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.

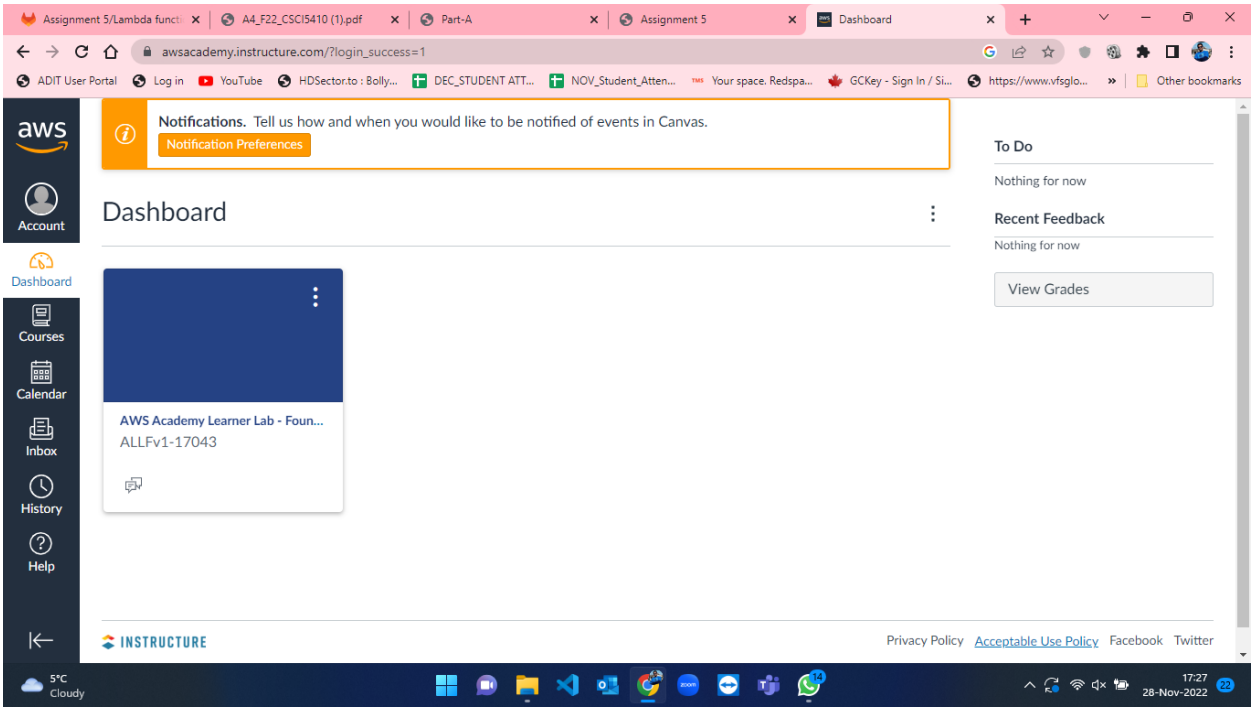

Figure 1: AWS Dashboard

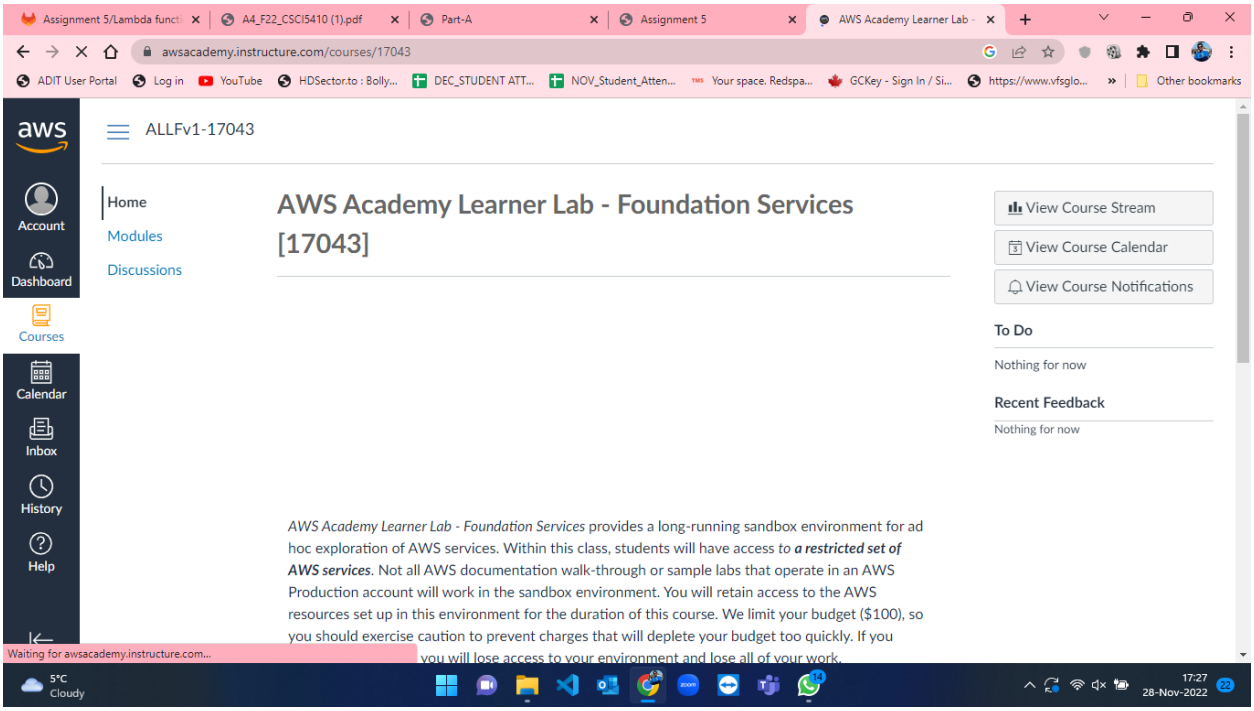

Figure 2: Homepage after course selection from dashboard in AWS.

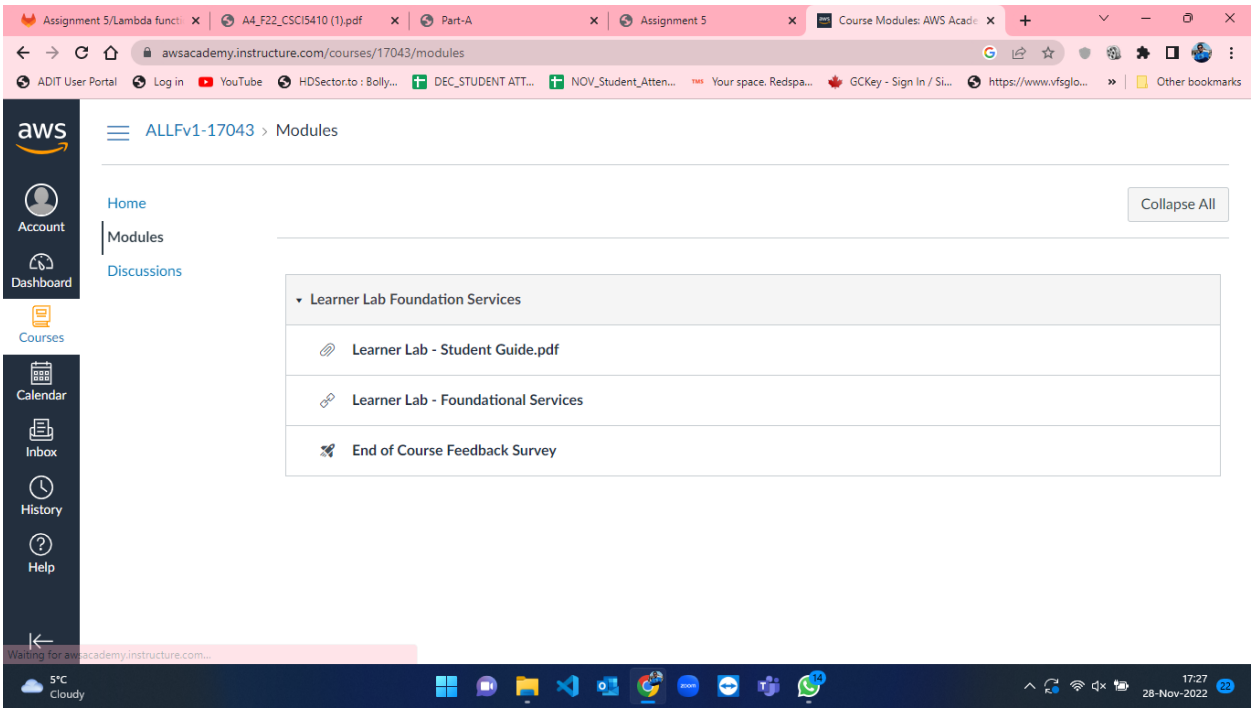

Figure 3: Modules page in AWS.

- 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.

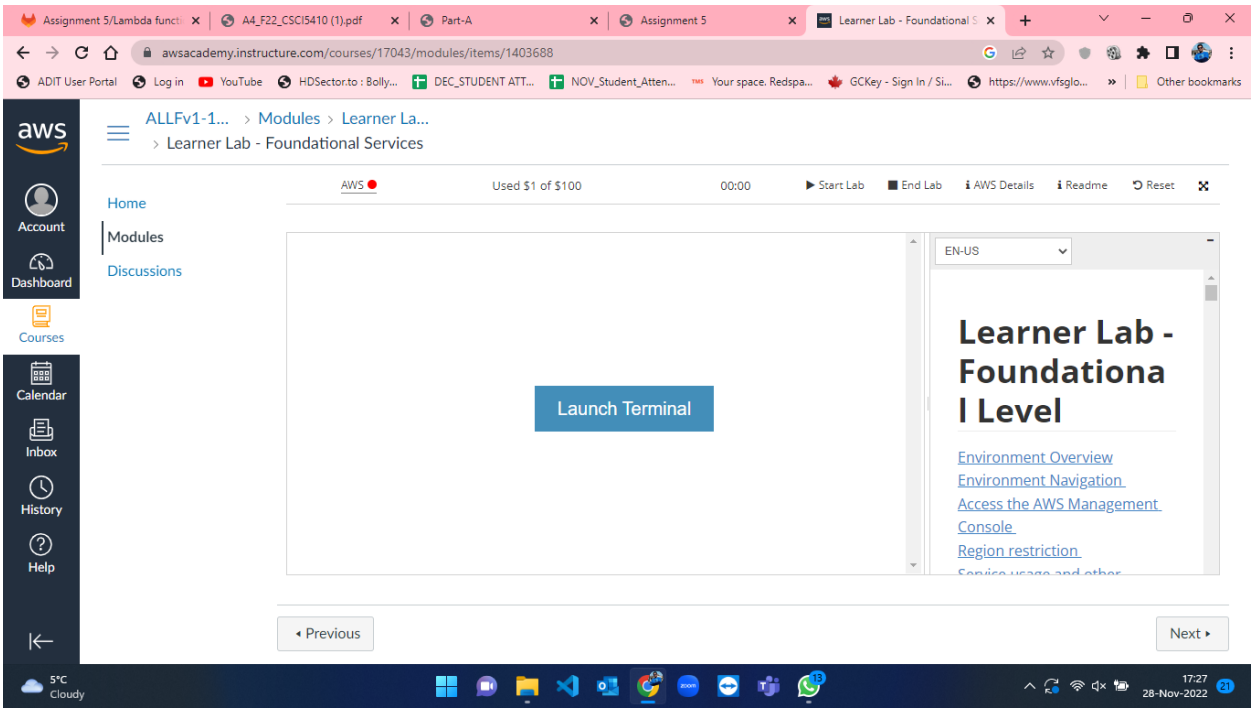

Figure 4: Lab not running in Modules page in AWS.

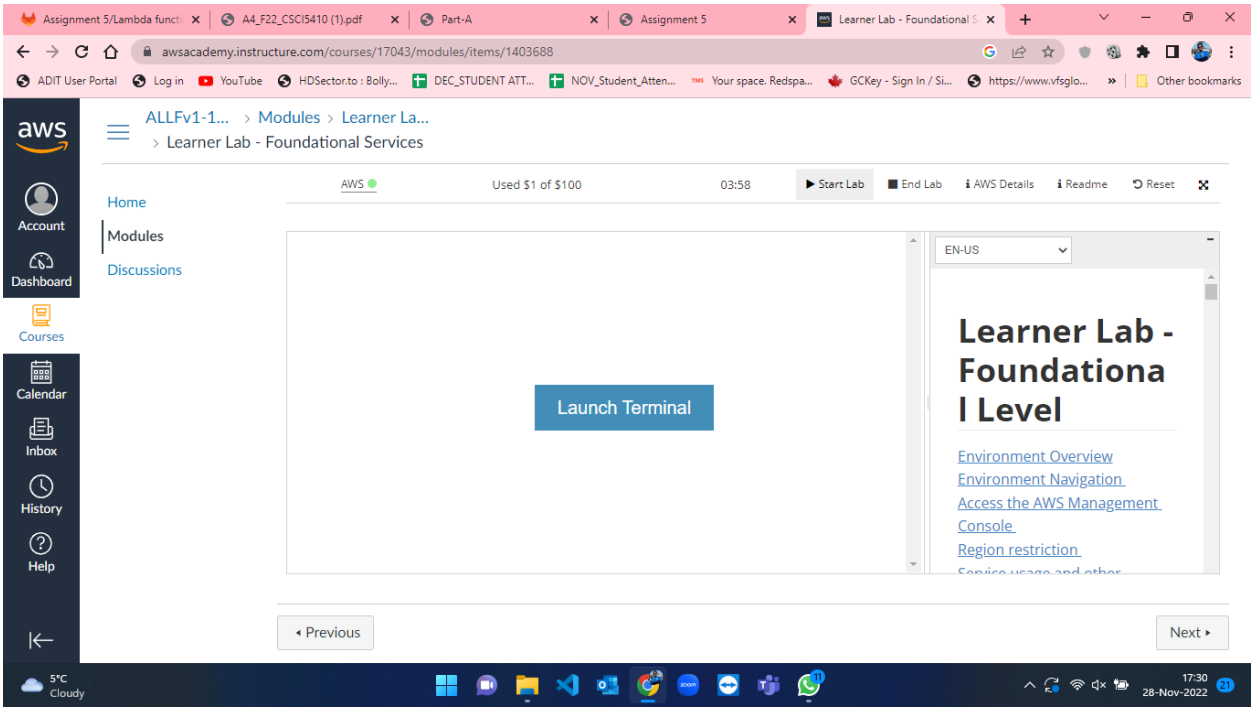

Figure 5: Lab started under Modules page in AWS.

- 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.

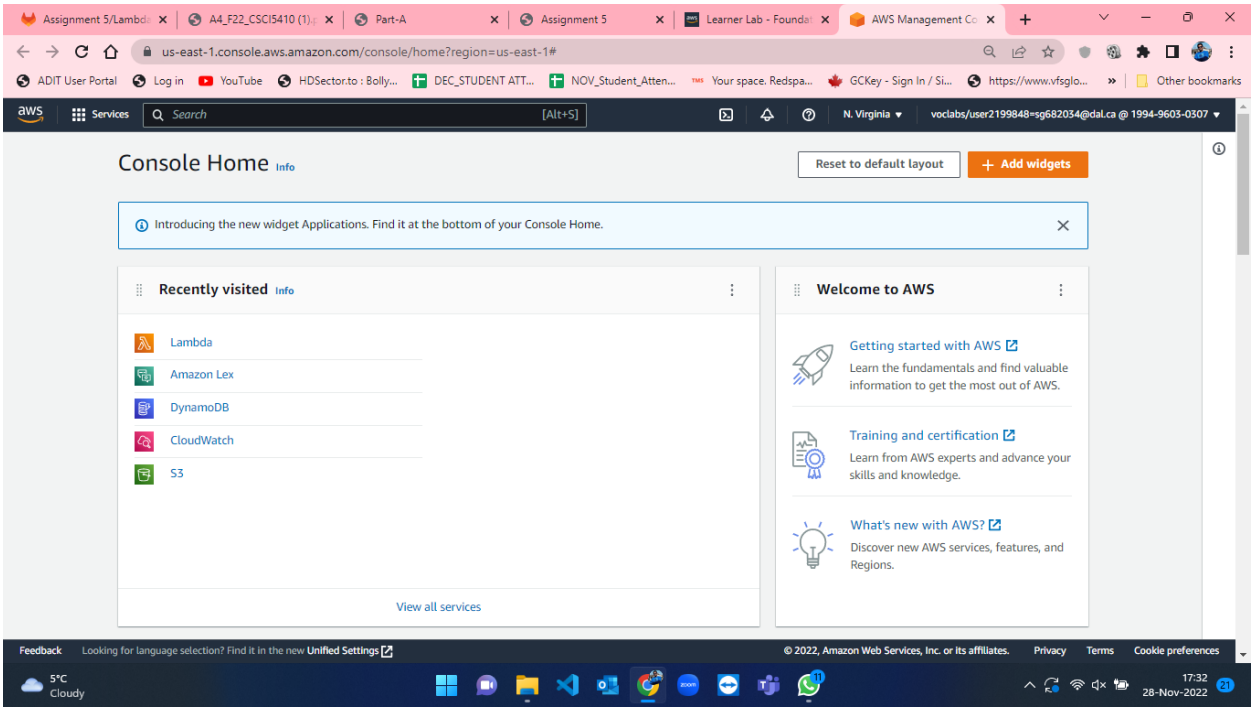

Figure 6: AWS Console Home.

- 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.

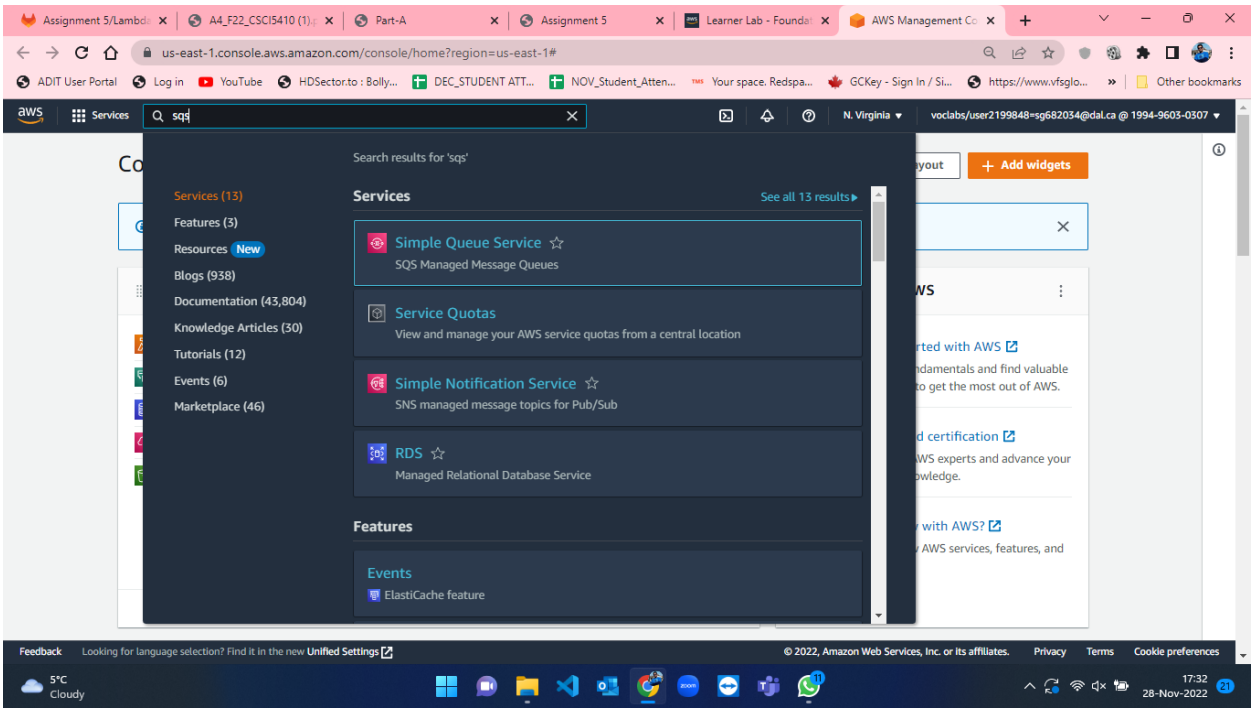

Figure 7: Searching for SQS in search bar from Console Home in AWS.

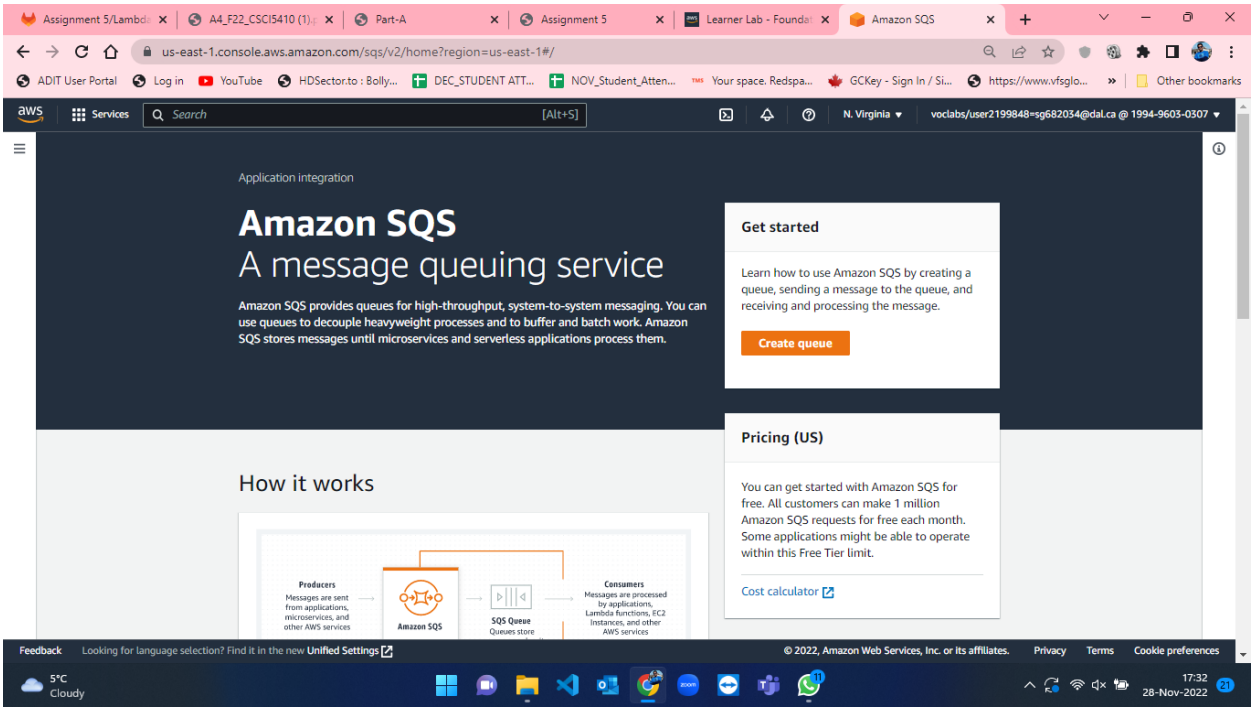

Figure 8: SQS homepage in AWS Console.

- 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 “assignmet4”[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 gifure-9 to figure-13.

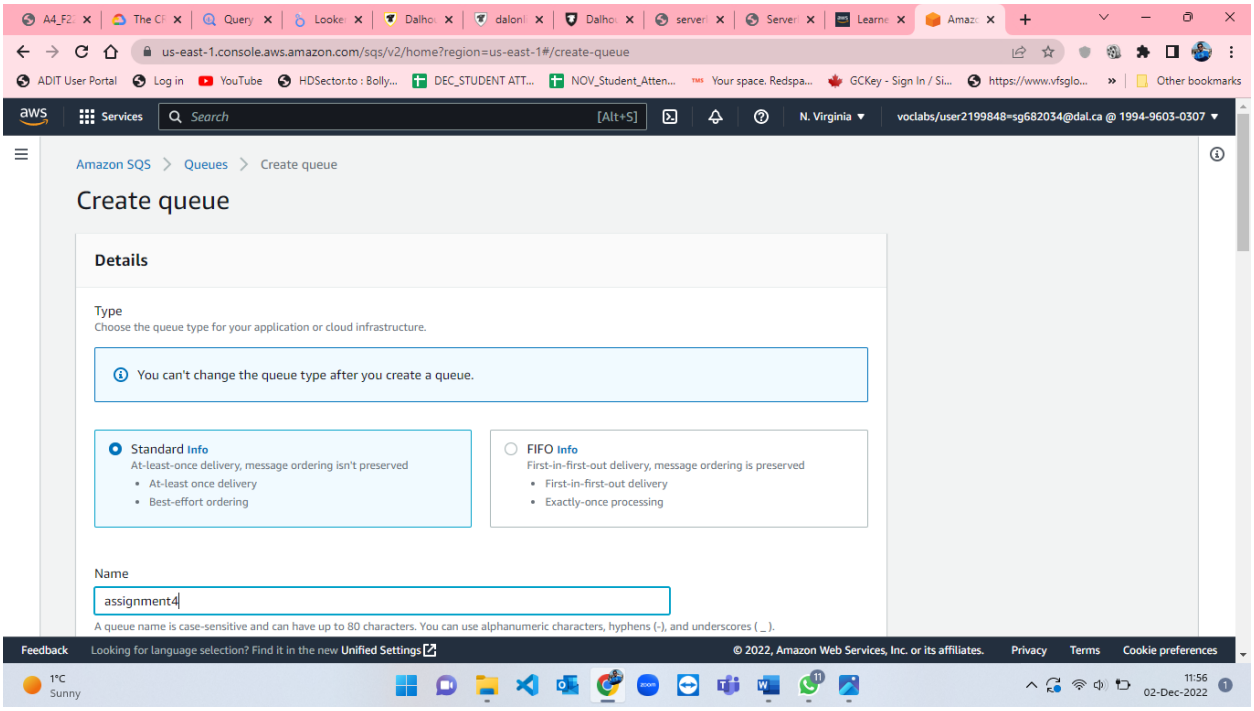

Figure 9: SQS create queue page – 1[1].

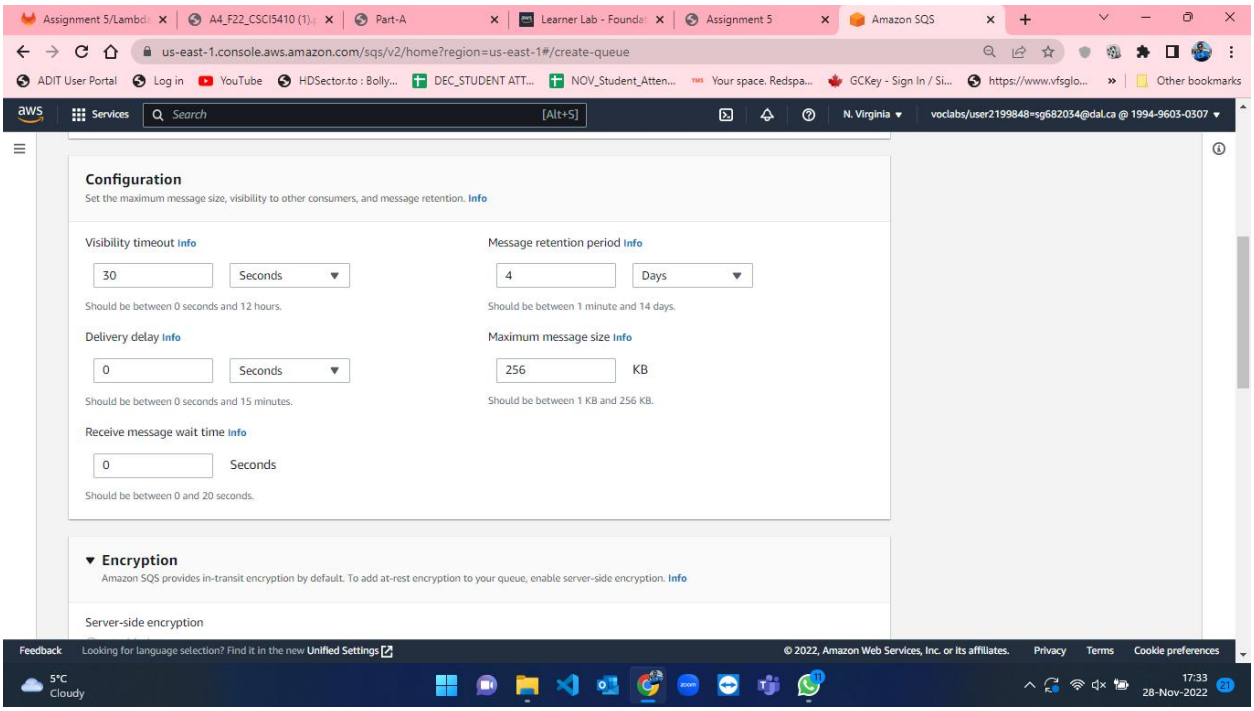

Figure 10: SQS create queue page – 2[1].

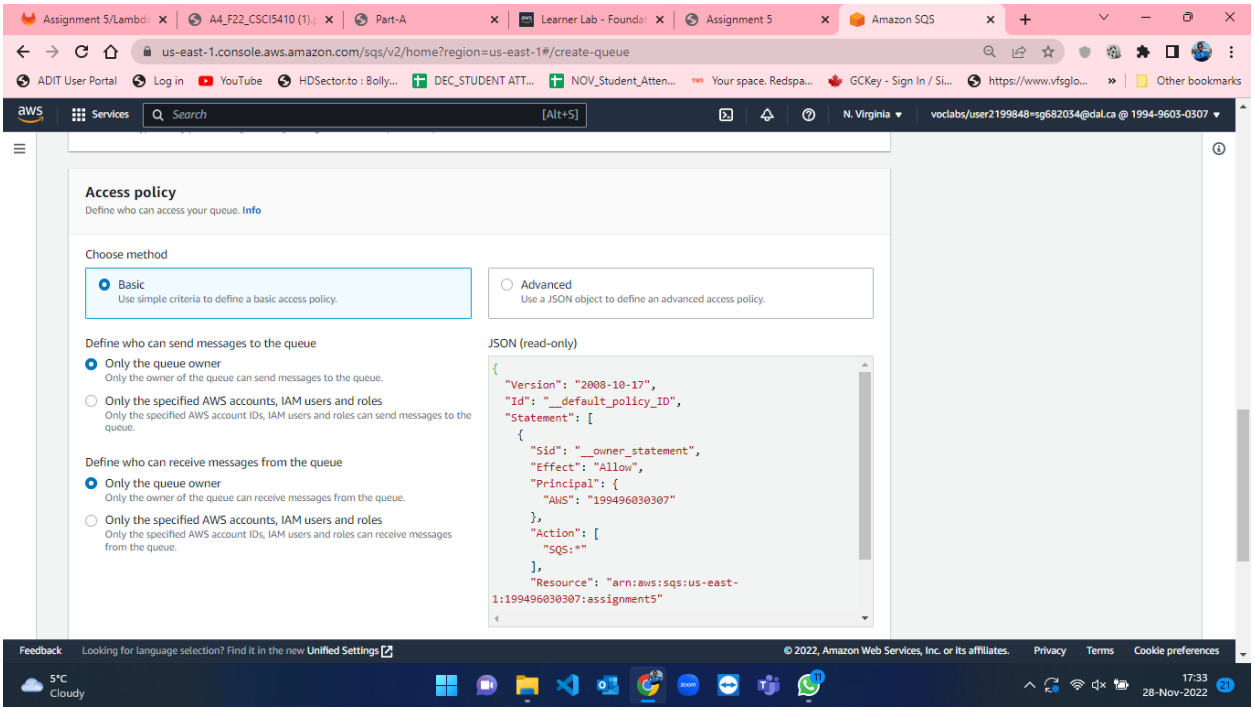

Figure 11: SQS create queue page – 3[1].

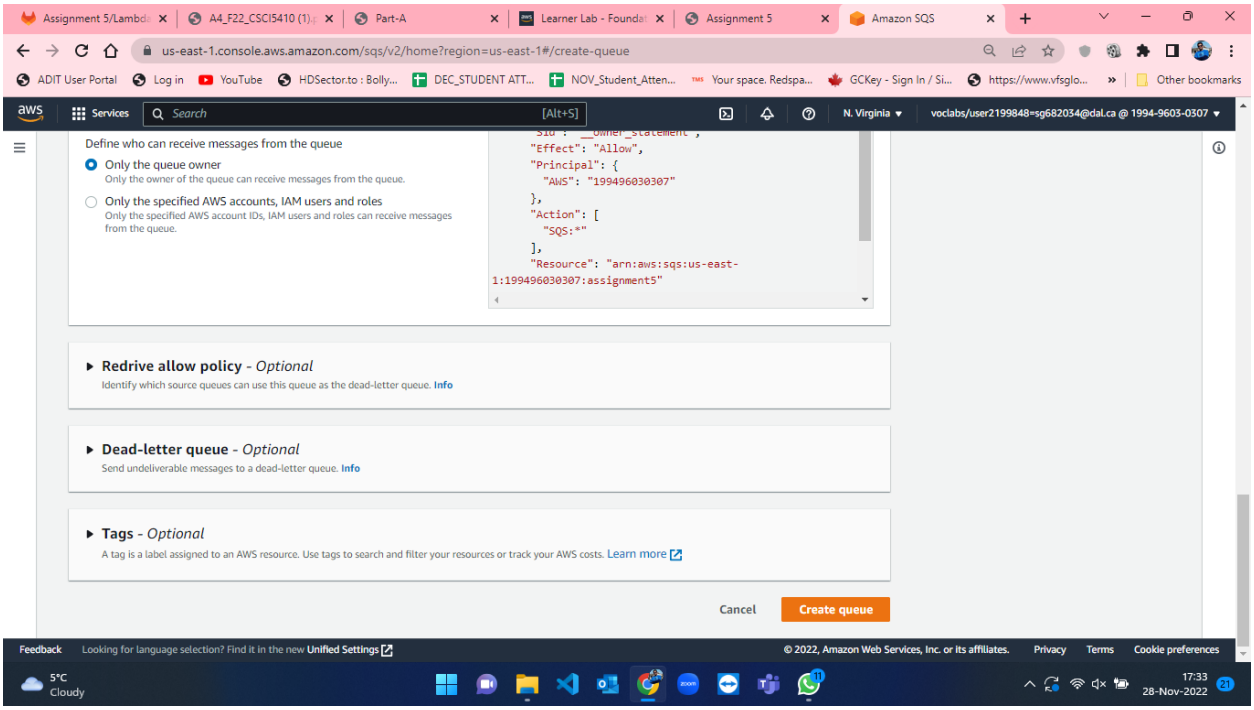

Figure 12: SQS create queue page – 4[1].

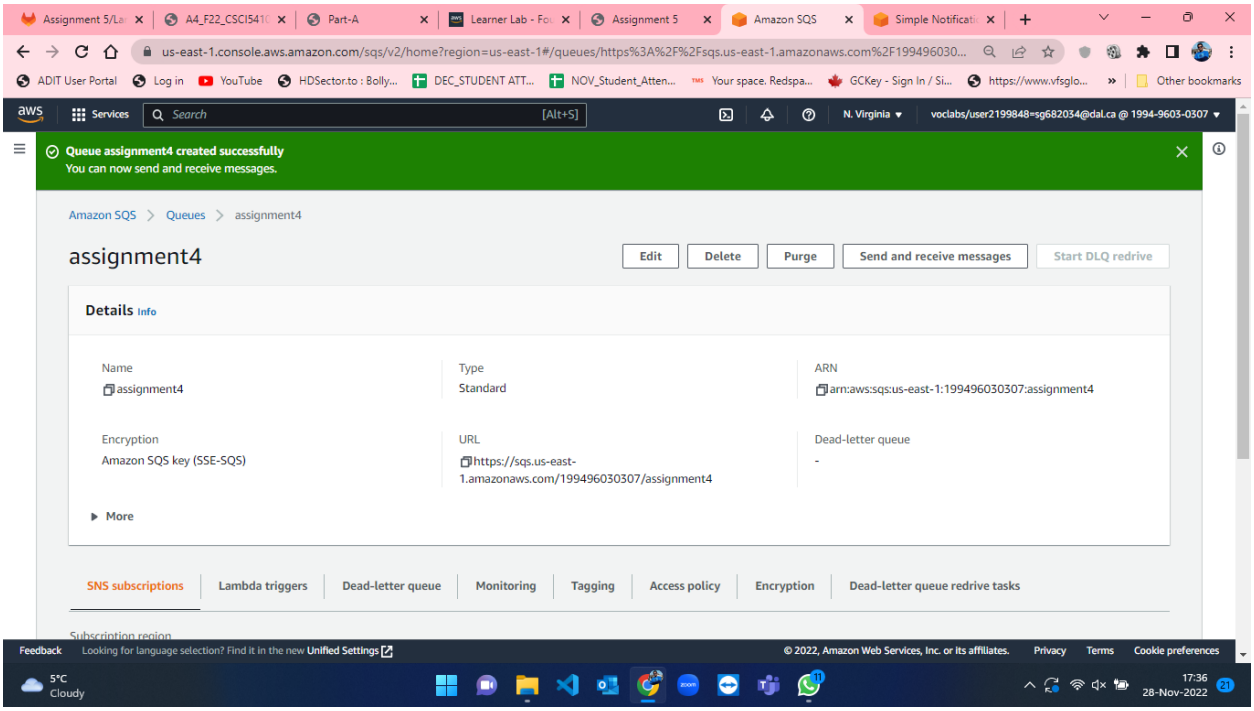

Figure 13: Successful creation of assignment4 queue.

- 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.

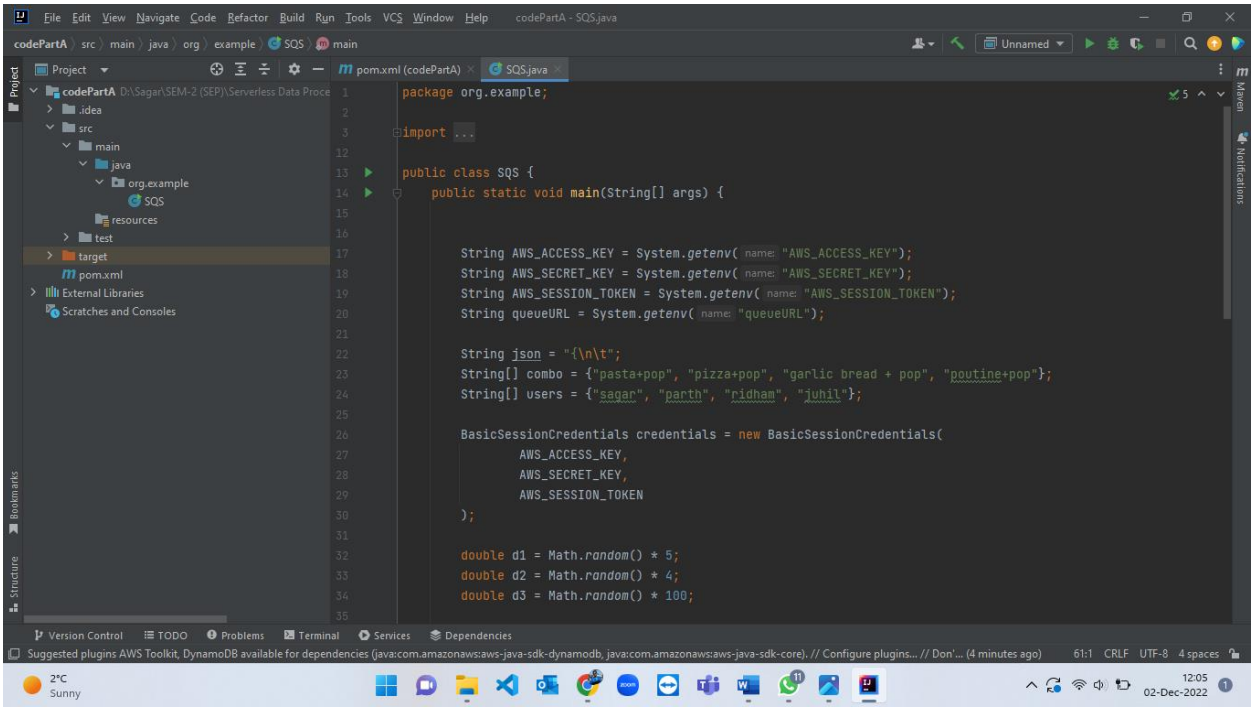

Figure 14: JAVA code for sending message in SQS queue part-1.

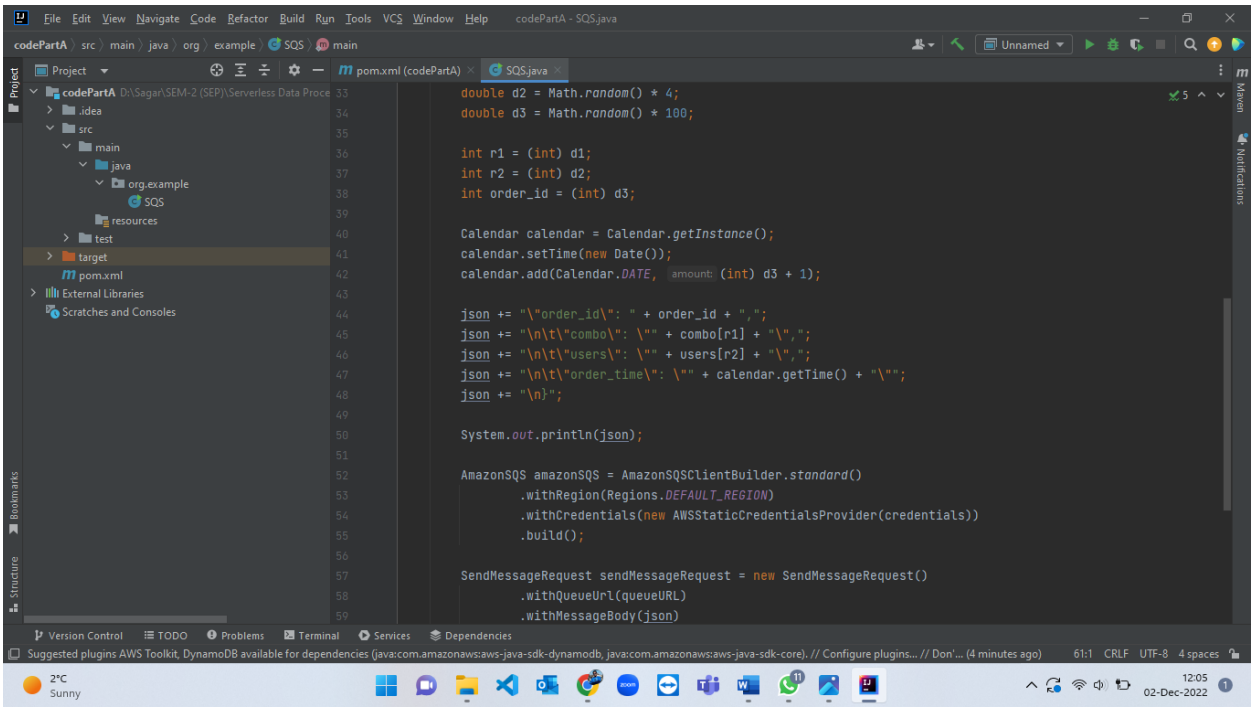

Figure 15: JAVA code for sending message in SQS queue part-2.

