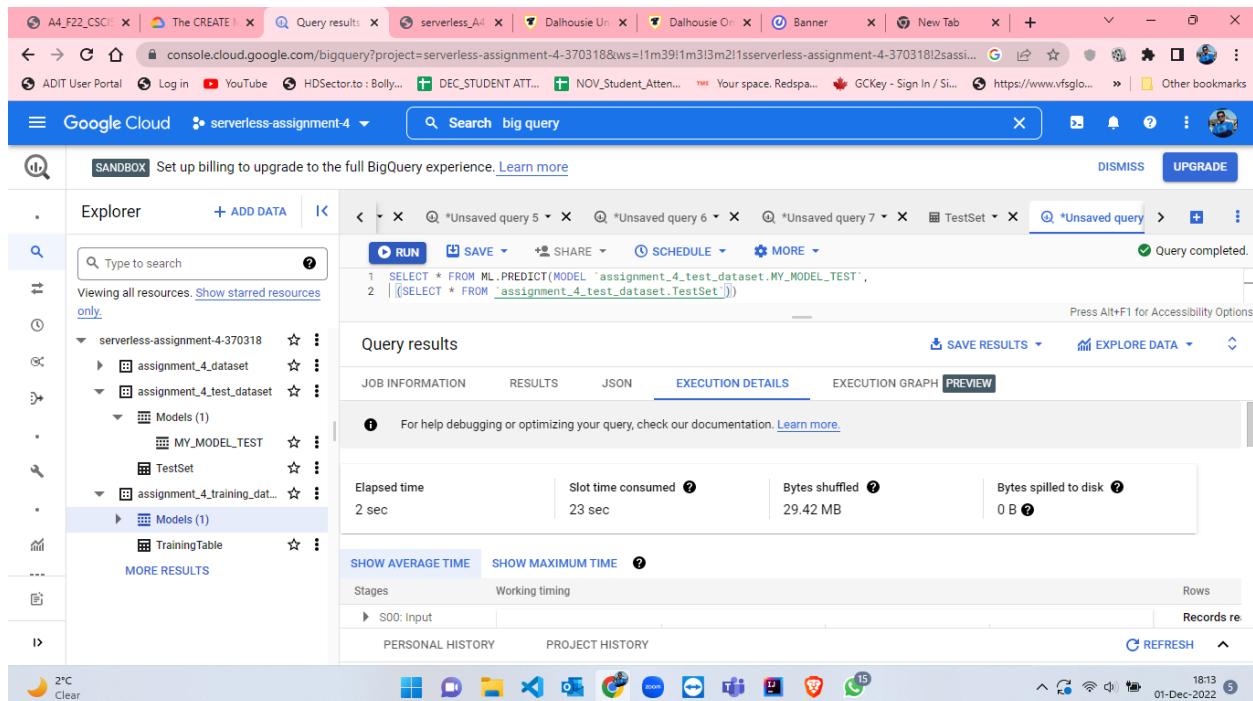
```
1 SELECT * FROM ML.PREDICT(MODEL 'assignment_4_test_dataset.MY_MODEL_TEST',
2 | (SELECT * FROM `assignment_4_test_dataset.TestSet`))
```

The results table has columns: Row, CENTROID\_ID, CENTROID\_ID, N...\_DISTANCE, V\_1, and V\_2. The data is as follows:

Row	CENTROID_ID	CENTROID_ID	N..._DISTANCE	V_1	V_2
1	16	16	0.27565477...	0.013757	3e-06
		11	1.12974809...		
		19	1.79544688...		
		10	1.80215780...		
		20	2.14648909...		
2	9	9	0.10428210...	0.000705	5e-06
		2	0.12604948...		
		7	0.13635852		

Below the table, it says "Results per page: 50 1 – 50 of 14502".

Figure 116: Results from ML.PREDICT query for TestSet table.



The screenshot shows the Google BigQuery interface. The left sidebar displays the project structure, including datasets like assignment\_4\_dataset and assignment\_4\_test\_dataset, and models like MY\_MODEL\_TEST and TestSet. The main area shows the execution details of a query. The query is:

```
1 SELECT * FROM ML.PREDICT(MODEL 'assignment_4_test_dataset.MY_MODEL_TEST',
2 | (SELECT * FROM `assignment_4_test_dataset.TestSet`))
```

The execution details table has columns: Elapsed time, Slot time consumed, Bytes shuffled, and Bytes spilled to disk. The data is as follows:

Elapsed time	Slot time consumed	Bytes shuffled	Bytes spilled to disk
2 sec	23 sec	29.42 MB	0 B

Below the table, it says "For help debugging or optimizing your query, check our documentation. [Learn more](#)".

