“Project: Building an Intelligent Health Monitoring & Heart Attack Prediction System Using AWS EMR, SageMaker, Lambda, Athena, and SNS”

1. Project Description

The Amazon Web Services (AWS) ecosystem was used to create the cloud-based predictive analytics solution known as the Intelligent Health Monitoring and Heart Attack Prediction System. The study shows how cloud computing, big data, and artificial intelligence may be used by contemporary healthcare systems to identify cardiac risks, continually monitor patient health, and promptly notify physicians and caregivers.

The solution creates a completely automated, end-to-end data pipeline by integrating several AWS services. A centralized data lake is created by safely storing patient data in Amazon S3, including both history records and simulated real-time vitals. The system integrates and aggregates this data at scale using Amazon EMR (Apache Spark), computing weekly averages of blood pressure, heart rate, sleep duration, and activity levels to produce significant health insights.

A machine learning model (XGBoost) that forecasts each patient's risk of a heart attack based on their combined demographic and physiological data is trained and deployed using Amazon SageMaker after the data has been cleansed and converted. Instant risk scoring is thus made possible by deploying the model as a real-time inference endpoint.

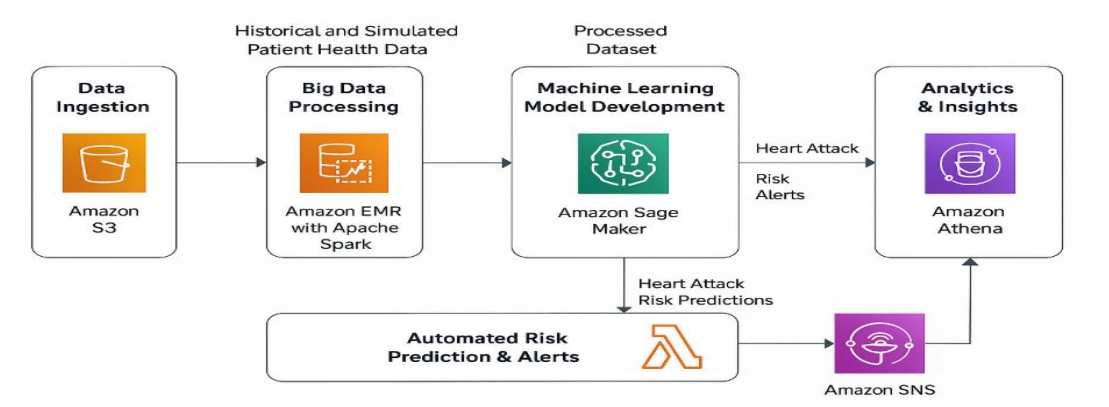
When a patient surpasses a predetermined risk threshold, an AWS Lambda function accesses the SageMaker endpoint for fresh patient data, assesses risk levels, and sends out Amazon

SNS (Simple Notification Service) notifications to healthcare providers in order to automate reactions. Lastly, Amazon Athena gives clinicians data-driven insights for long-term health monitoring and preventive care by enabling SQL-based querying and visualization of past predictions and trends straight from S3.

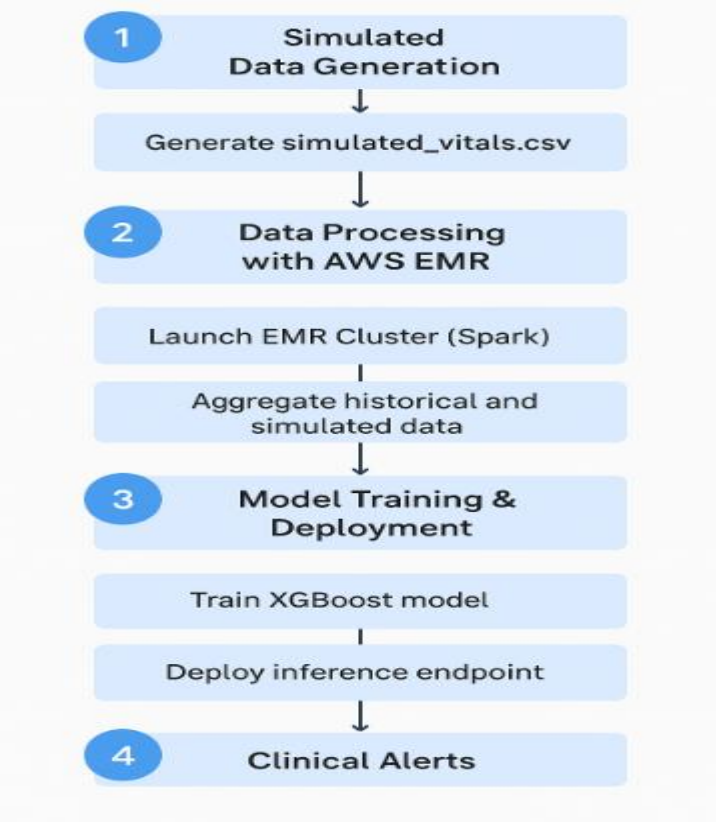
This project demonstrates how AWS technologies may be coordinated to create a proactive, scalable, and intelligent healthcare system. It illustrates how cloud-based machine learning can enhance early diagnosis, lessen medical emergencies, and promote better clinical decision-making by mimicking continuous monitoring, risk prediction, and automatic warning.

2. Project Workflow

An end-to-end cloud-based workflow that links several AWS services into a smooth data processing and prediction pipeline is used by the Intelligent Health Monitoring and Heart Attack Prediction System. From absorbing and processing health data to training models, generating alarms, and carrying out analytics, each stage of the workflow has a distinct purpose.



Intelligent Health Monitoring and Heart Attack Prediction Workflow



1. **Data Ingestion (Amazon S3)**
   A consolidated, secure data lake is created by storing all historical and simulated patient health data in an Amazon S3 bucket. While simulated vitals (heart rate, blood pressure, sleep hours, activity) replicate real-time wearable sensor readings, the historical dataset offers patient demographics and prior medical data.

2. **Big Data Processing (Apache Spark and Amazon EMR)**
   The system cleans, aggregates, and transforms the raw datasets using Apache Spark on Amazon EMR. Meaningful weekly health summaries are computed by averaging seven days of simulated vital signs for each patient. After processing and merging, the dataset is saved back to S3 so that the model can be trained.

3. **Developing Machine Learning Models using Amazon SageMaker**
   An XGBoost classification model that forecasts heart attack risk based on patient health indicators and lifestyle characteristics is trained using the processed dataset in SageMaker. For continuous prediction, the trained model is used as a real-time inference endpoint.

4. **Automated Risk Forecasting and Alerts (SNS + AWS Lambda)**
   Every time fresh patient data is received, a serverless Lambda function instantly initiates the SageMaker model. Each patient's risk score is determined by the function, which then sends out an email alert to clinicians via Amazon SNS if the score surpasses a certain level.

5. **Amazon Athena's Analytics & Insights**
   Healthcare professionals can compare past and anticipated risks, examine patterns, and obtain important clinical insights by using Athena to conduct SQL queries directly on the S3 data. The feedback loop for data-driven health monitoring and decision-making is completed in this step.

This process shows how AWS may be used to create a scalable, intelligent healthcare monitoring pipeline. It enables real-time, proactive cardiac risk identification and effective healthcare intervention by automating every step, from data acquisition and processing to prediction, alerting, and analytics.