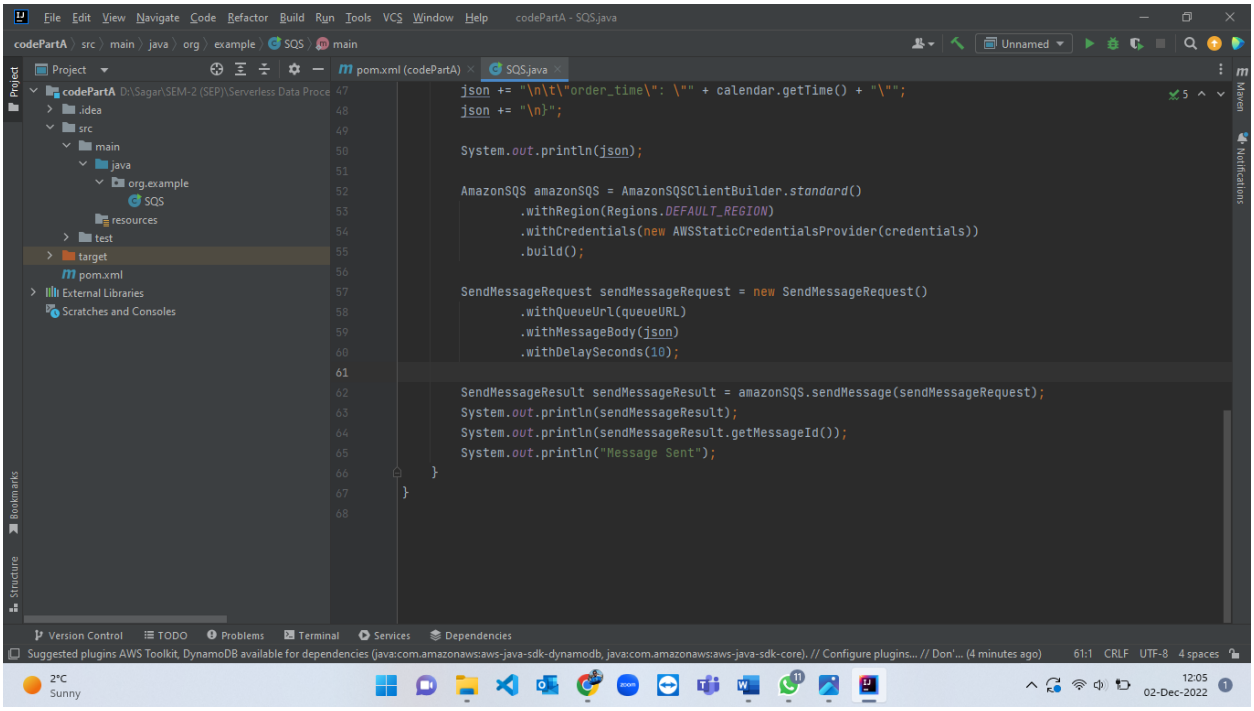

Figure 16: JAVA code for sending message in SQS queue part-3.

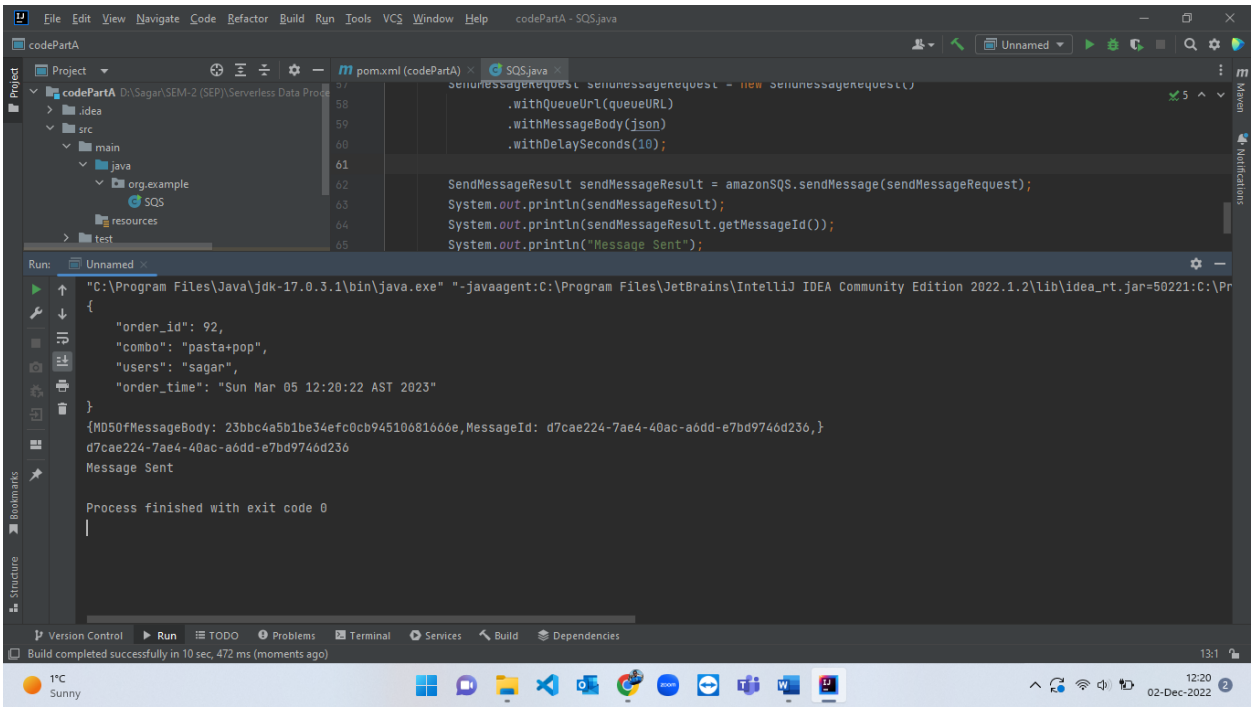

Figure 17: Execution logs from JAVA code displaying that message has been successfully sent to SQS.

- 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.

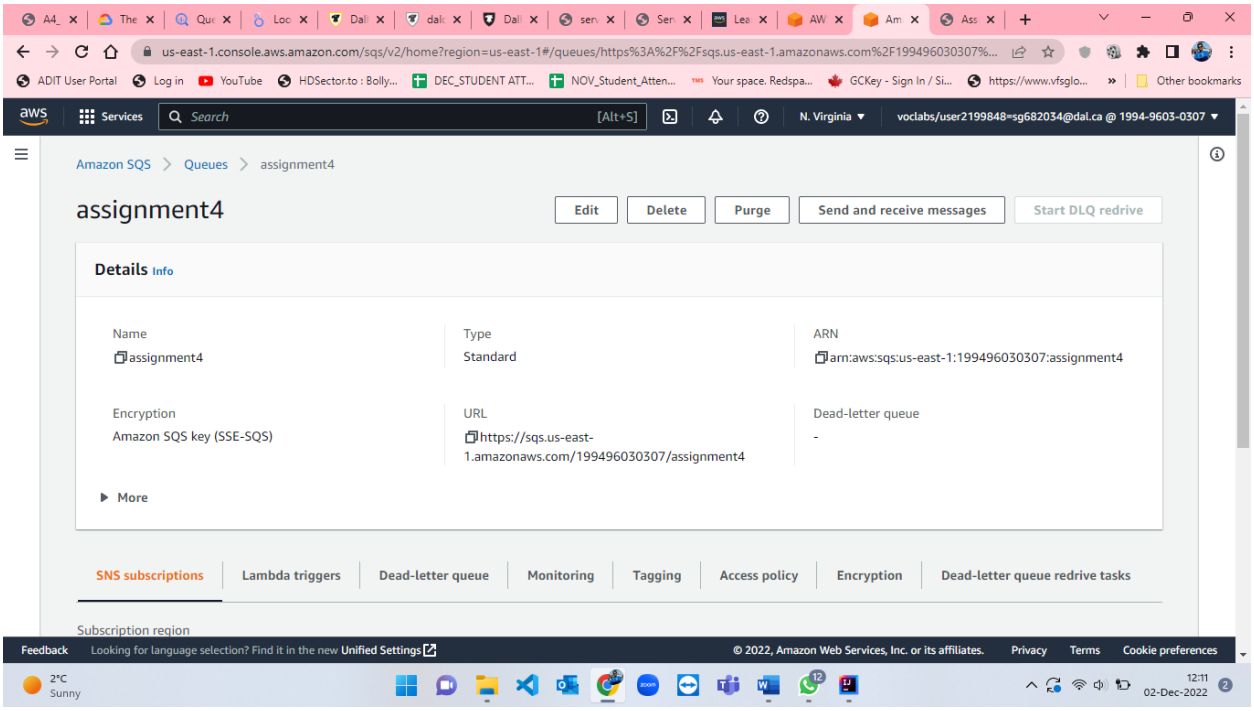

Figure 18: SQS page to select " Send and receive messages".

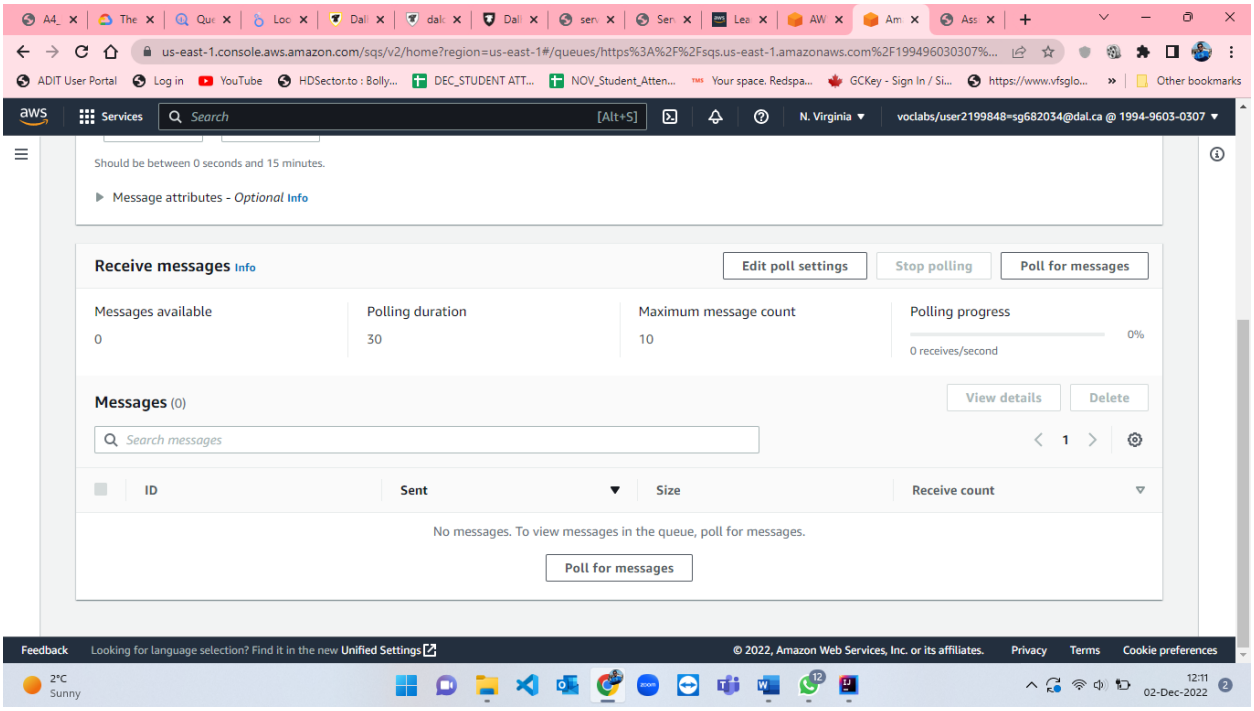

Figure 19: "Poll for messages" option from SQS.

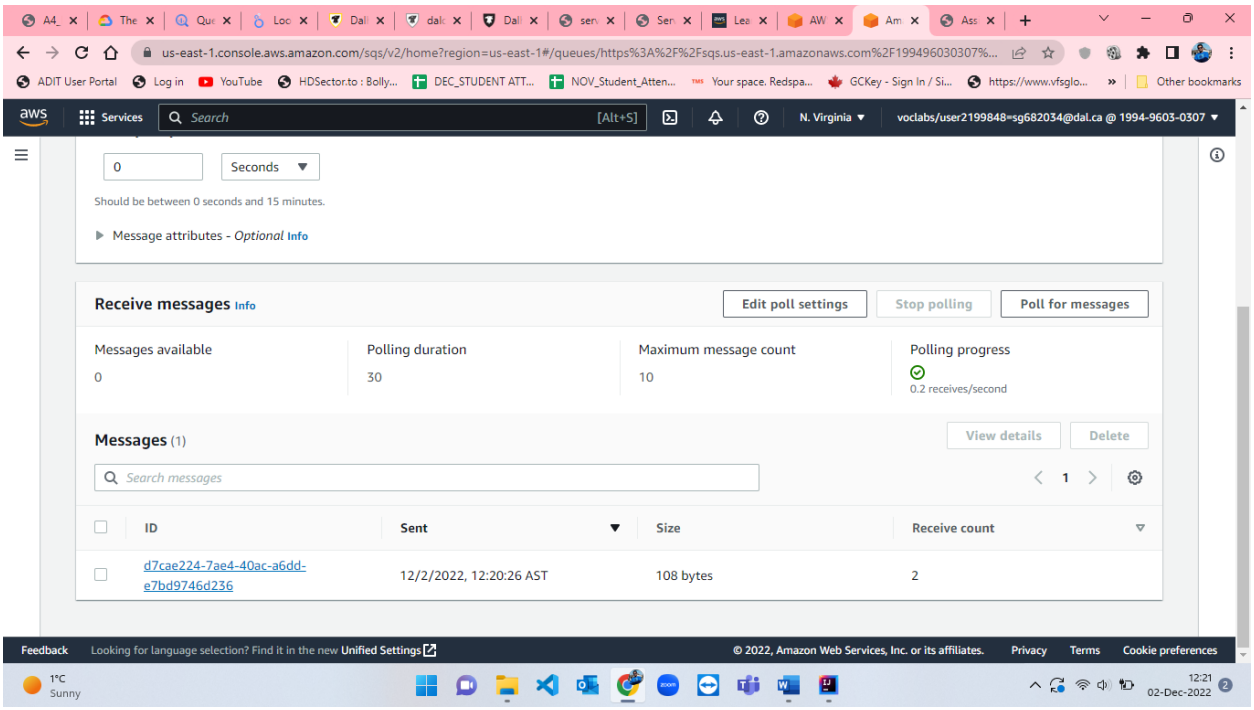

Figure 20: Message received from JAVA code to SQS.

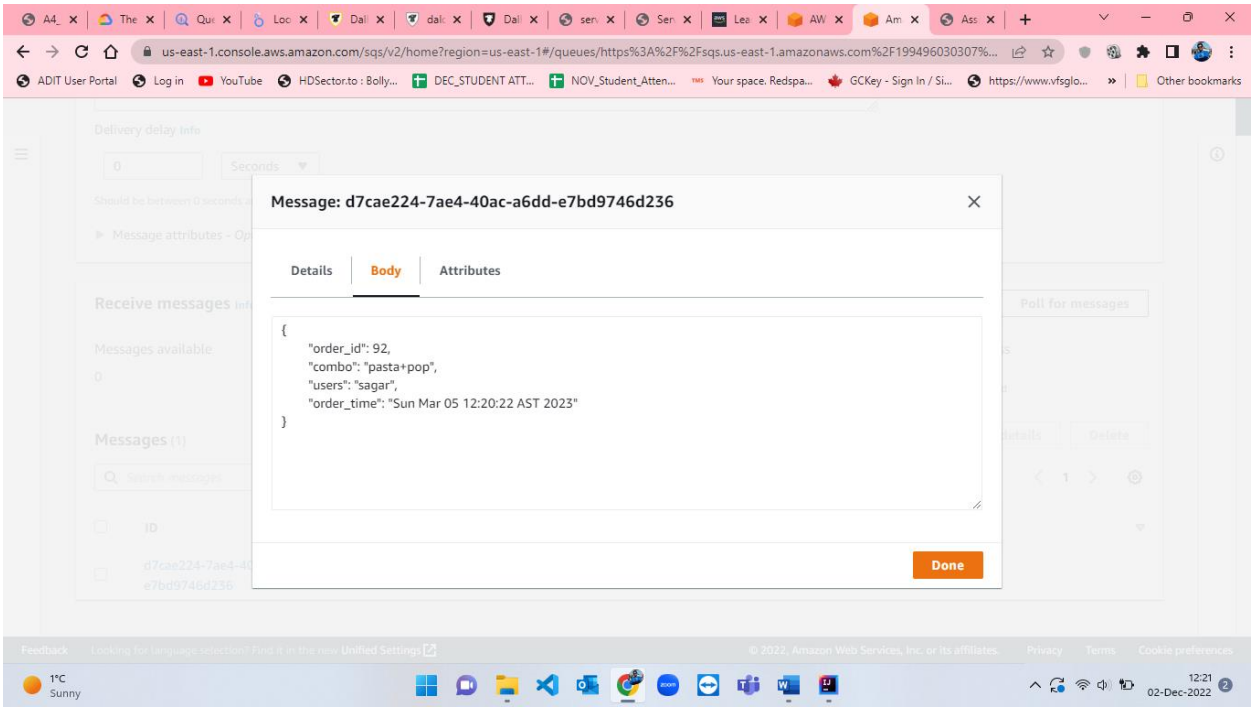

Figure 21: Displaying JSON format message received in SQS.

- 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.

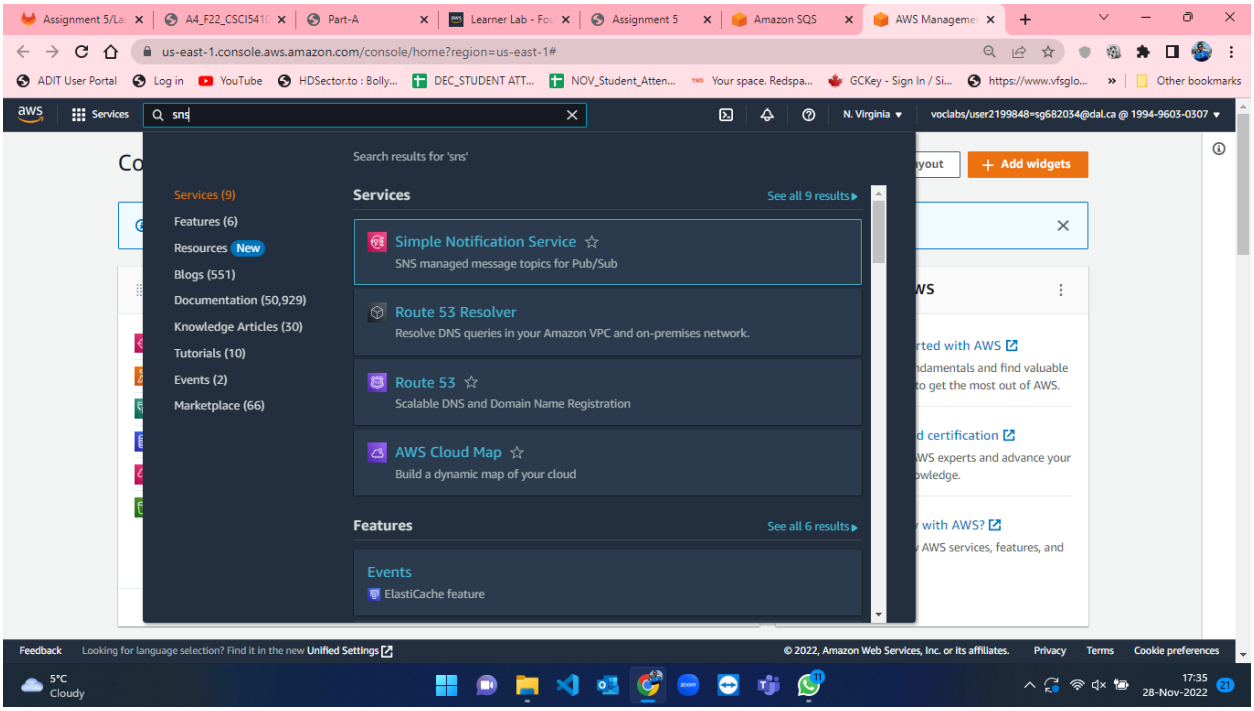

Figure 22: Searching for SNS in search bar from Console Home in AWS.

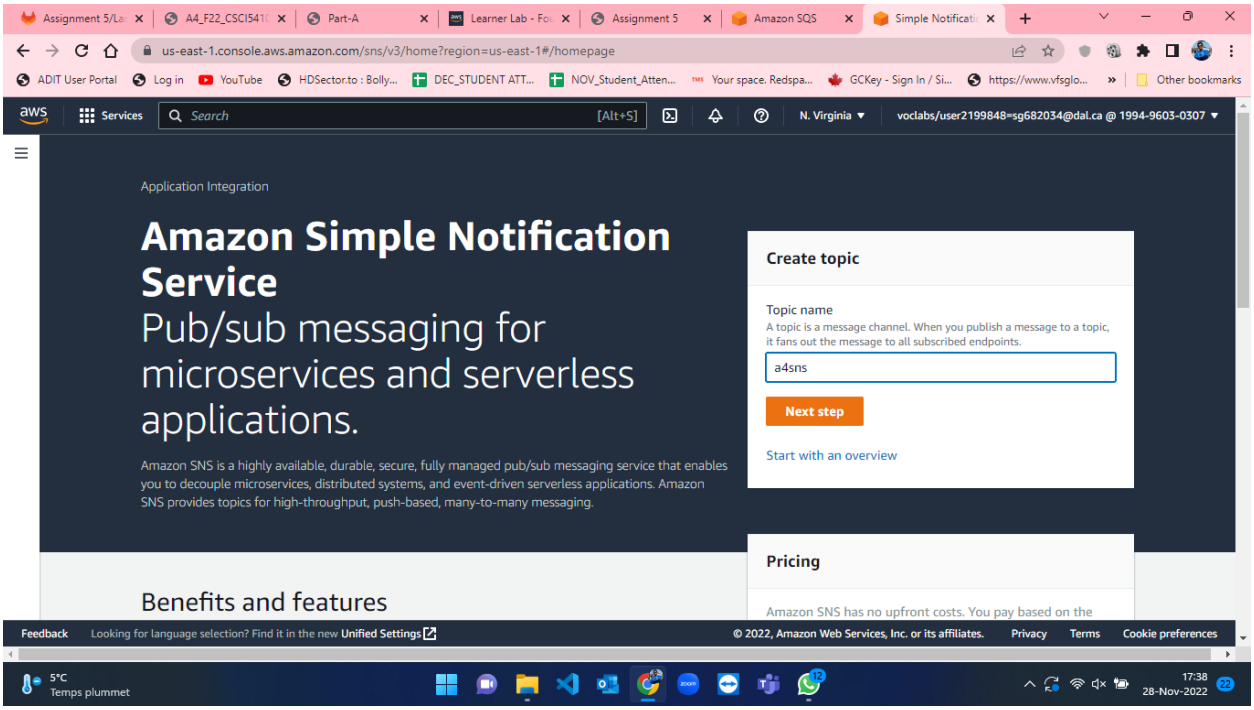

Figure 23: SNS homepage in AWS Console.

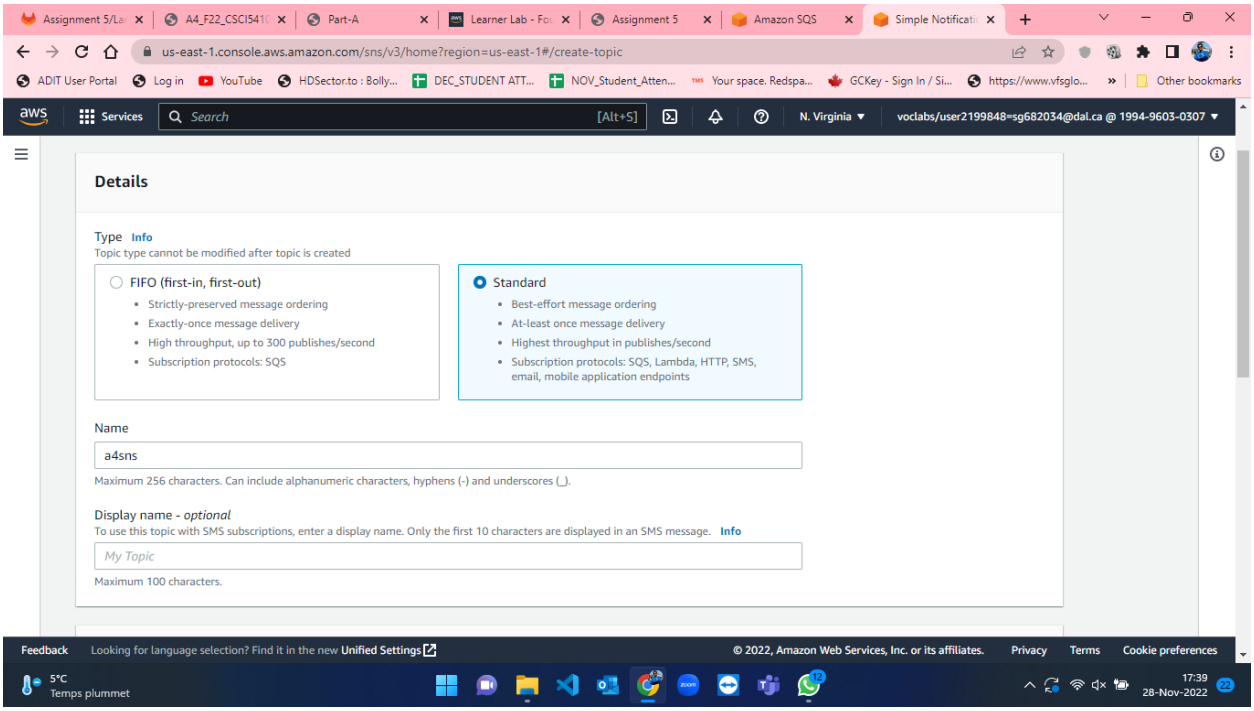

Figure 24: Create SNS part-1.

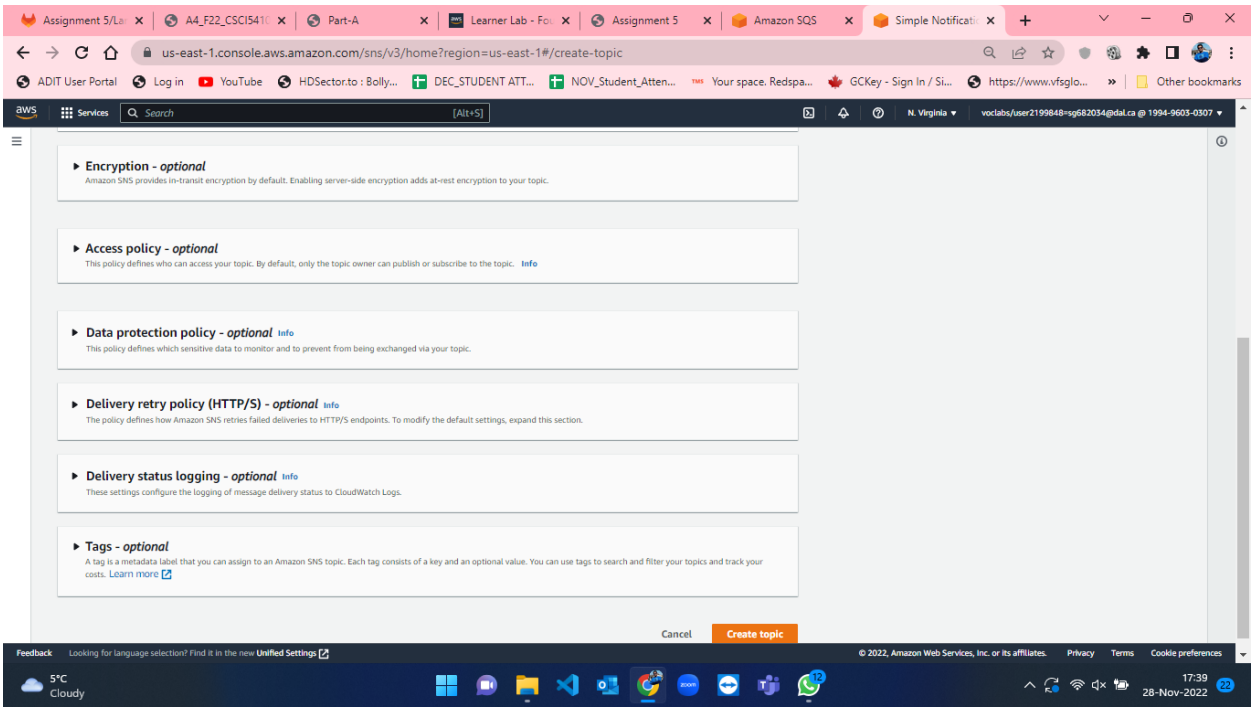

Figure 25: Create SNS part-2.

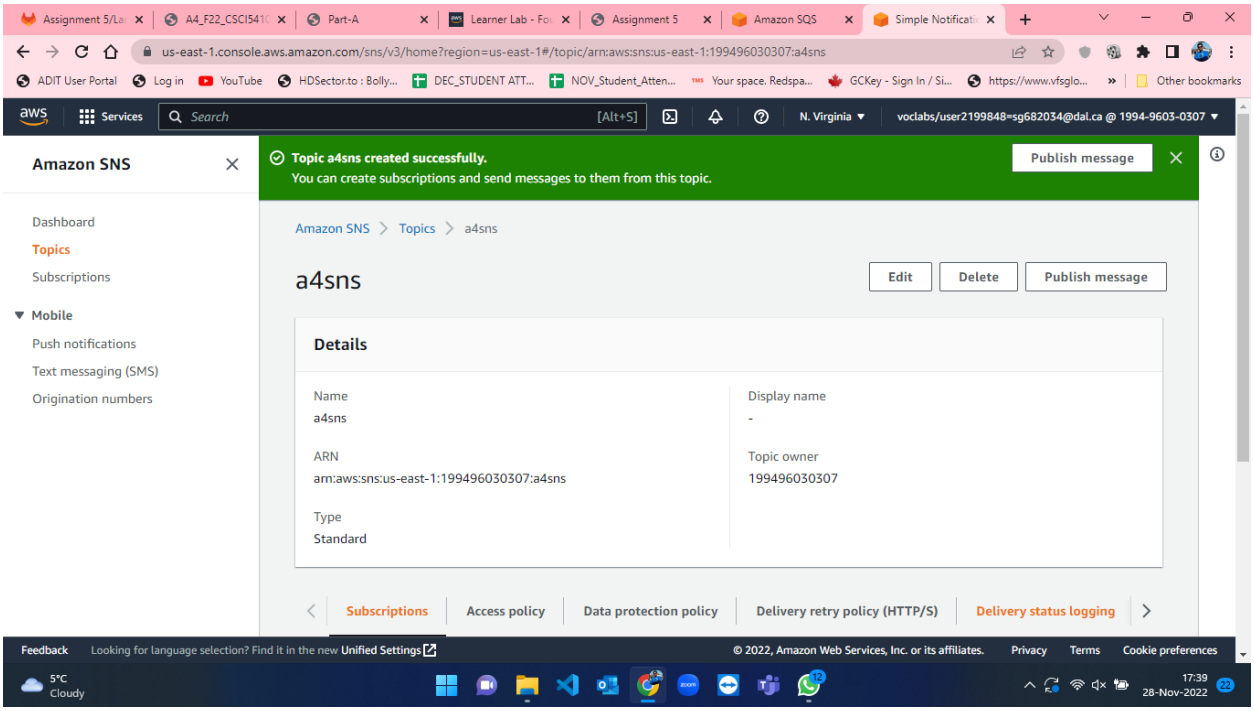

Figure 26: Successful creation of a4sns.

- 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.

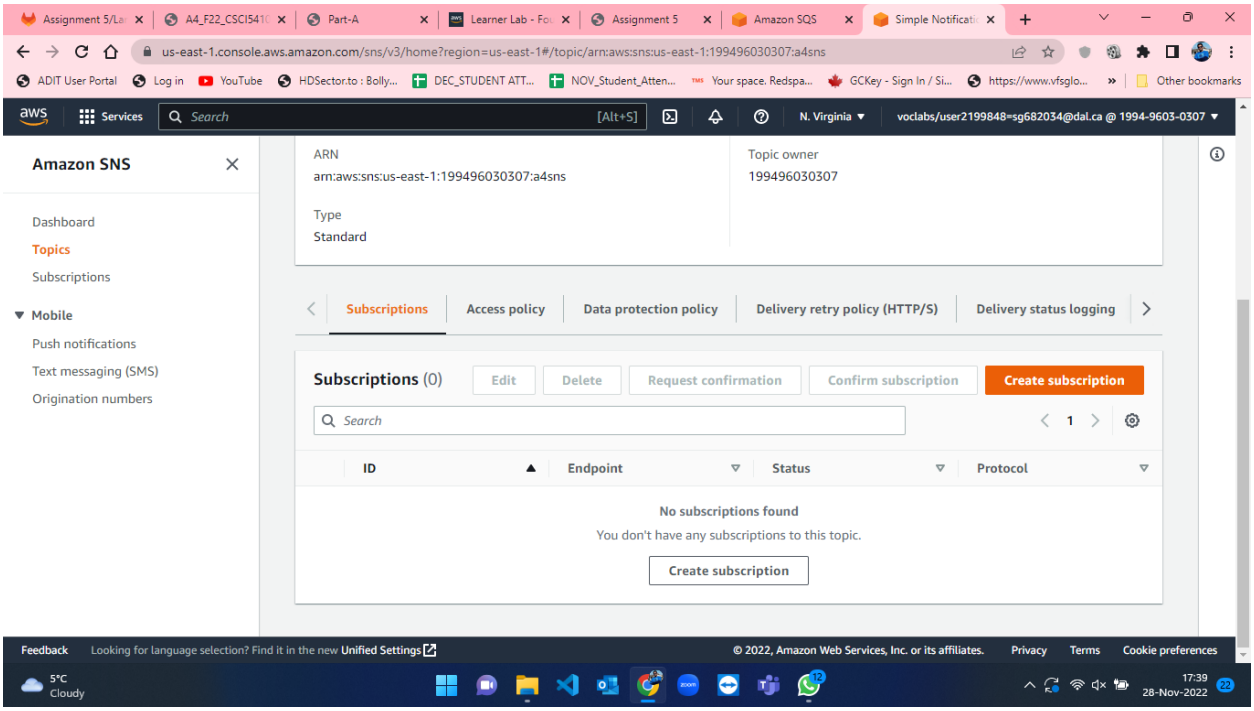

Figure 27: Create SNS subscription step-1.

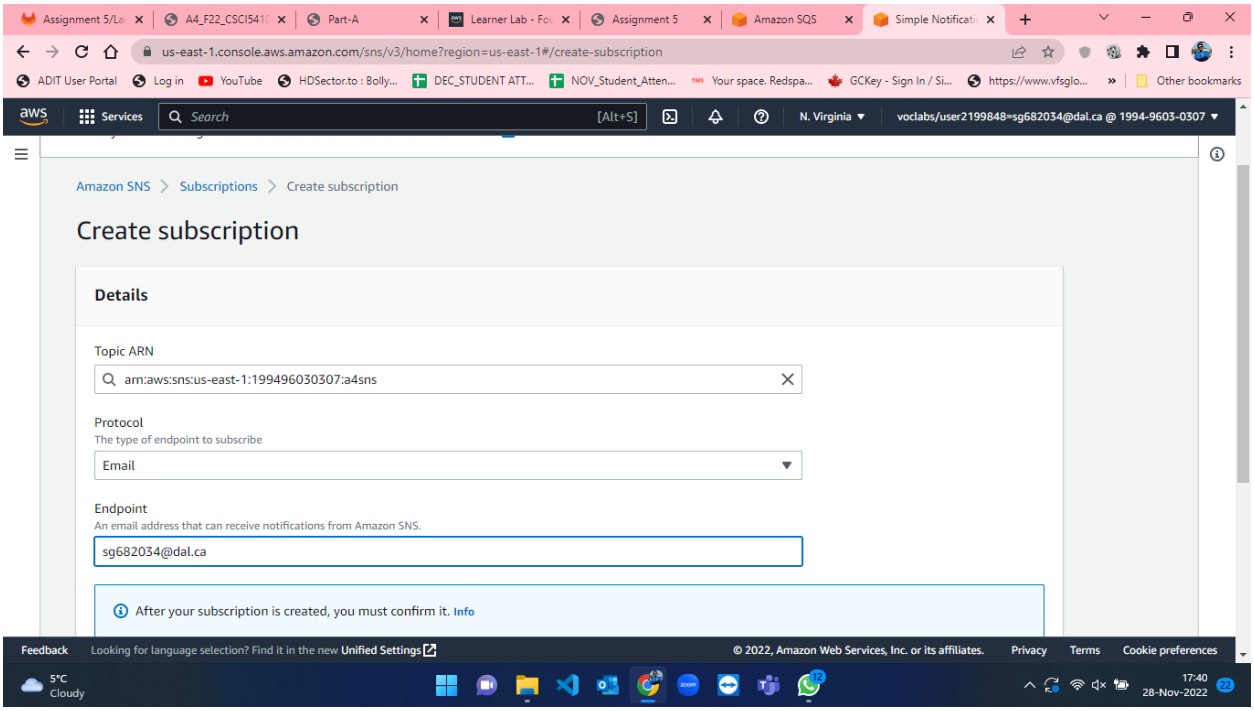

Figure 28: Create SNS subscription step-2.

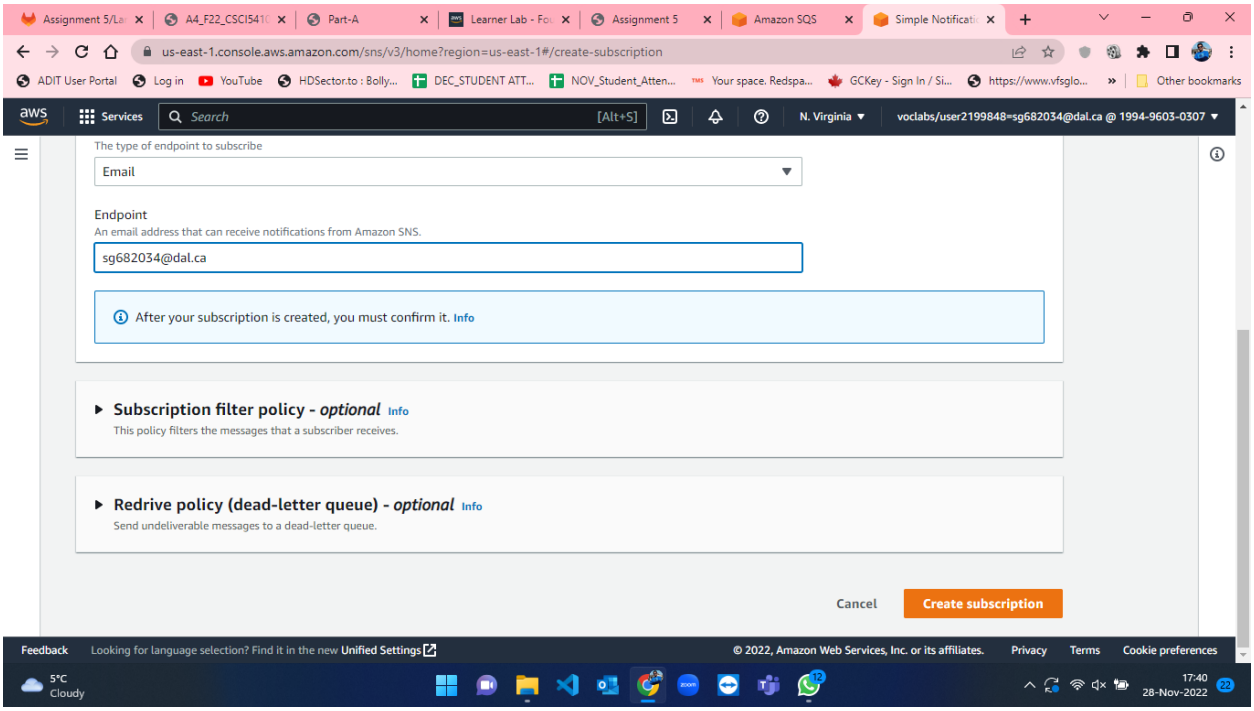

Figure 29: Create SNS subscription step-3.

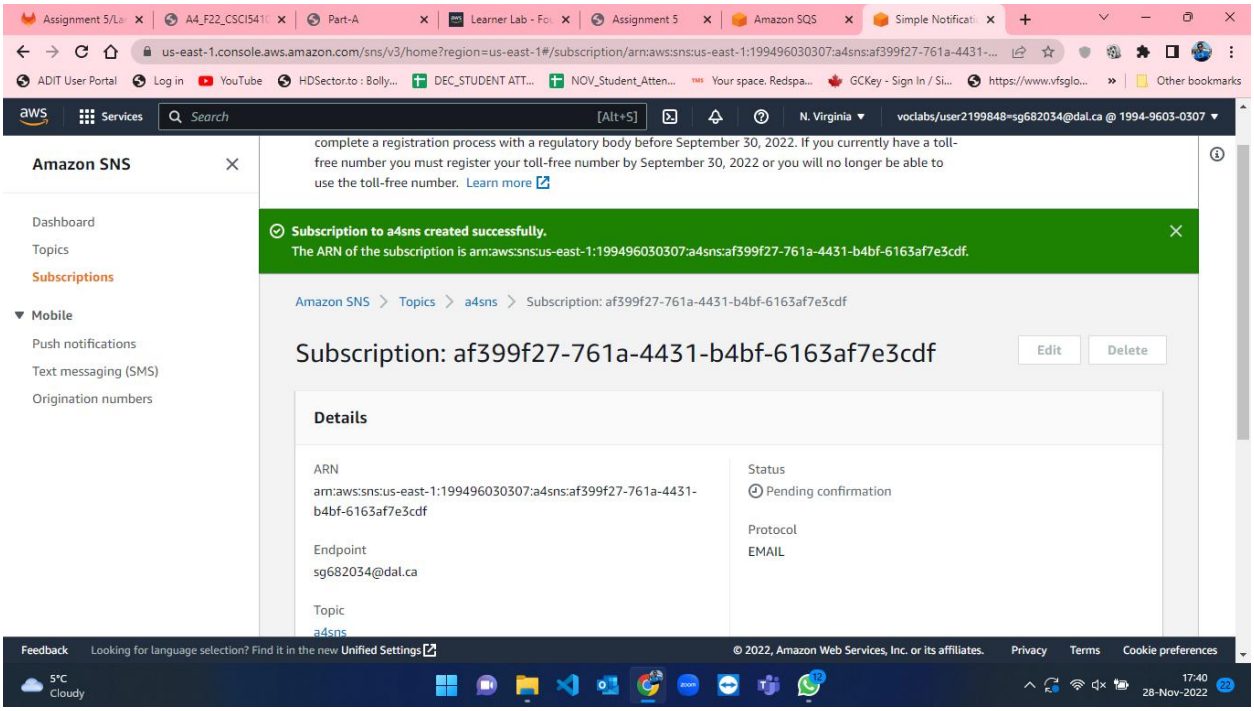

Figure 30: Successful creation of SNS subscription.