Figure 117: Execution details from ML.PREDICT query for TestSet table.

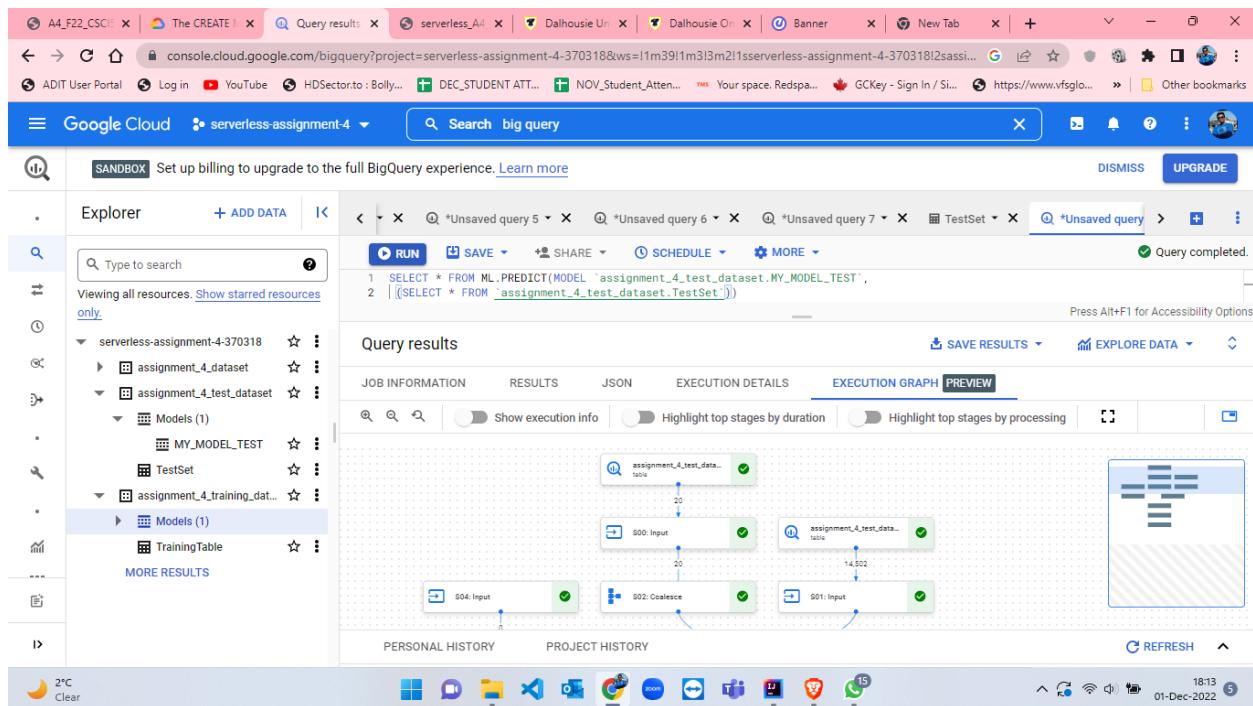


Figure 118: Execution Graph from ML.PREDICT query for TestSet table.