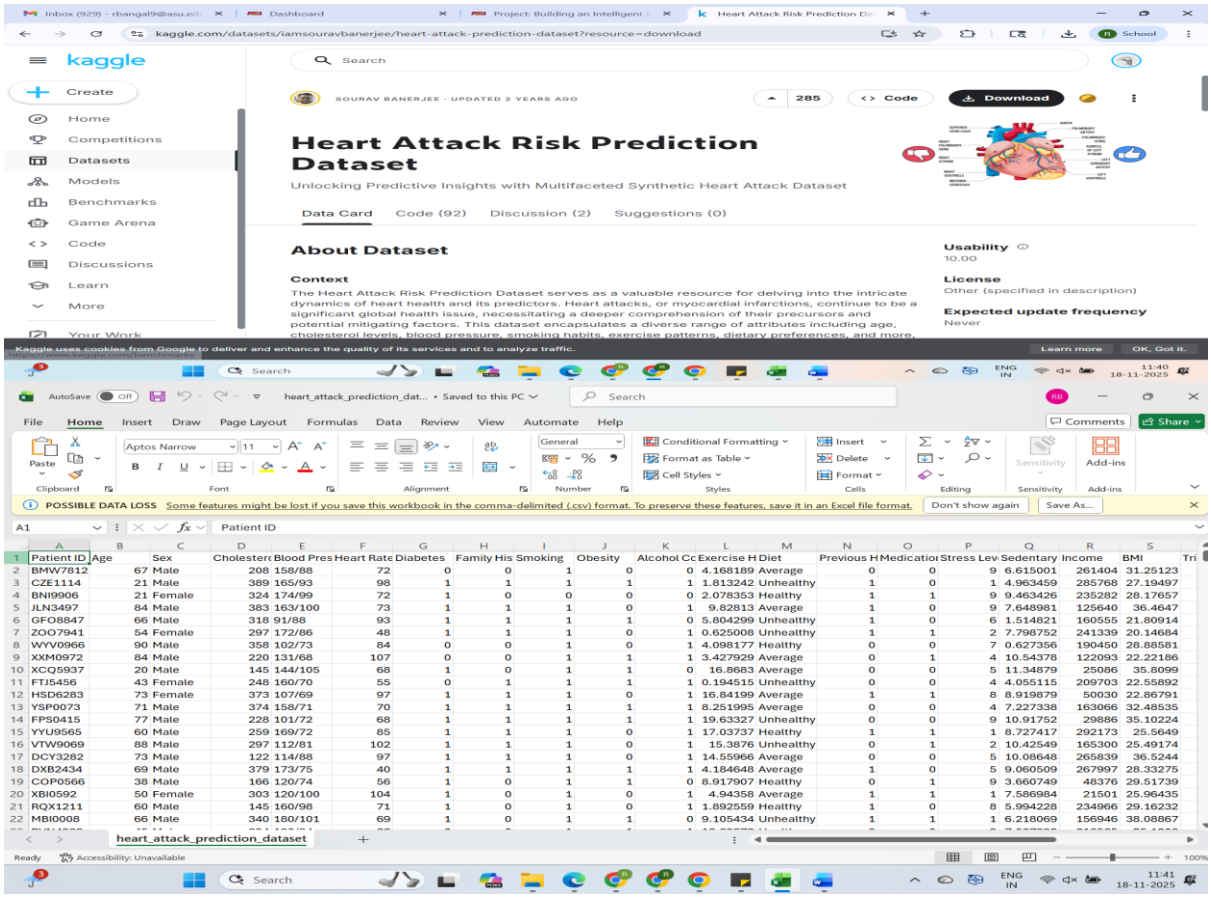
Download and Upload Historical Dataset
Upload to Amazon S3

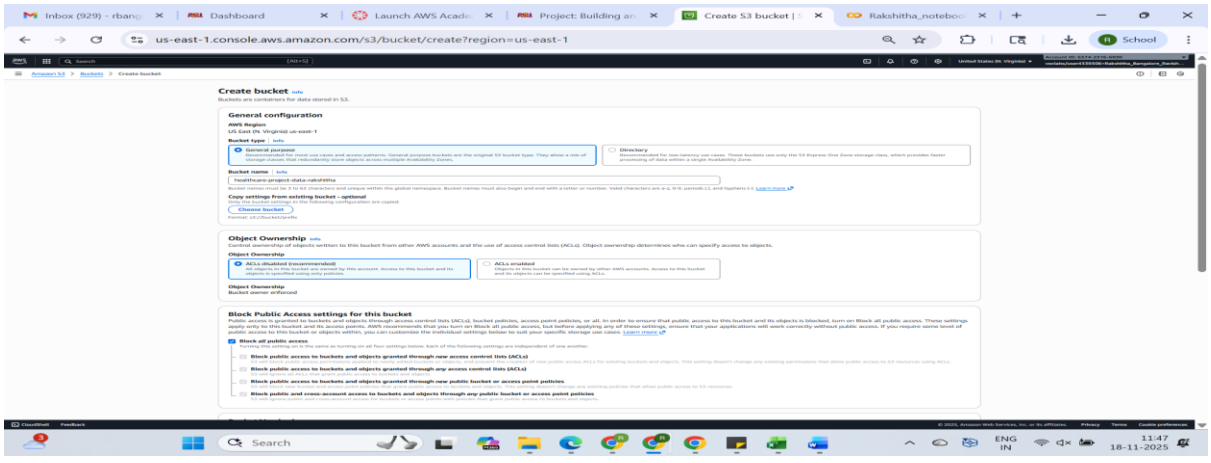
1.
    1. **Go to Amazon S3 Console → Create bucket**
        - **Name:**

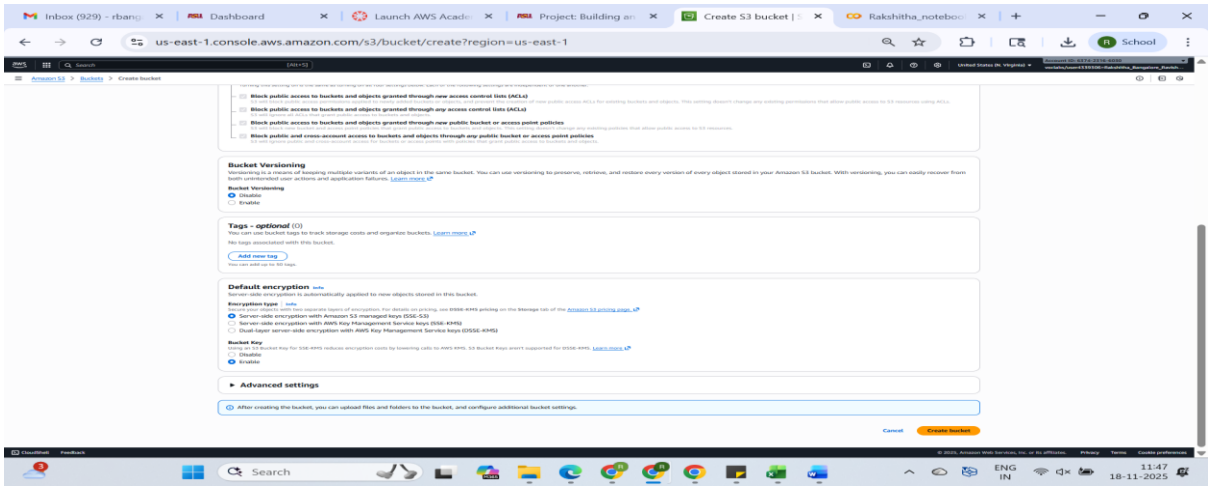
healthcare-project-data-rakshitha (replace rakshitha with your own)

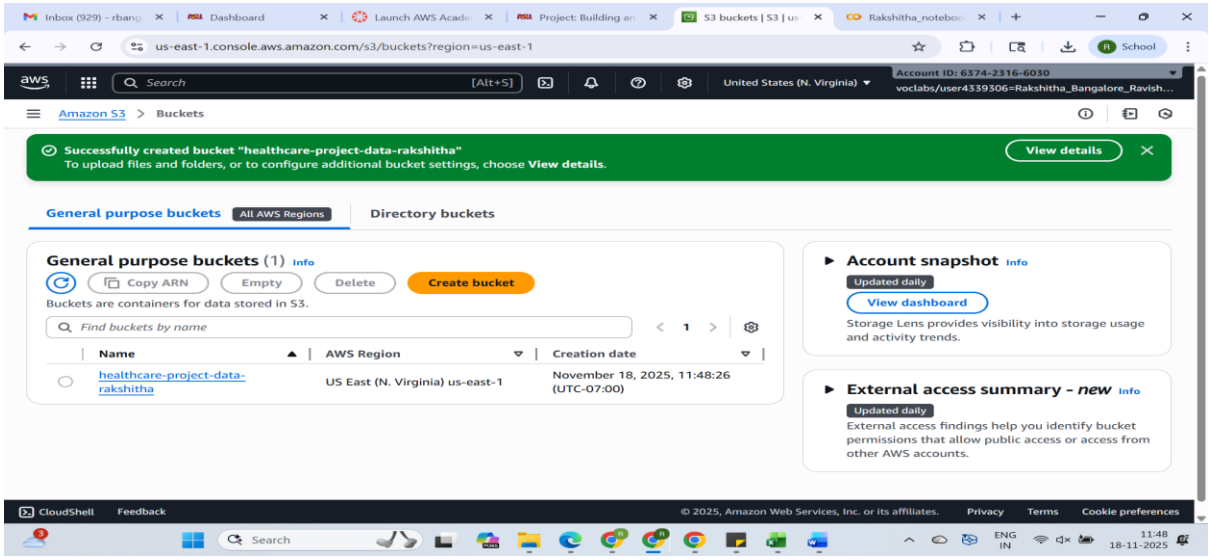
        - **Region: same as your AWS Lab environment**
        - **Leave defaults → Create bucket**
    2. **Inside the bucket, create folders (raw, and historical inside raw):**