- 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).

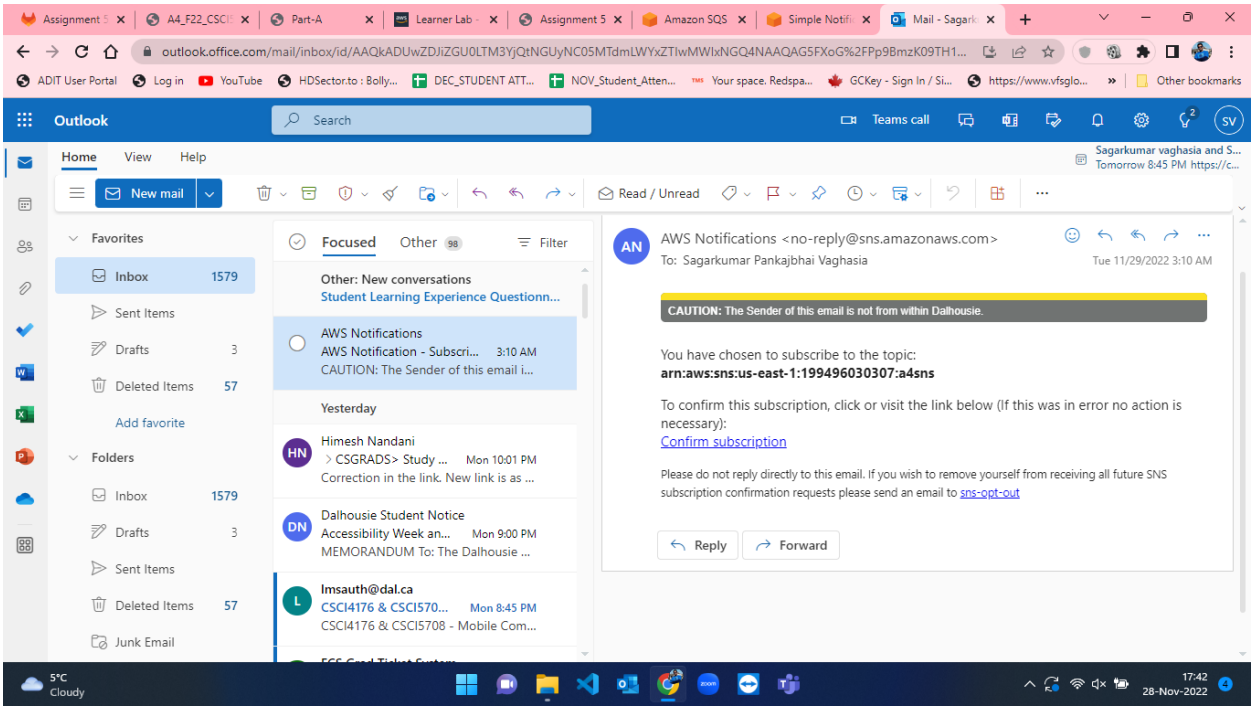

Figure 31: Confirm Subscription mail from AWS.

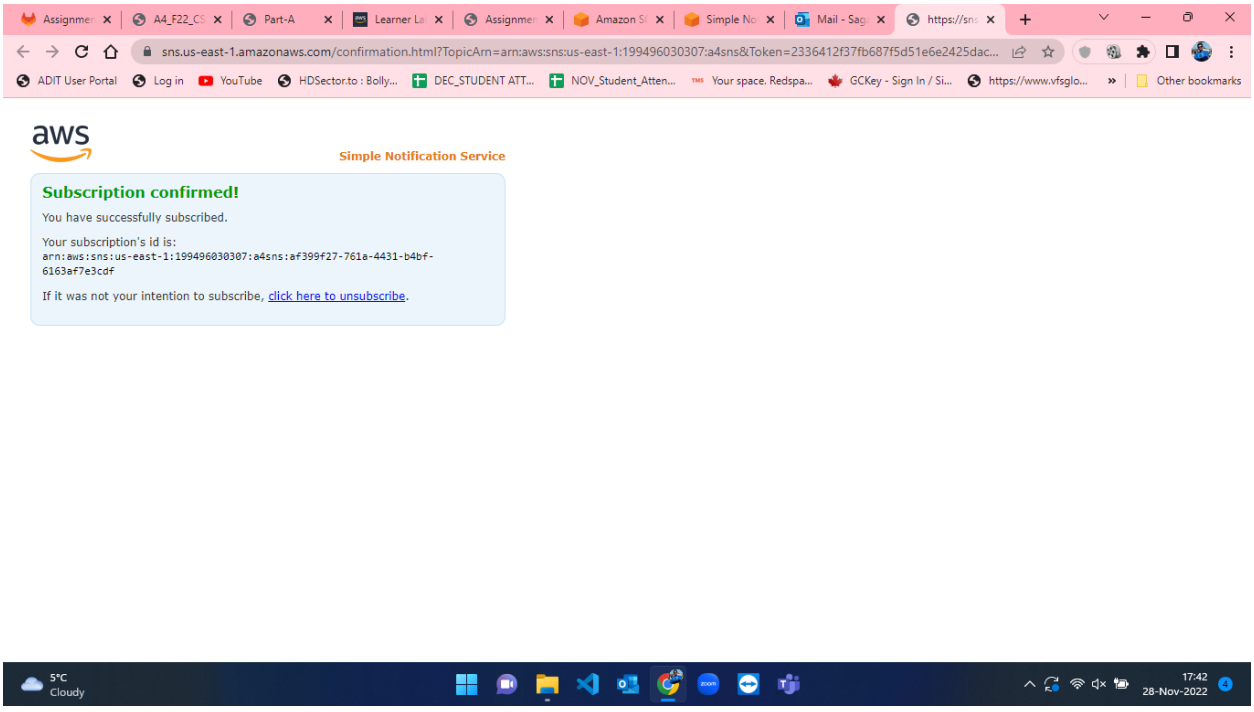

Figure 32: Subscription confirmation.

- 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.

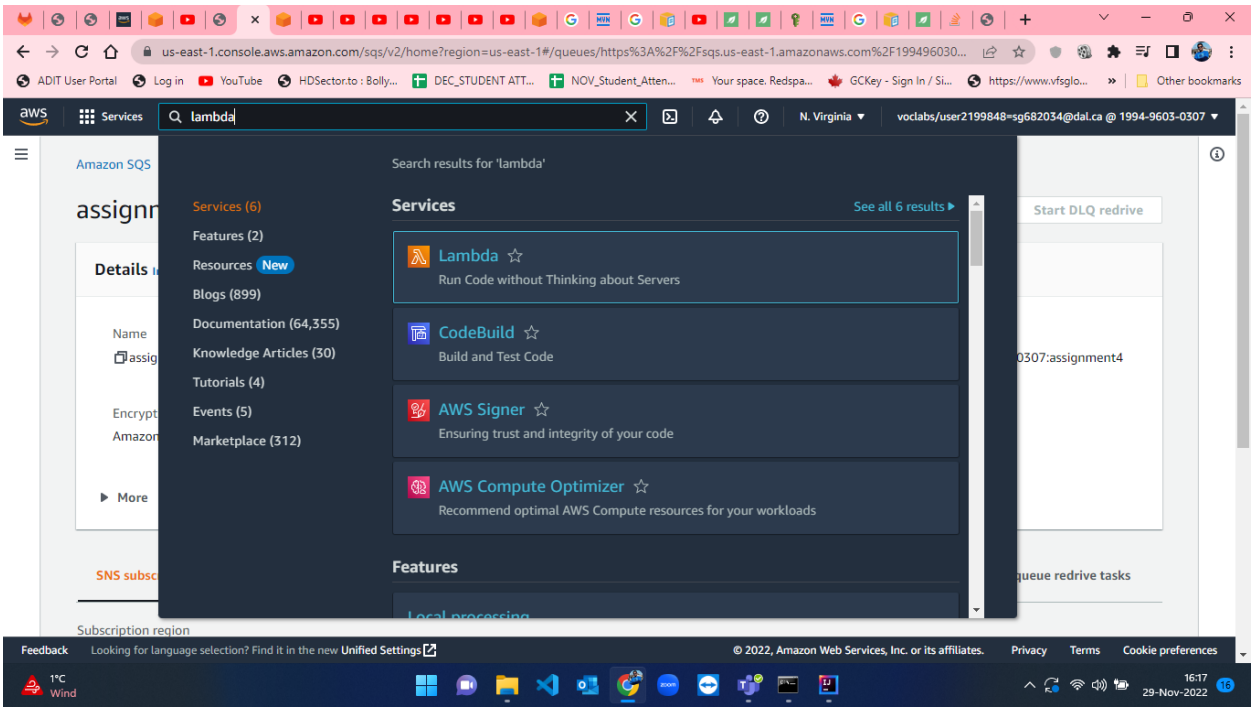

Figure 33: Lambda Function creation step-1.

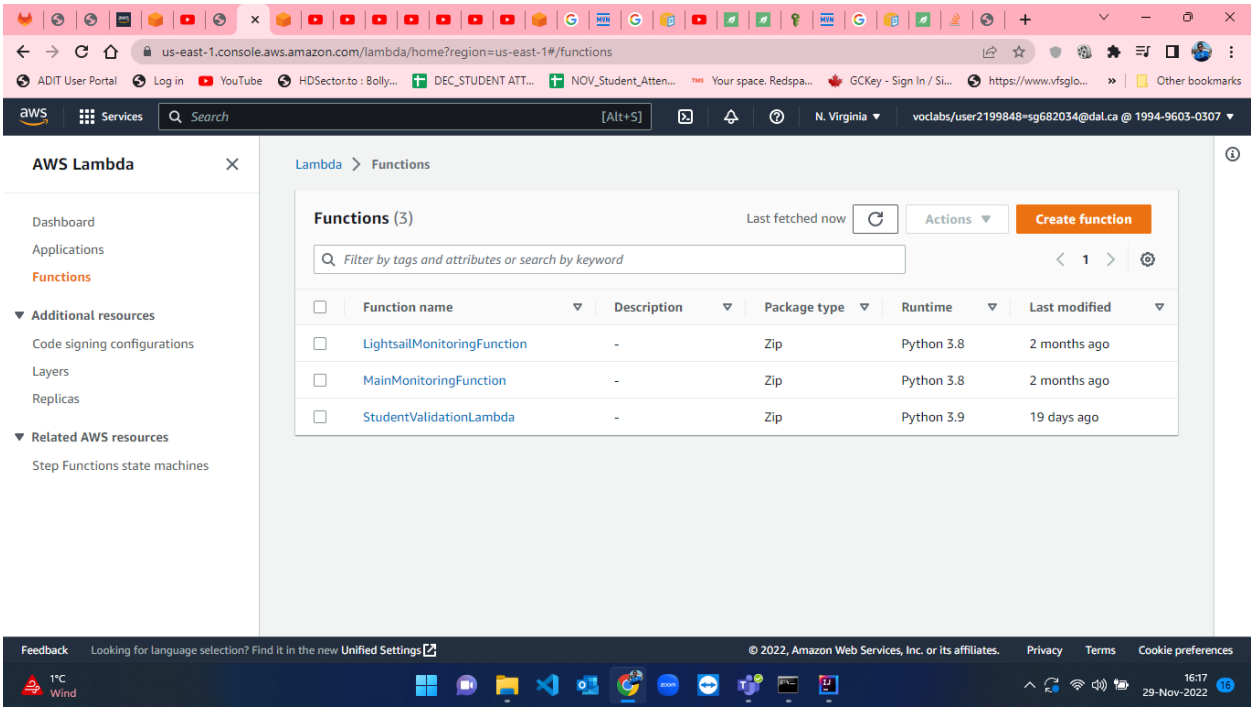

Figure 34: Lambda Function creation step-2.

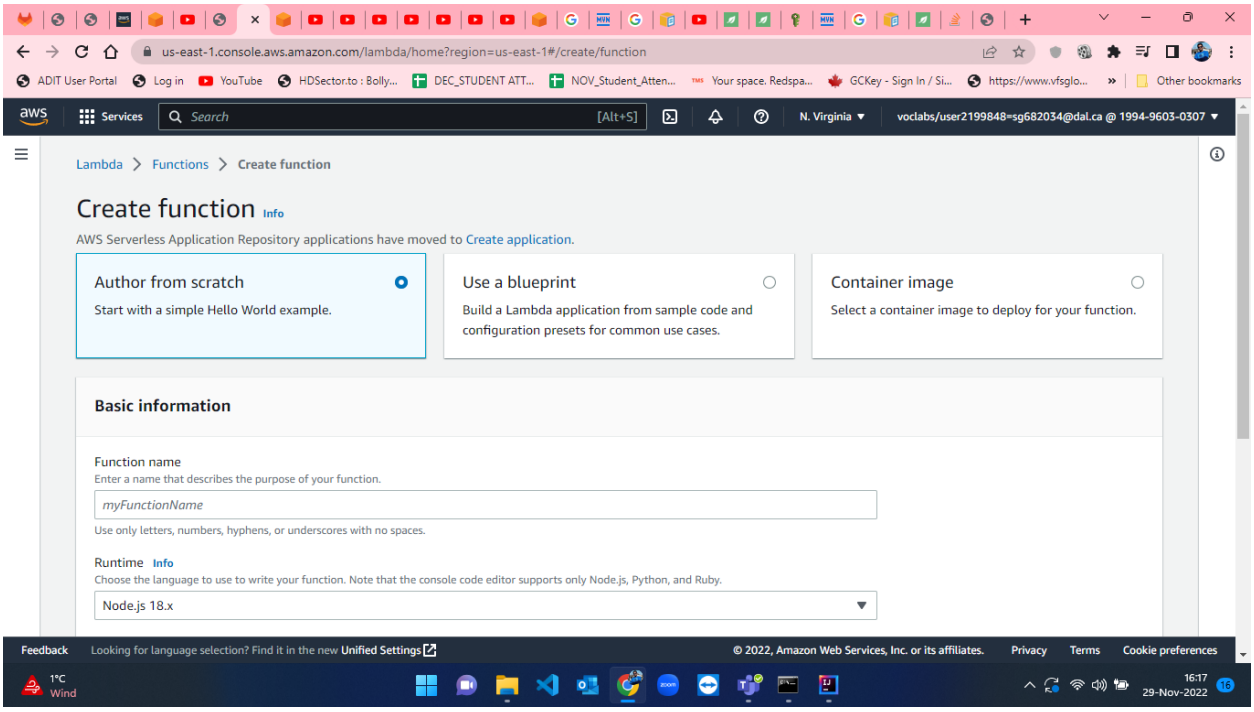

Figure 35: Lambda Function creation step-3.

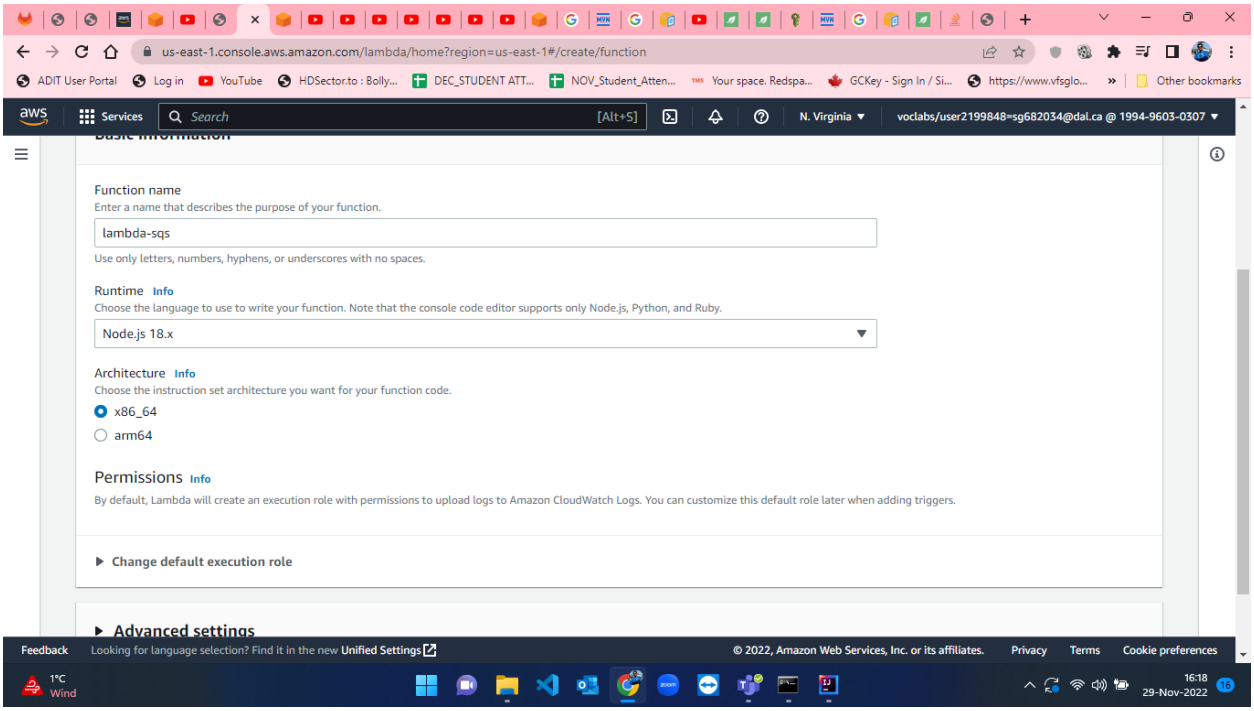

Figure 36: Lambda Function creation step-4.

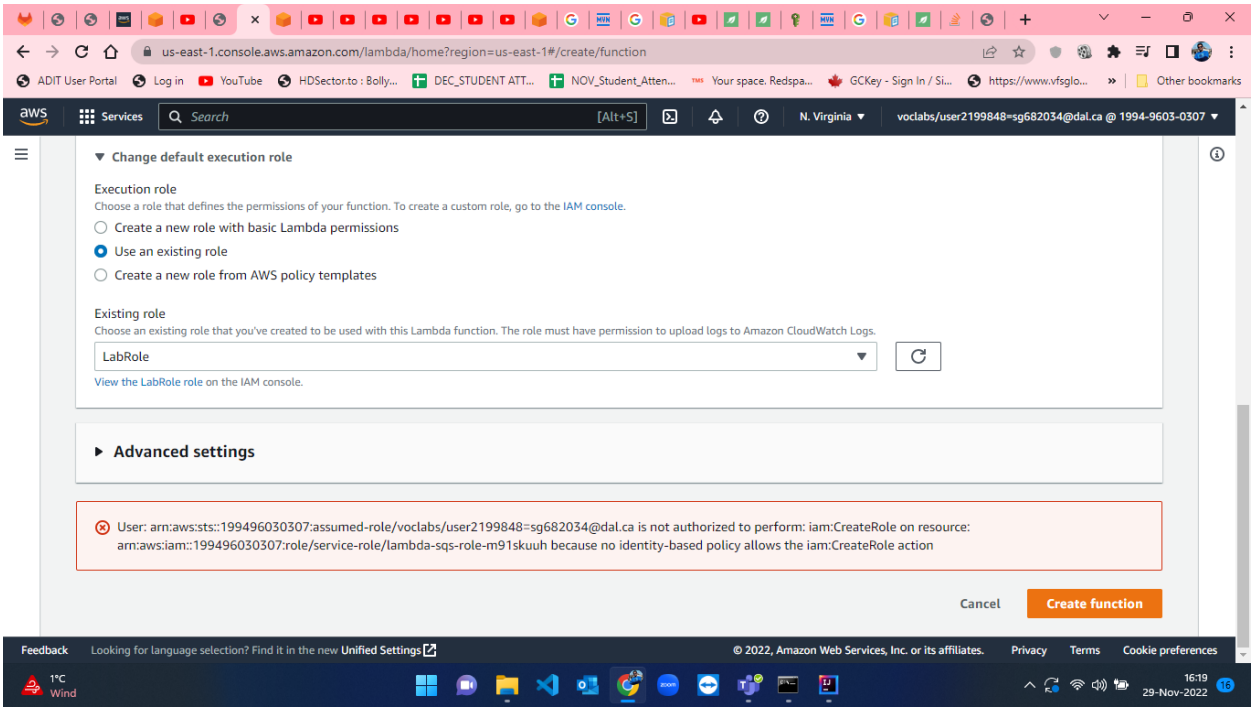

Figure 37: Lambda Function creation step-5.

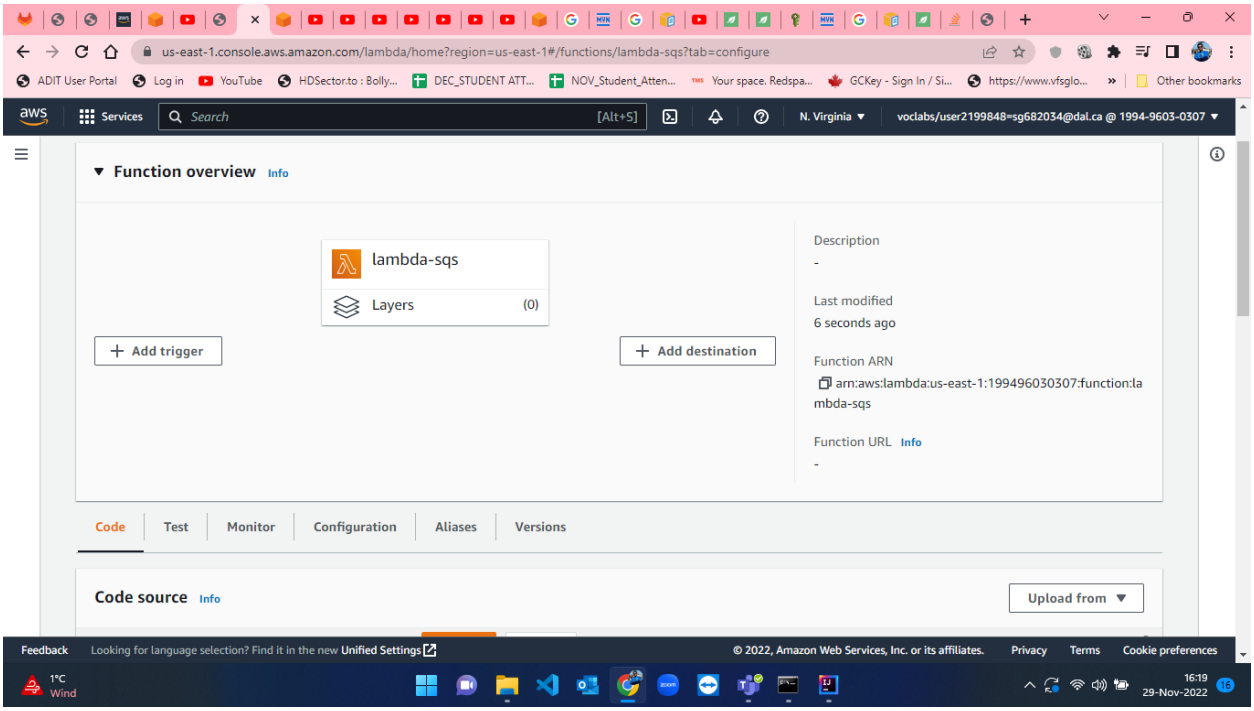

Figure 38: Lambda Function creation step-6.

- 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.

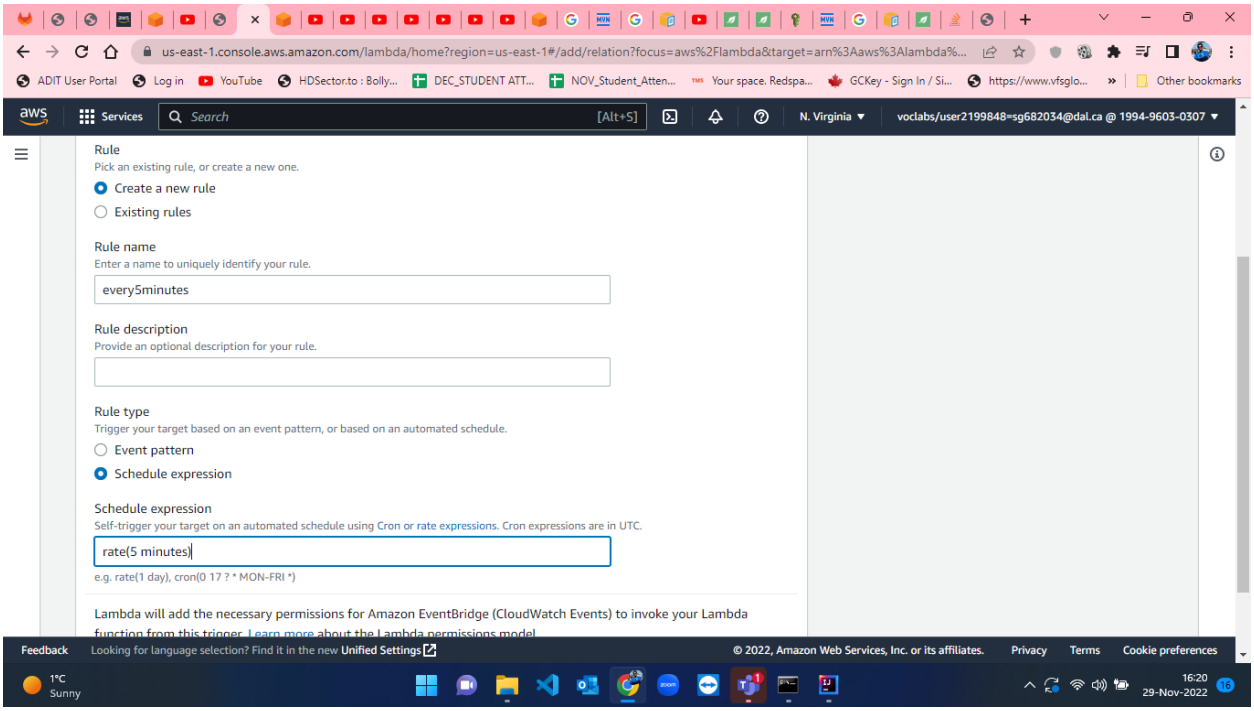

Figure 39: Adding trigger to lambda function.

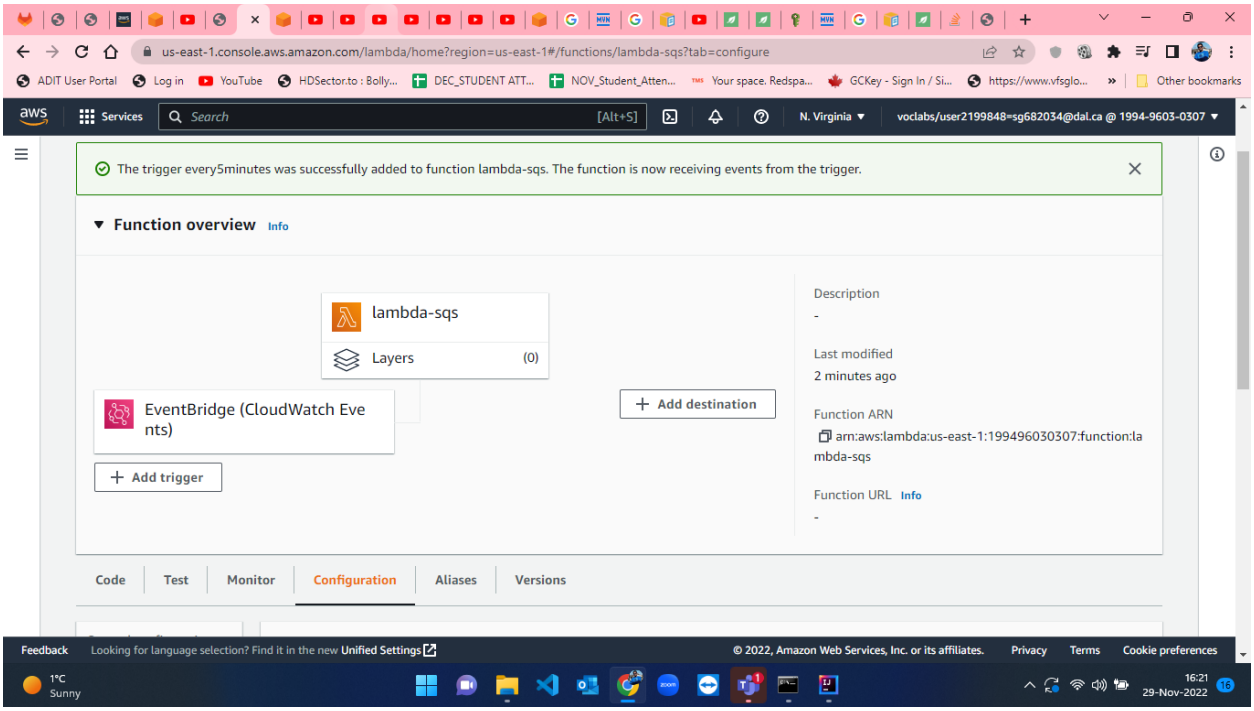

Figure 40: Successful trigger addition to lambda function.

- 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.

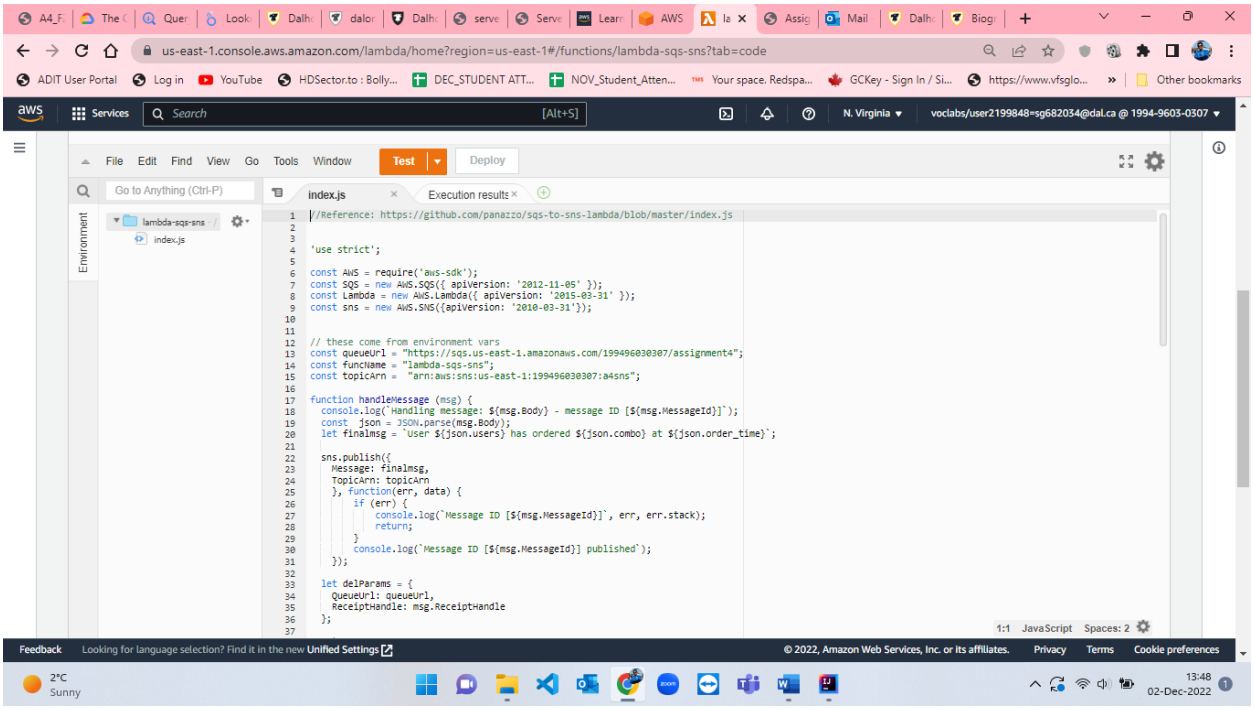

Figure 41: lambda function part-1.

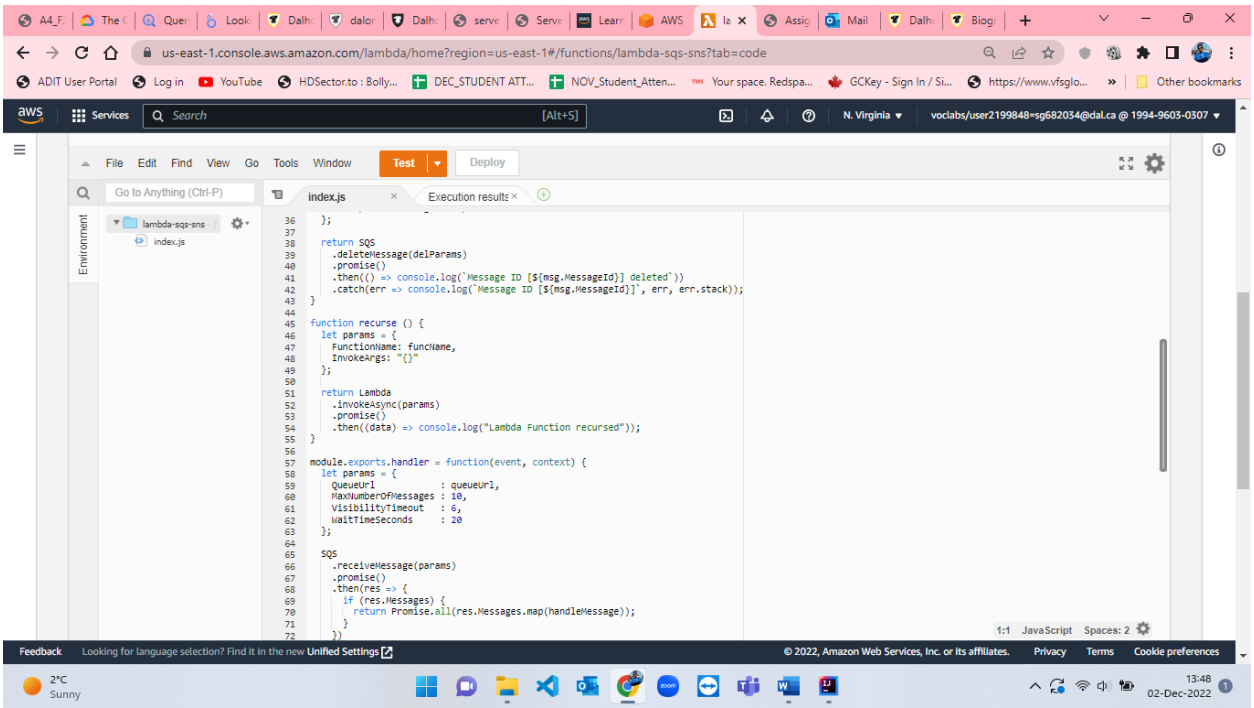

Figure 42: lambda function part-2.

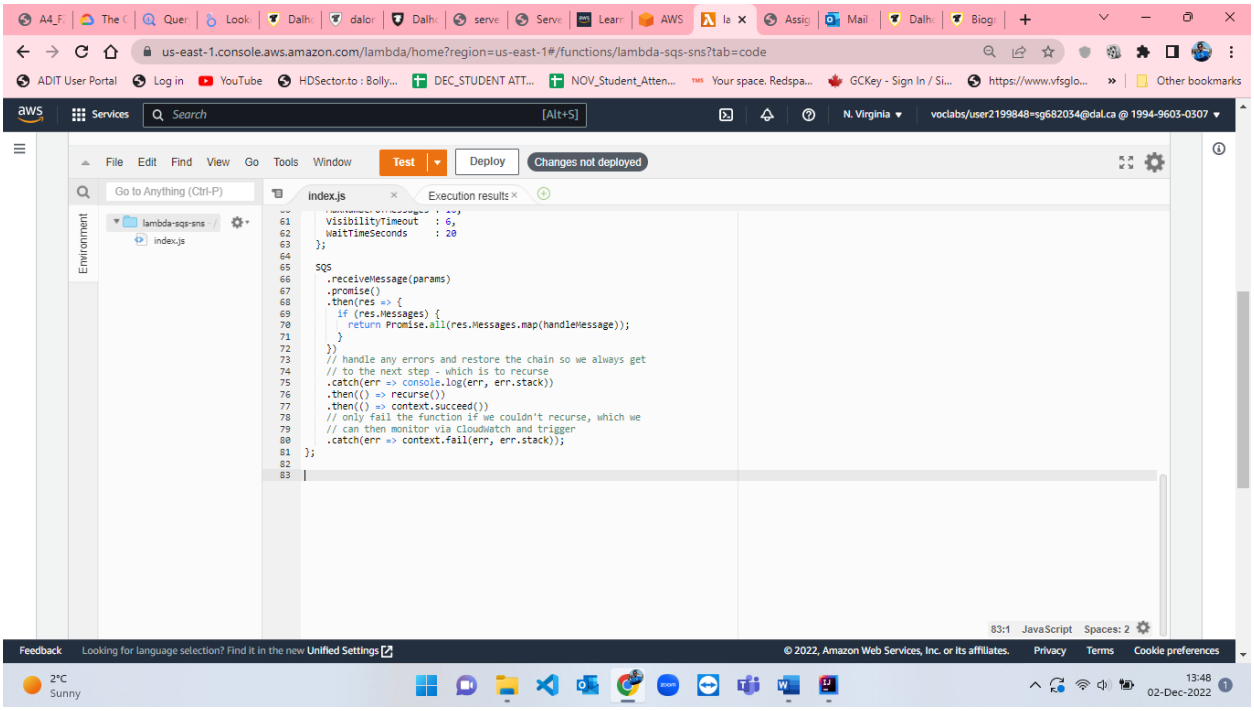

Figure 43: lambda function part-3.

- 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”.

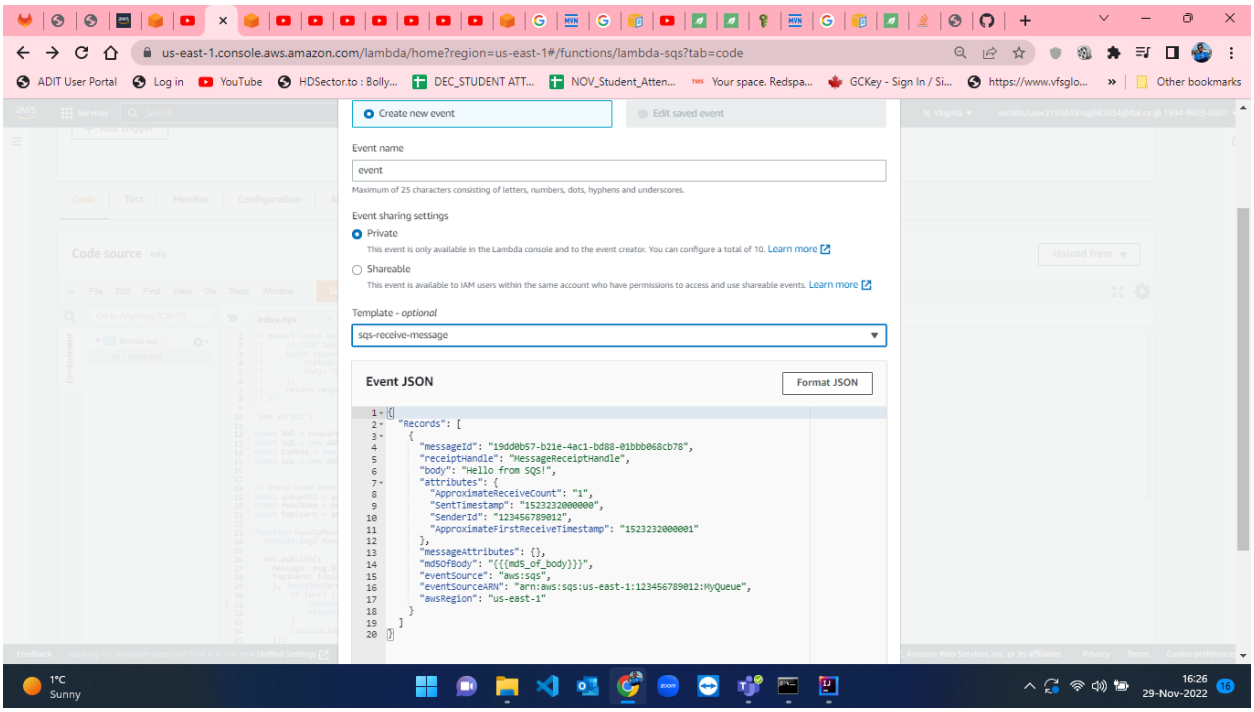

Figure 44: Event creation to test the lambda function.

- 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.

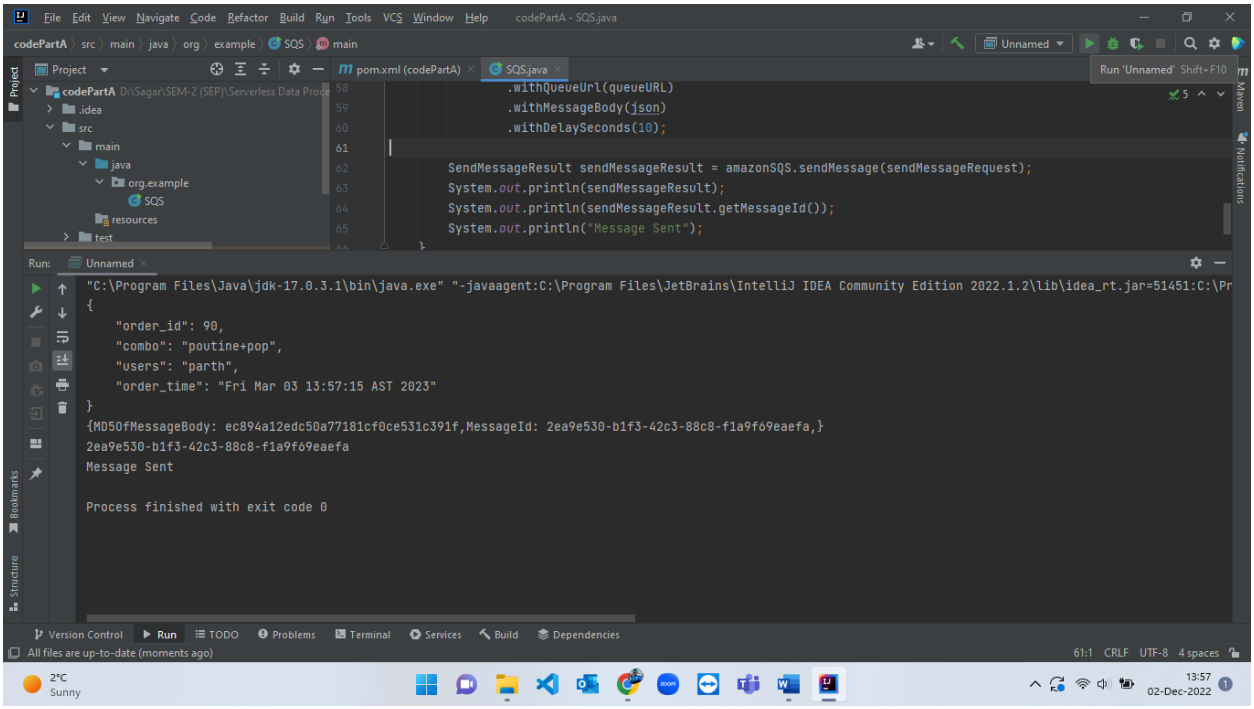

Figure 45: Random order creation - 1.