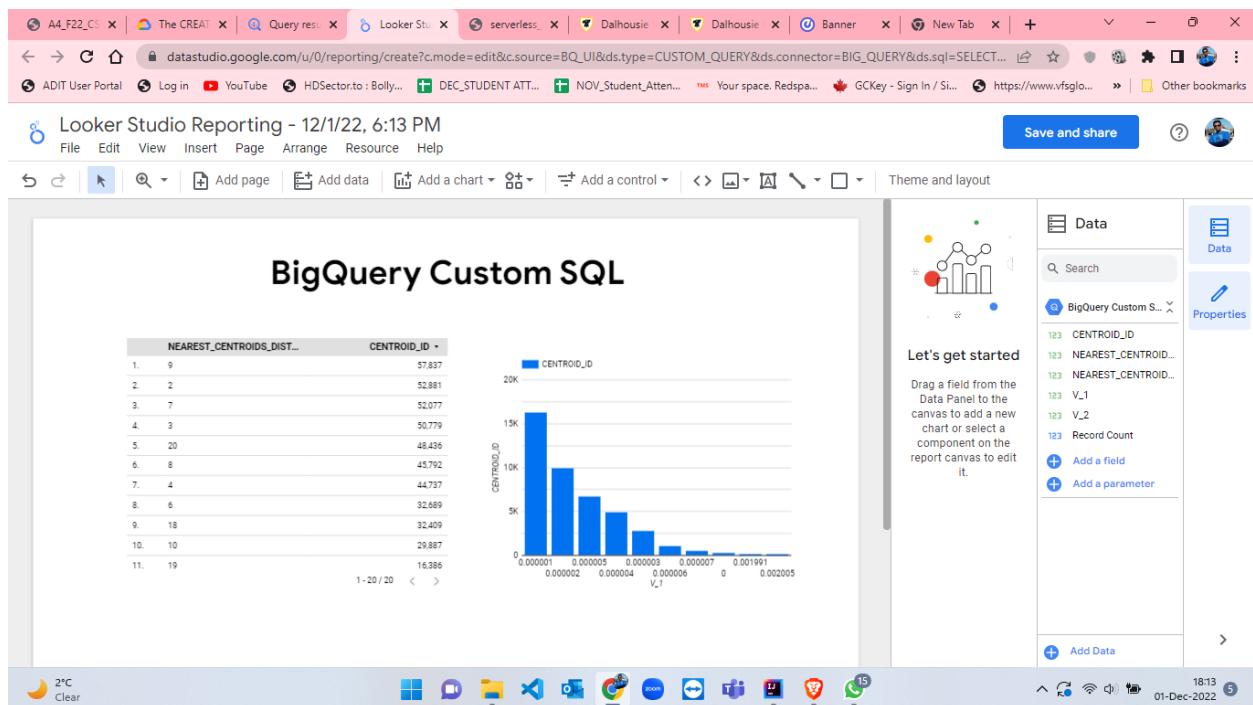


Figure 119: Looker Studio Reporting for visualization for TestSet table.

**References:**

- [1] Amazon Web Services, Inc., “Create a queue (console)”, Amazon Web Services, Inc., 2022 [Online]. Available: <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/step-create-queue.html> [Accessed: 01-Dec-2022].
- [2] “How to generate random numbers in Java,” Eduative: Interactive Courses for Software Developers, 2022 [Online]. <https://www.educative.io/answers/how-to-generate-random-numbers-in-java> [Accessed: 01-Dec-2022].
- [3] Amazon Web Services, Inc., “Sending, Receiving, and Deleting Amazon SQS Messages”, Amazon Web Services, Inc., 2022 [Online]. Available: <https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/examples-sqs-messages.html> [Accessed: 01-Dec-2022].
- [4] Baeldung, “Java System.getProperty vs System.getenv | Baeldung,” www.baeldung.com, Jun. 21, 2018 [Online]. Available: <https://www.baeldung.com/java-system-get-property-vs-system-getenv> [Accessed: 01-Dec-2022].
- [5] Amazon Web Services, Inc., “Creating an Amazon SNS topic”, Amazon Web Services, Inc., 2022 [Online]. Available: <https://docs.aws.amazon.com/sns/latest/dg/sns-create-topic.html> [Accessed: 01-Dec-2022].
- [6] Amazon Web Services, Inc., “Using Lambda with Amazon SQS”, Amazon Web Services, Inc., 2022 [Online]. Available: <https://docs.aws.amazon.com/lambda/latest/dg/with-sqs.html> [Accessed: 01-Dec-2022].
- [7] Amazon Web Services, Inc., “Tutorial: Schedule AWS Lambda functions using EventBridge”, Amazon Web Services, Inc., 2022 [Online]. Available: <https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-run-lambda-schedule.html> [Accessed: 01-Dec-2022].
- [8] R. P. Roizman, “SQS-TO-SNS-LAMBDA,” GitHub, 2022[Online]. Available: <https://github.com/panazzo/sqs-to-sns-lambda/blob/master/index.js> [Accessed: 01-Dec-2022].
- [9] “The CREATE MODEL statement for K-means models | BigQuery ML,” Google Cloud, 2022[Online]. Available: <https://cloud.google.com/bigquery-ml/docs/reference/standard-sql/bigqueryml-syntax-create-kmeans> [Accessed: 02-Dec-2022].
- [10] “Quickstart: Create machine learning models in BigQuery ML,” Google Cloud, 2022[Online]. Available: <https://cloud.google.com/bigquery-ml/docs/create-machine-learning-model> [Accessed: 02-Dec-2022].