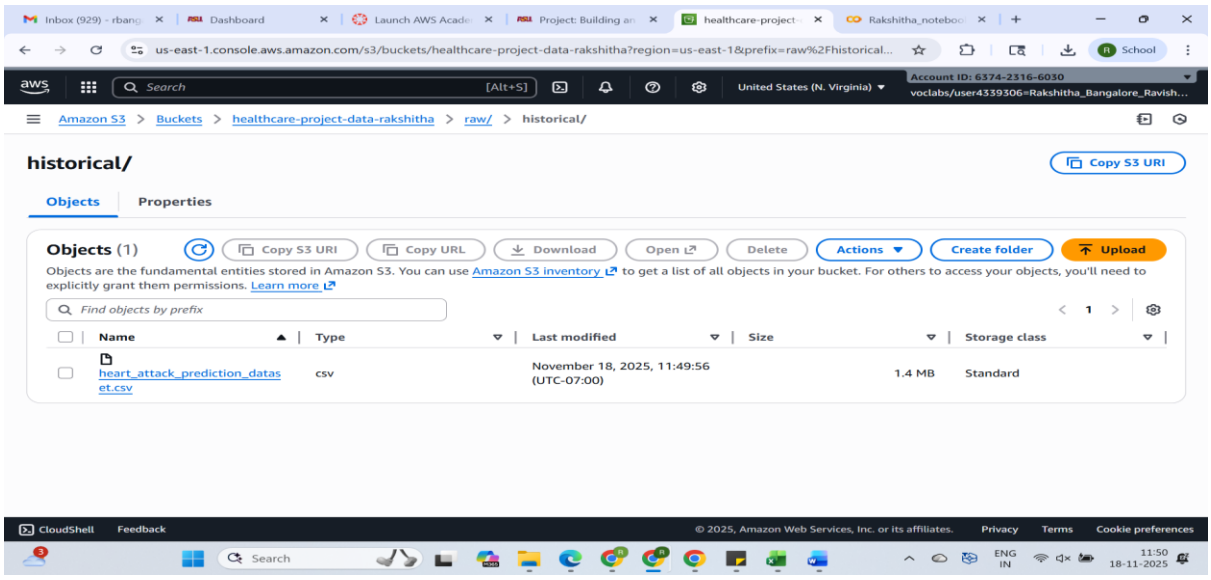
raw/
└──historical/


kaggle
Heart Attack Risk Prediction Dataset
Unlocking Predictive Insights with Multifaceted Synthetic Heart Attack Dataset
Data Card
Code (92)
Discussion (2)
Suggestions (0)
About Dataset
Context
The Heart Attack Risk Prediction Dataset serves as a valuable resource for delving into the intricate dynamics of heart health and its predictors. Heart attacks, or myocardial infarctions, continue to be a significant global health issue, necessitating a deeper comprehension of their precursors and potential mitigating factors. This dataset encapsulates a diverse range of attributes including age, cholesterol levels, blood pressure, smoking habits, exercise patterns, dietary preferences, and more.
Usability
10.00
License
Other (specified in description)
Expected update frequency
Never
heart_attack_prediction_dat... • Saved to this PC
POSSIBLE DATA LOSS
Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.
Don't show again
Save As...
A1
Patient ID
heart_attack_prediction_dataset

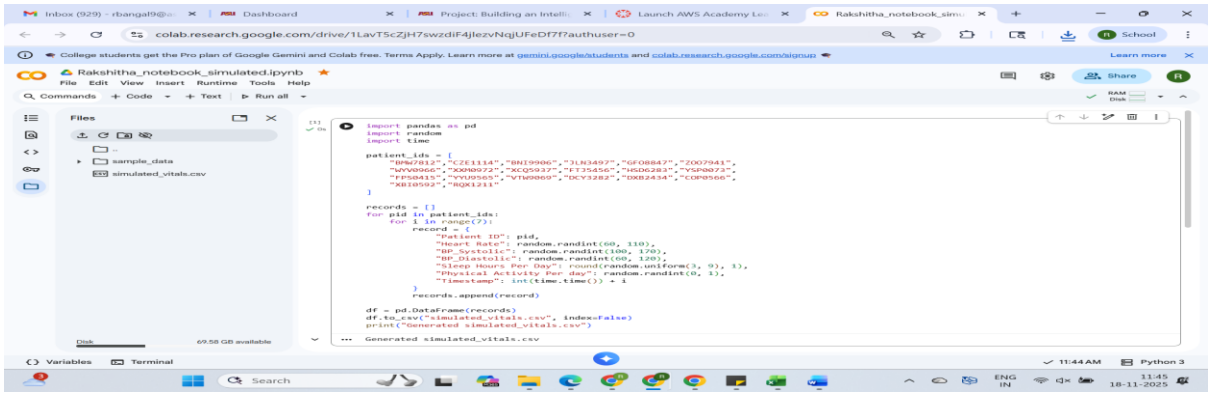

us-east-1.console.aws.amazon.com/s3/bucket/create?region=us-east-1
Create bucket
General configuration
AWS Region
US East (N. Virginia) us-east-1
Bucket type
Bucket name
healthcare-project-data-rakshitha
Object Ownership
Block Public Access settings for this bucket