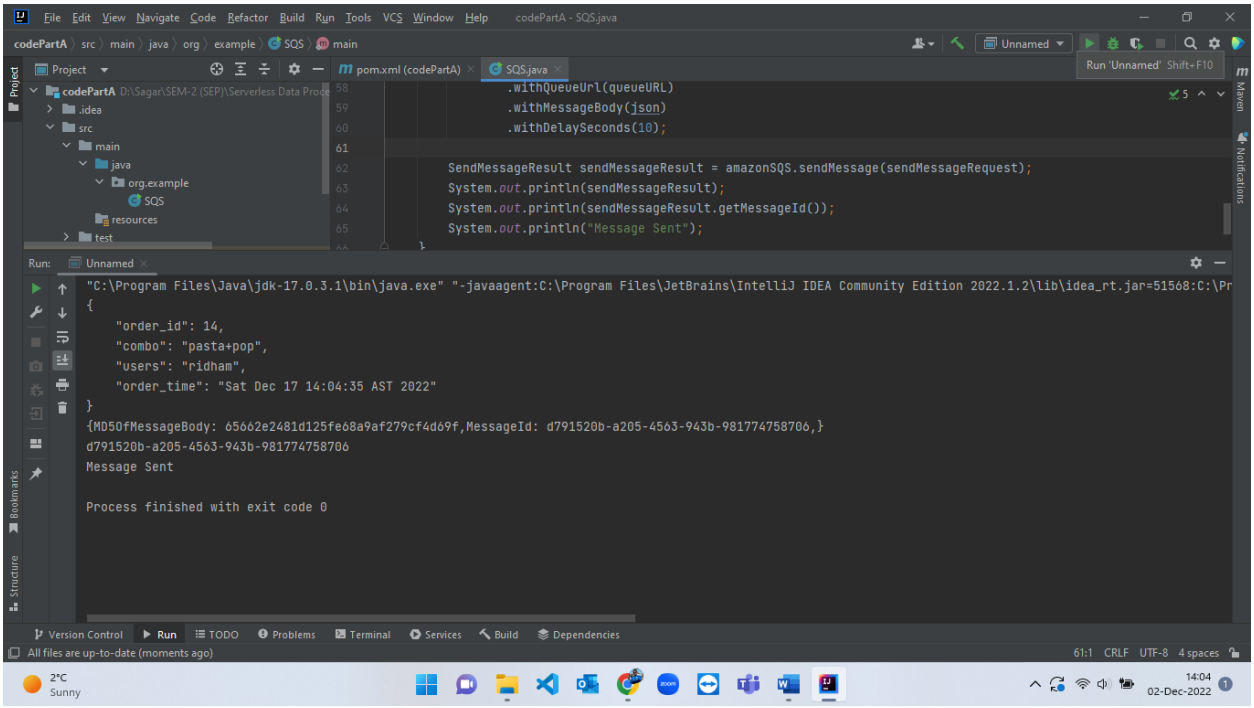

Figure 46: Random order creation - 2.

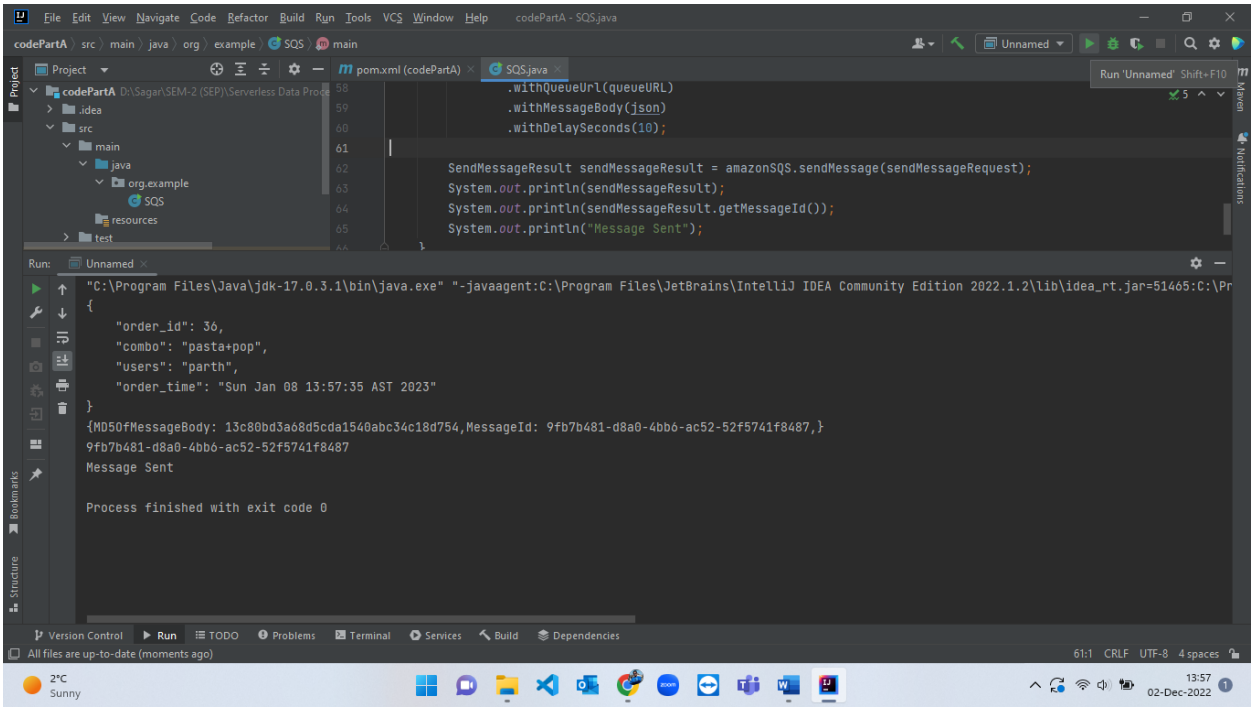

Figure 47: Random order creation - 3.

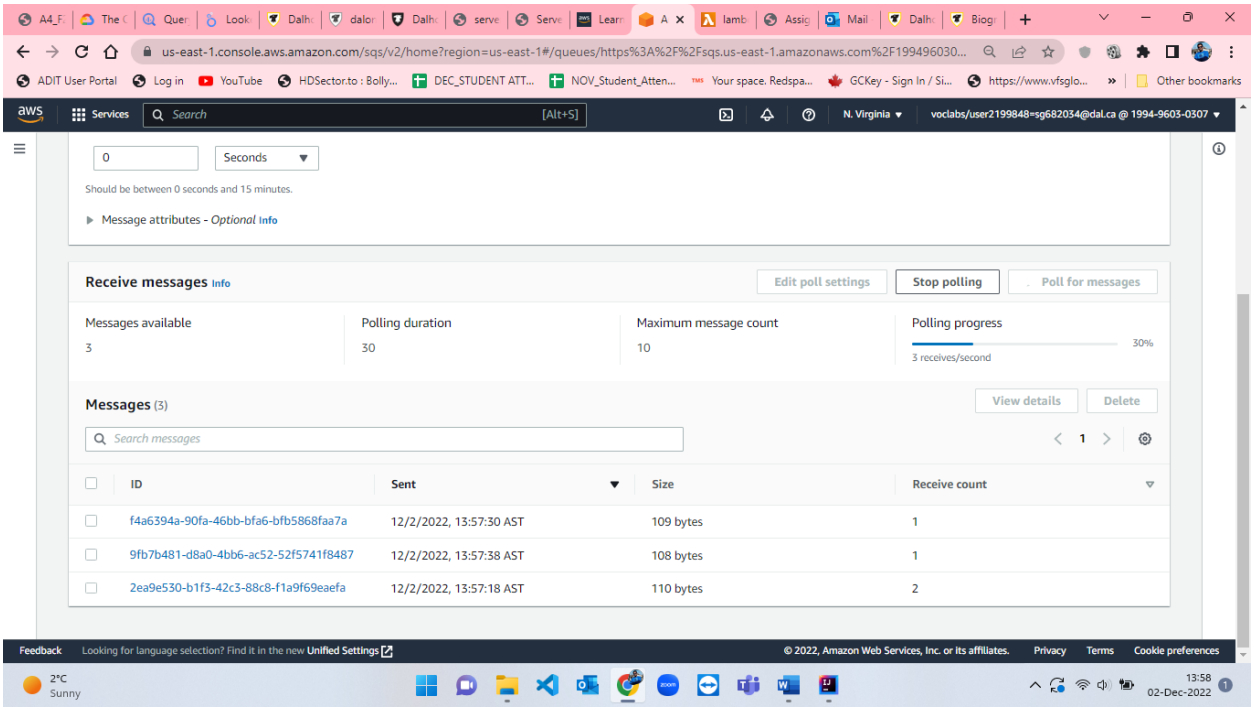

Figure 48: 3 orders messages polled in SQS queue.

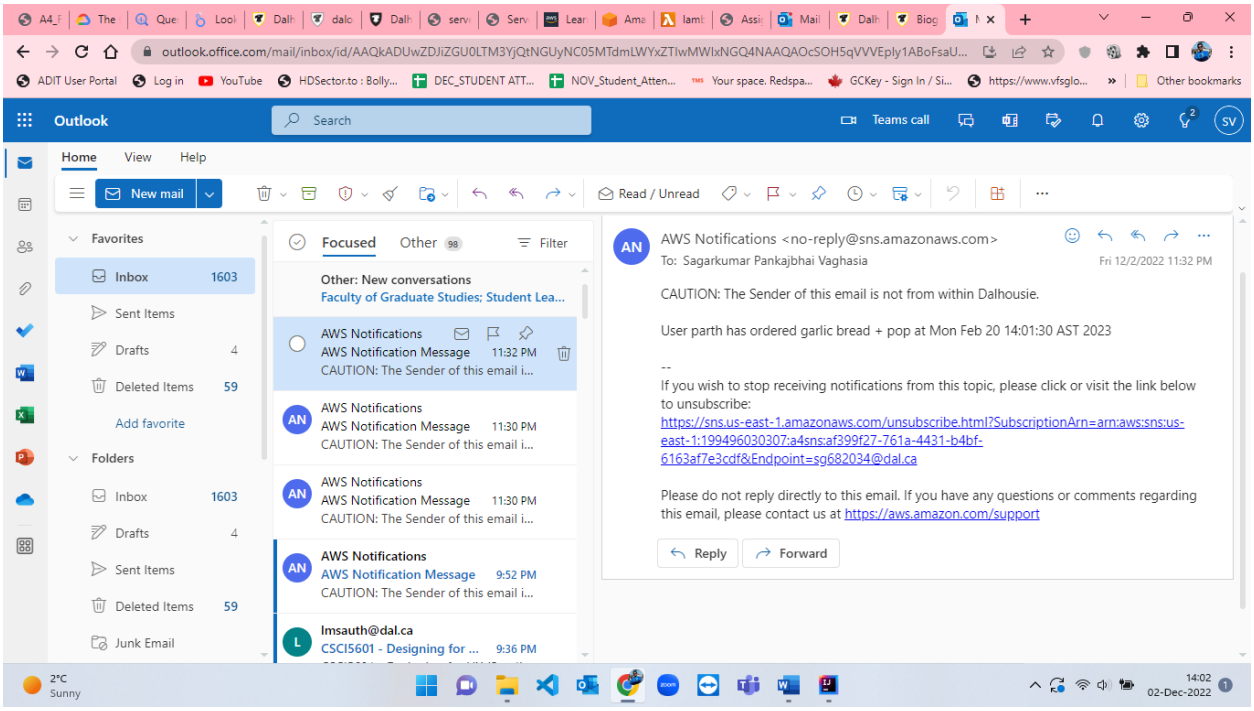

Figure 49: Email notification from SNS - 1.

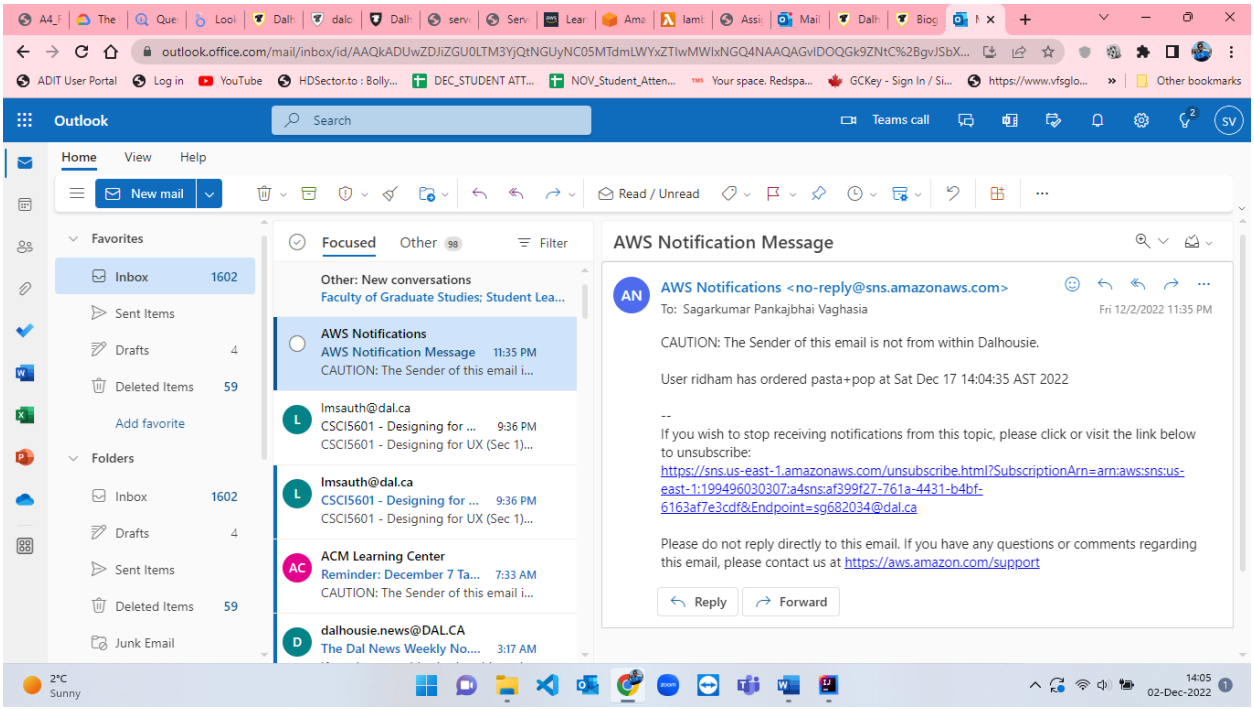

Figure 50: Email notification from SNS - 2.

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].

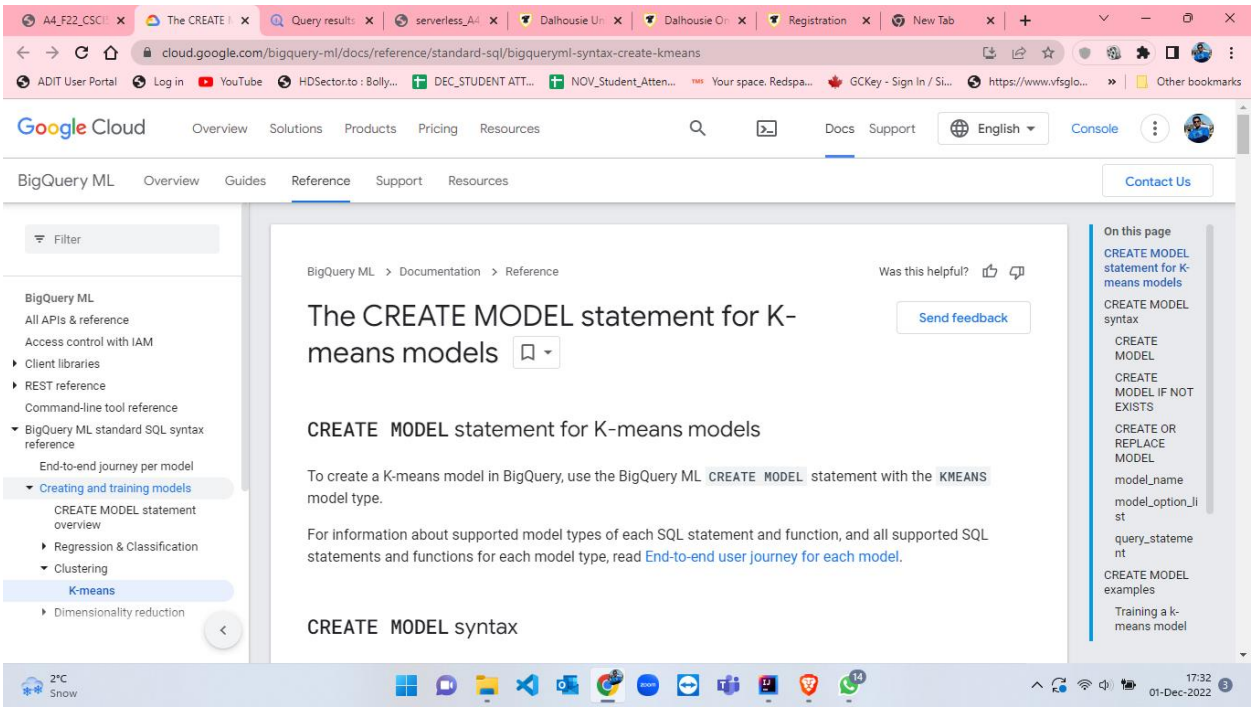

Figure 51: GCP BigQueryML documentation.

- 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.

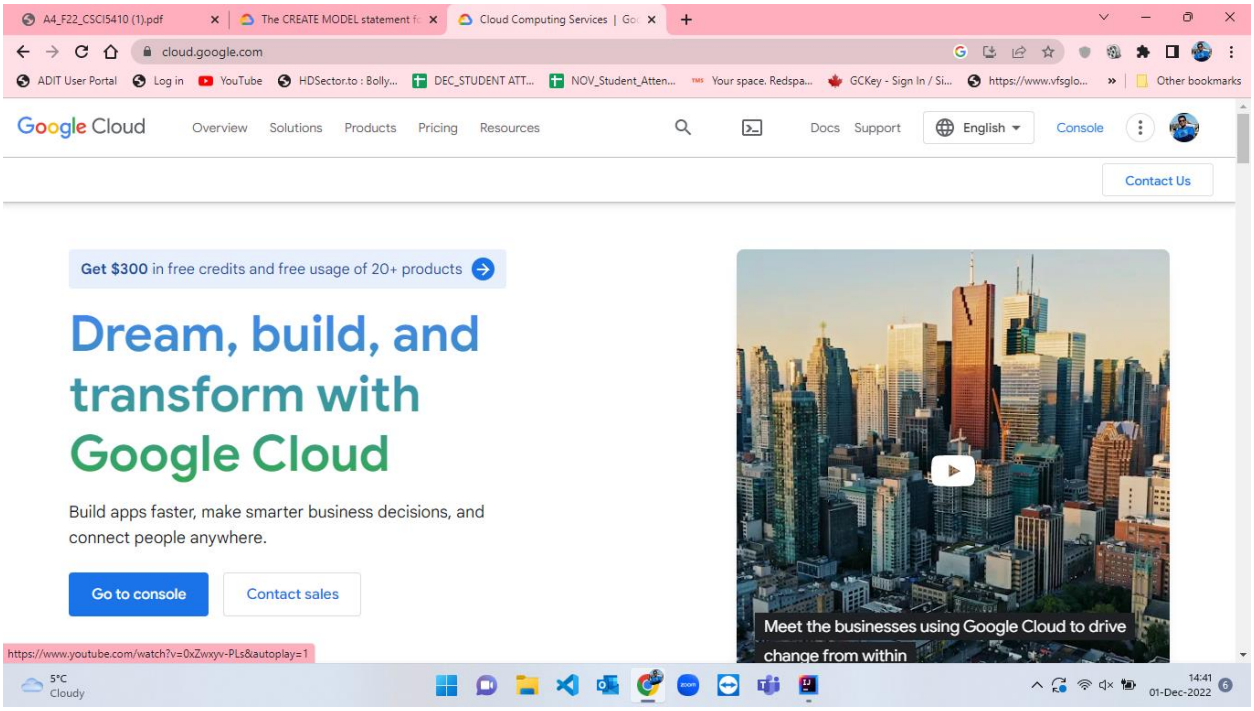

Figure 52: GCP homepage.

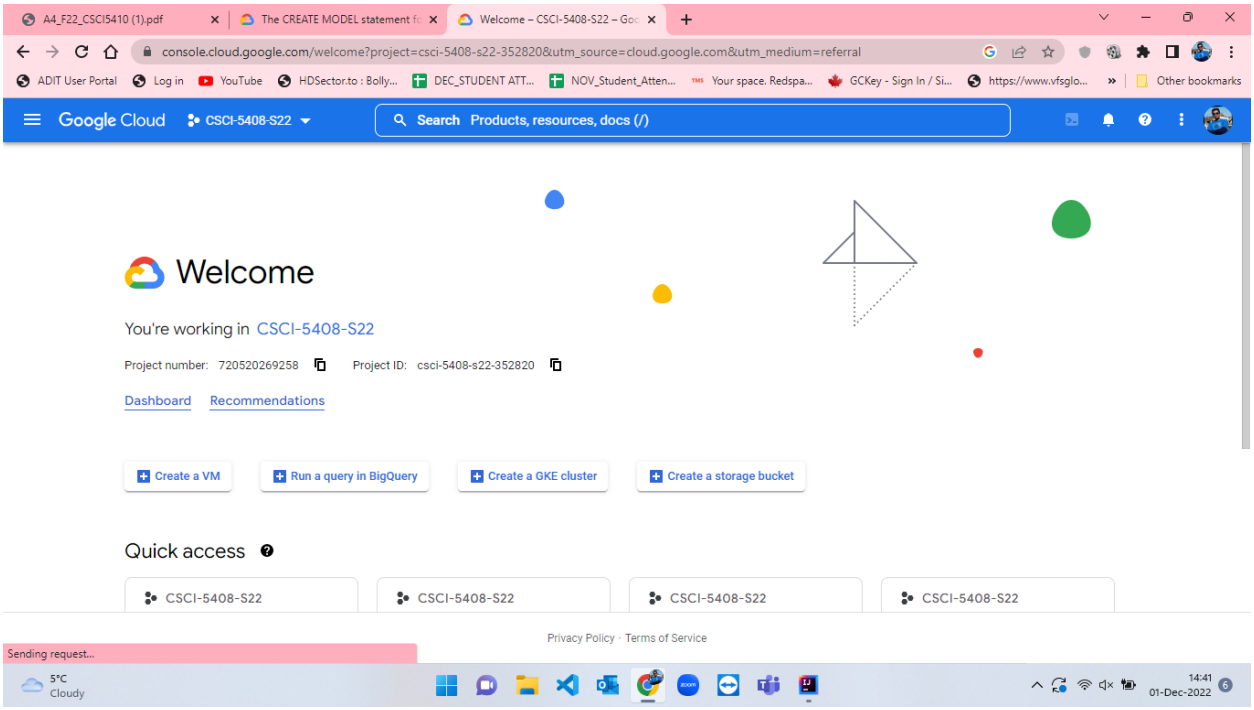

Figure 53: GCP console page.

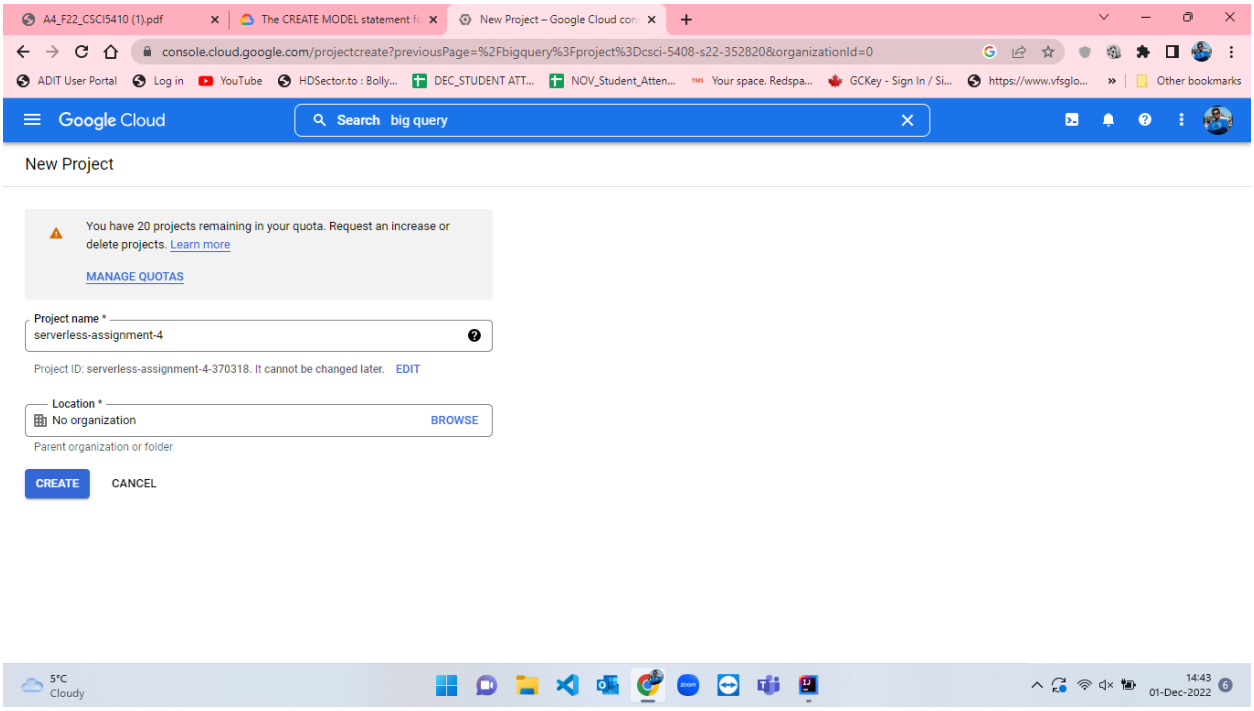

Figure 54: Project creation in GCP.

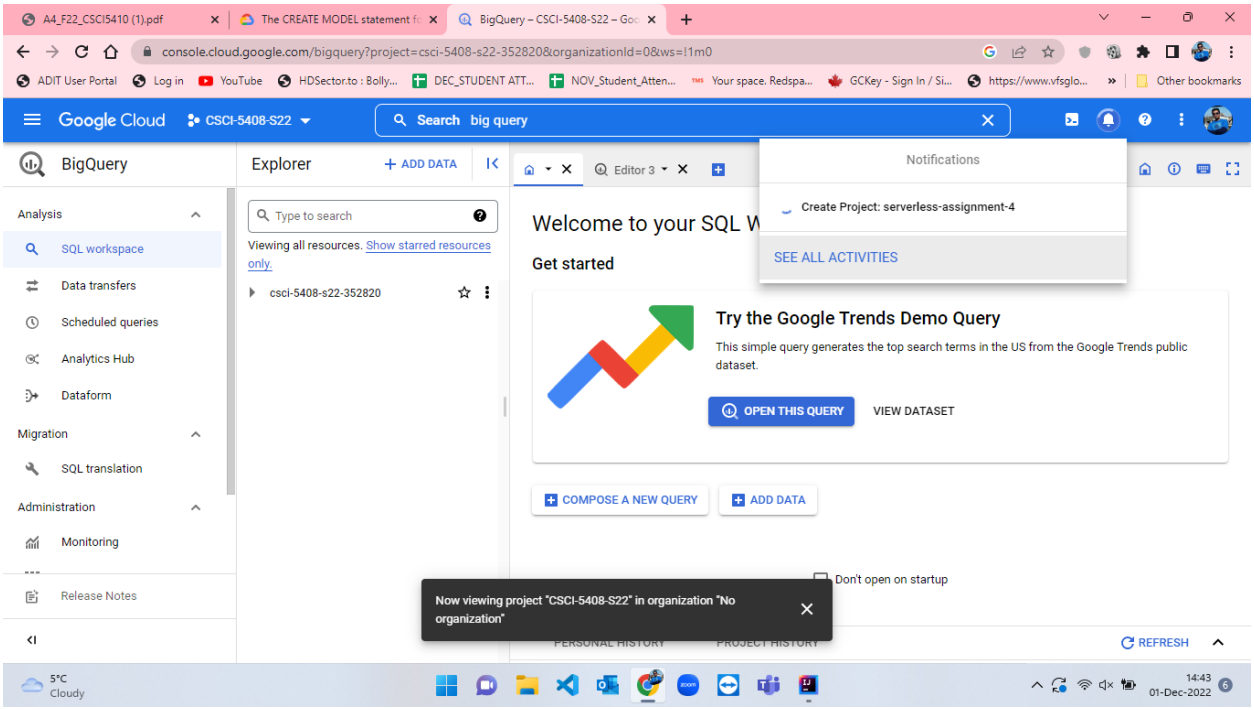

Figure 55: Project creation process taking place.

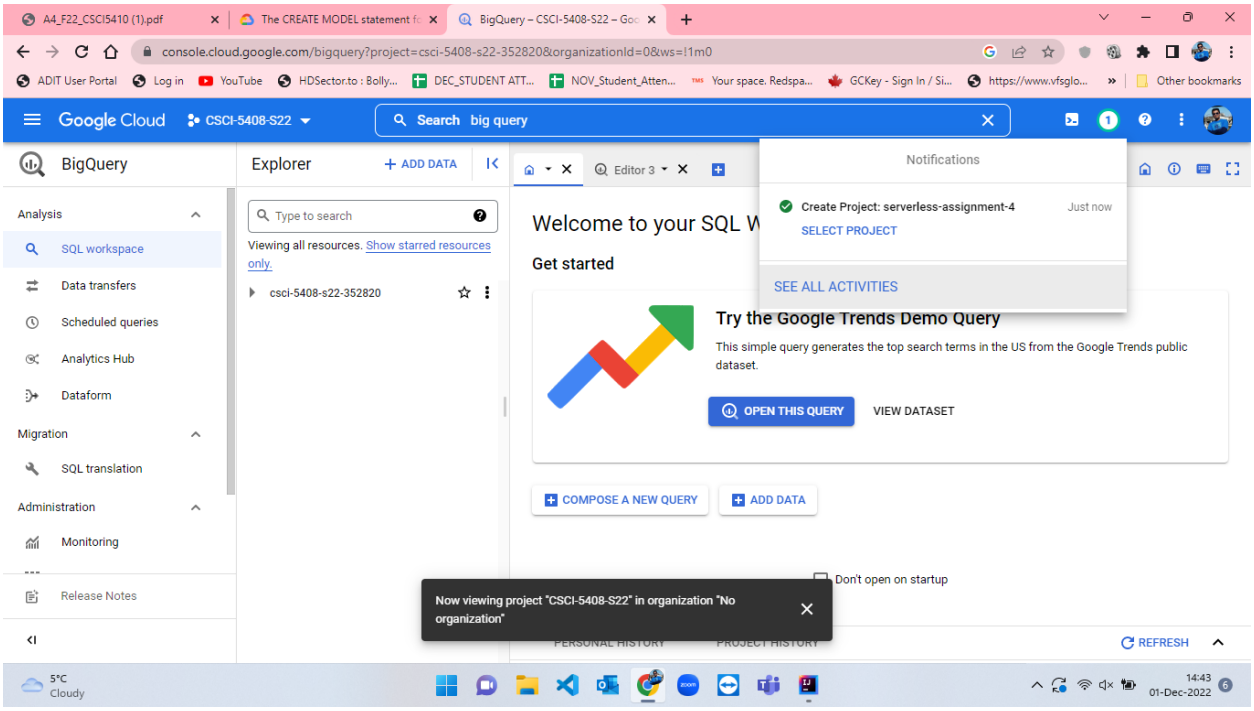

Figure 56: Successfully creation of project.

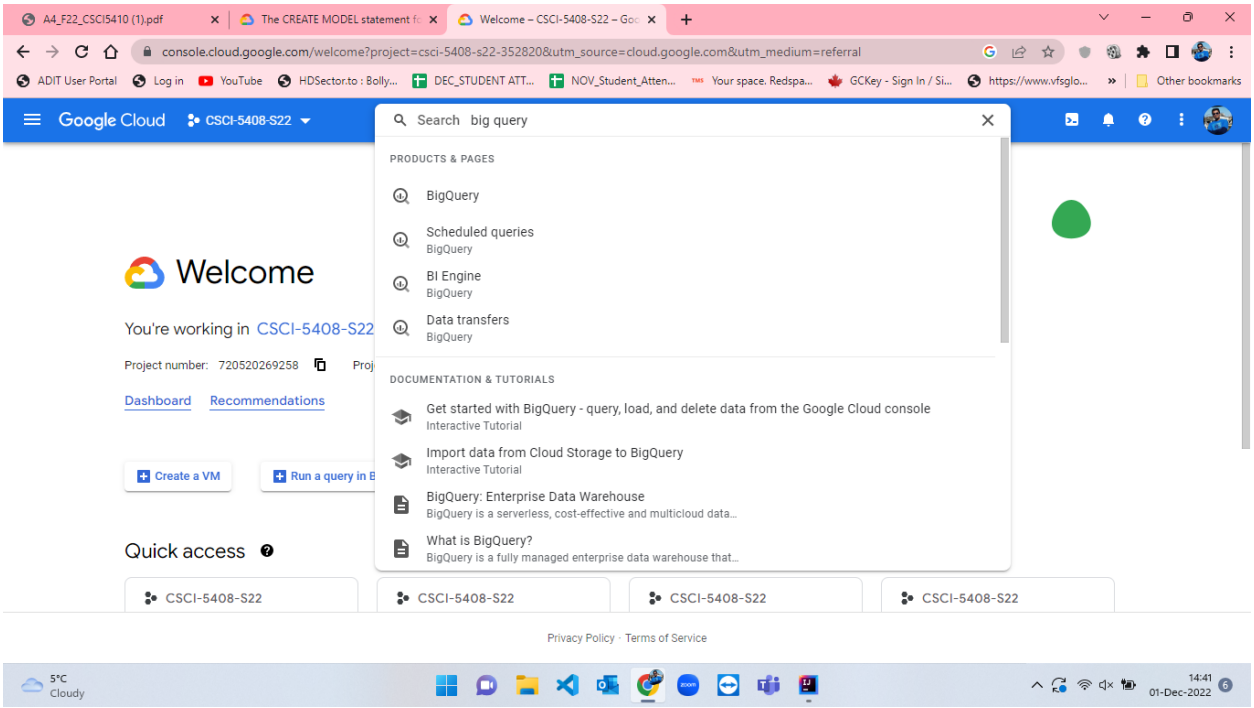

Figure 57: Searching for big query in search bar in GCP.

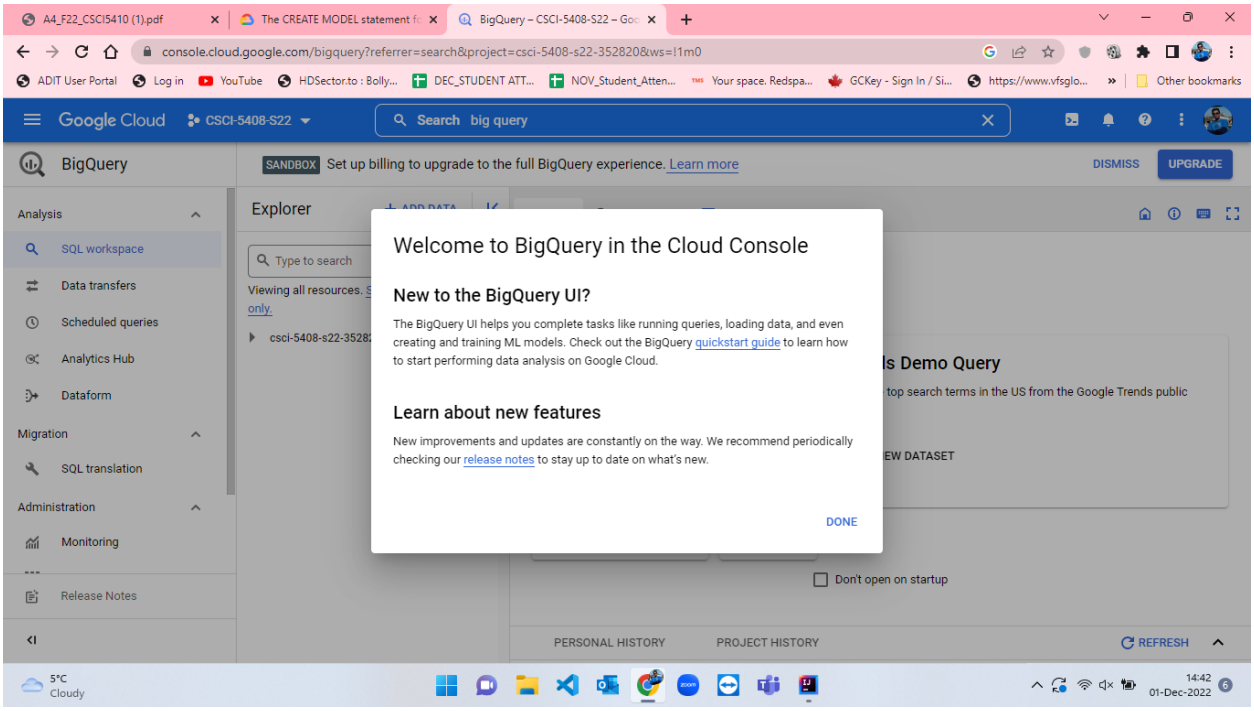

Figure 58: BigQuery welcome message.

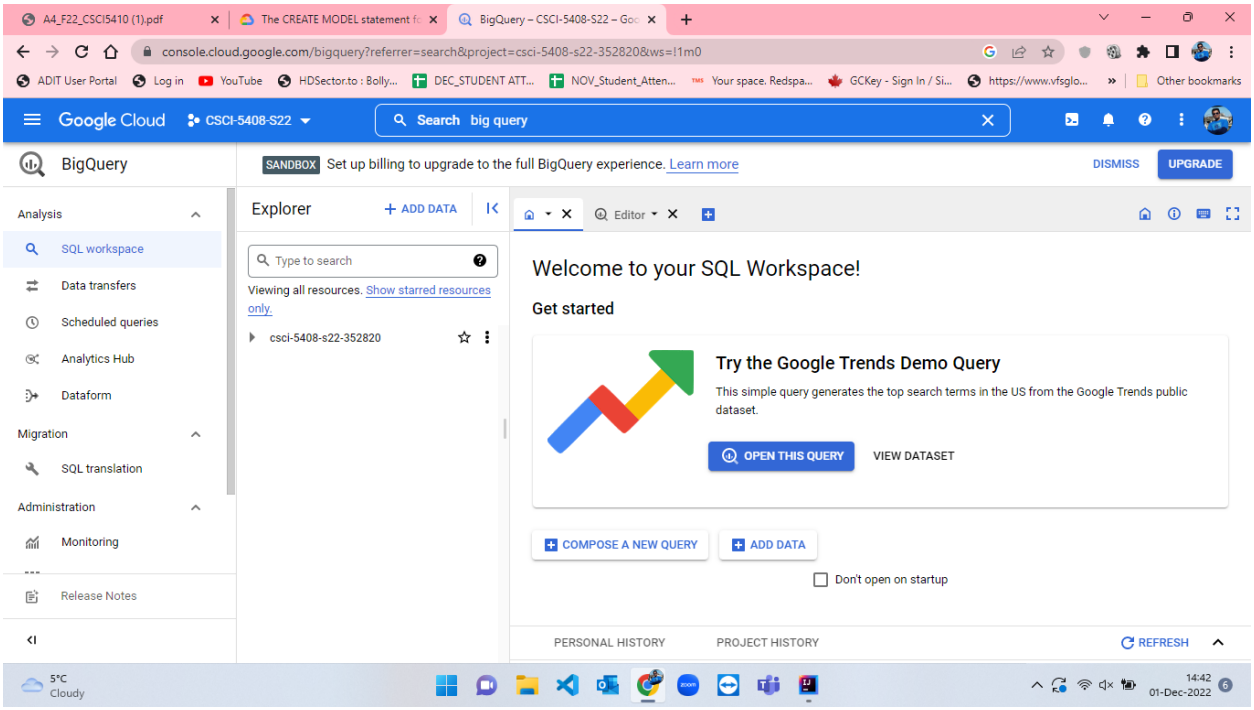

Figure 59: BigQuery Homepage.

- 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).

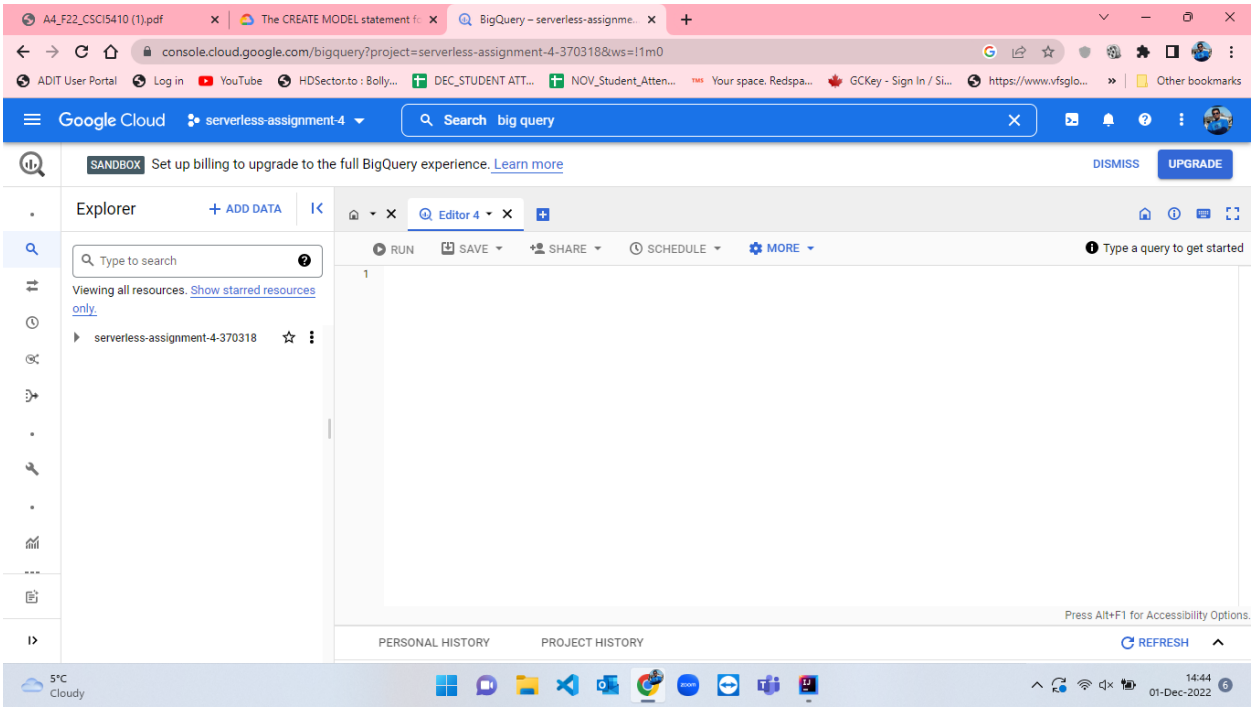

Figure 60: Selecting current project to work and opening editor to write query in GCP BigQuery.