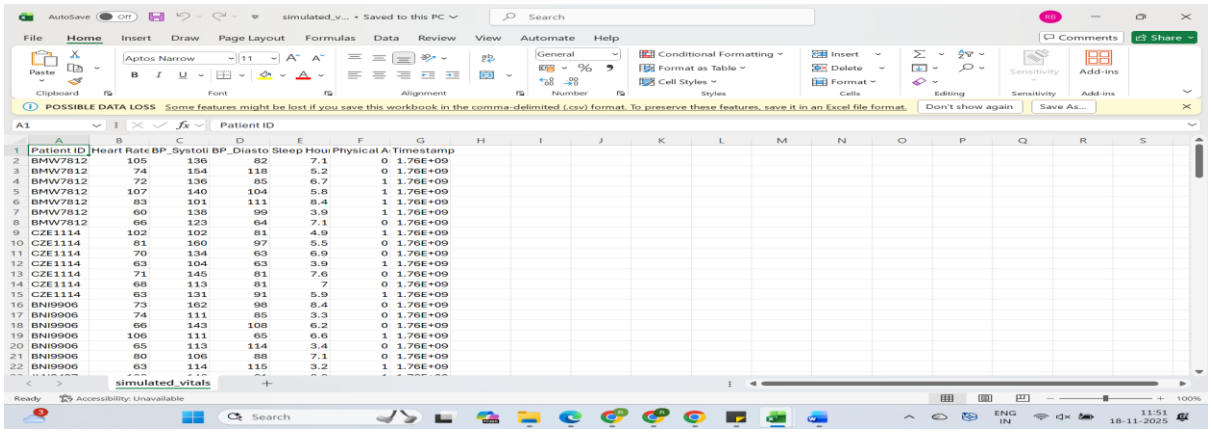


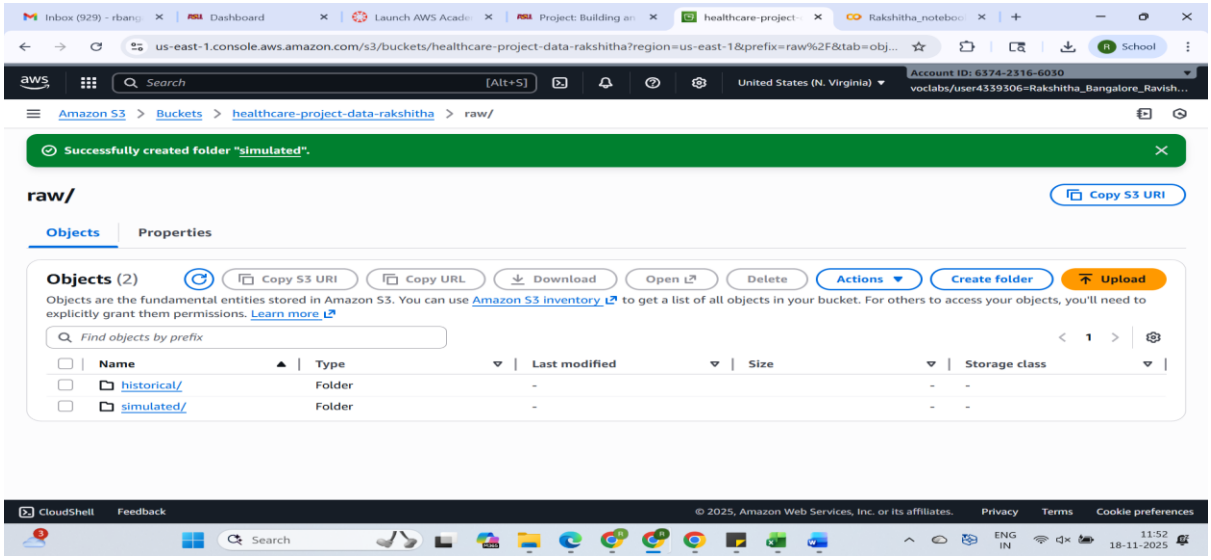
Phase 1 . Prepare Input Data

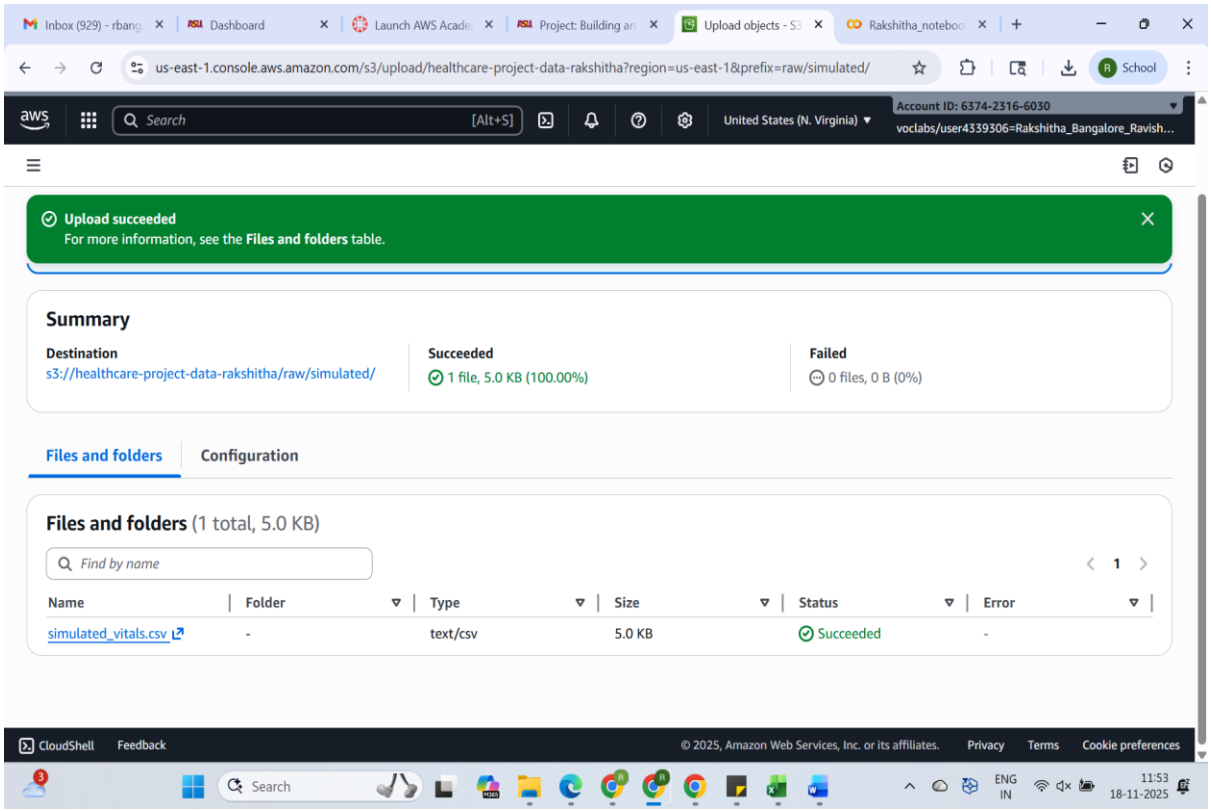


Today

simulated_vitals	18-11-2025 11:44	Microsoft Excel Com...	5 KB
heart_attack_prediction_dataset	18-11-2025 11:38	Microsoft Excel Com...	1,423 KB







Phase 2: Process Data Using AWS EMR (Spark)

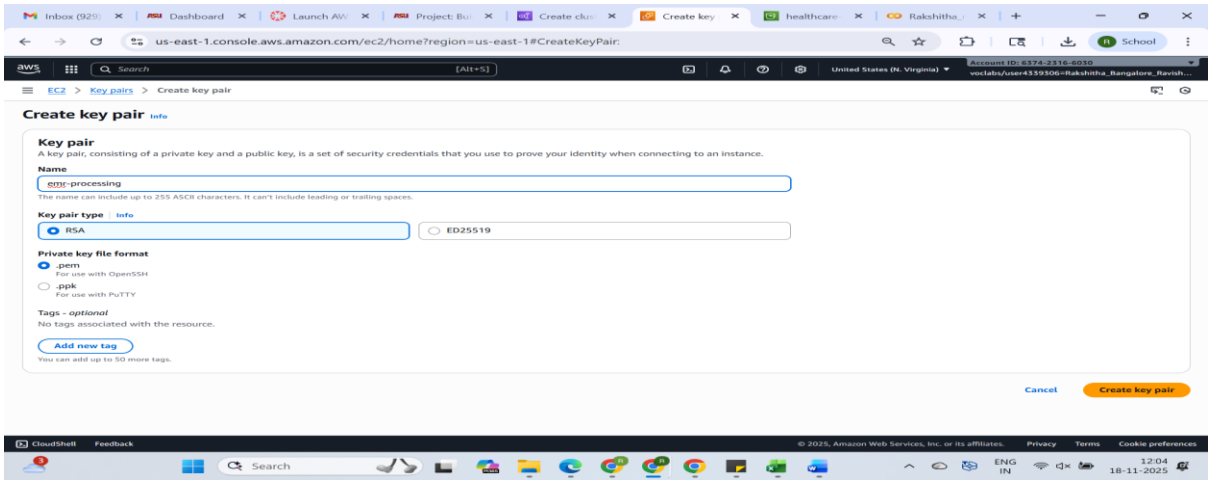
2.1. Launch an EMR Cluster

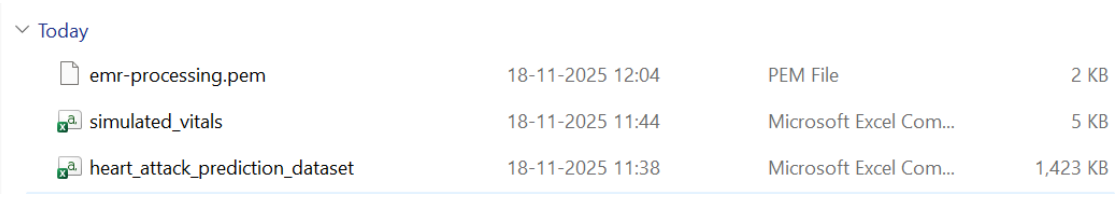
Go to Amazon EMR Console and Launch an EMR Cluster of EMR Lab to launch the cluster and download the key pair .pem file

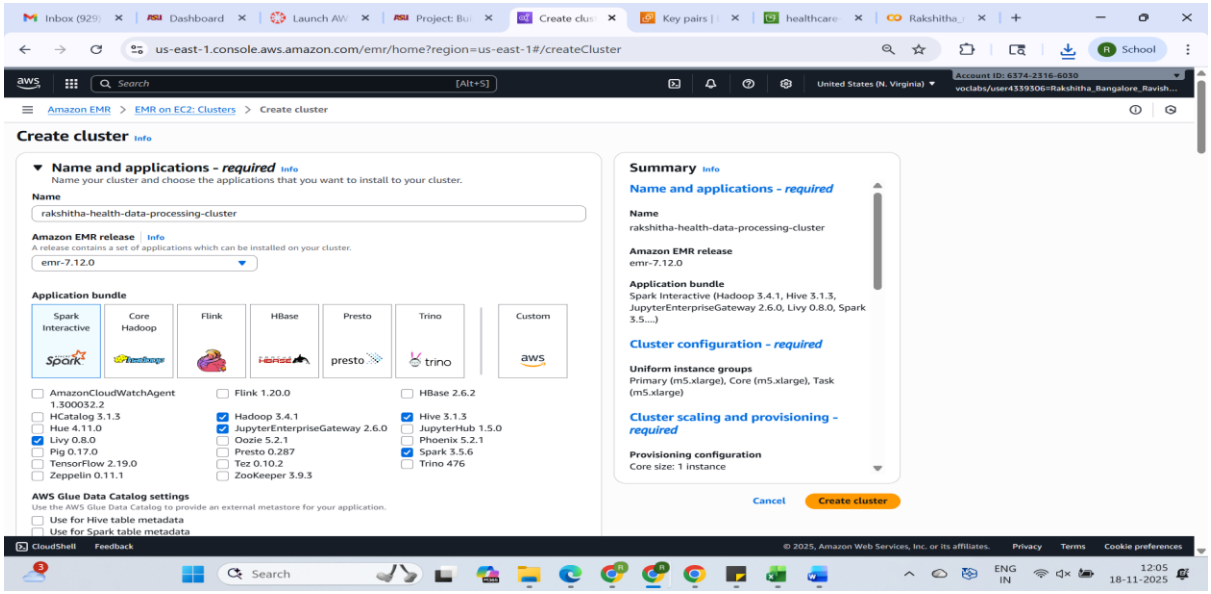
name: rakshitha-health-data-processing-cluster

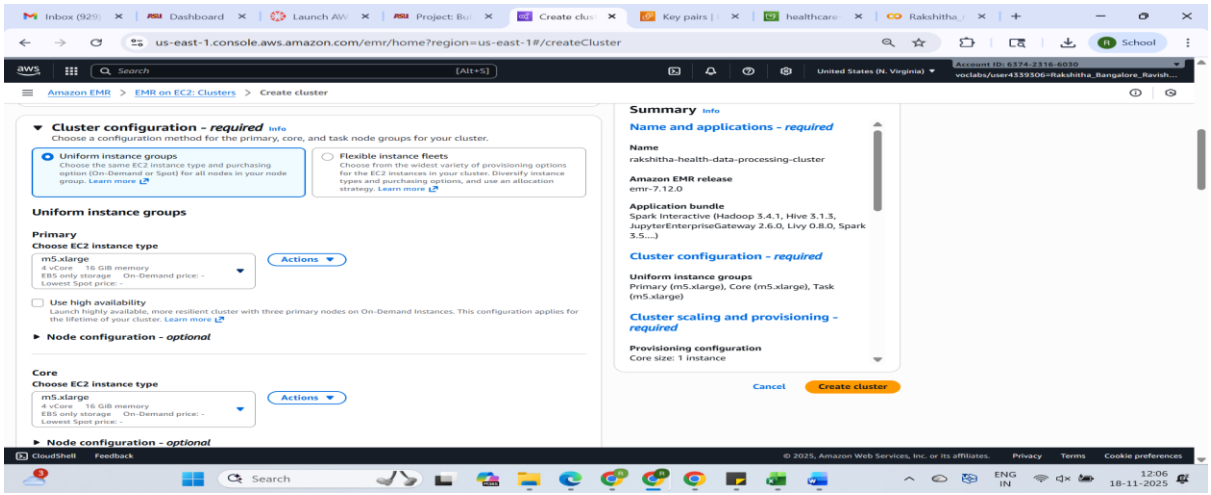
logs: s3://healthcare-project-data-rakshitha/logs