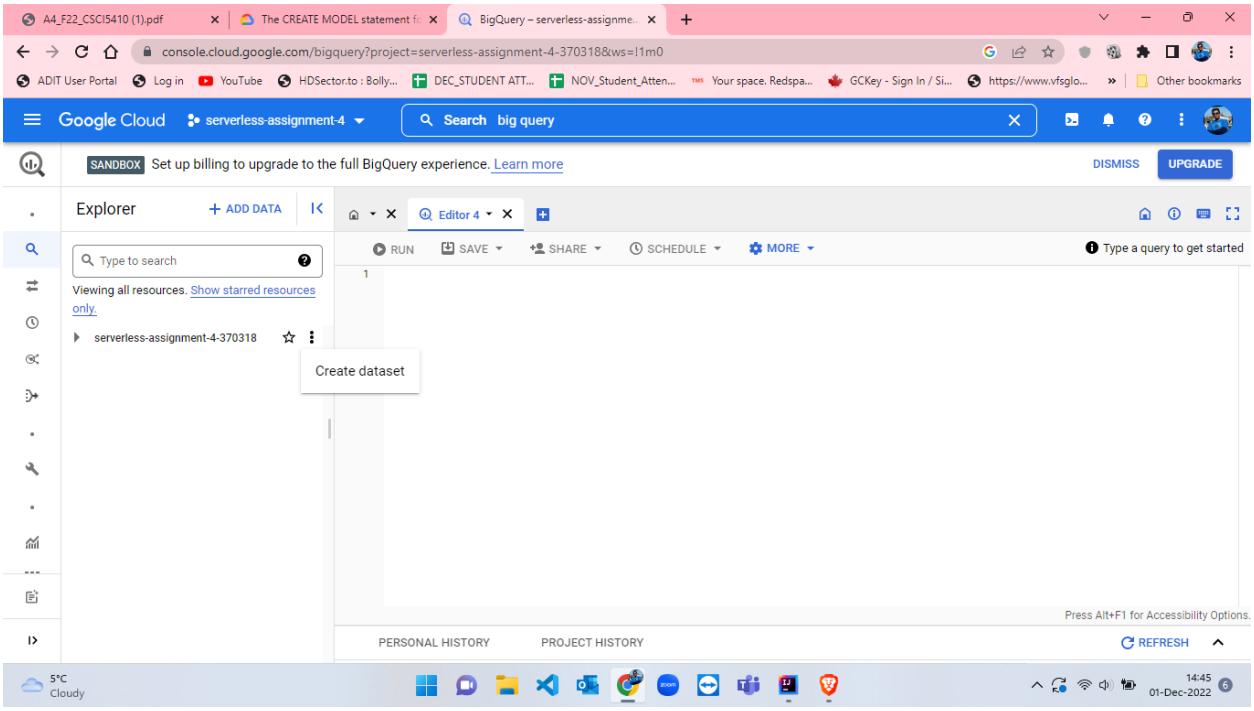

Figure 61: Create dataset in BigQuery.

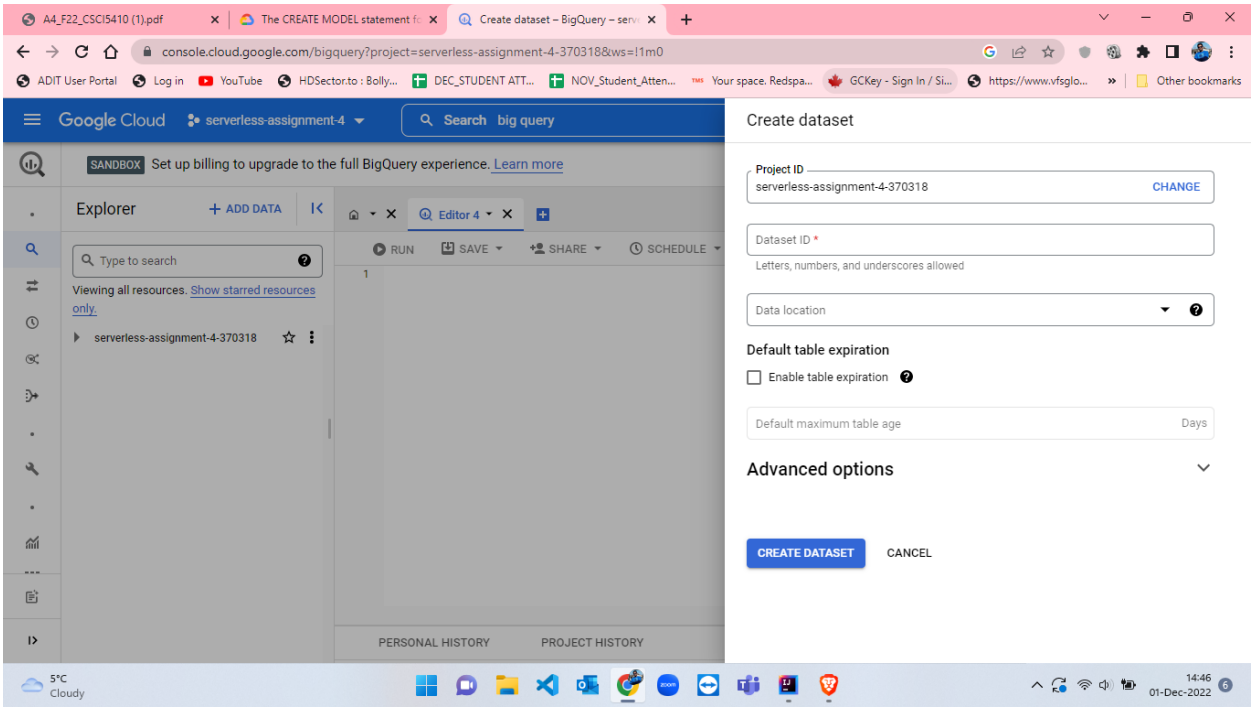

Figure 62: Create dataset blank page in BigQuery.

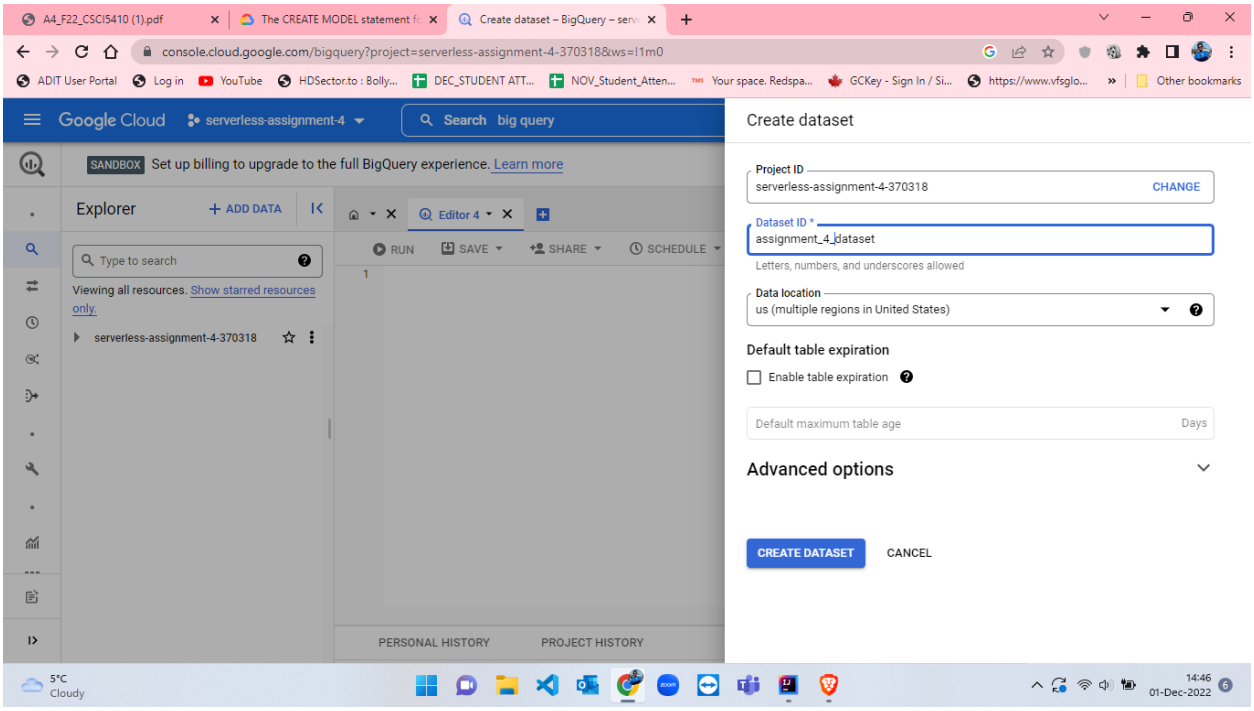

Figure 63: Filling details to create dataset.

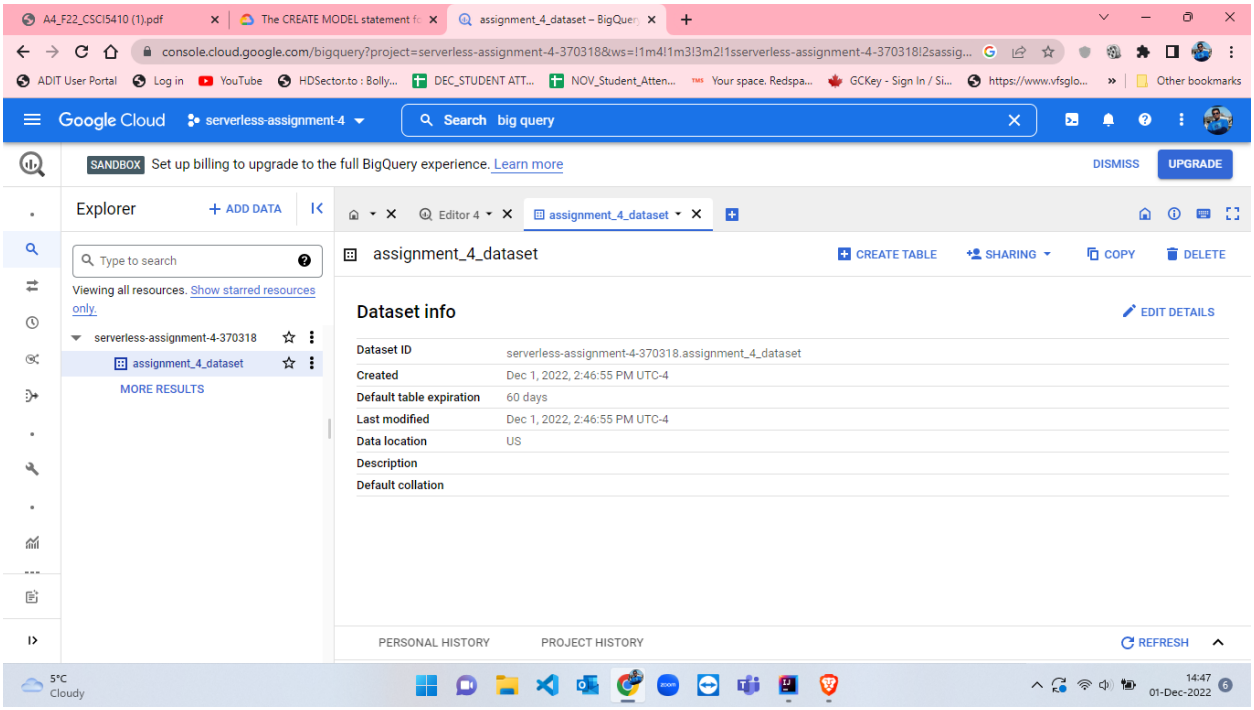

Figure 64: Successful creation of dataset

- 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.

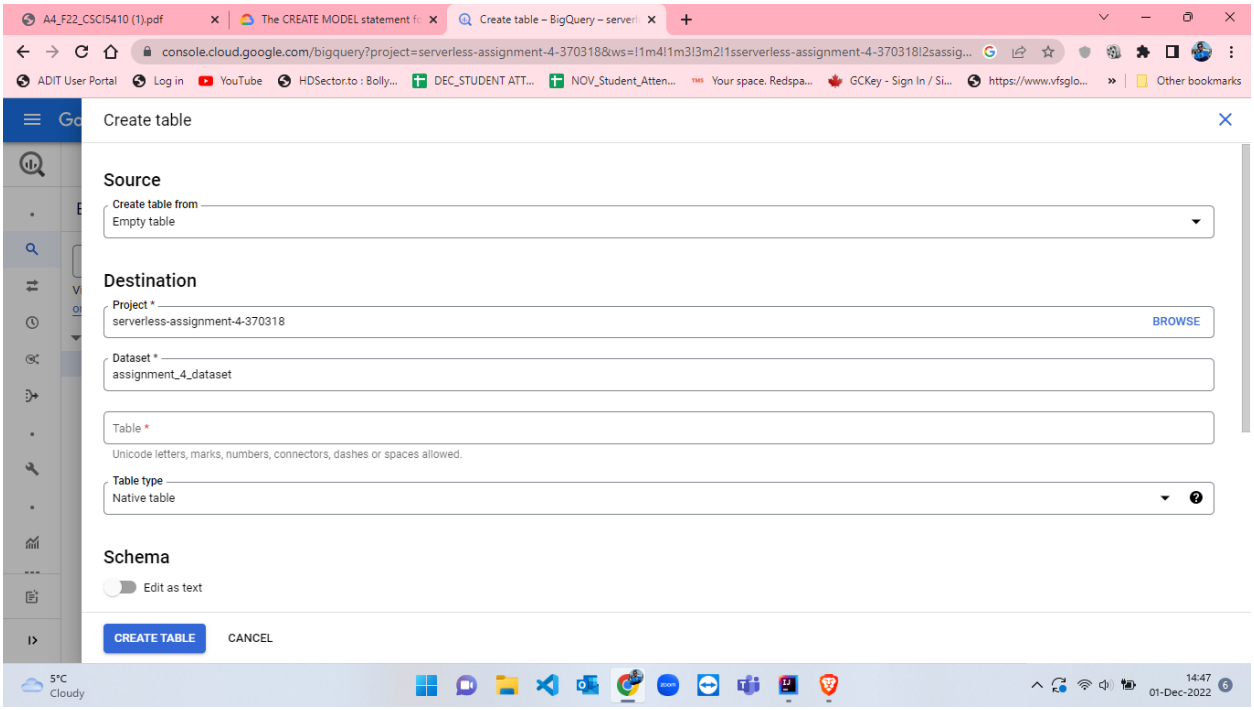

Figure 65: Create table empty form in BigQuery.

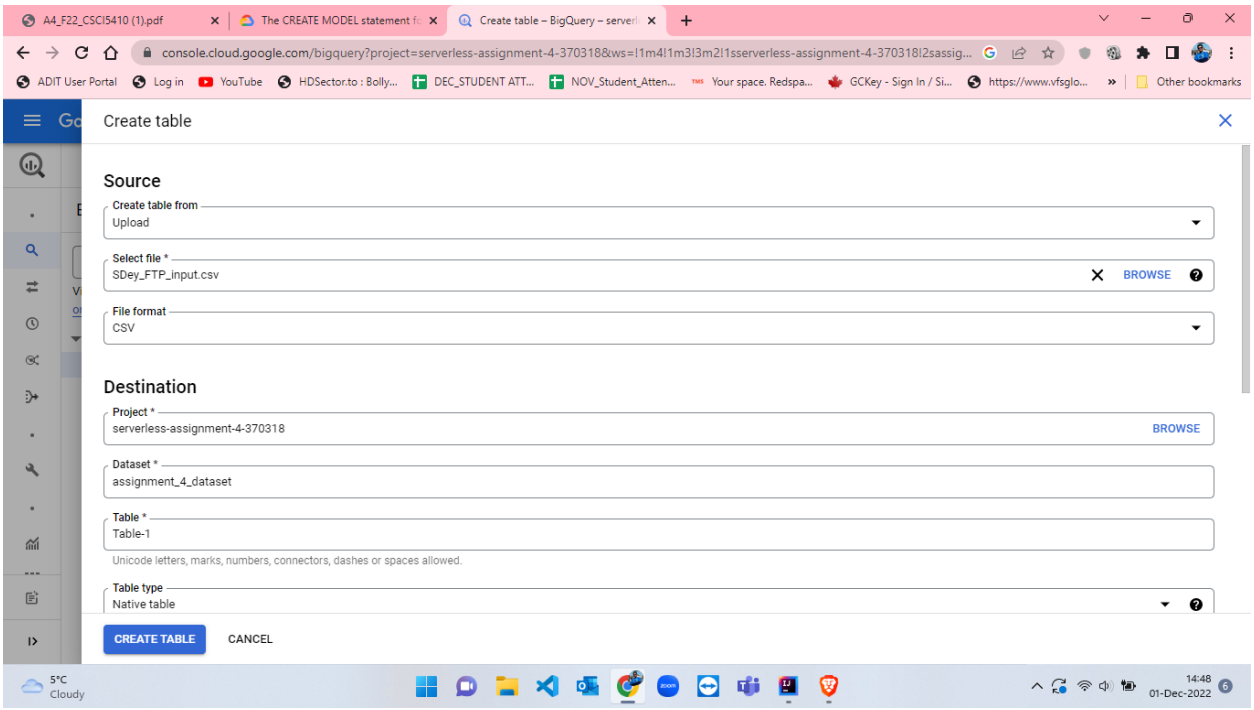

Figure 66: Filling details in create table form part-1.

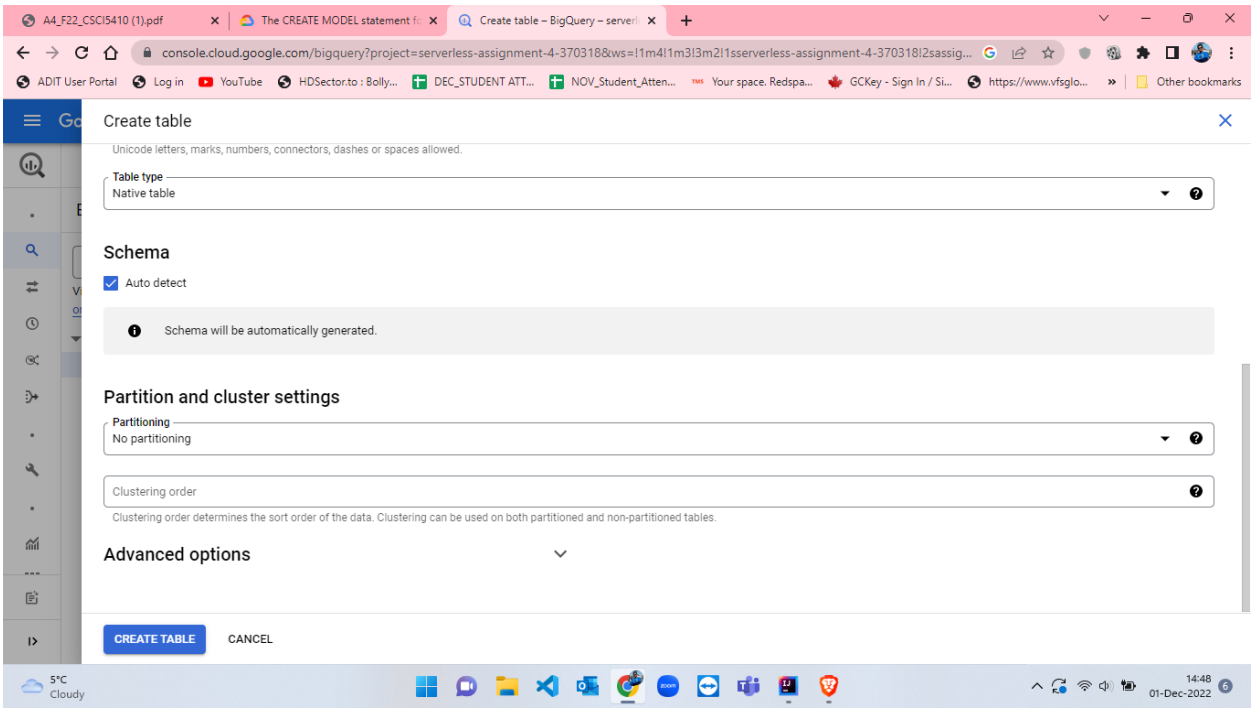

Figure 67: Filling details in create table form part-2.

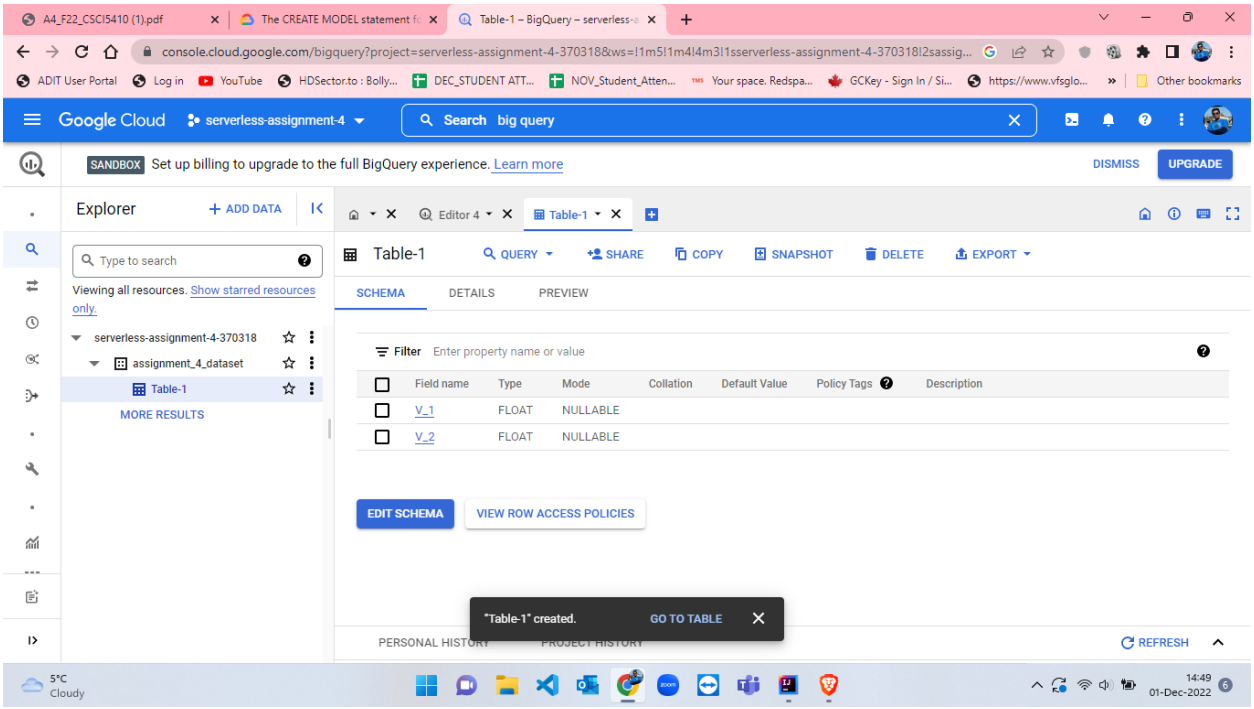

Figure 68: Successful creation of Table-.

- 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.

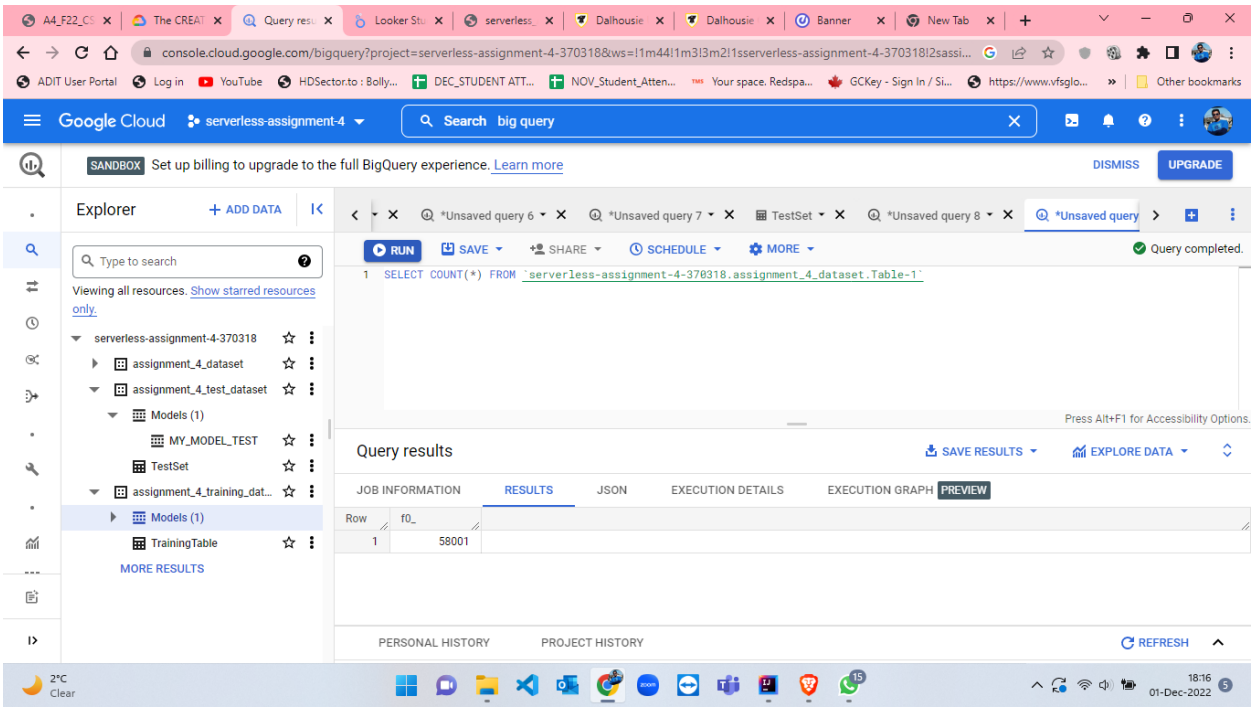

Figure 69: Table-1 number of rows.

- After creating Table-1, I have to change the query settings to execute the query. So, I have clicked on “More” option(figire-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.

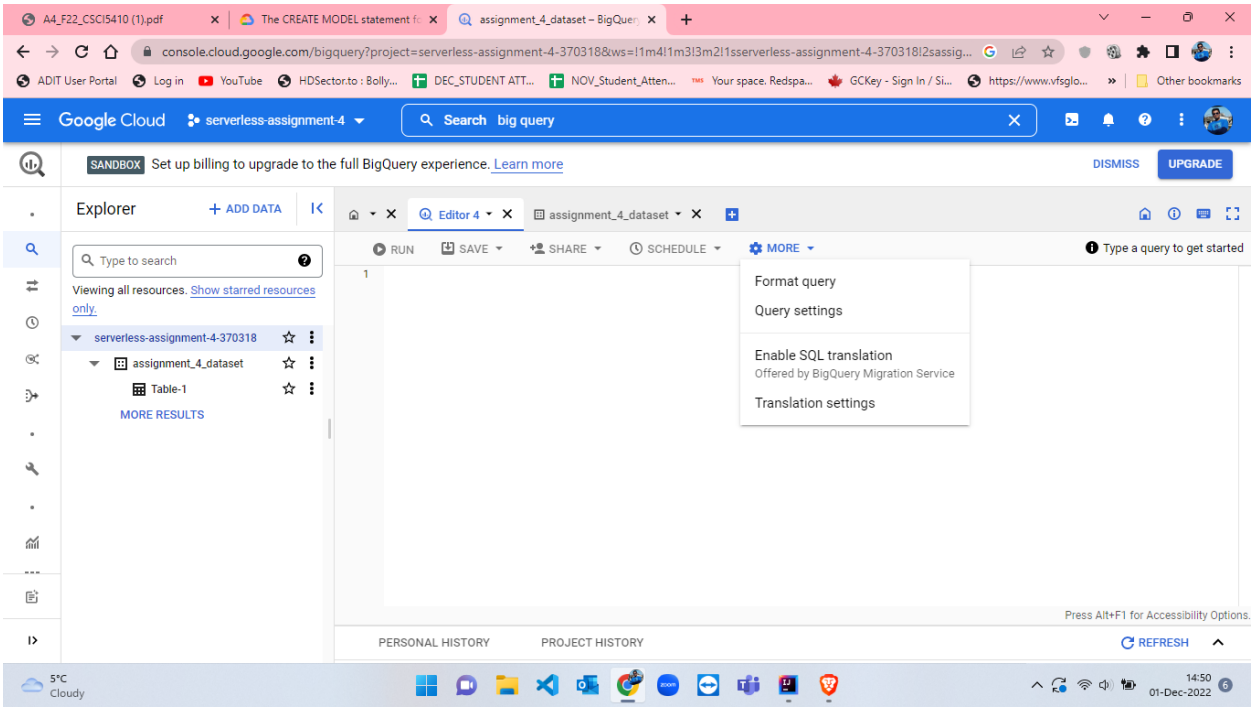

Figure 70: More options to change Query Settings.

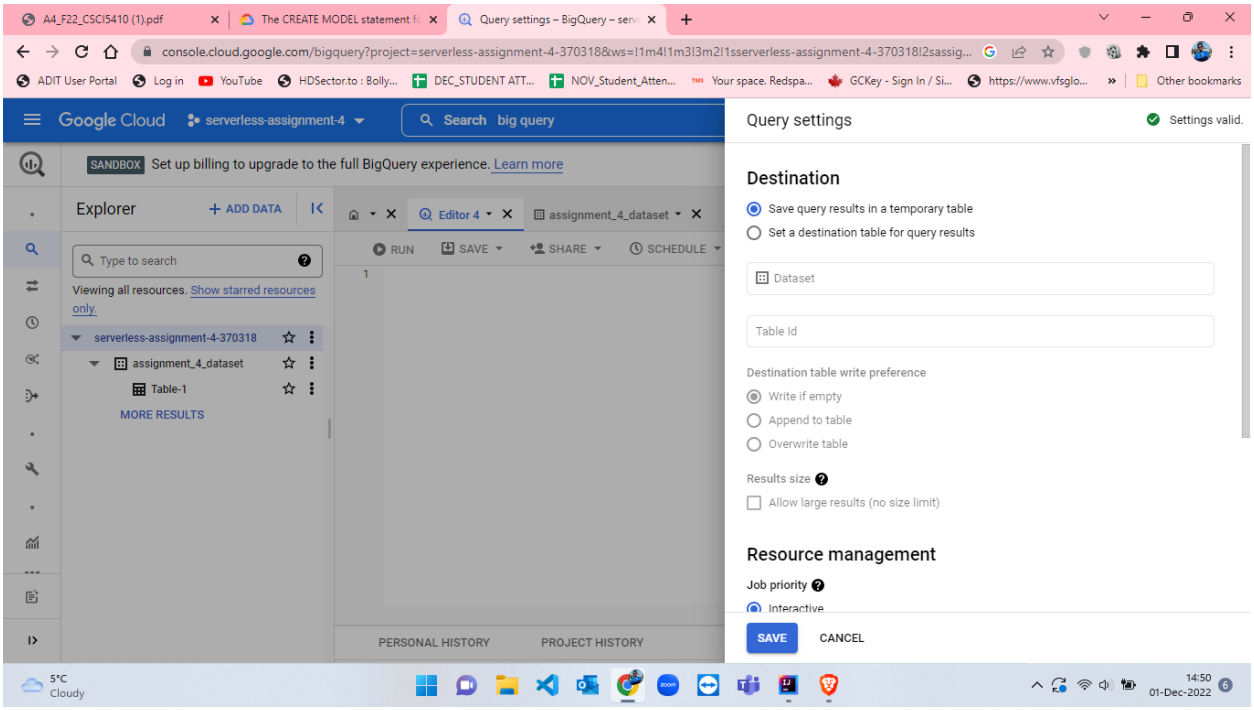

Figure 71: Query Settings in BigQuery step-1.

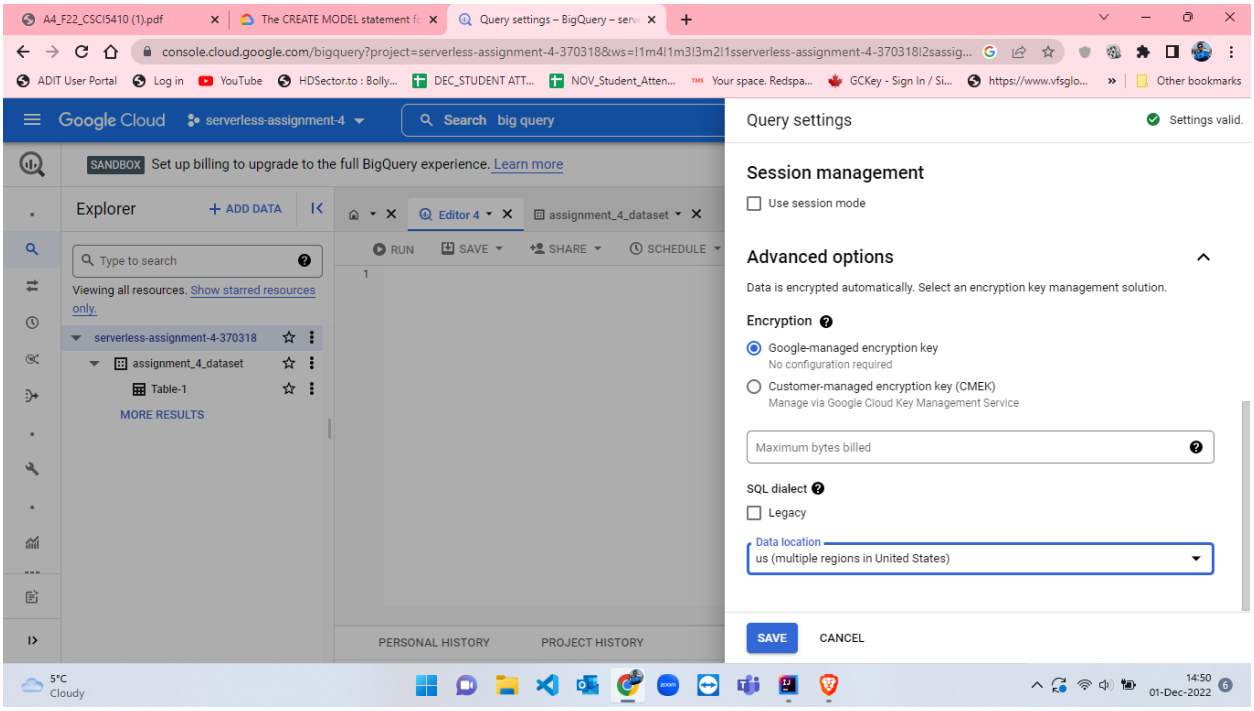

Figure 72: Query Settings in BigQuery step-2.

- 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.

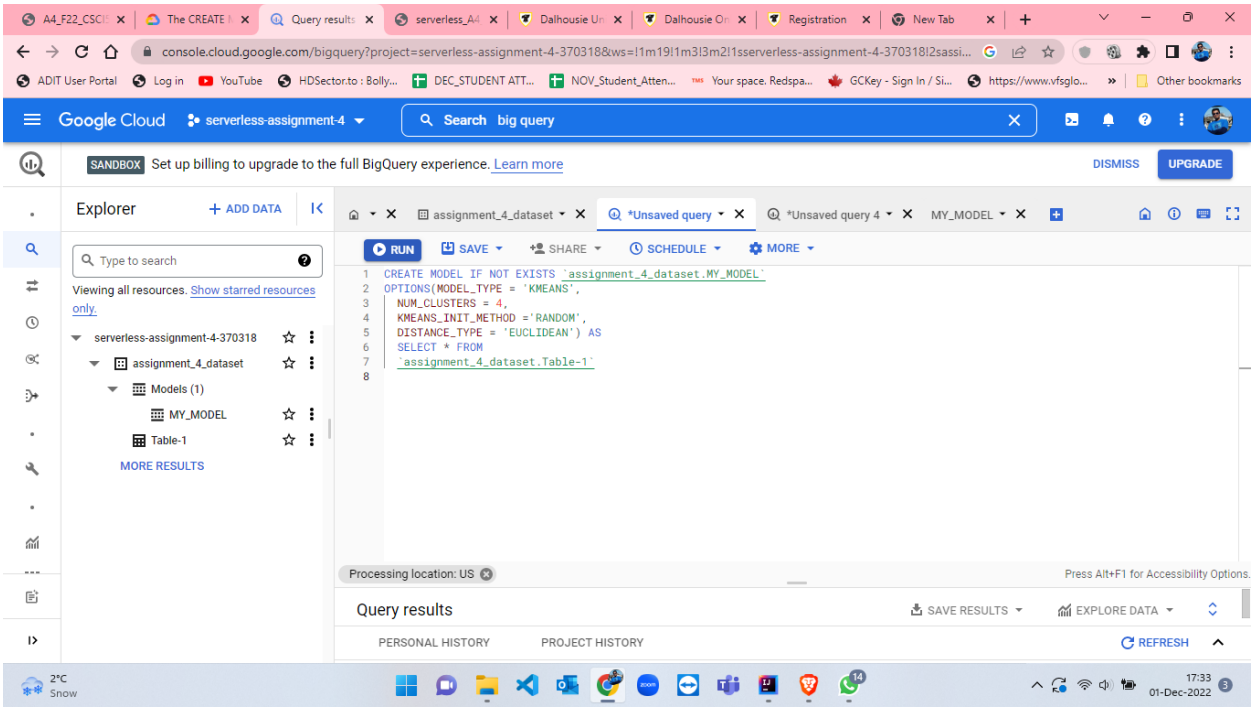

Figure 73: Query to create model in BigQuery[10].

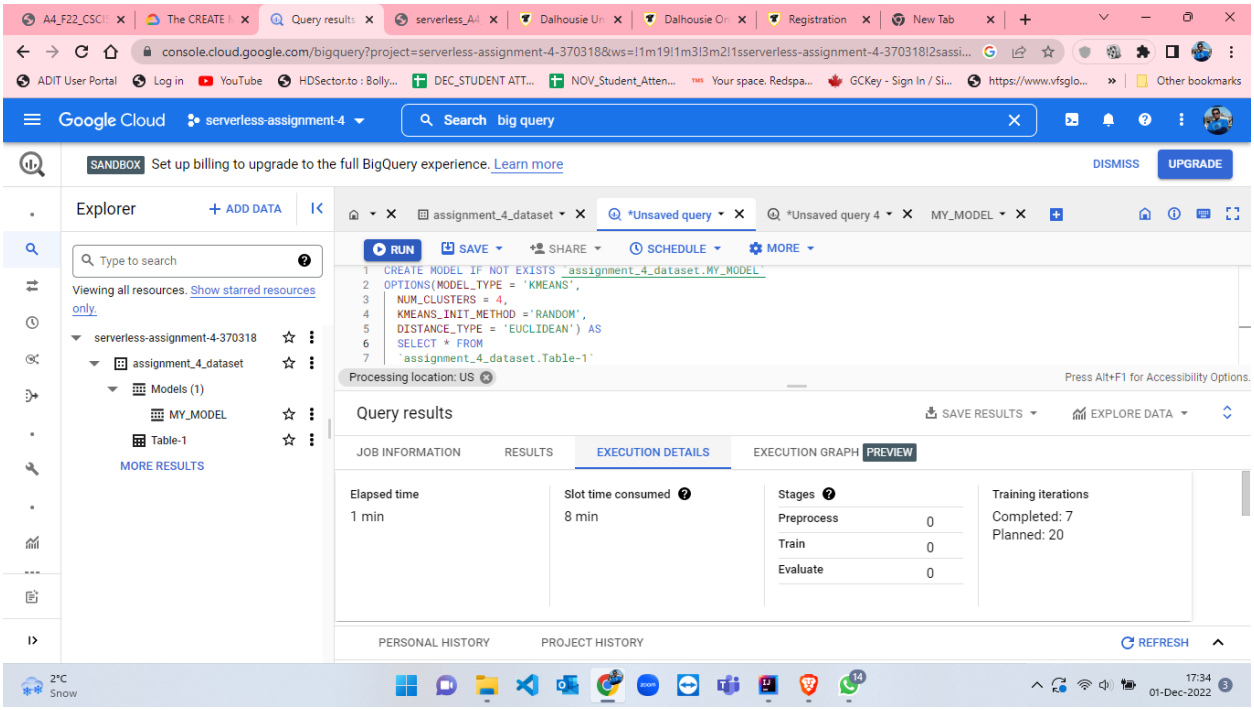

Figure 74: Query Execution details when KMeans model is ready part-1.