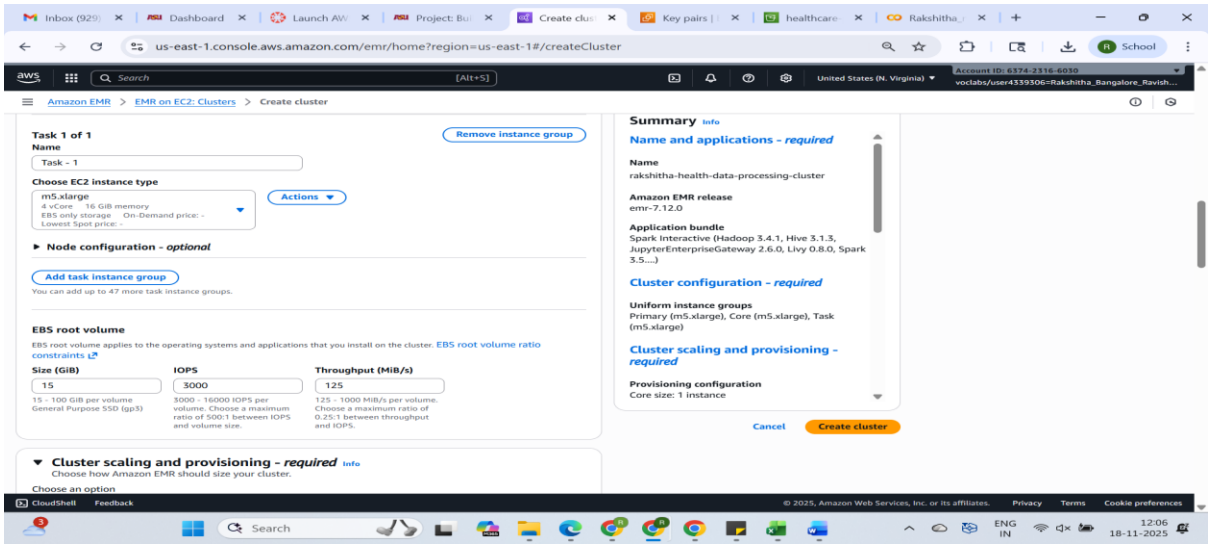
key pair .pem file name: emr-processing

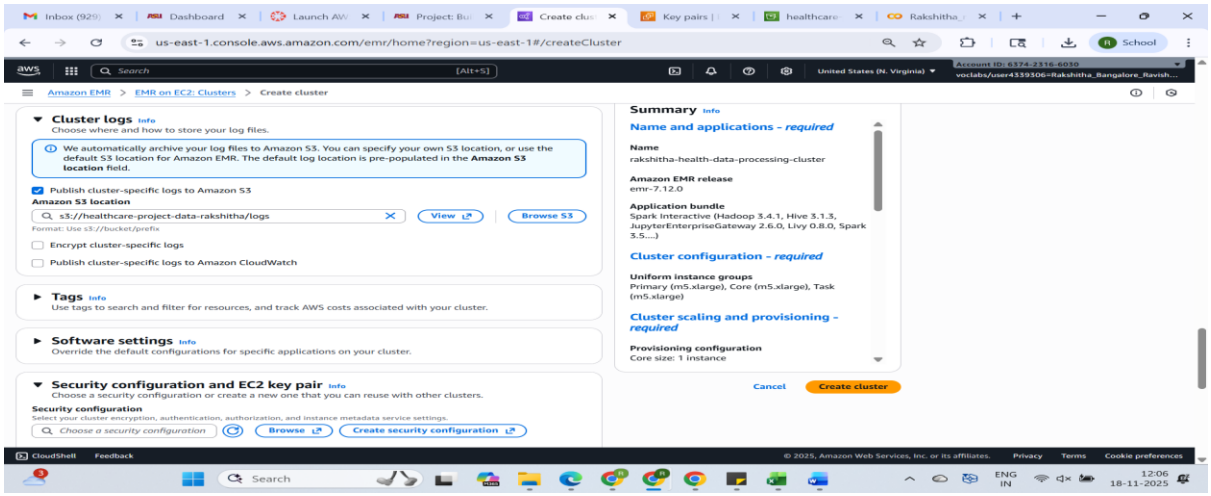

Create key pair
Key pair
A key pair, consisting of a private key and a public key, is a set of security credentials that you use to prove your identity when connecting to an instance.
Name
emr-processing
Key pair type
RSA
ED25519
Private key file format
.pem
.ppk
Tags - optional
No tags associated with the resource.
Add new tag
Cancel
Create key pair

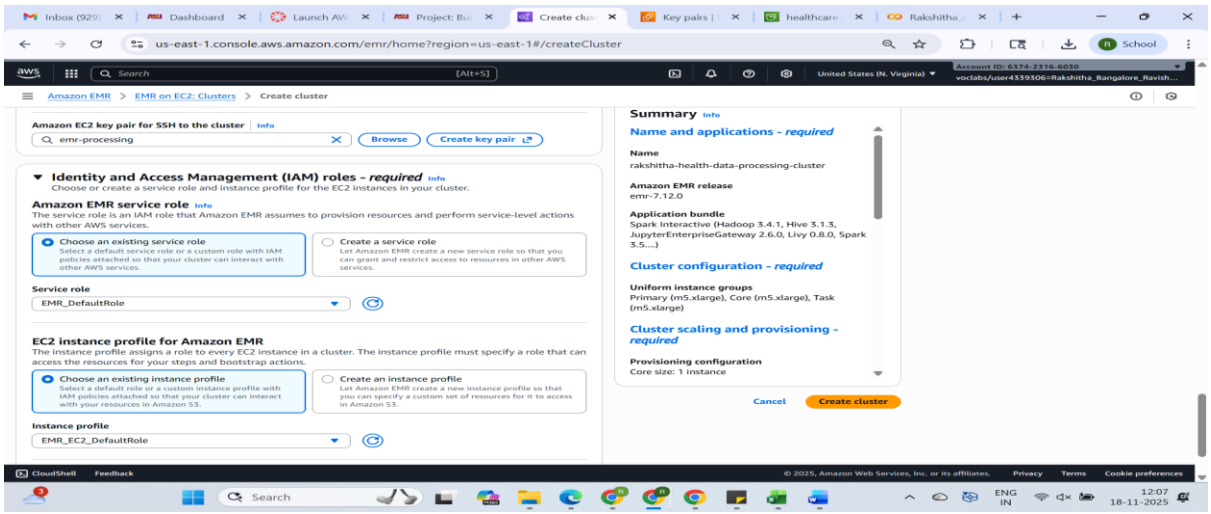

Today
emr-processing.pem 18-11-2025 12:04 PEM File 2 KB
simulated_vitals 18-11-2025 11:44 Microsoft Excel Com... 5 KB
heart_attack_prediction_dataset 18-11-2025 11:38 Microsoft Excel Com... 1,423 KB

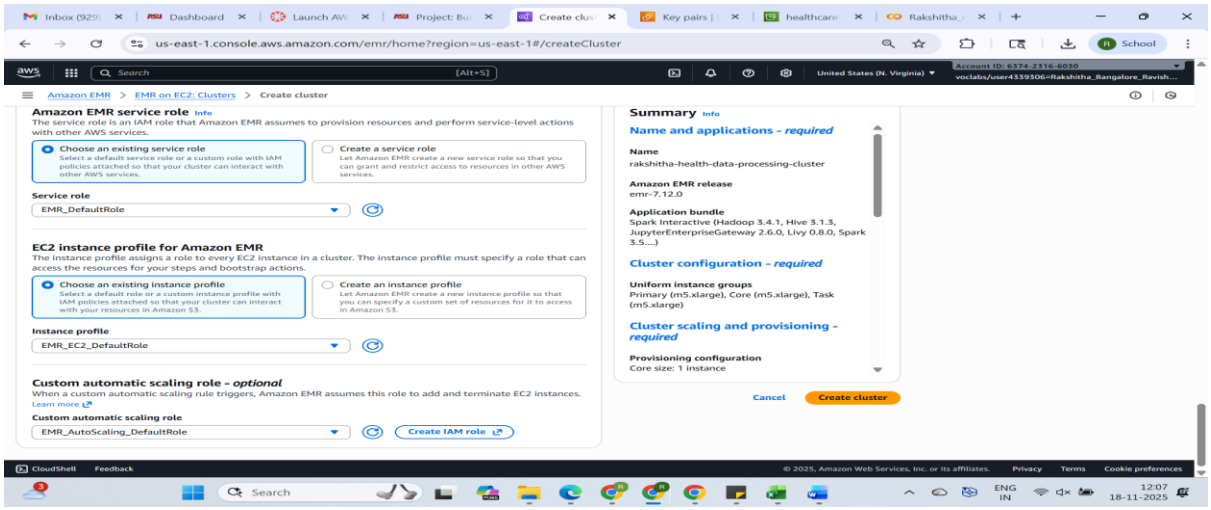

Create cluster
Name and applications - required
Name
rakshitha-health-data-processing-cluster
Amazon EMR release
emr-7.12.0
Application bundle
Spark Interactive
Core Hadoop
Flink
HBase
Presto
Trino
Custom
Summary
Cluster configuration - required
Cluster scaling and provisioning - required
Cancel
Create cluster

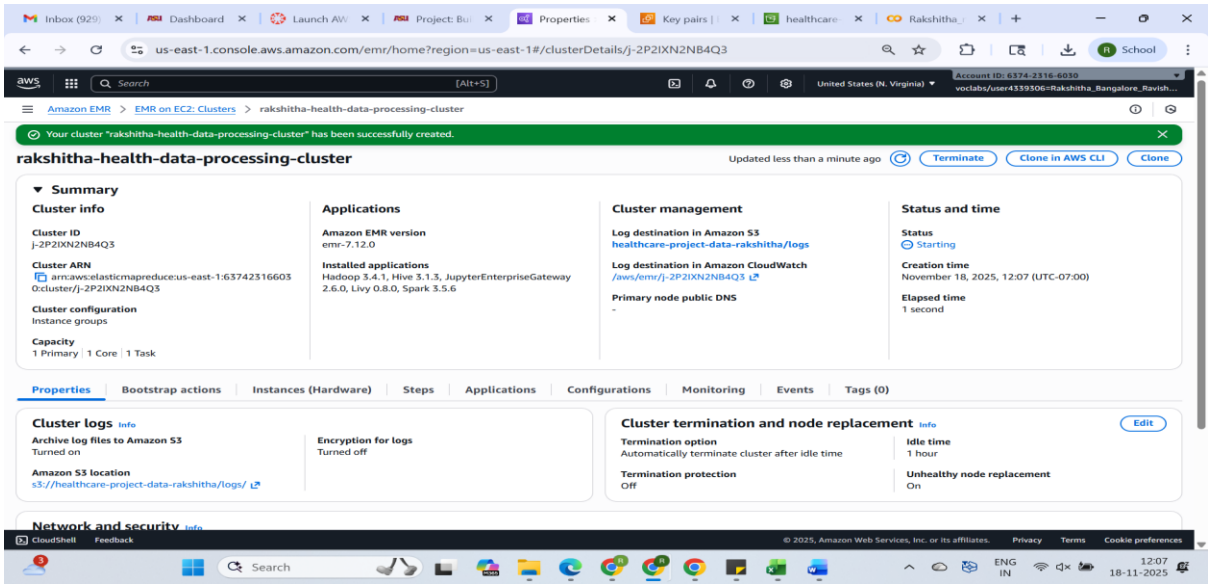

Cluster configuration - required
Uniform instance groups
Flexible instance fleets
Uniform instance groups
Primary
Choose EC2 instance type
m5.xlarge
Actions
Use high availability
Node configuration - optional
Core
Summary
Cancel
Create cluster

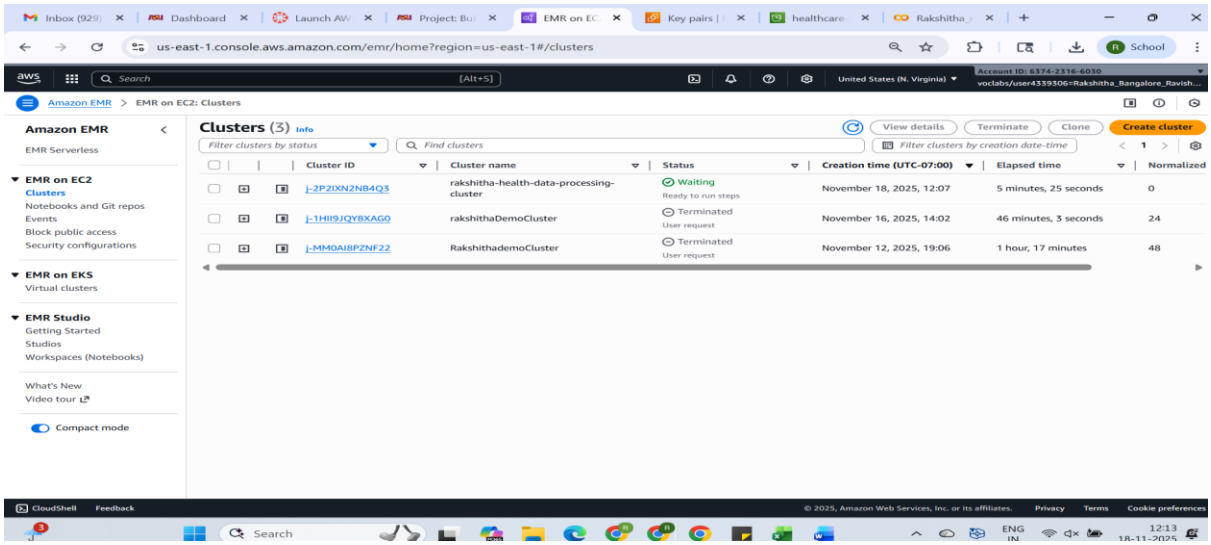

us-east-1.console.aws.amazon.com/emr/home?region=us-east-1#/createCluster
Amazon EMR > EMR on EC2: Clusters > Create cluster
Task 1 of 1
Name
Task - 1
Choose EC2 instance type
m5.xlarge
Actions
Remove instance group
Node configuration - optional
Add task instance group
You can add up to 47 more task instance groups.
EBS root volume
Size (GiB)
15
IOPS
3000
Throughput (MiB/s)
125
Cluster scaling and provisioning - required Info
Choose how Amazon EMR should size your cluster.
Choose an option
Summary Info
Name and applications - required
Name
rakshitha-health-data-processing-cluster
Amazon EMR release
emr-7.12.0
Application bundle
Spark Interactive (Hadoop 3.4.1, Hive 3.1.3, JupyterEnterpriseGateway 2.6.0, Livy 0.8.0, Spark 3.5....)
Cluster configuration - required
Uniform instance groups
Primary (m5.xlarge), Core (m5.xlarge), Task (m5.xlarge)
Cluster scaling and provisioning - required
Provisioning configuration
Core size: 1 instance
Cancel
Create cluster


us-east-1.console.aws.amazon.com/emr/home?region=us-east-1#/createCluster
Amazon EMR > EMR on EC2: Clusters > Create cluster
Cluster logs Info
Choose where and how to store your log files.
We automatically archive your log files to Amazon S3. You can specify your own S3 location, or use the default S3 location for Amazon EMR. The default log location is pre-populated in the Amazon S3 location field.
Publish cluster-specific logs to Amazon S3
Amazon S3 location
s3://healthcare-project-data-rakshitha/logs
View
Browse S3
Encrypt cluster-specific logs
Publish cluster-specific logs to Amazon CloudWatch
Tags Info
Use tags to search and filter for resources, and track AWS costs associated with your cluster.
Software settings Info
Override the default configurations for specific applications on your cluster.
Security configuration and EC2 key pair Info
Choose a security configuration or create a new one that you can reuse with other clusters.
Security configuration
Choose a security configuration
Browse
Create security configuration
Summary Info
Name and applications - required
Name
rakshitha-health-data-processing-cluster
Amazon EMR release
emr-7.12.0
Application bundle
Spark Interactive (Hadoop 3.4.1, Hive 3.1.3, JupyterEnterpriseGateway 2.6.0, Livy 0.8.0, Spark 3.5....)
Cluster configuration - required
Uniform instance groups
Primary (m5.xlarge), Core (m5.xlarge), Task (m5.xlarge)
Cluster scaling and provisioning - required
Provisioning configuration
Core size: 1 instance
Cancel
Create cluster


us-east-1.console.aws.amazon.com/emr/home?region=us-east-1#/createCluster
Amazon EMR > EMR on EC2: Clusters > Create cluster
Amazon EC2 key pair for SSH to the cluster Info
emr-processing
Browse
Create key pair
Identity and Access Management (IAM) roles - required Info
Choose or create a service role and instance profile for the EC2 instances in your cluster.
Amazon EMR service role Info
The service role is an IAM role that Amazon EMR assumes to provision resources and perform service-level actions with other AWS services.
Choose an existing service role
Create a service role
Service role
EMR_DefaultRole
EC2 instance profile for Amazon EMR
The instance profile assigns a role to every EC2 instance in a cluster. The instance profile must specify a role that can access the resources for your steps and bootstrap actions.
Choose an existing instance profile
Create an instance profile
Instance profile
EMR_EC2_DefaultRole
Summary Info
Name and applications - required
Name
rakshitha-health-data-processing-cluster
Amazon EMR release
emr-7.12.0
Application bundle
Spark Interactive (Hadoop 3.4.1, Hive 3.1.3, JupyterEnterpriseGateway 2.6.0, Livy 0.8.0, Spark 3.5....)
Cluster configuration - required
Uniform instance groups
Primary (m5.xlarge), Core (m5.xlarge), Task (m5.xlarge)
Cluster scaling and provisioning - required
Provisioning configuration
Core size: 1 instance
Cancel
Create cluster