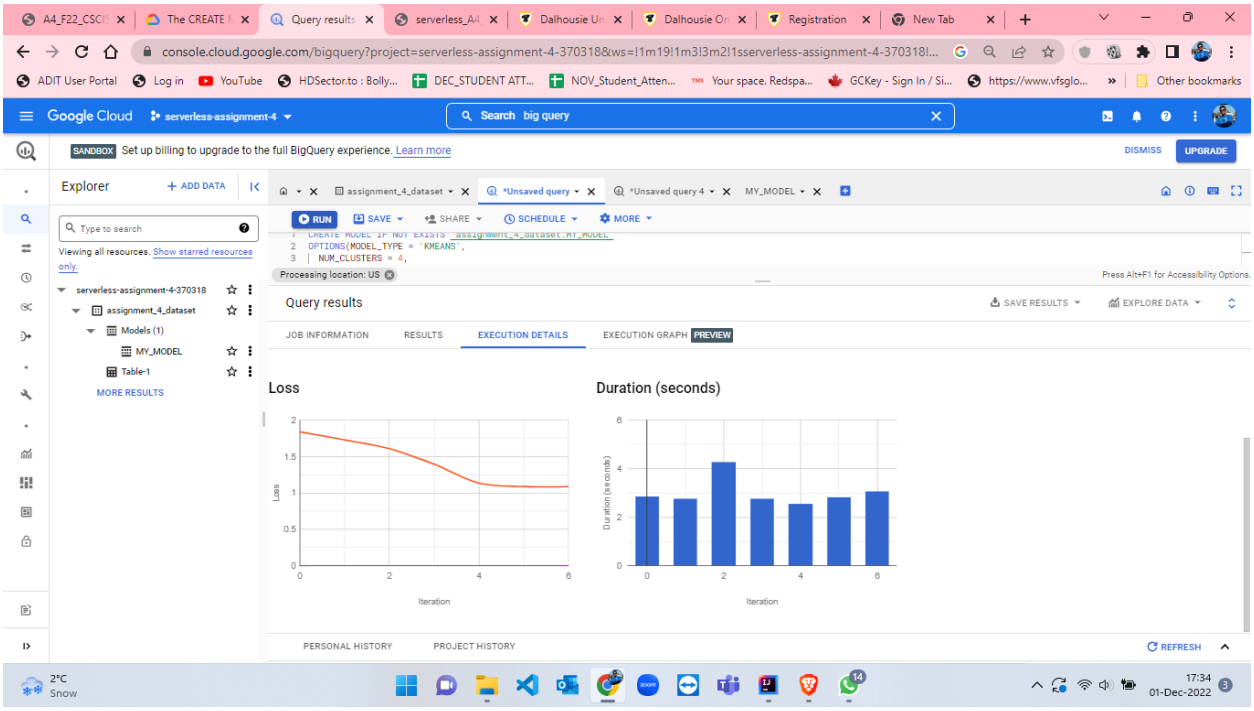

Figure 75: Query Execution details when KMeans model is ready part-2.

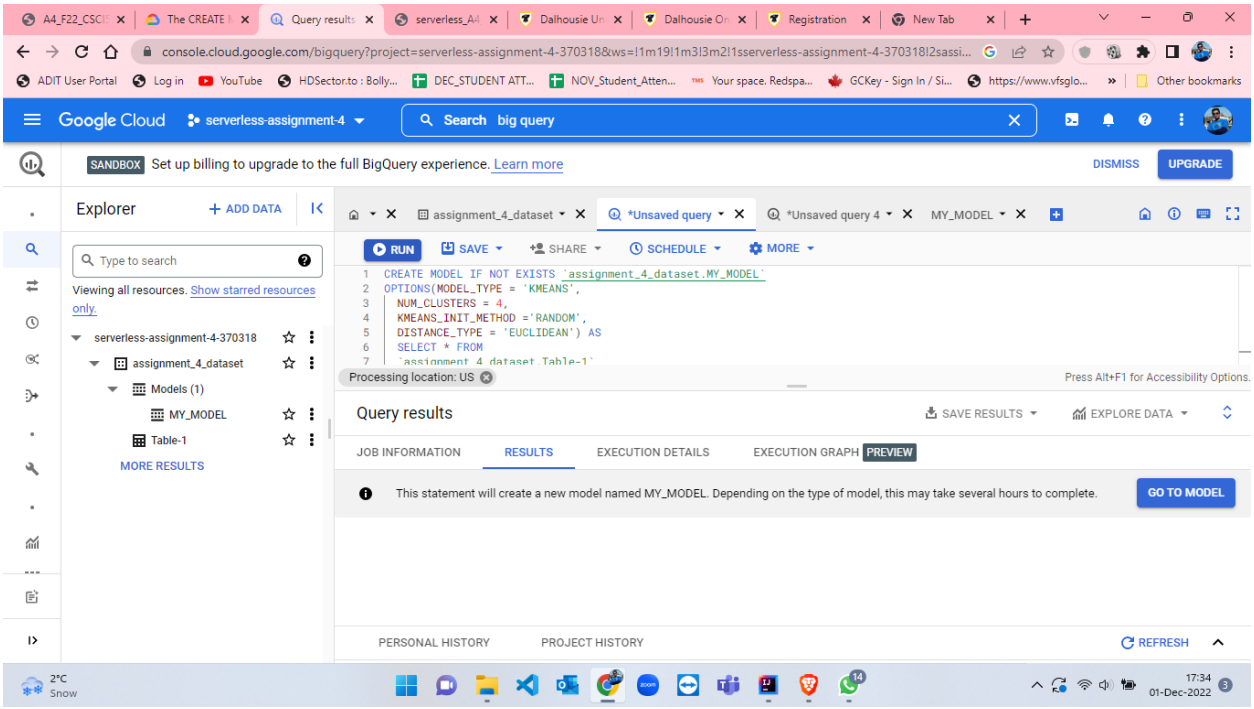

Figure 76: Query results when KMeans model is ready.

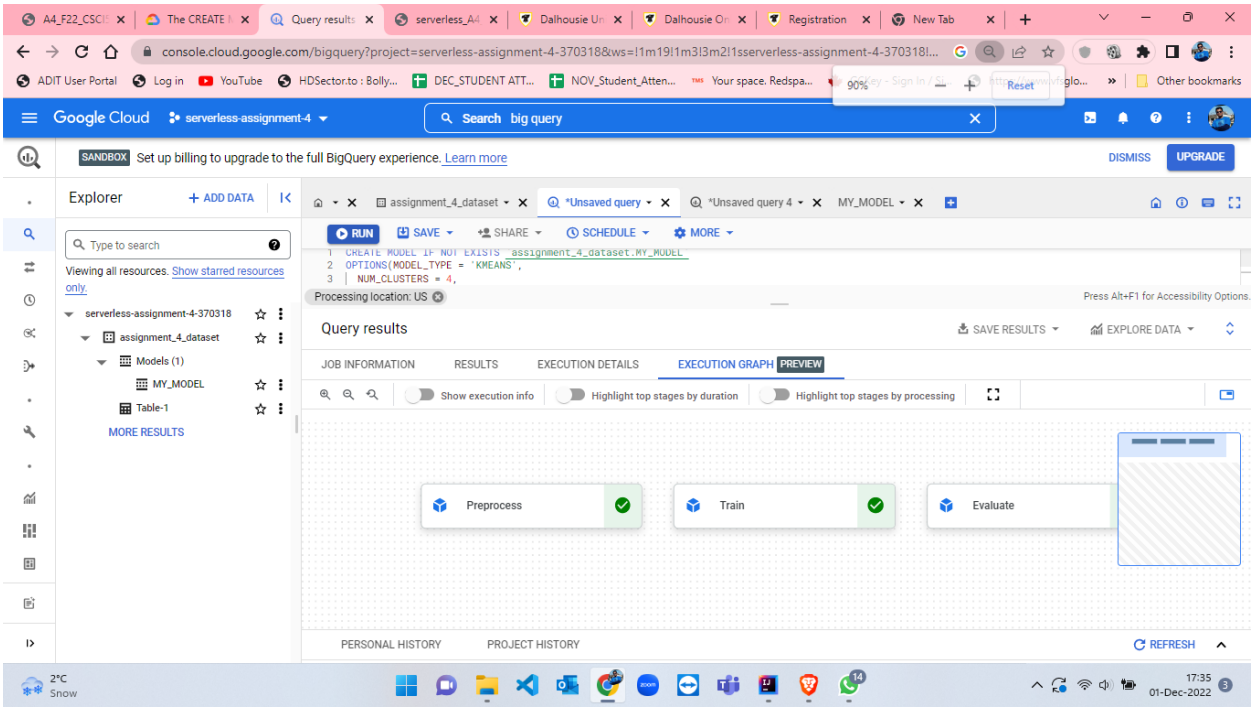

Figure 77: Query Execution Graph when KMeans model is ready.

- 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.

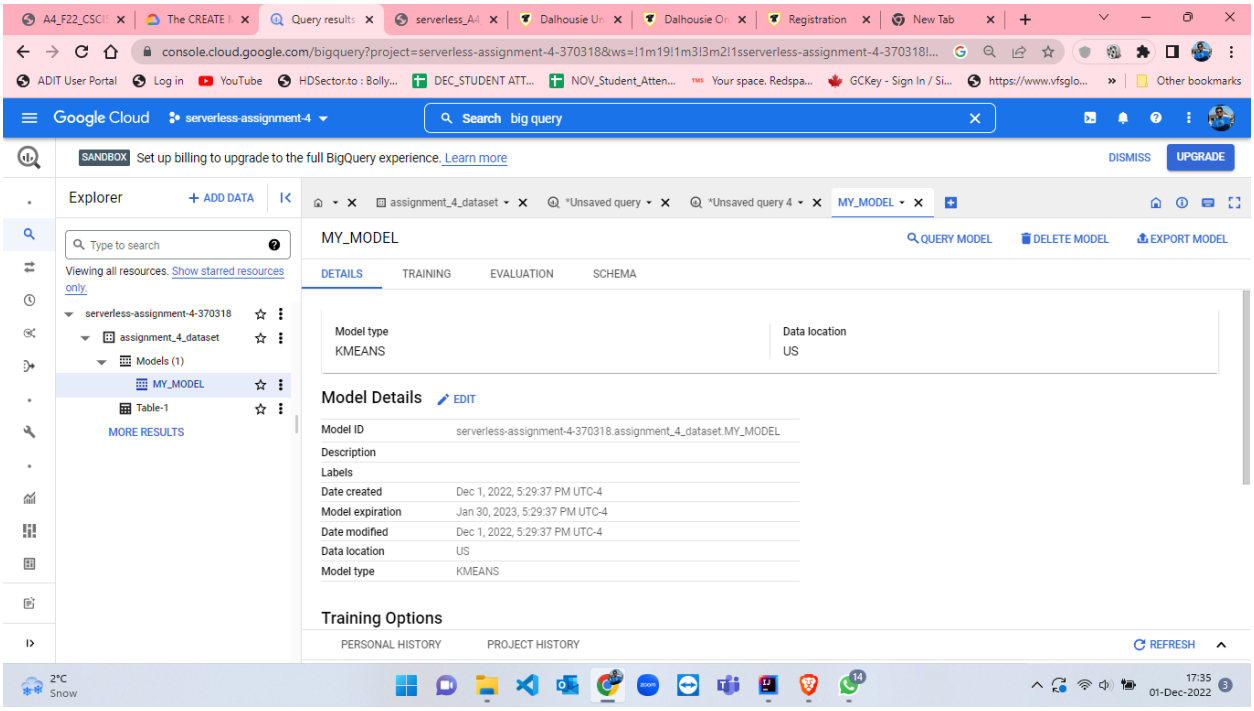

Figure 78: Model details of MY_MODEL part-1.

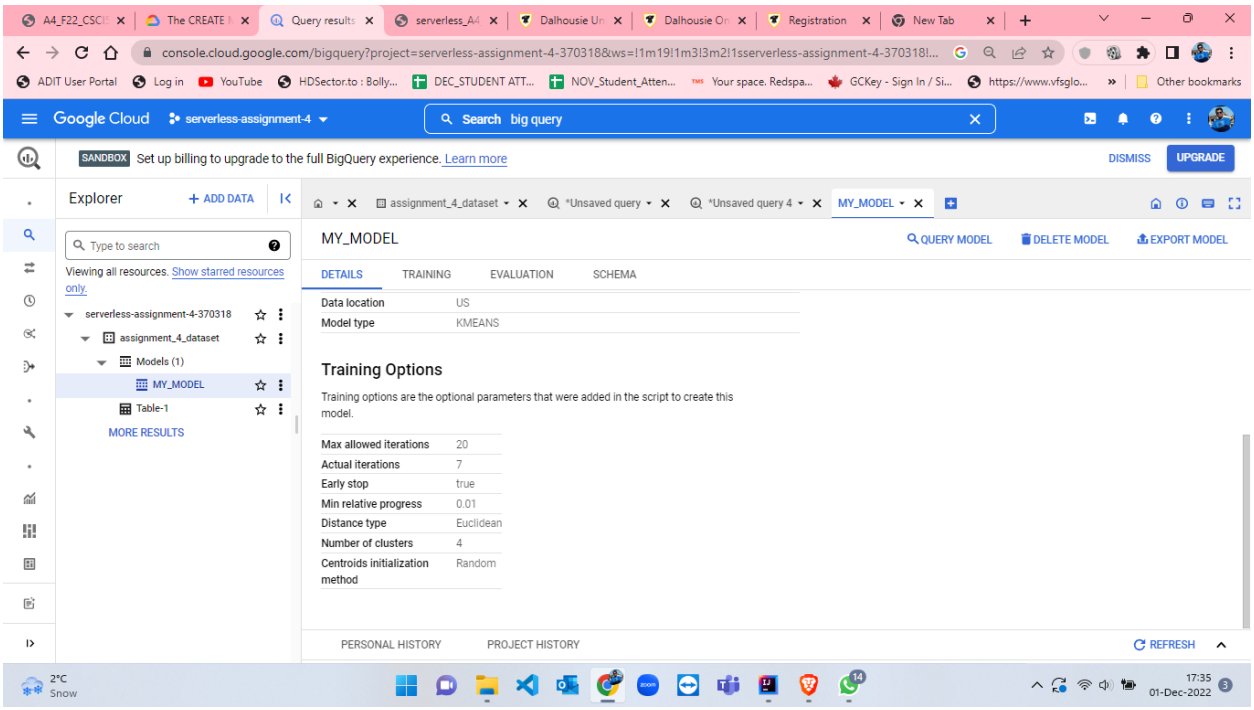

Figure 79: Model details of MY_MODEL part-2.

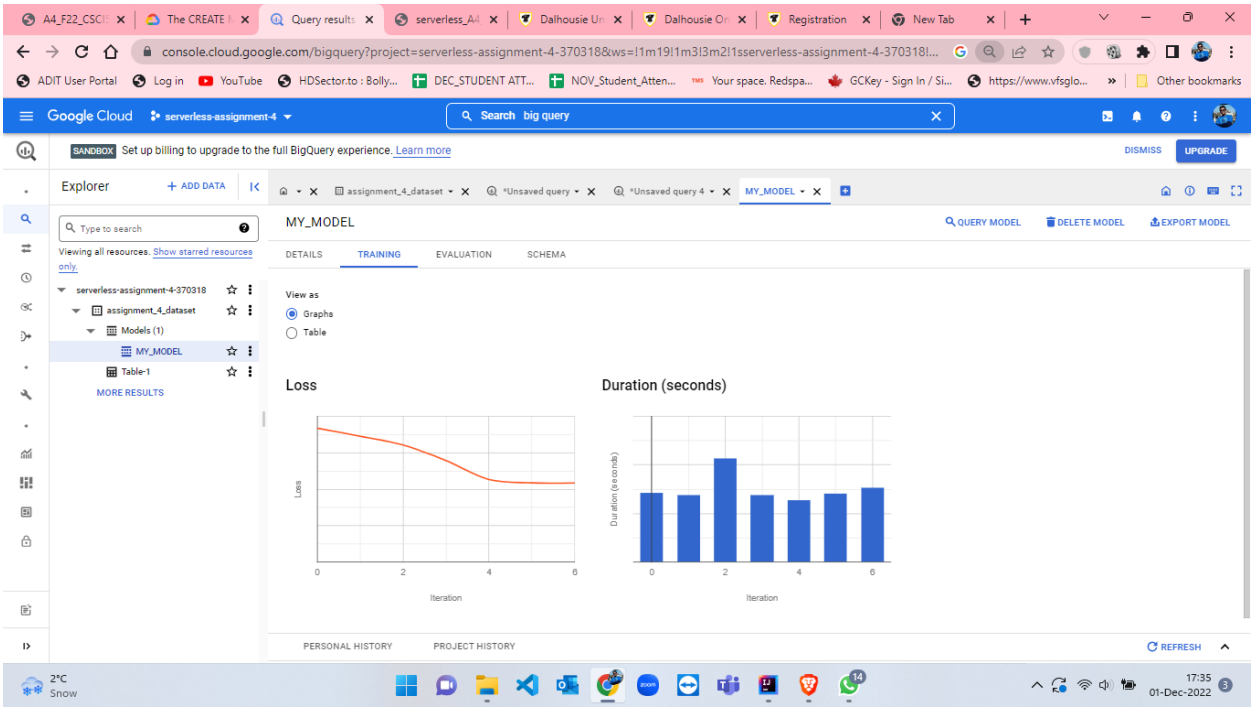

Figure 80: Training details of MY_MODEL part-1.

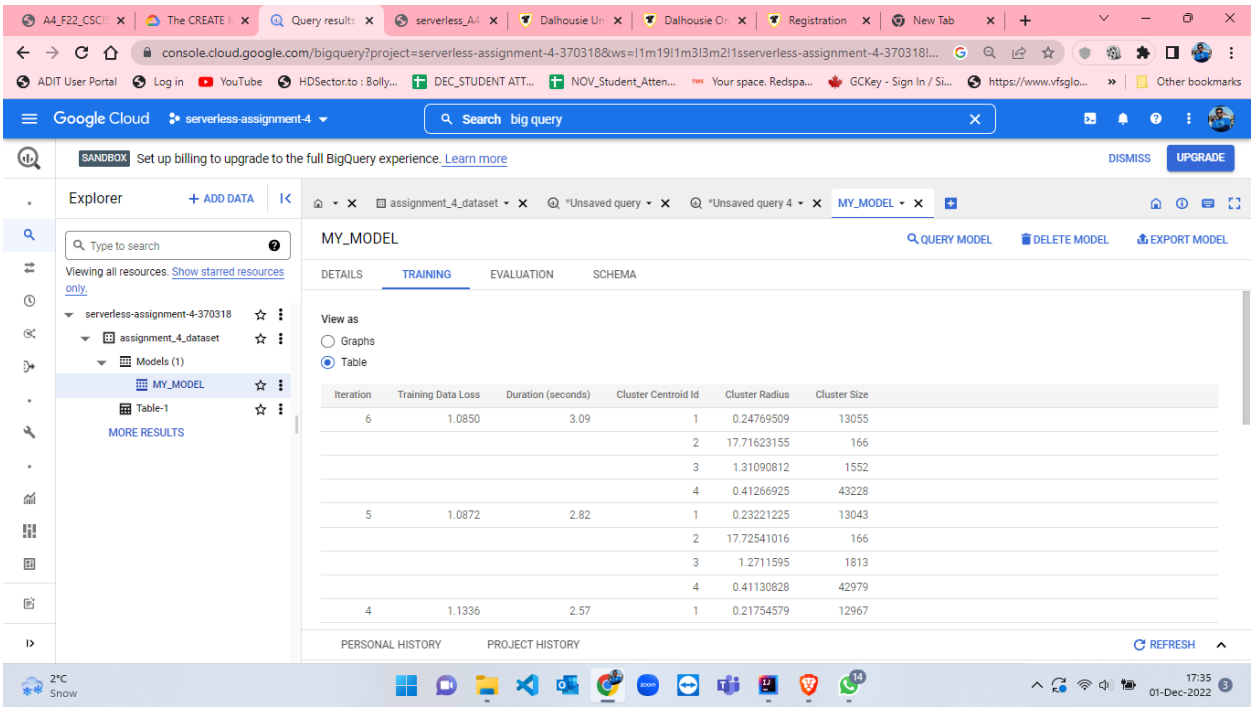

Figure 81: Training details of MY_MODEL part-2.

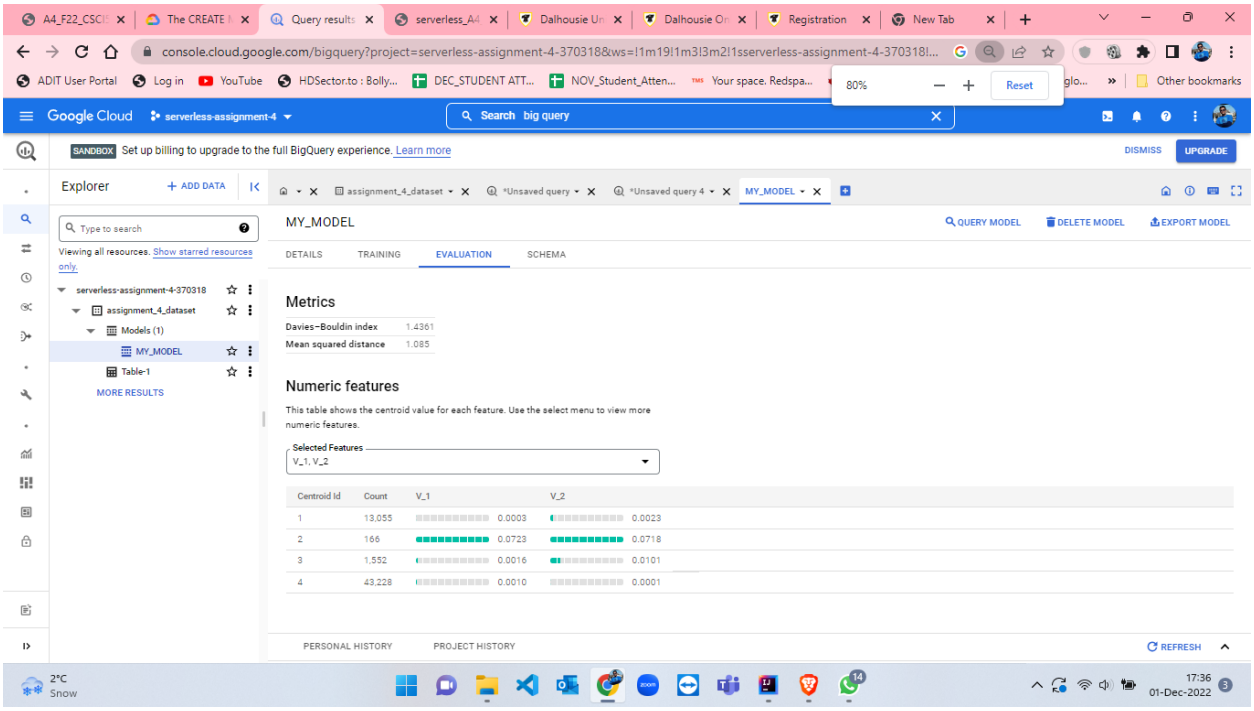

Figure 82: Evaluation details of MY_MODEL.

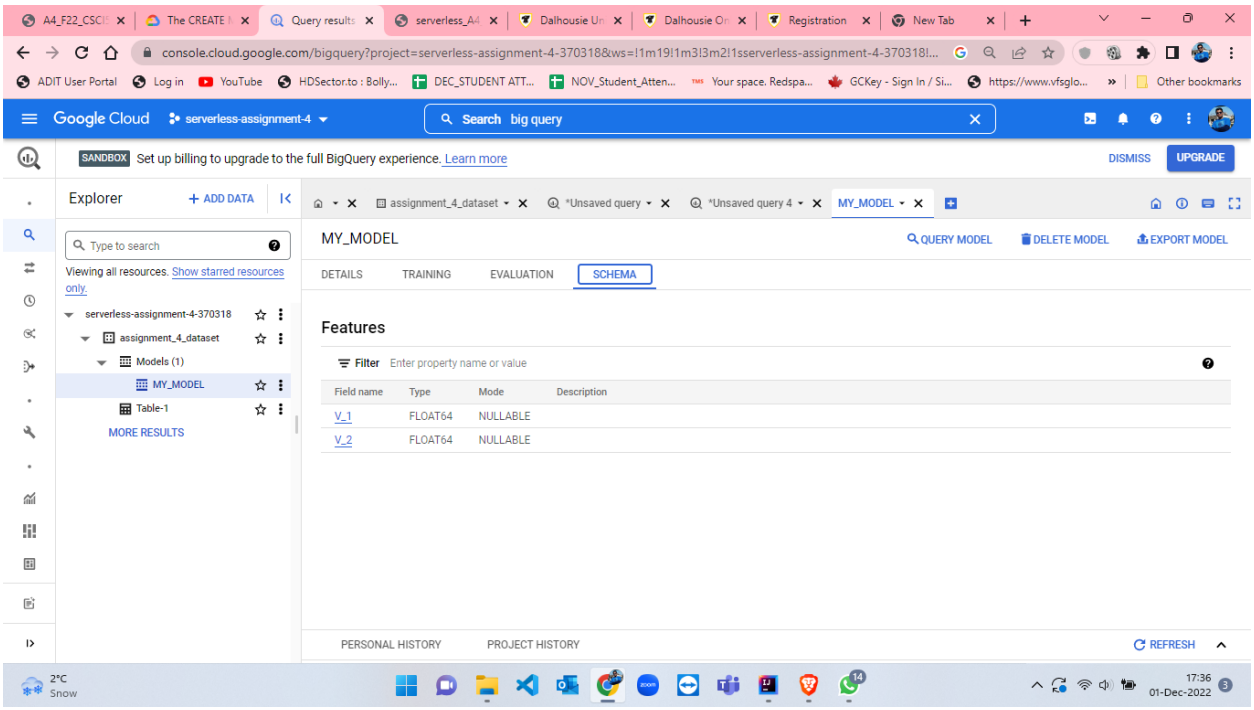

Figure 83: Schema details of MY_MODEL.

- 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`))
```

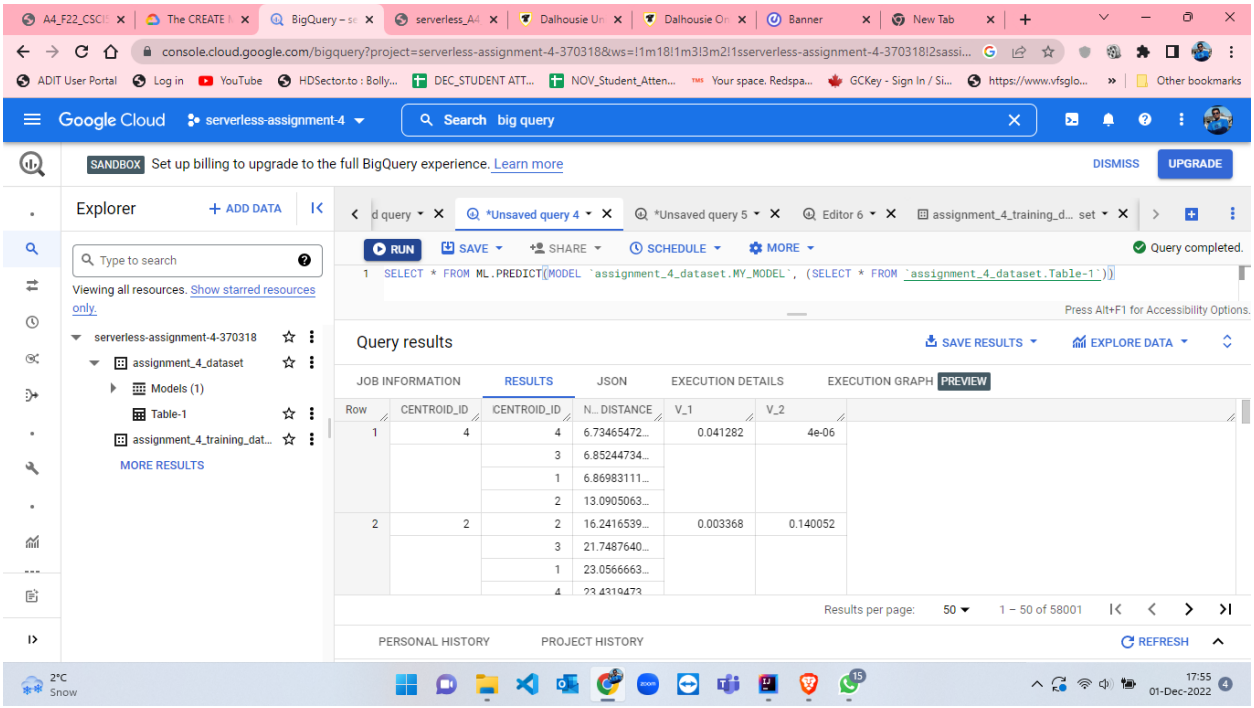

Figure 84: Results from ML.PREDICT query.

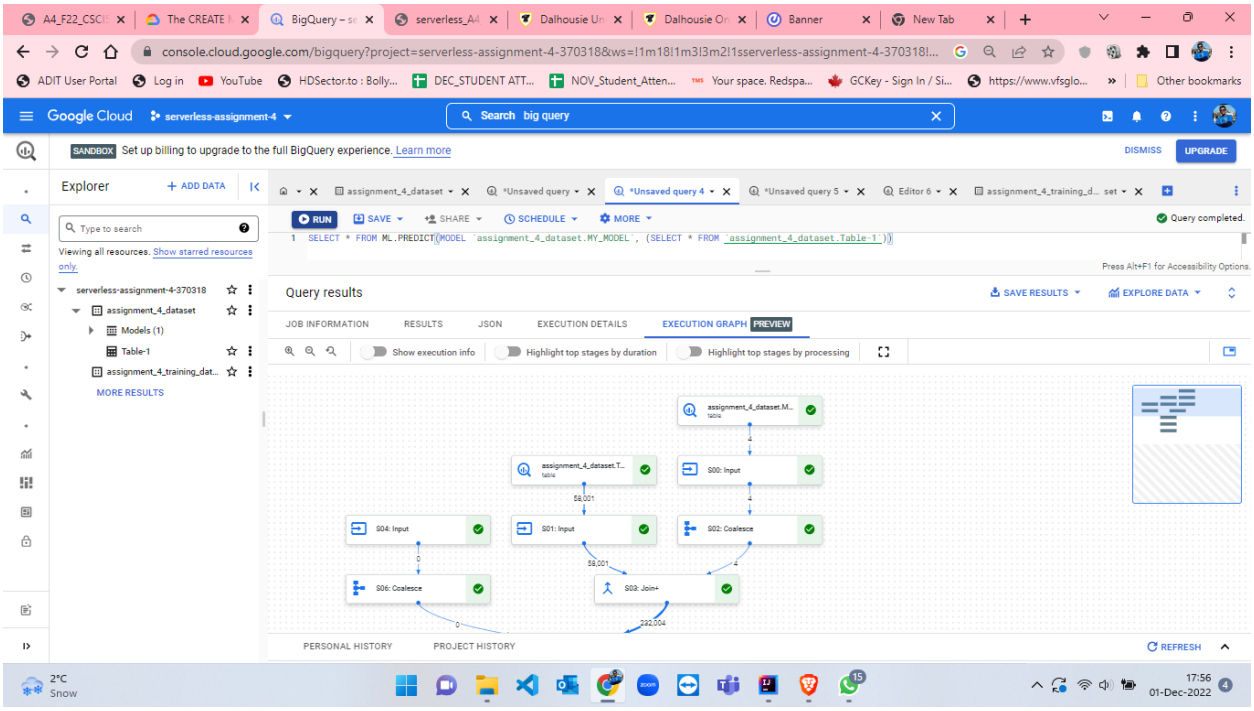

Figure 85: Execution Graph from ML.PREDICT query.

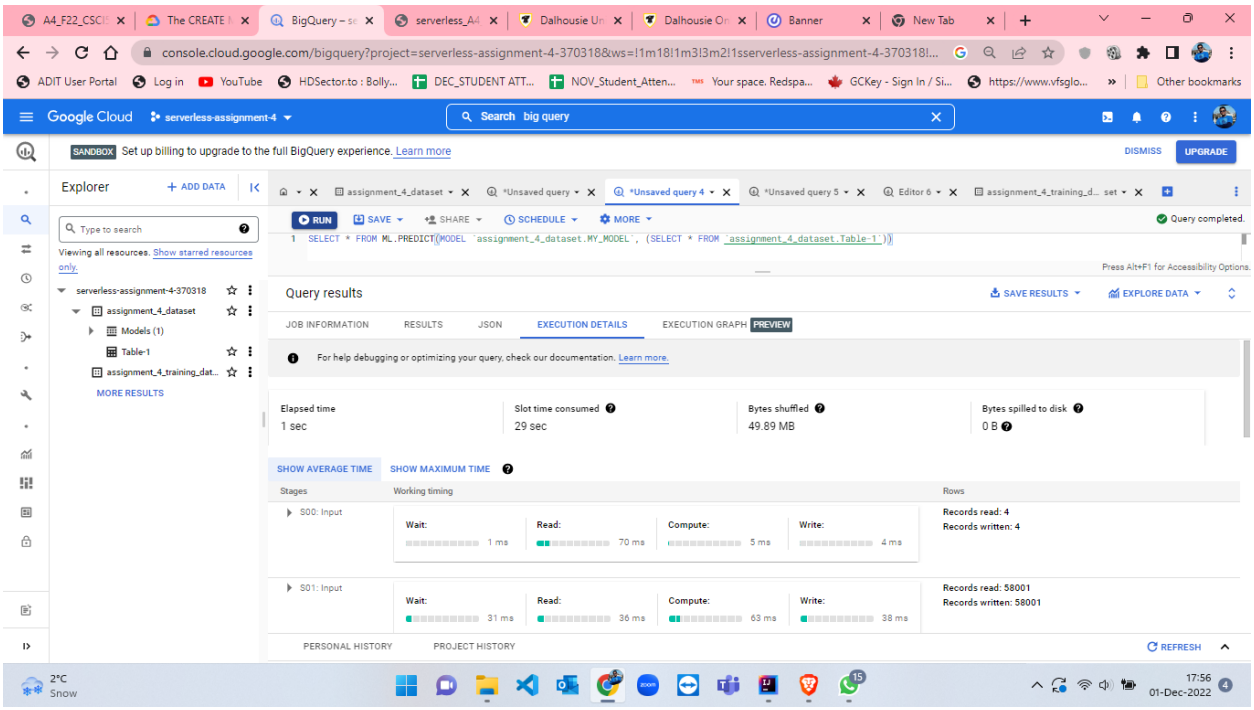

Figure 86: Execution Details from ML.PREDICT query part-1.

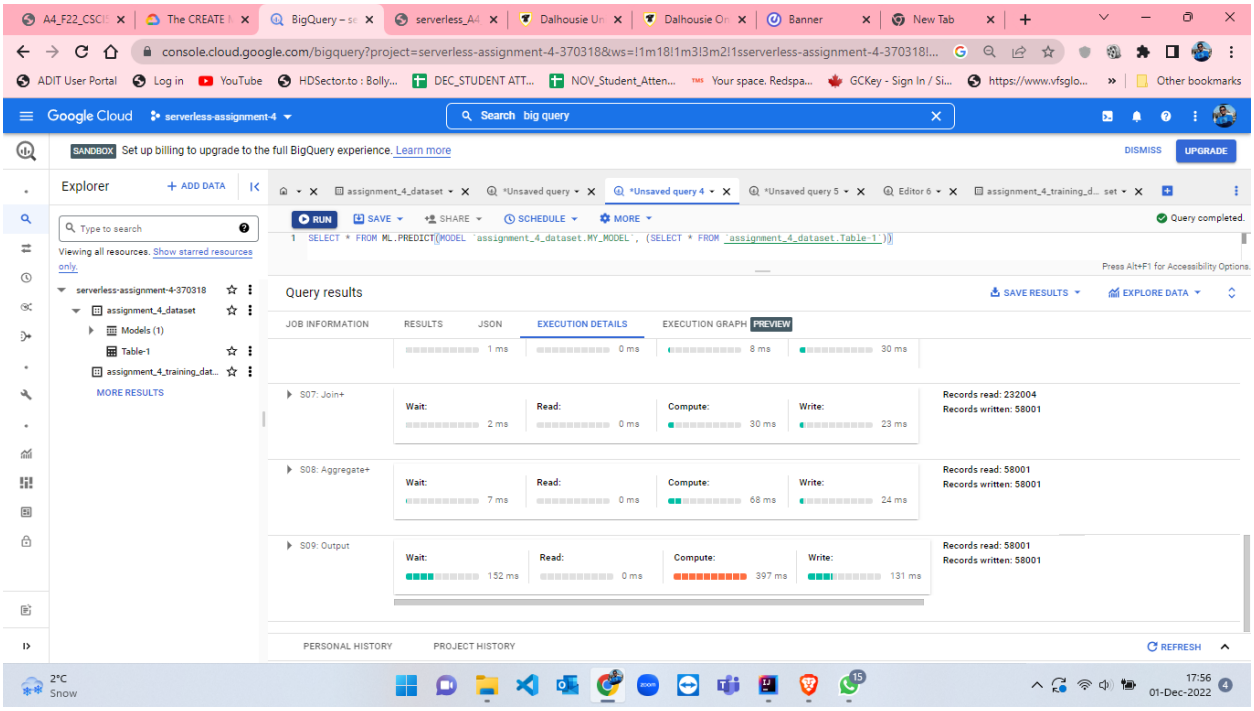

Figure 87: Execution Graph from ML.PREDICT query part-2.

- 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).

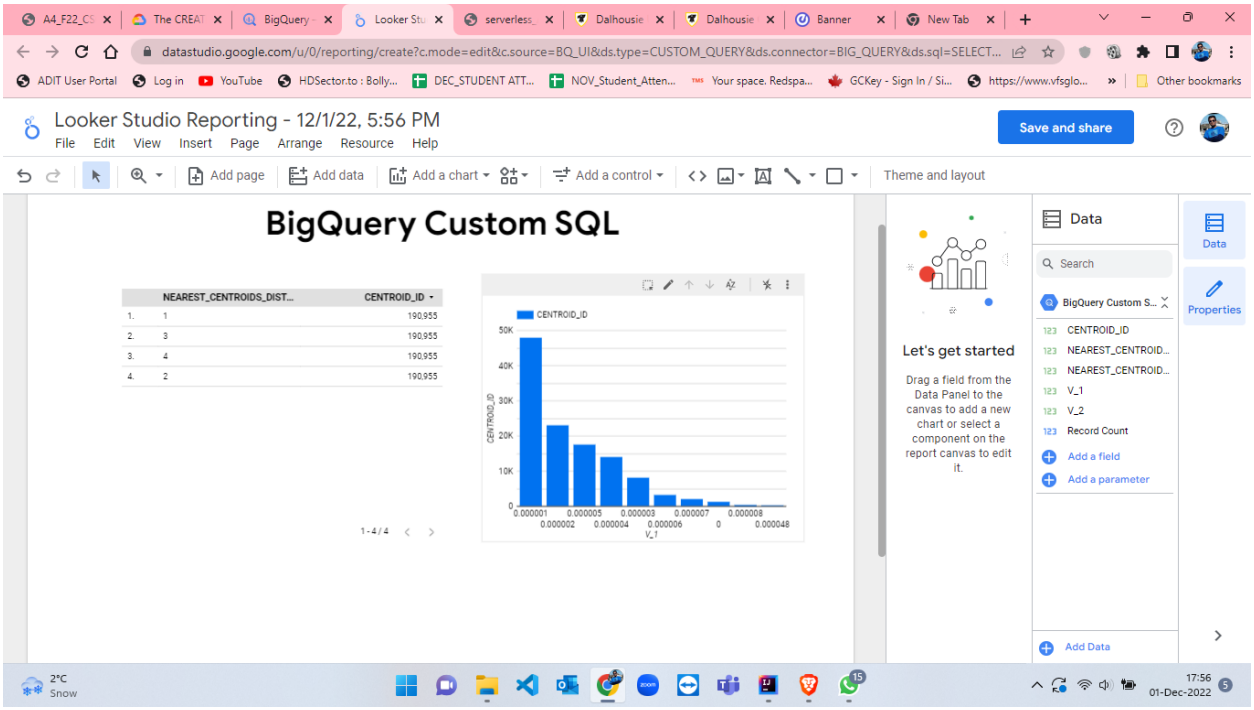

Figure 88: Looker Studio Reporting for visualization.

- 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.

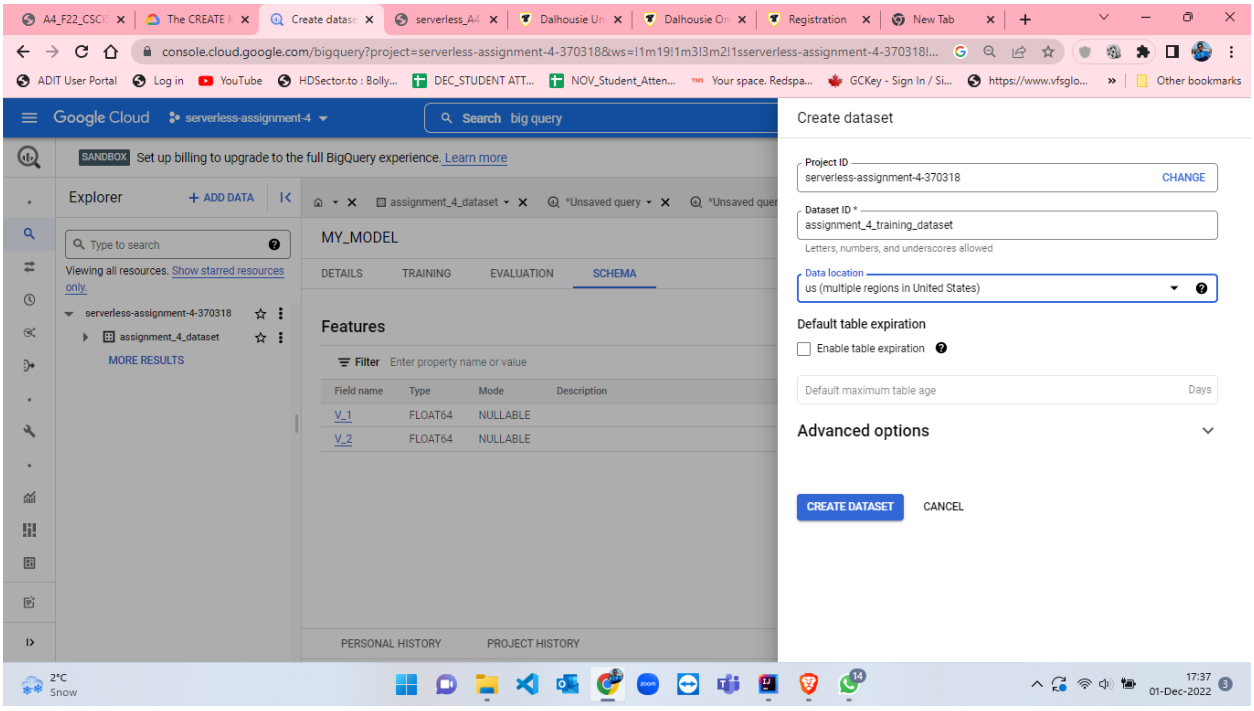

Figure 89: Create dataset named as training_dataset.

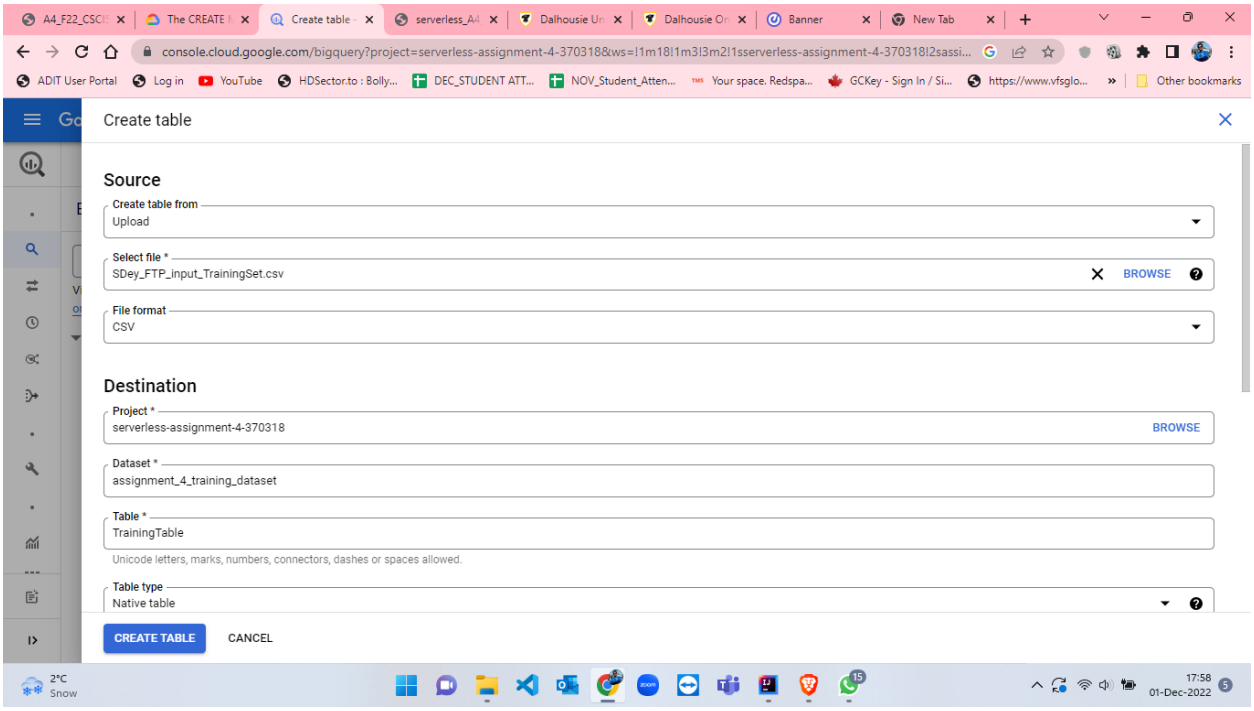

Figure 90: Create table for training named as TrainingTable having 75% of records.

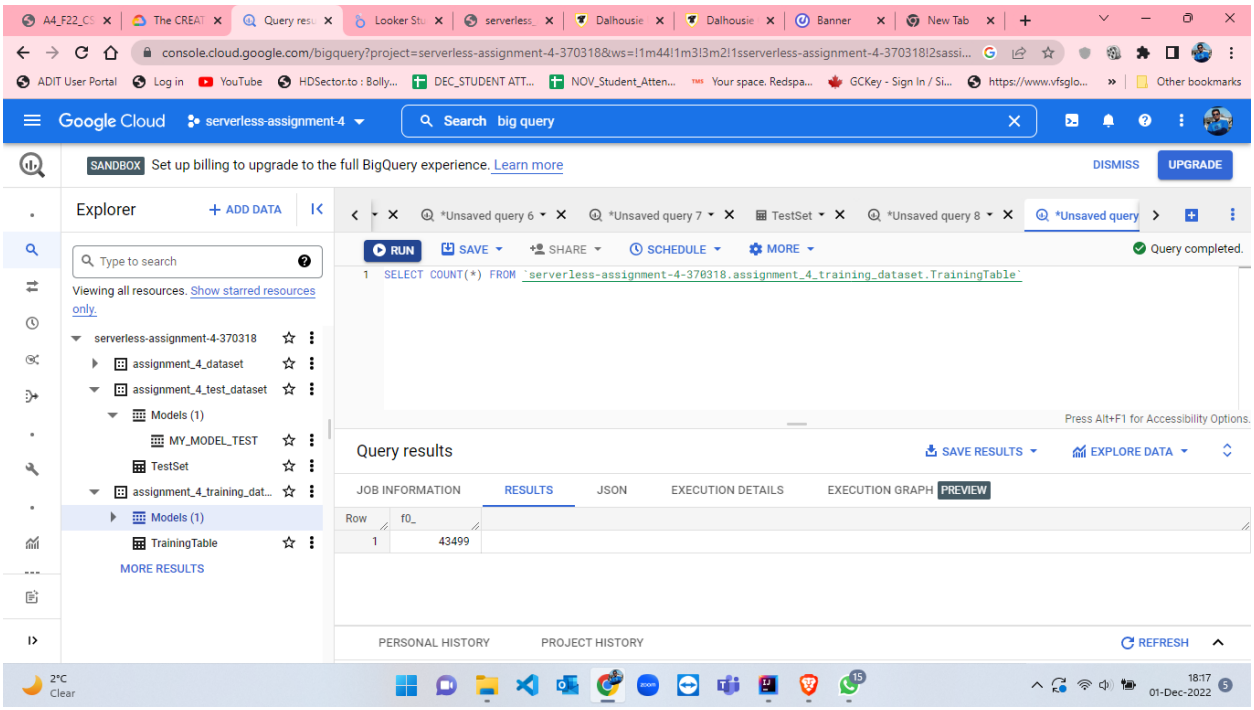

Figure 91: TrainingTable number of rows.

```
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.

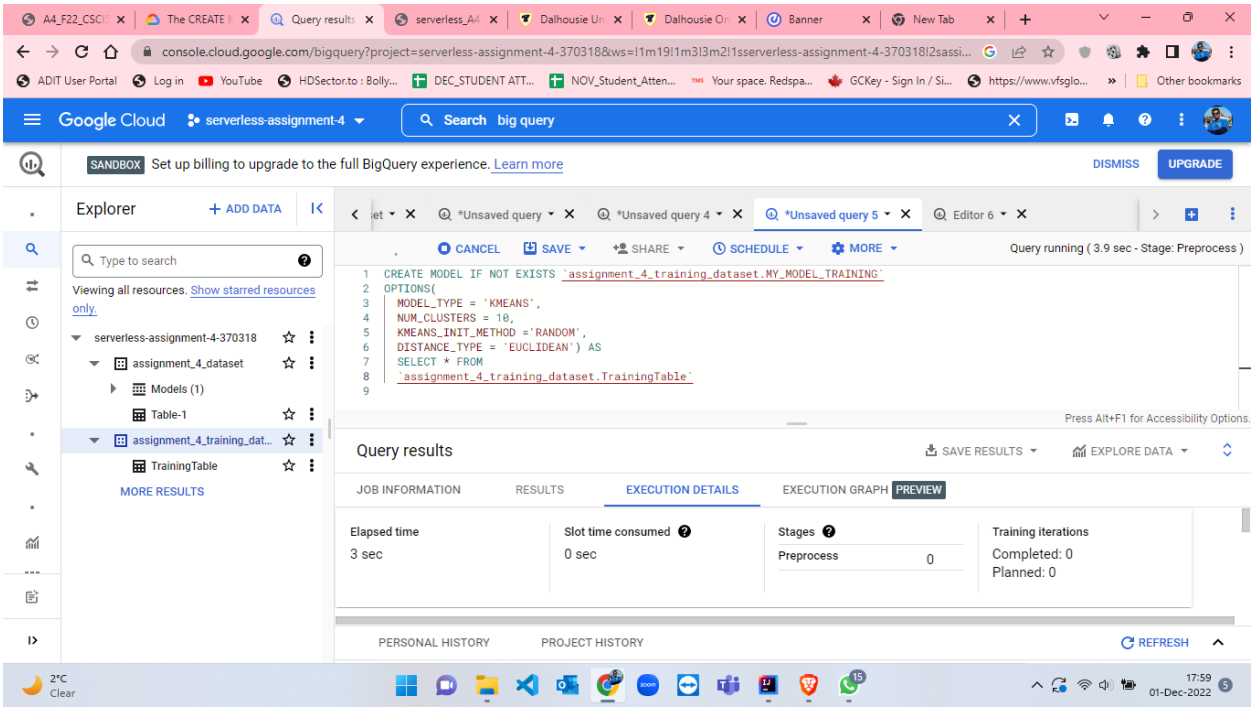

Figure 92: Query Execution details when KMeans model is ready for TrainingTable[10].

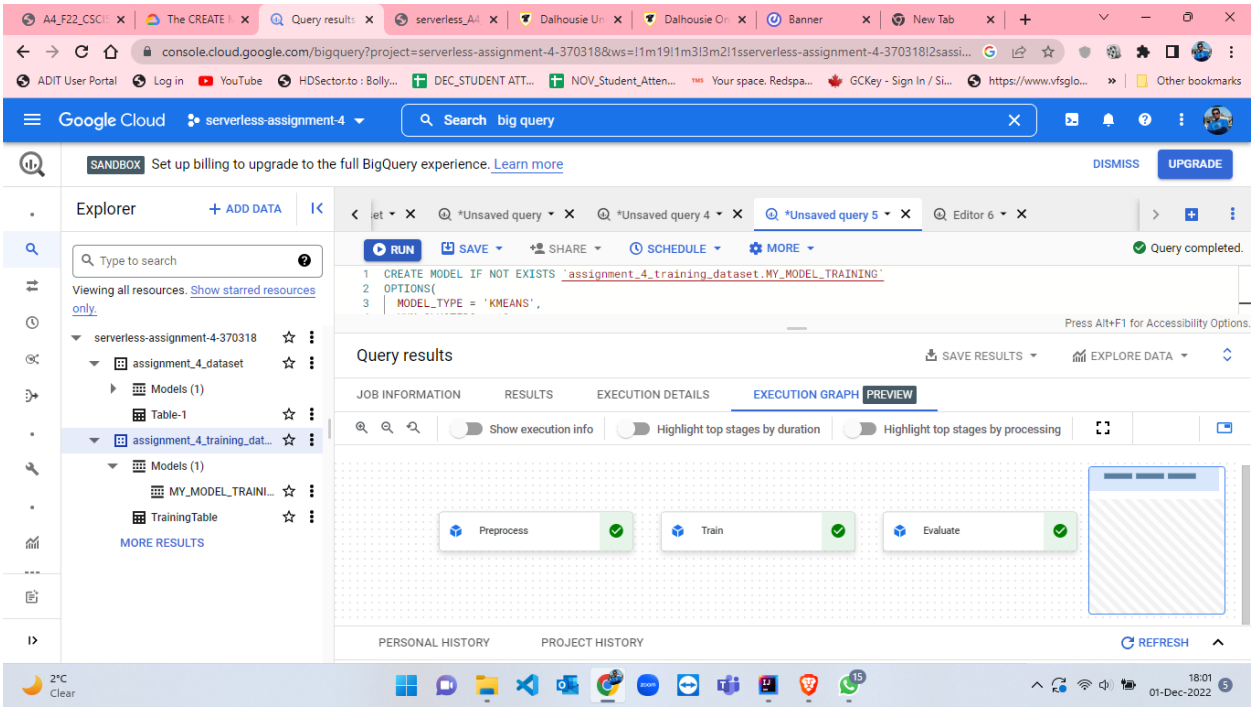

Figure 93: Execution Graph details when KMeans model is ready for TrainingTable.

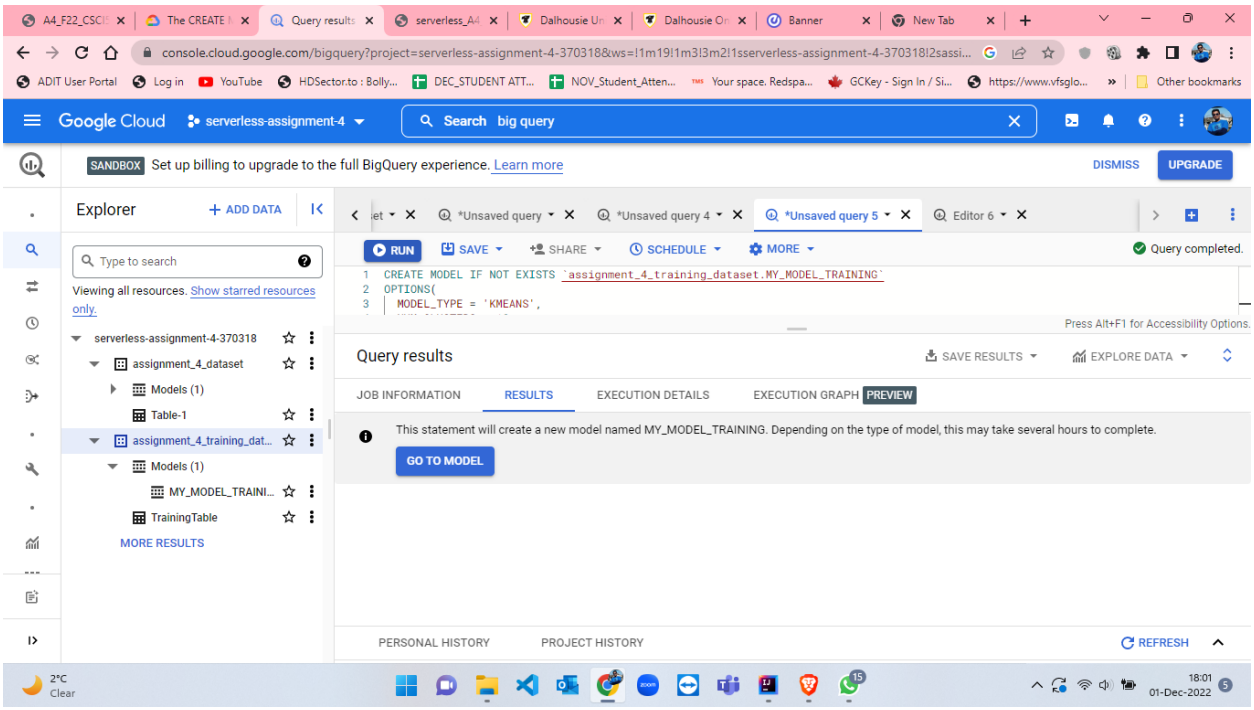

Figure 94: Query results when KMeans model is ready for TrainingTable.

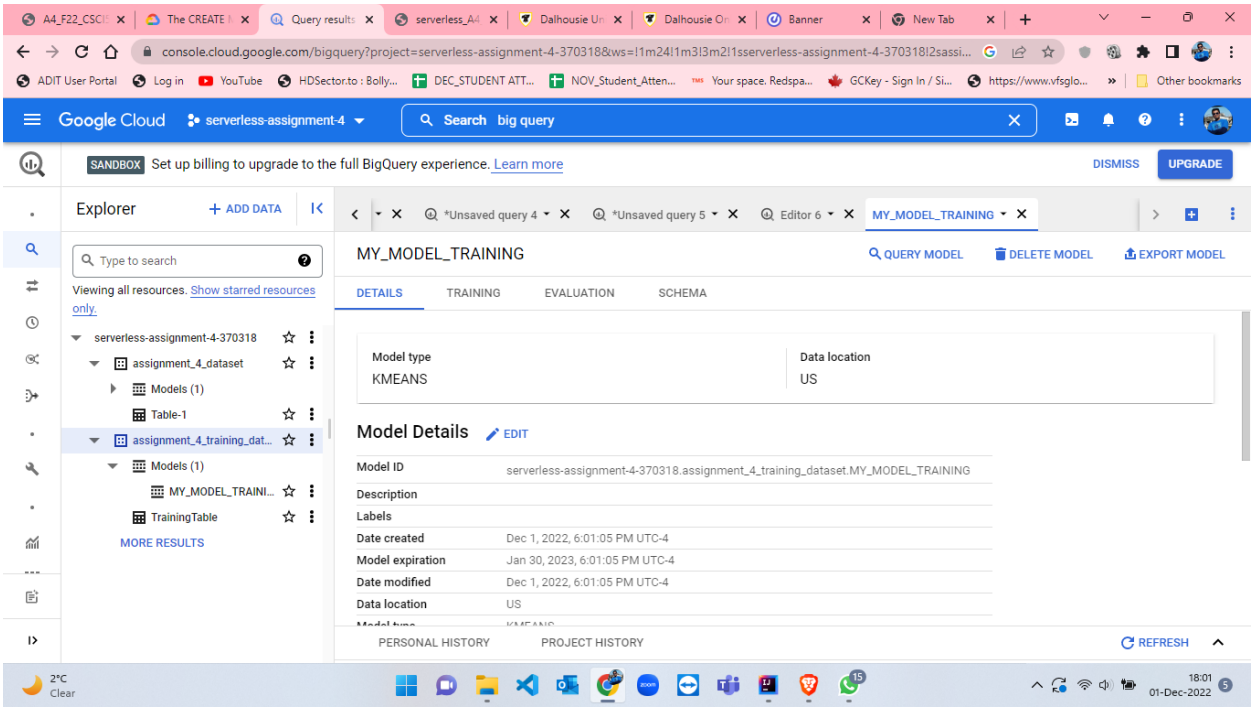

Figure 95: Model details of MY_MODEL_TRAINING part-1.

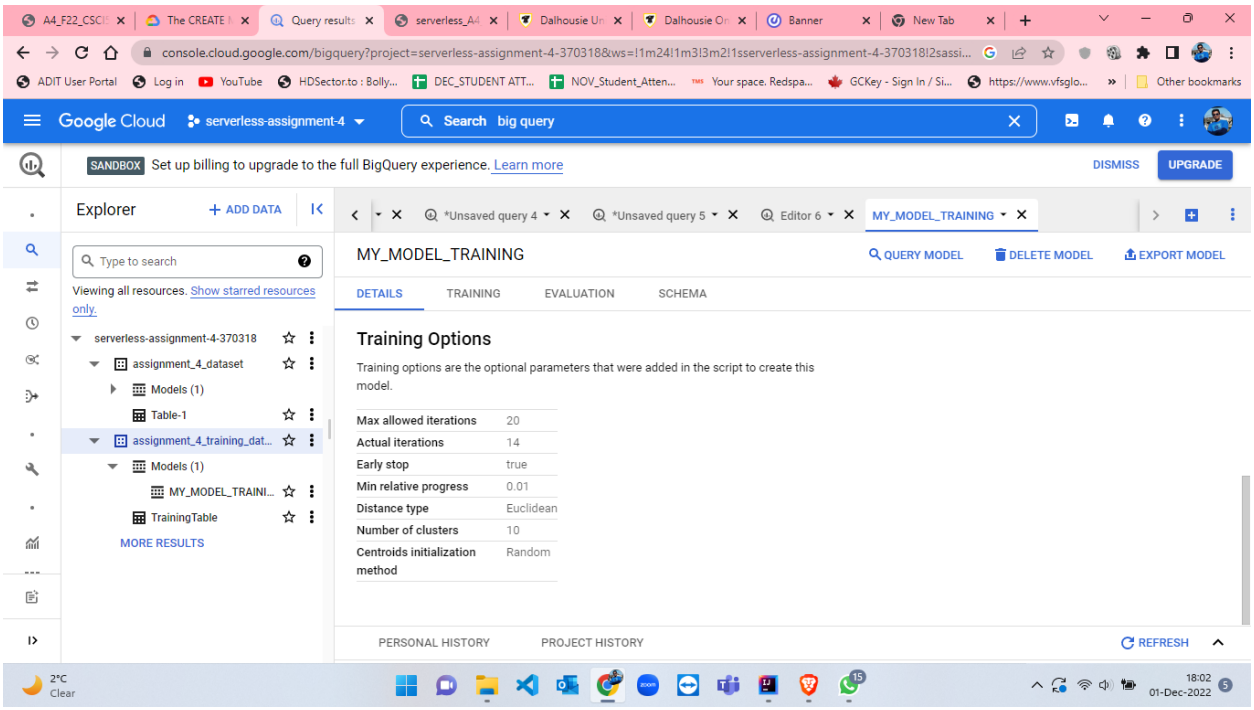

Figure 96: Model details of MY_MODEL_TRAINING part-2.

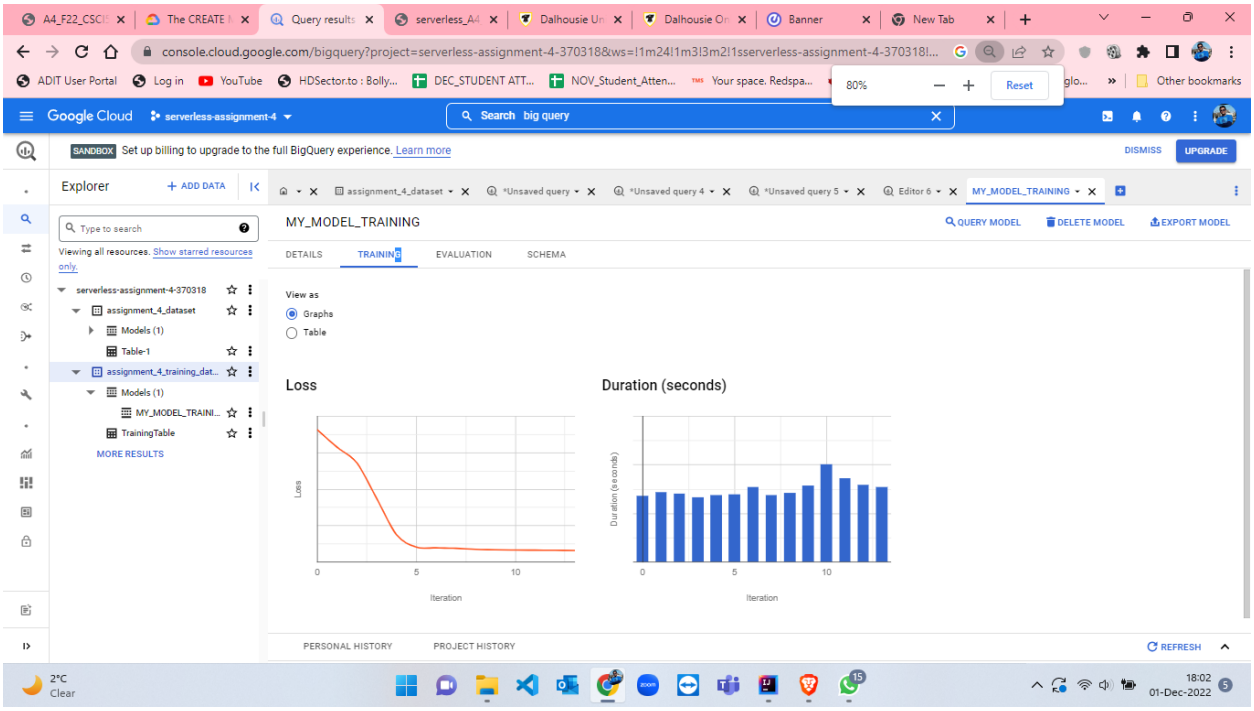

Figure 97: Model details of MY_MODEL_TRAINING part-3.

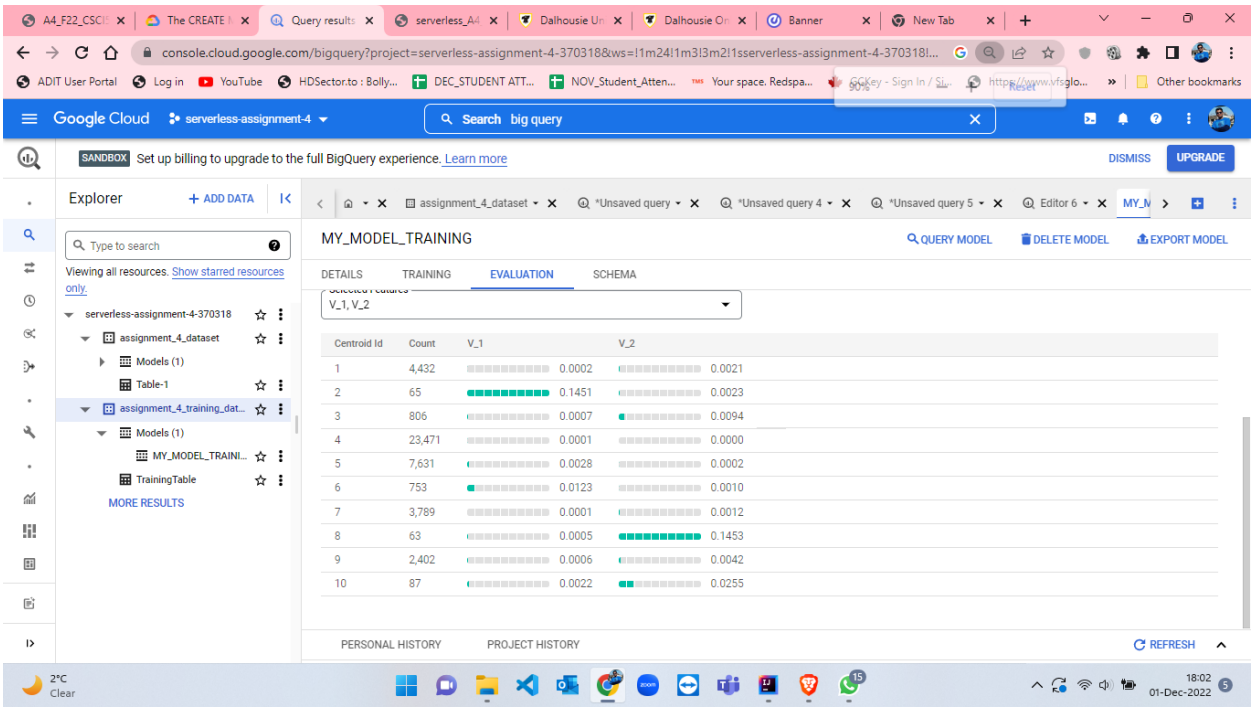

Figure 98: Model details of MY_MODEL_TRAINING part-4.

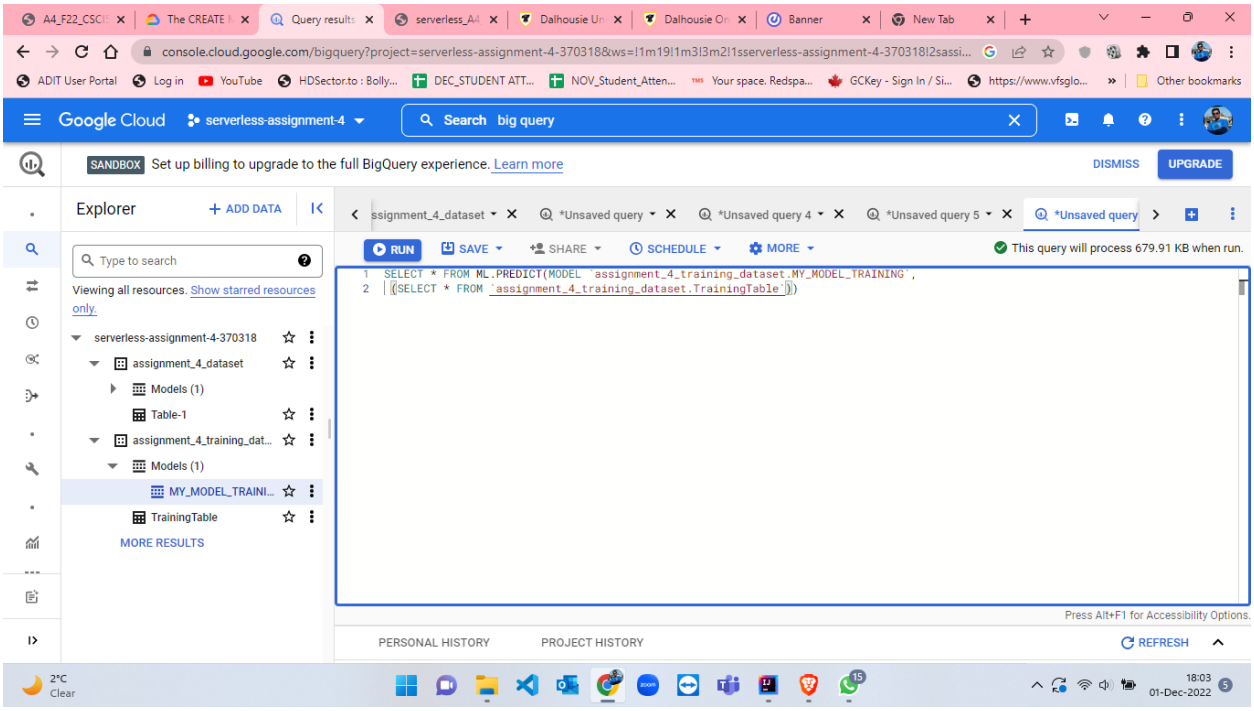

Figure 99:ML.PREDICT query.

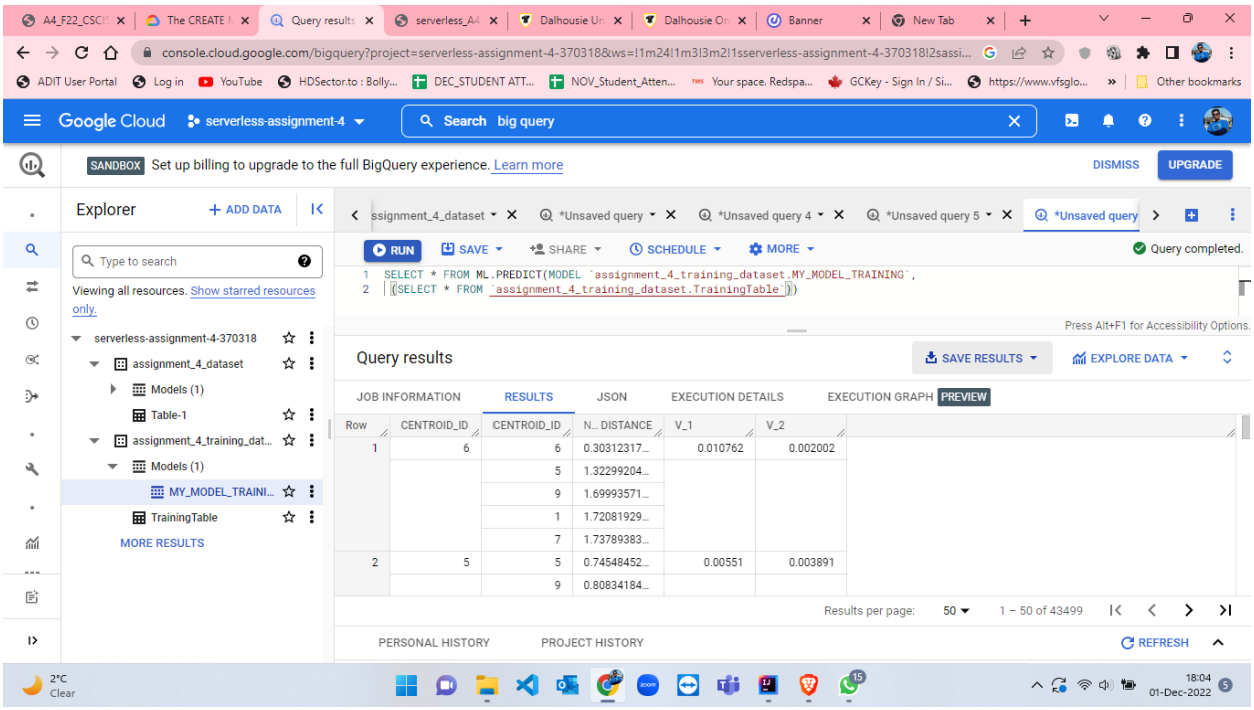

Figure 100: Results from ML.PREDICT query.

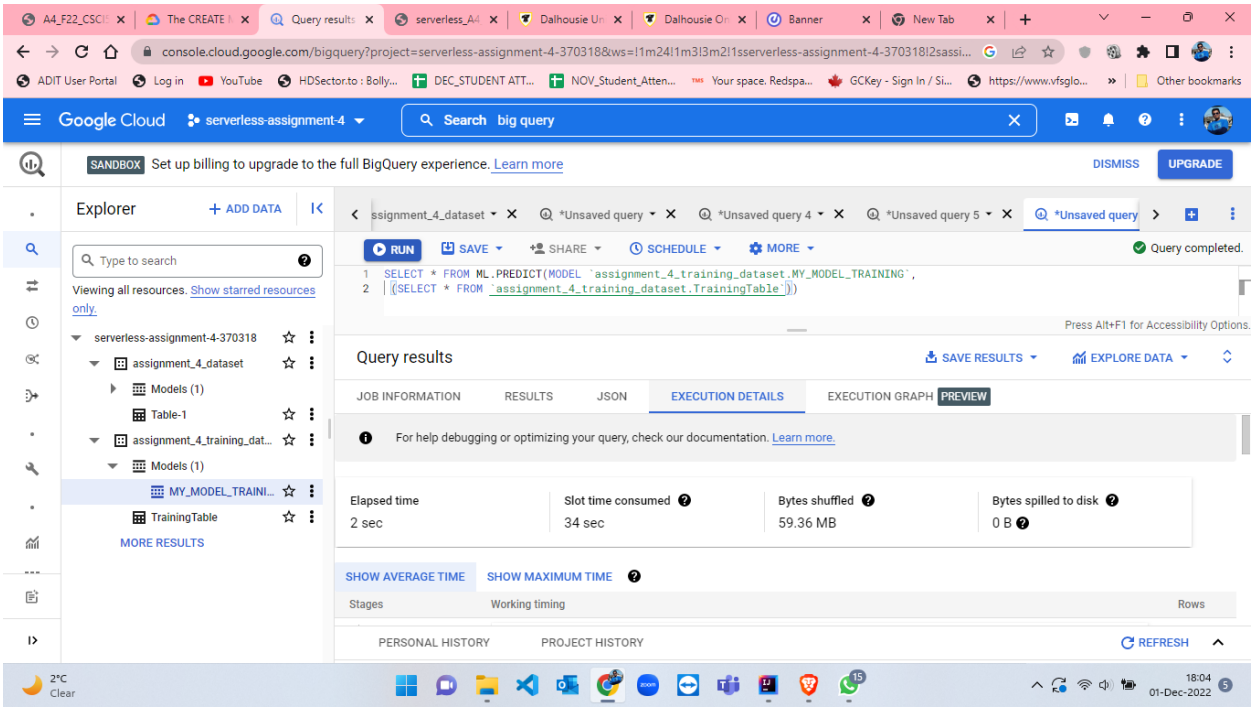

Figure 101: Execution Details from ML.PREDICT query part-1.

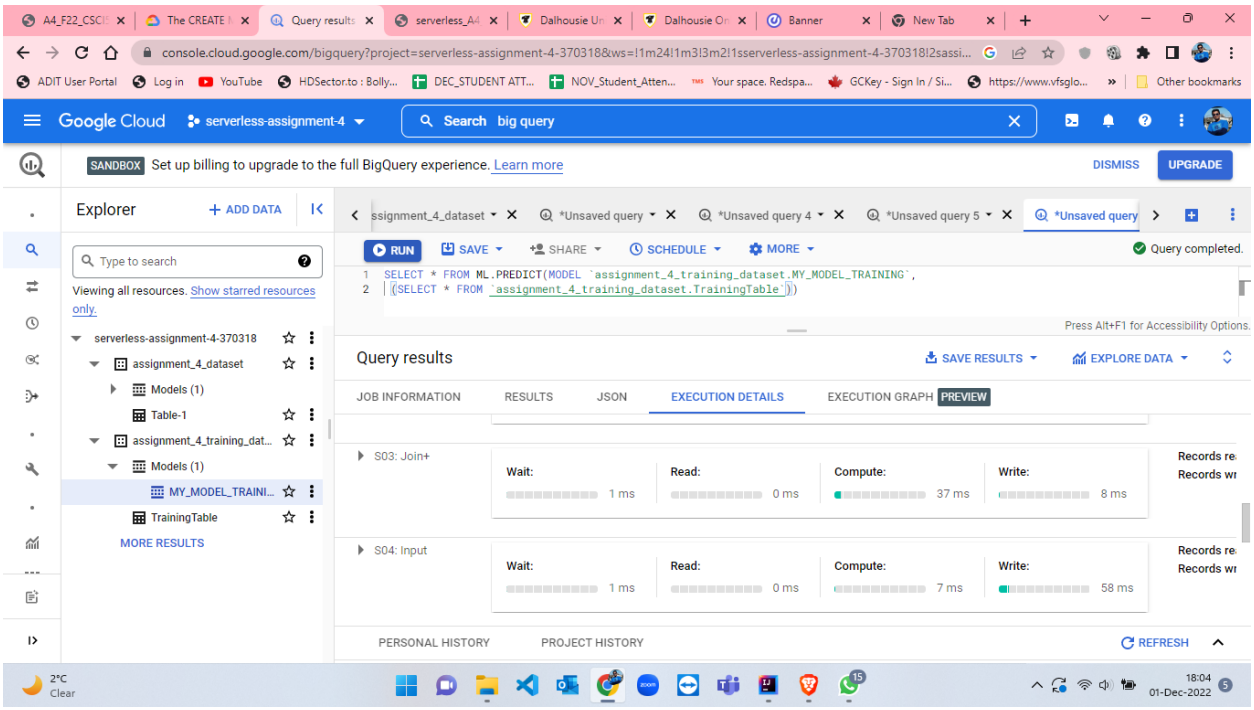

Figure 102: Execution Details from ML.PREDICT query part-2.

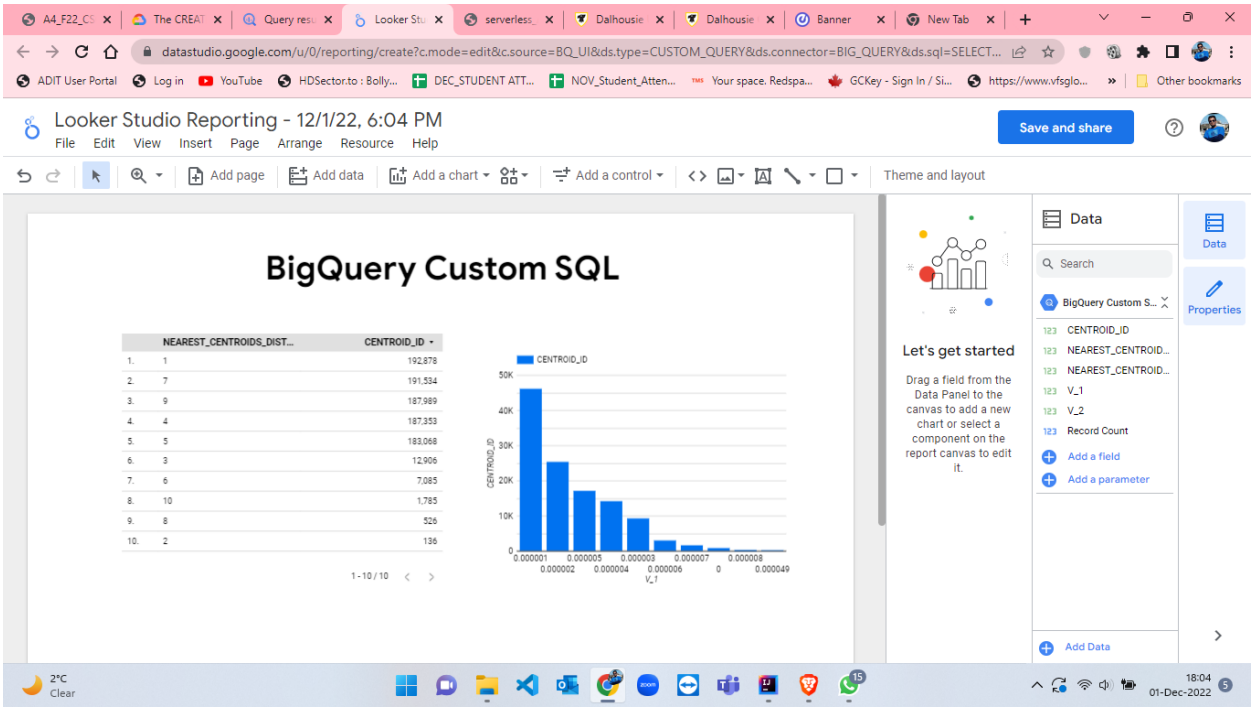

Figure 103: Looker Studio Reporting for visualization.

- 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.

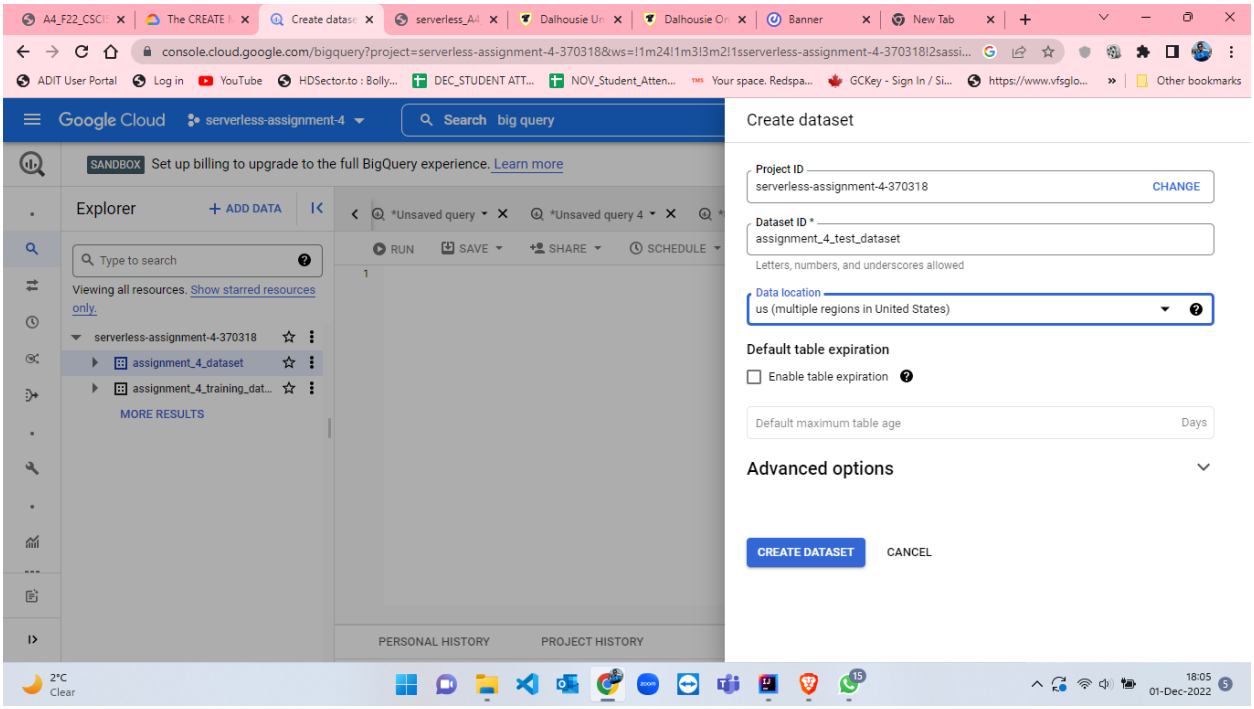

Figure 104: Create dataset named as test_dataset.