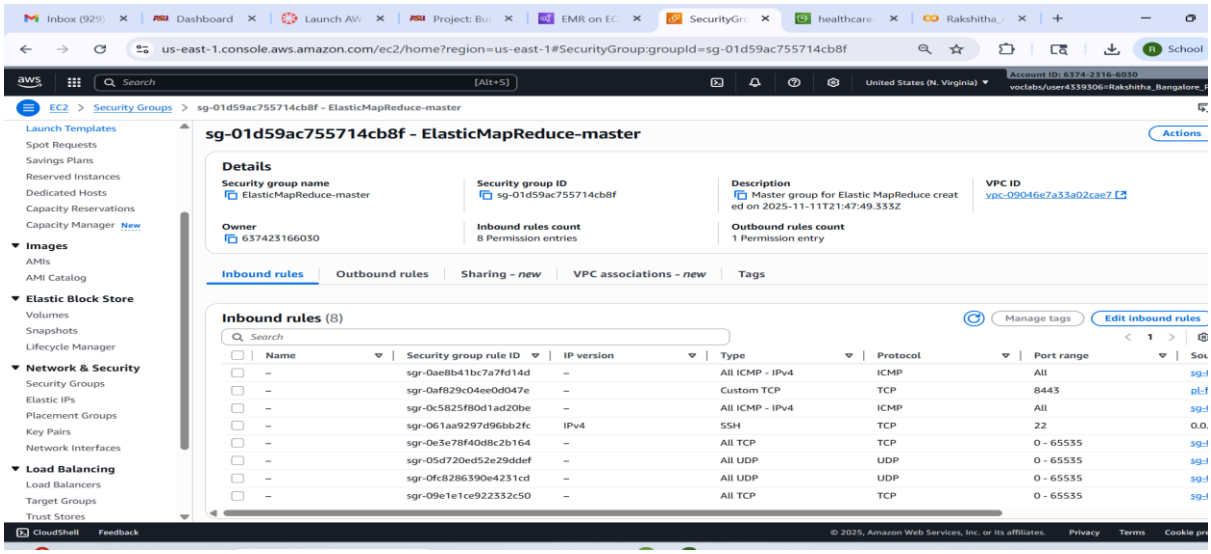


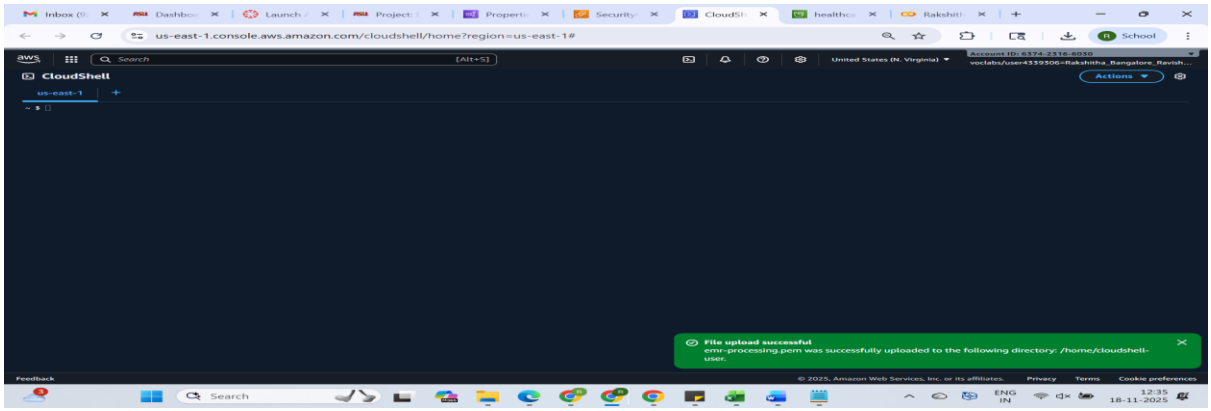


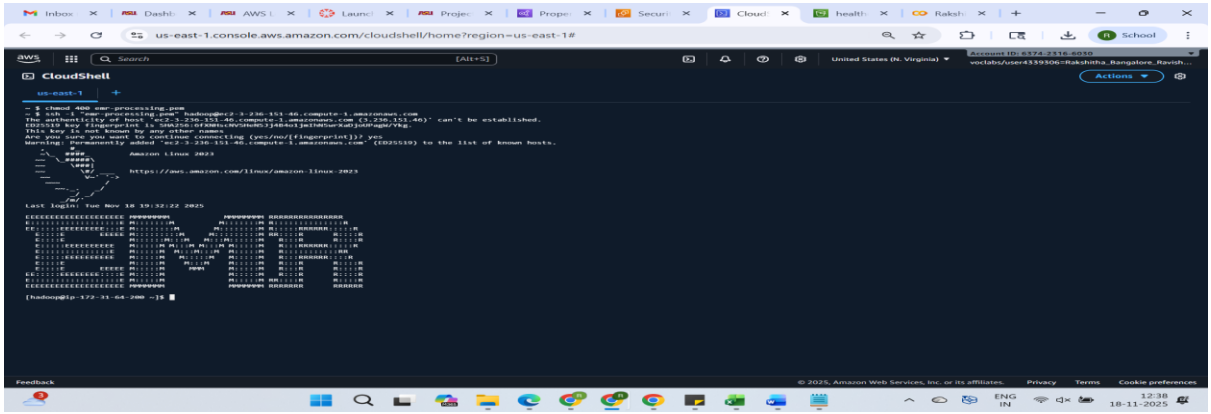
2.3. Connect to EMR Cluster via SSH

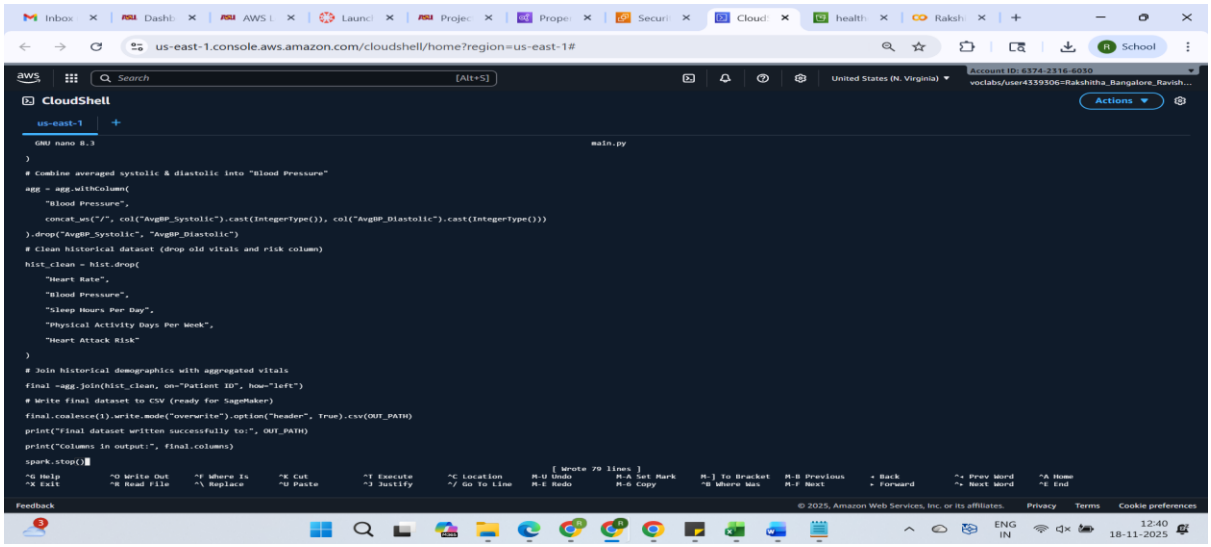
Use the Step 4: Connect to EMR Cluster via SSH of EMR LAB to connect to the EMR cluster.