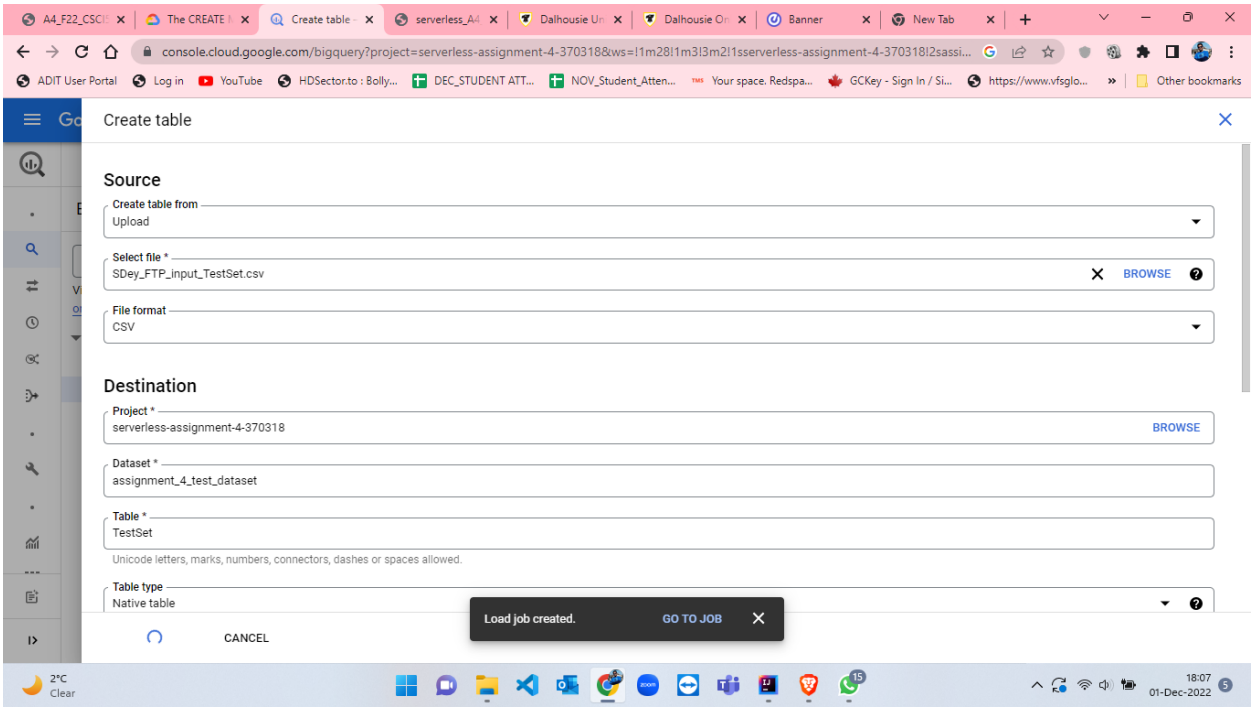

Figure 105: Create table for test named as TestSet having 25% of records.

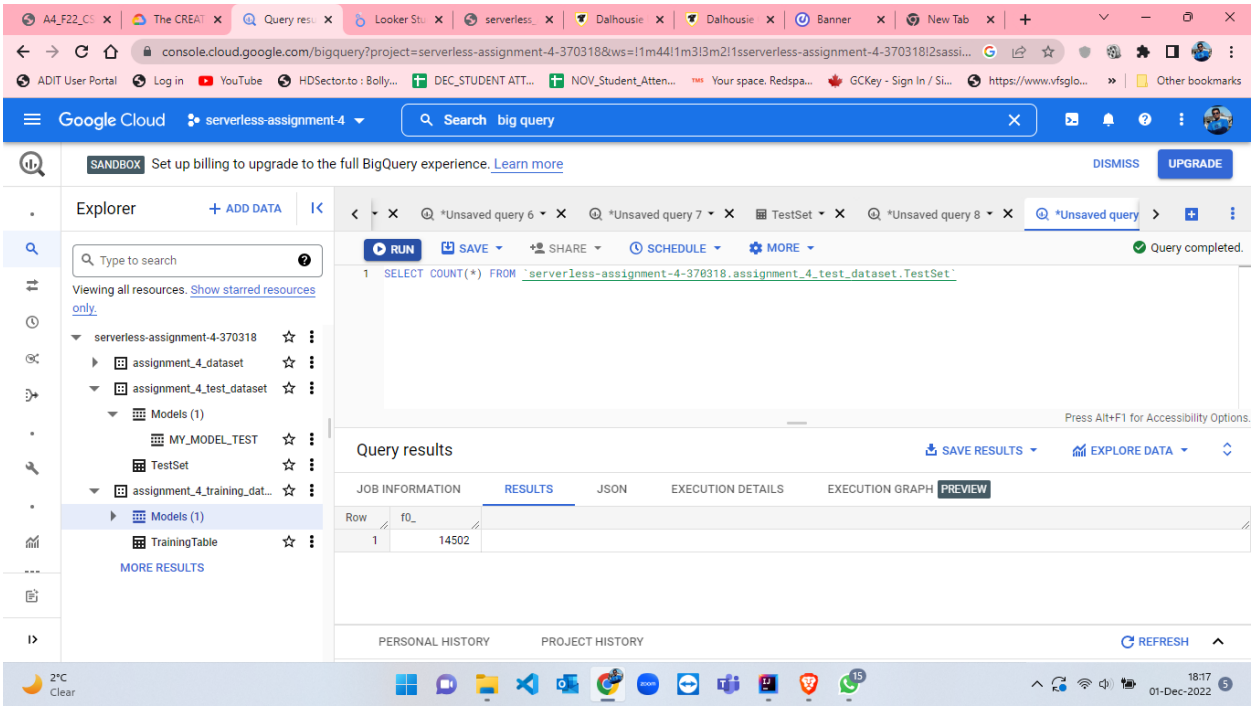

Figure 106: TestSet table number of rows.

```
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.

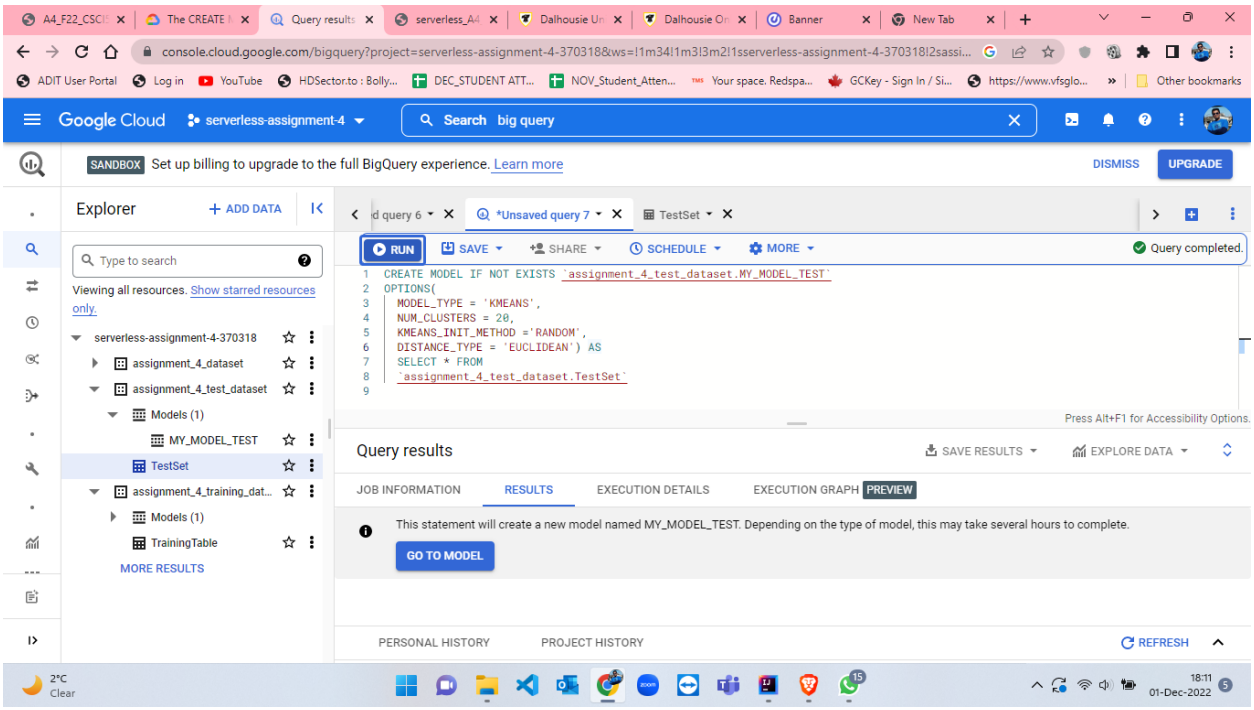

Figure 107: Query results when KMeans model is ready for TestSet table[10].

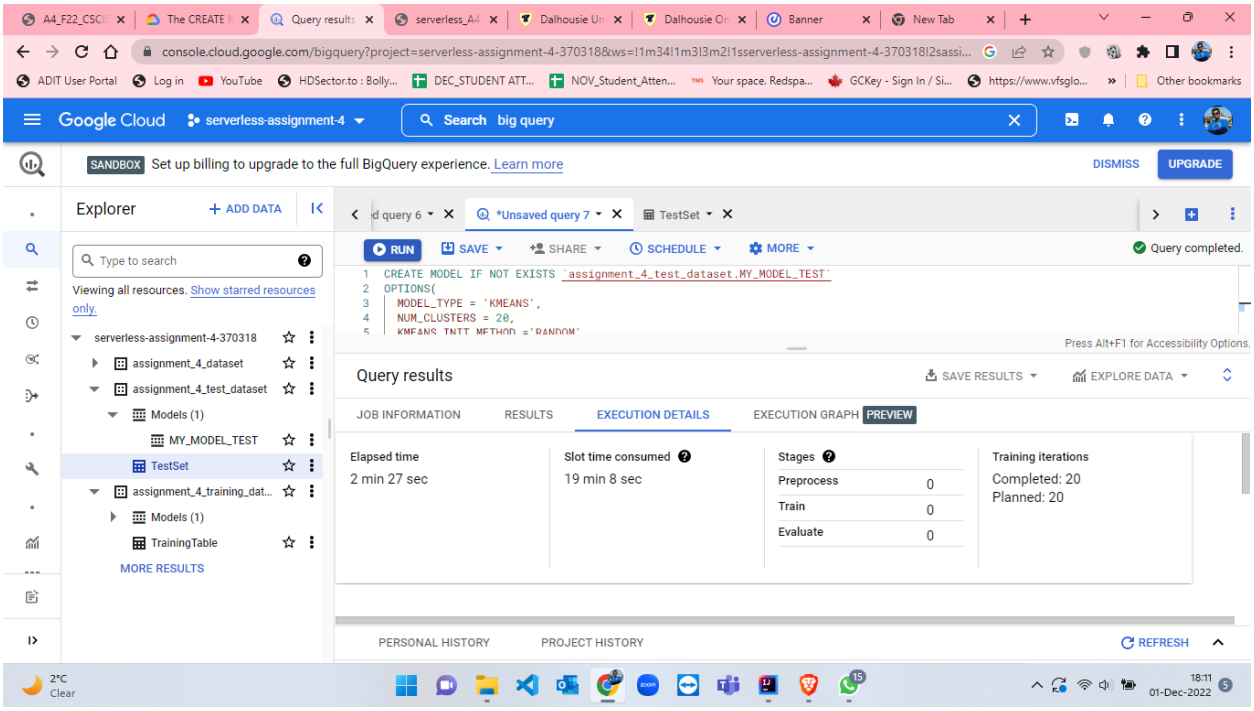

Figure 108: Execution details when KMeans model is ready for TestSet table.

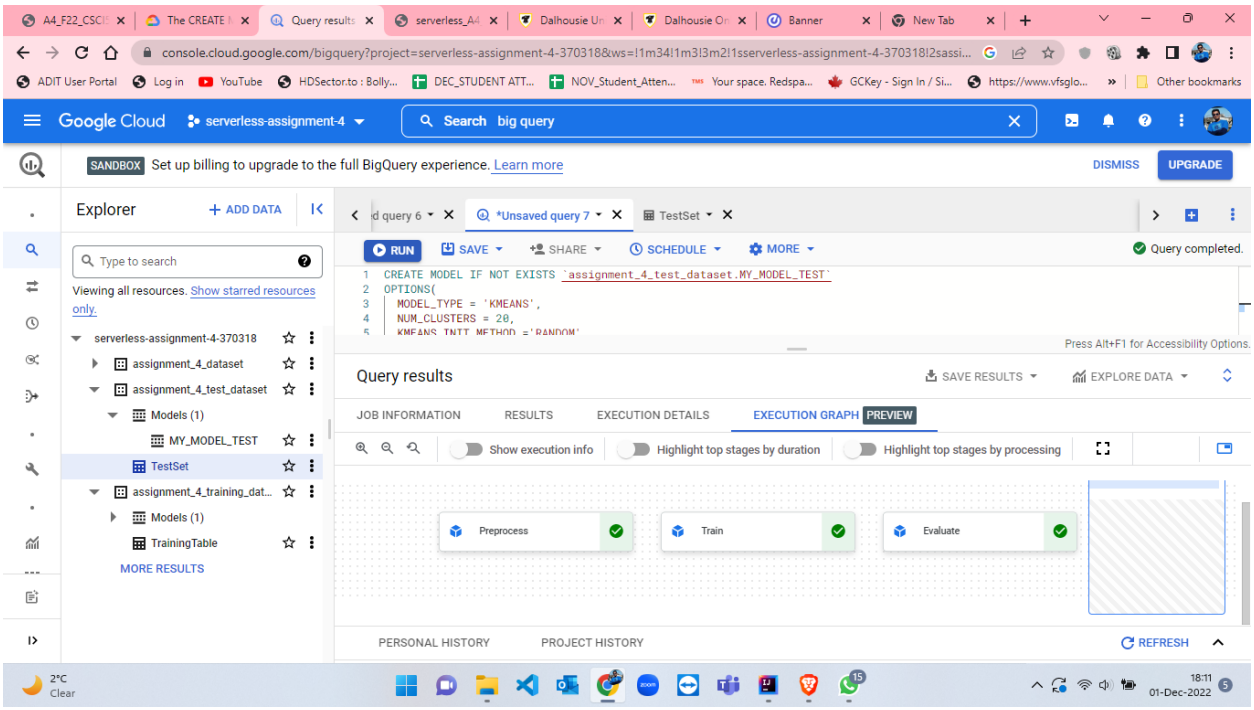

Figure 109: Execution Graph when KMeans model is ready for TestSet table.

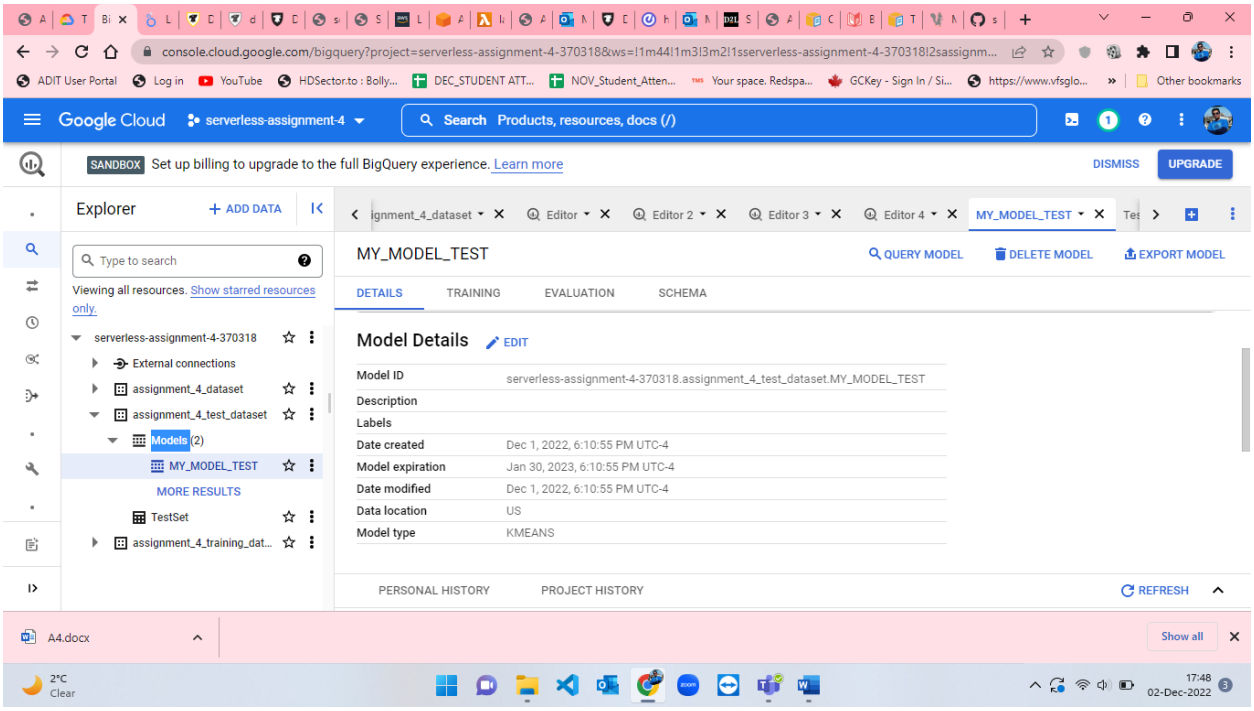

Figure 110: Model details of MY_MODEL_TEST part-1.

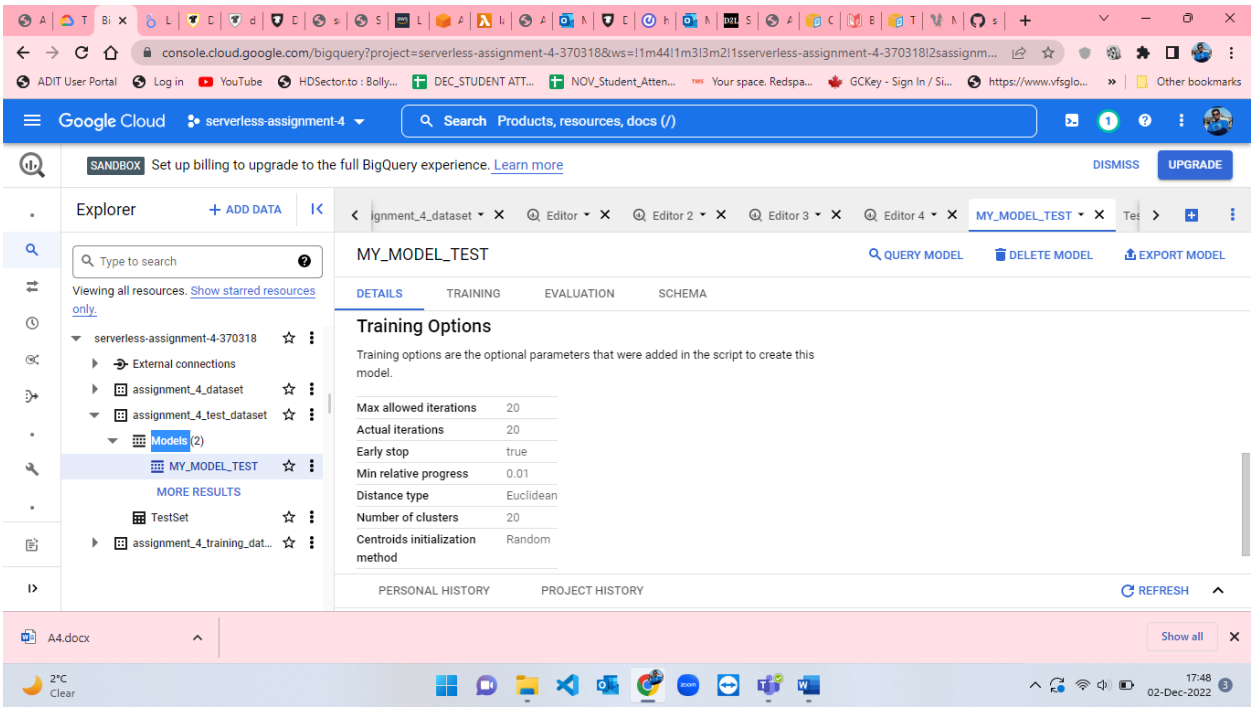

Figure 111: Model details of MY_MODEL_TEST part-2.

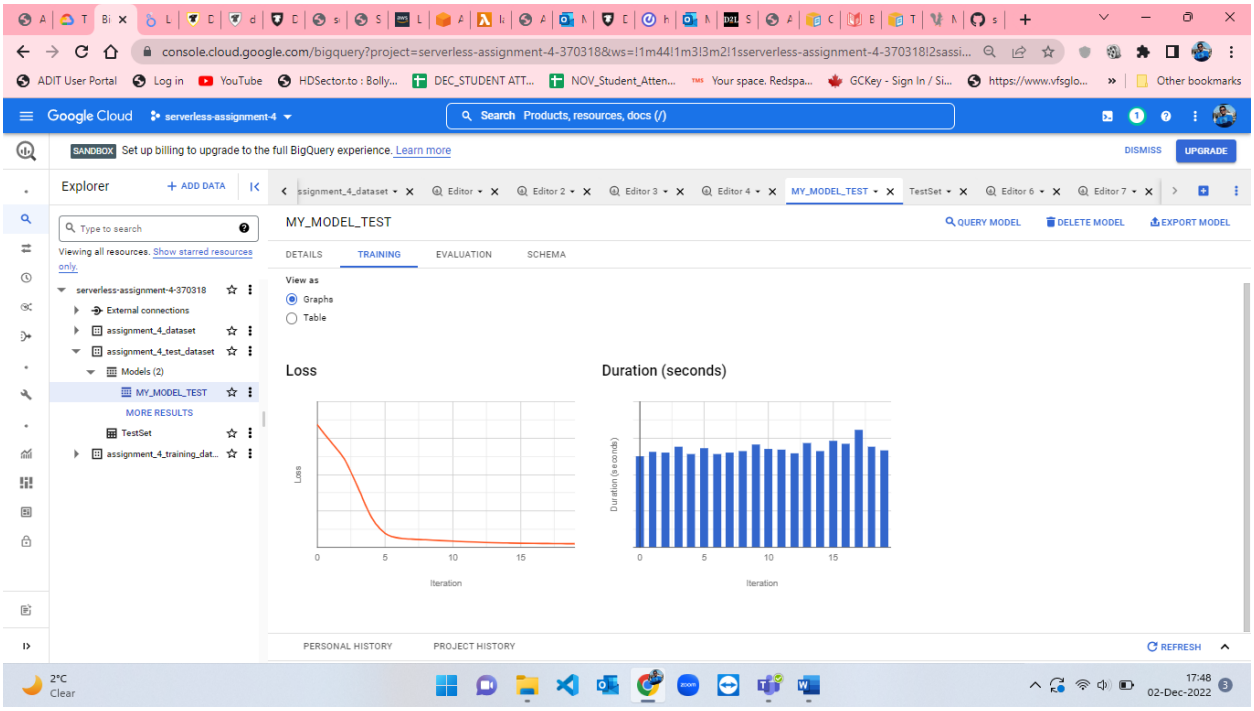

Figure 112: Model details of MY_MODEL_TEST part-3.

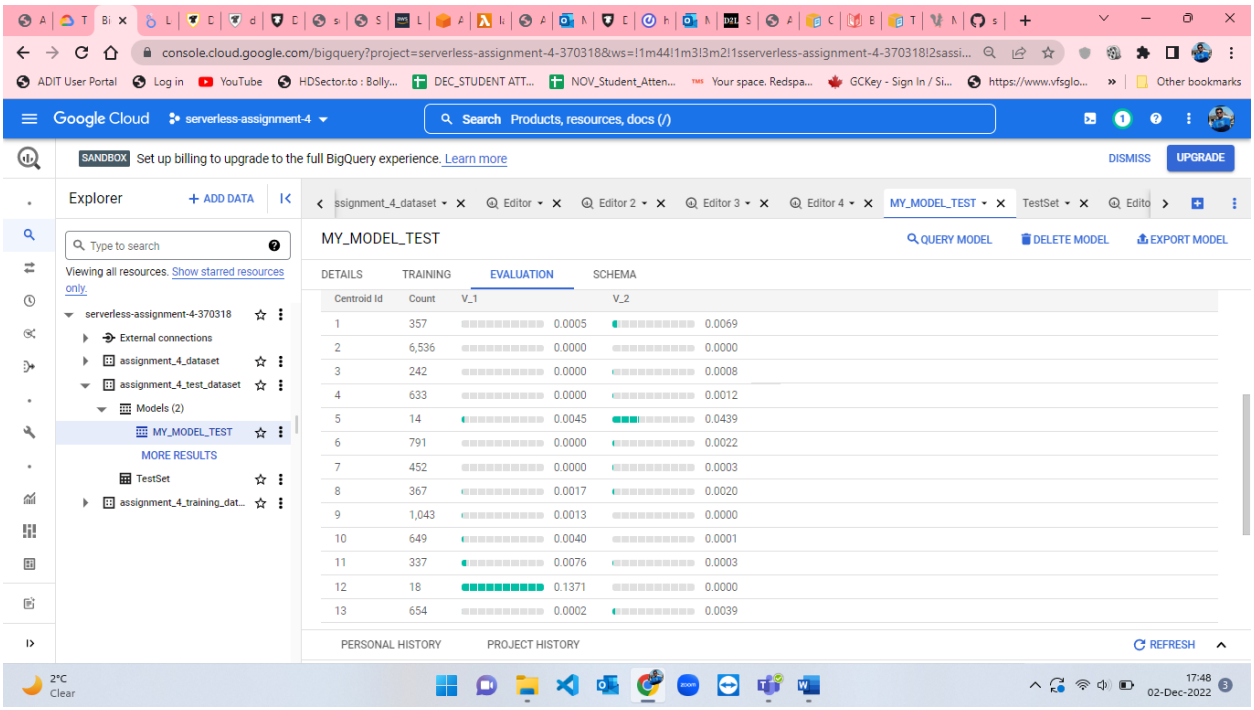

Figure 113: Model details of MY_MODEL_TEST part-4.

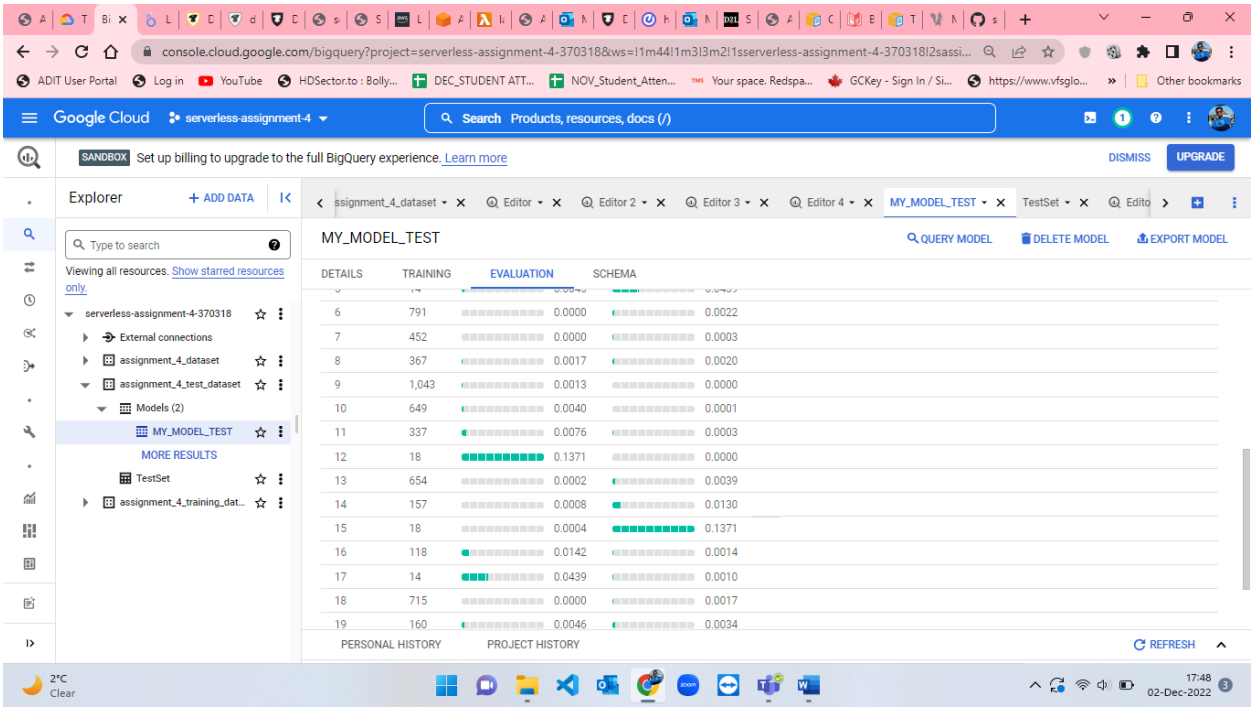

Figure 114: Model details of MY_MODEL_TEST part-5.

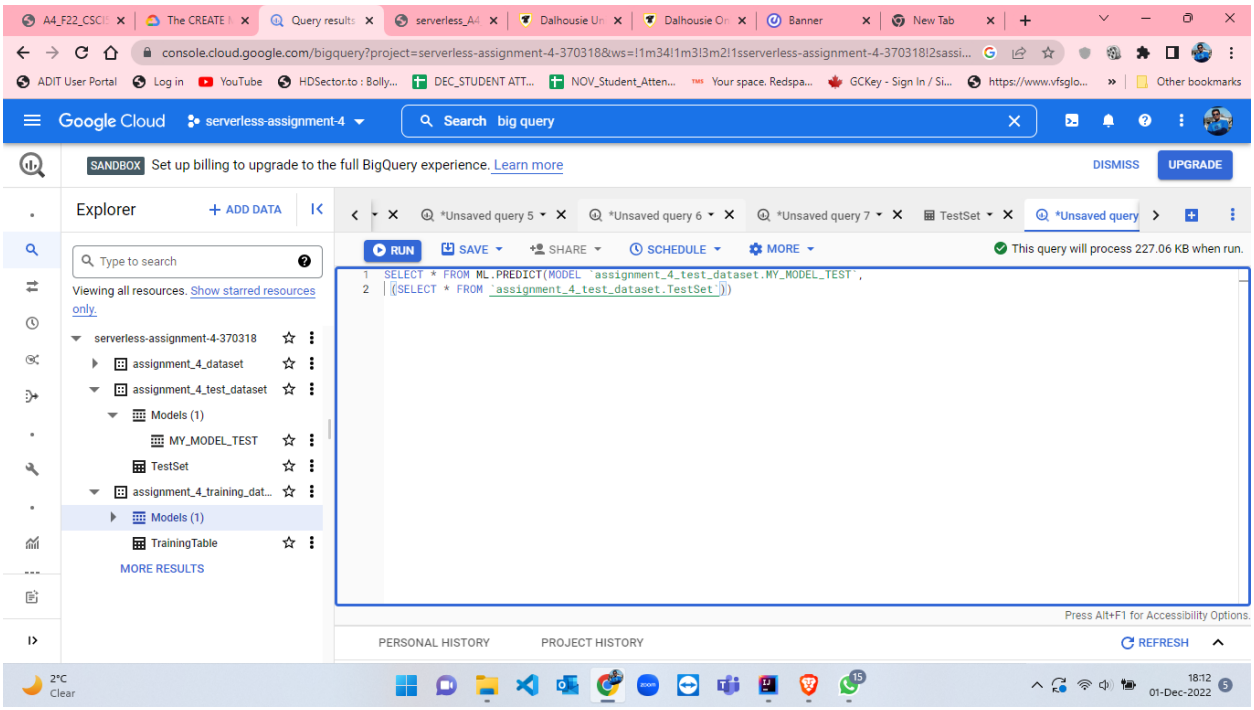

Figure 115:ML.PREDICT query for TestSet Table.

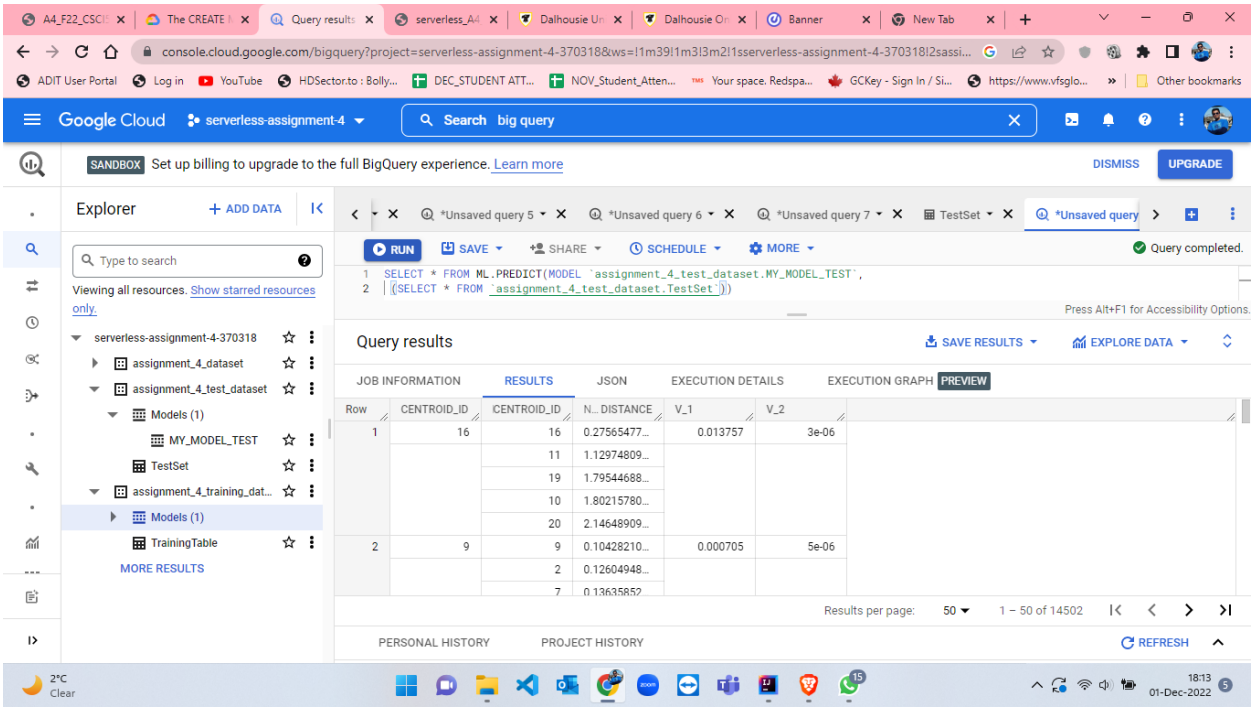

Figure 116: Results from ML.PREDICT query for TestSet table.

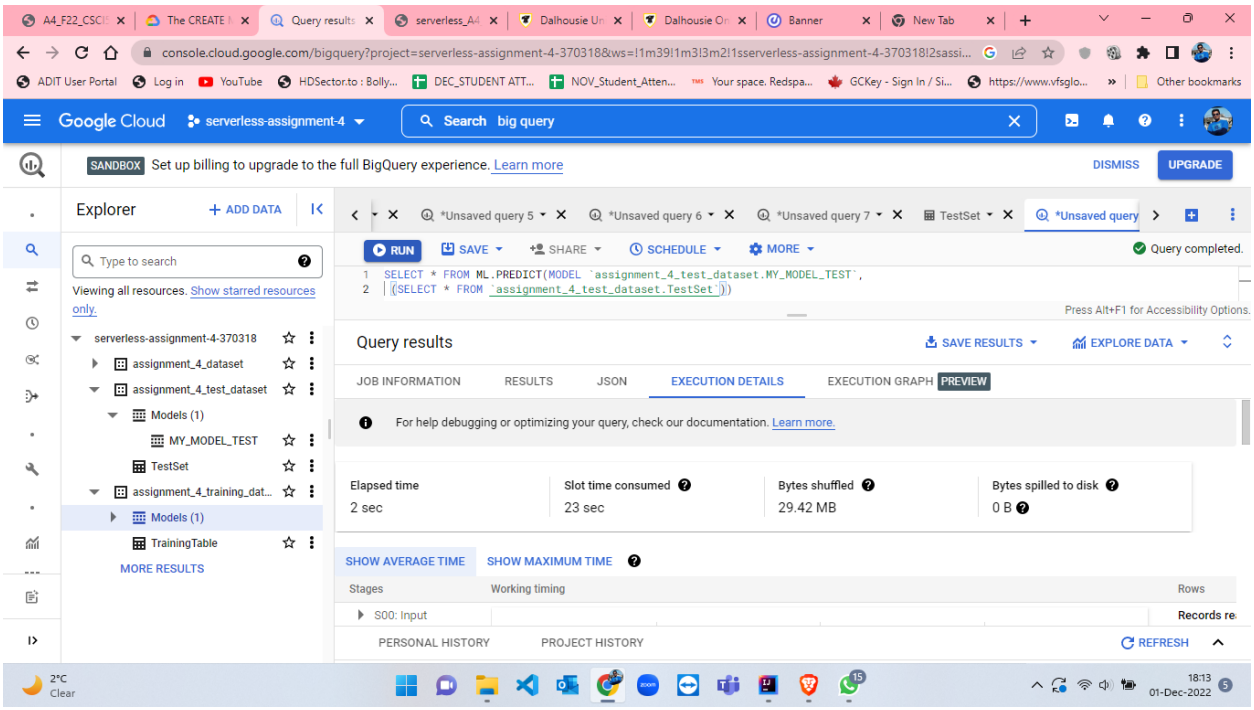

Figure 117: Execution details from ML.PREDICT query for TestSet table.

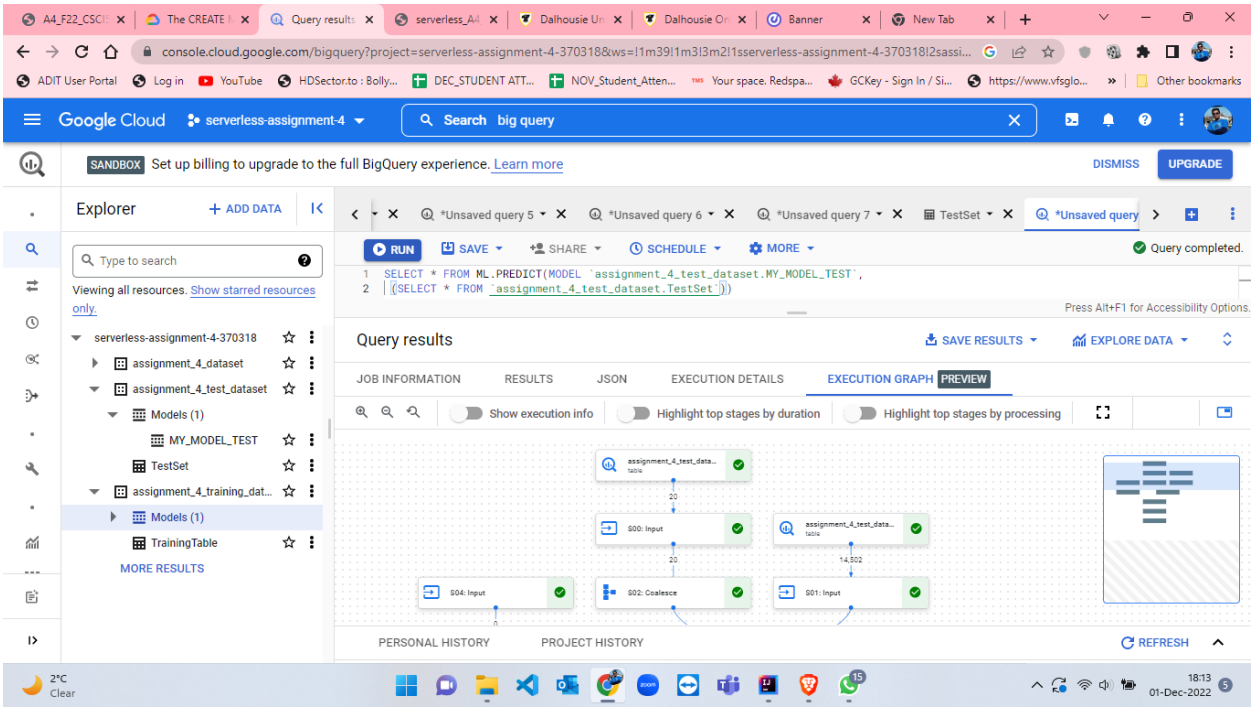

Figure 118: Execution Graph from ML.PREDICT query for TestSet table.

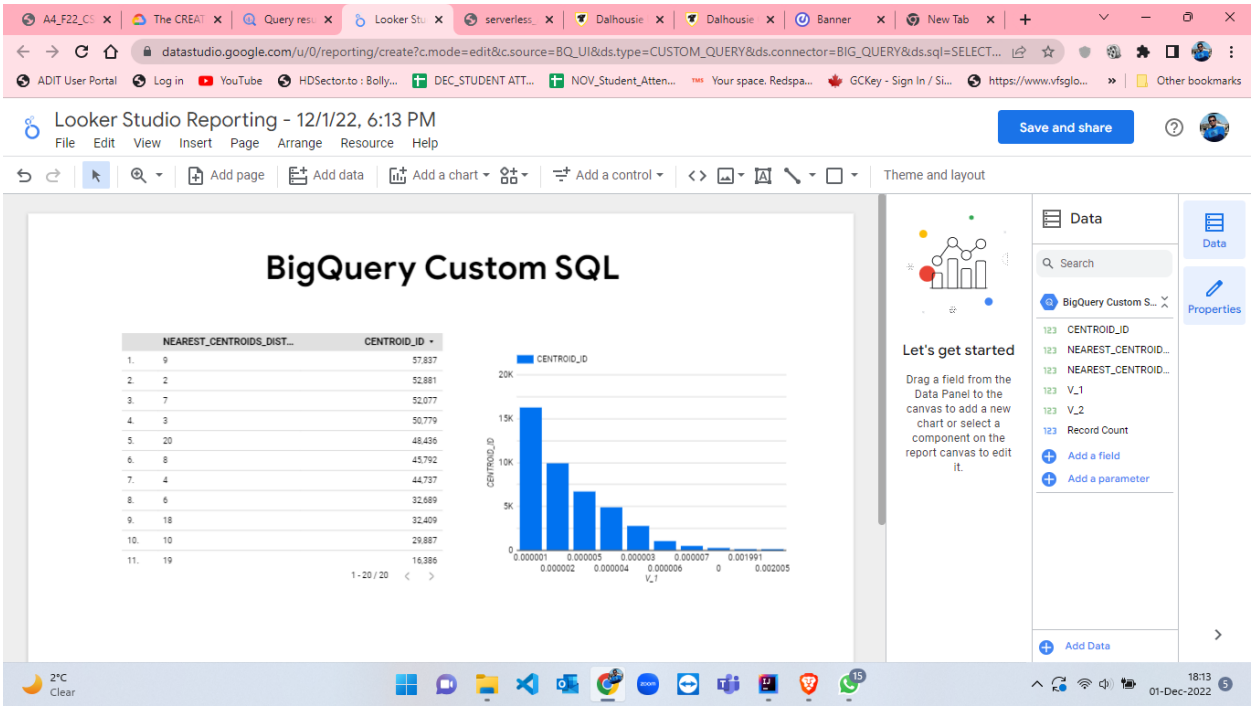

Figure 119: Looker Studio Reporting for visualization for TestSet table.

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,” Educative: Interactive Courses for Software Developers,2022 [Onine]. 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].