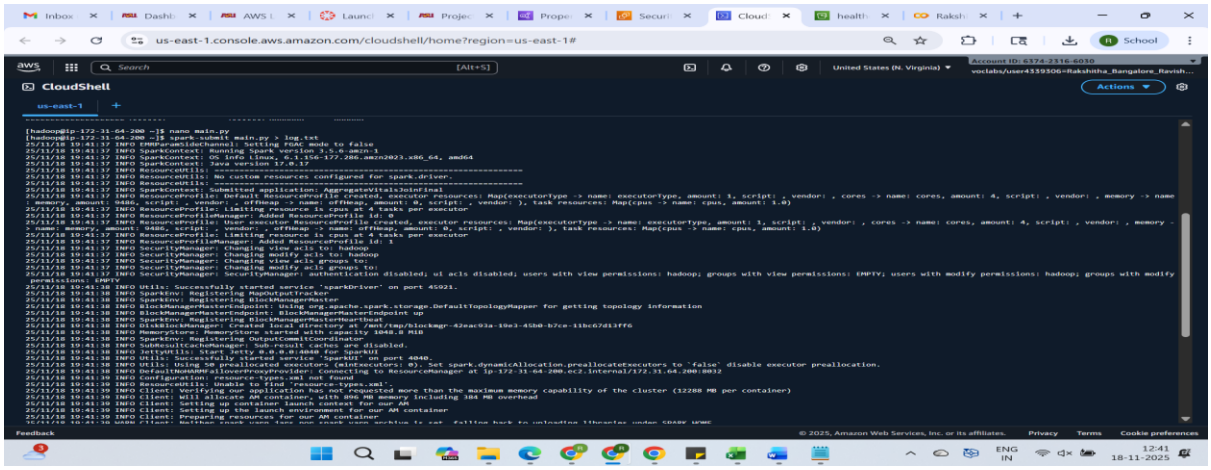
Make sure to use the emr-processing.pem file you downloaded previously and the above given main.py code

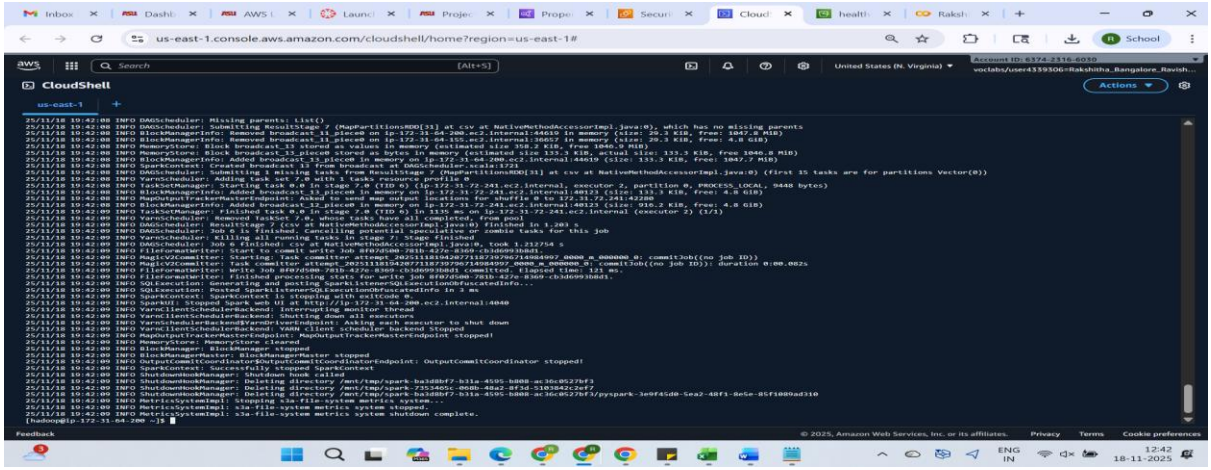


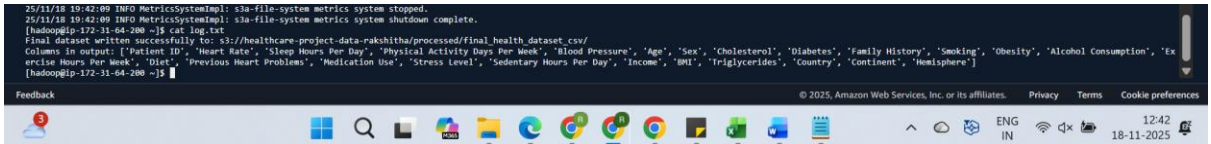


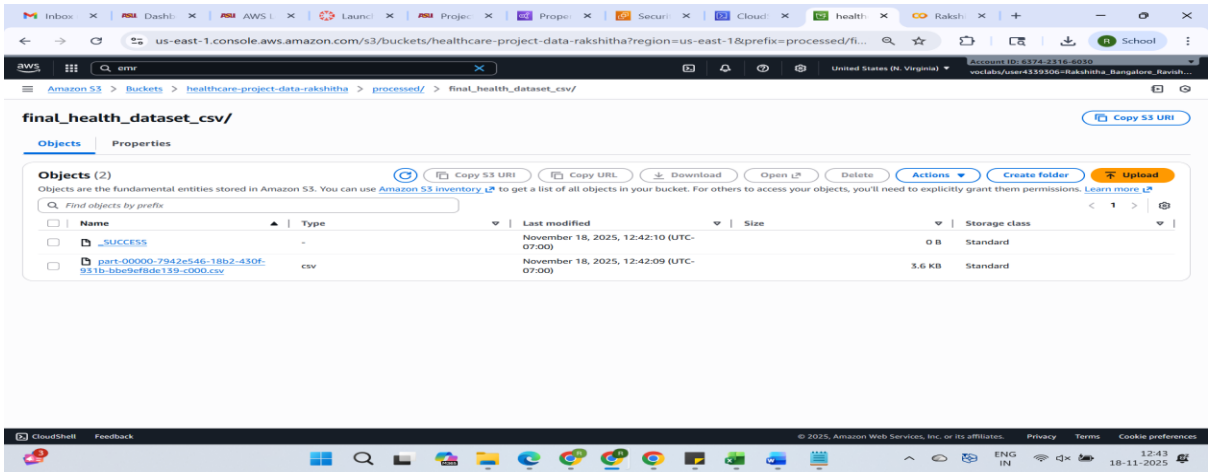


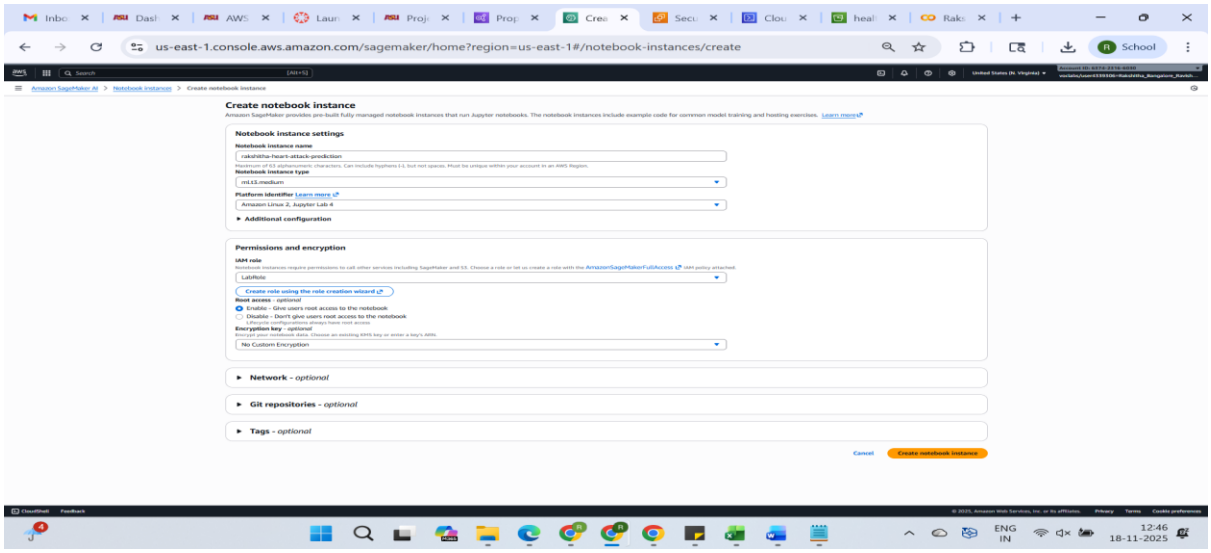

us-east-1.console.aws.amazon.com/cloudshell/home?region=us-east-1#
CloudShell
us-east-1
GNU nano 8.3
main.py
# Combine averaged systolic & diastolic into "Blood Pressure"
agg = agg.withColumn(
"Blood Pressure",
concat_ws("/", col("AvgBP_Systolic").cast(IntegerType()), col("AvgBP_Diastolic").cast(IntegerType()))
).drop("AvgBP_Systolic", "AvgBP_Diastolic")
# Clean historical dataset (drop old vitals and risk column)
hist_clean = hist.drop(
"Heart Rate",
"Blood Pressure",
"Sleep Hours Per Day",
"Physical Activity Days Per Week",
"Heart Attack Risk"
)
# Join historical demographics with aggregated vitals
final = agg.join(hist_clean, on="Patient ID", how="left")
# Write final dataset to CSV (ready for SageMaker)
final.coalesce(1).write.mode("overwrite").option("header", True).csv(OUT_PATH)
print("Final dataset written successfully to:", OUT_PATH)
print("Columns in output:", final.columns)
spark.stop()
[Wrote 79 lines]


us-east-1.console.aws.amazon.com/cloudshell/home?region=us-east-1#
CloudShell
us-east-1
[hadoop@ip-172-31-64-200 ~]$ nano main.py
[hadoop@ip-172-31-64-200 ~]$ spark-submit main.py > log.txt

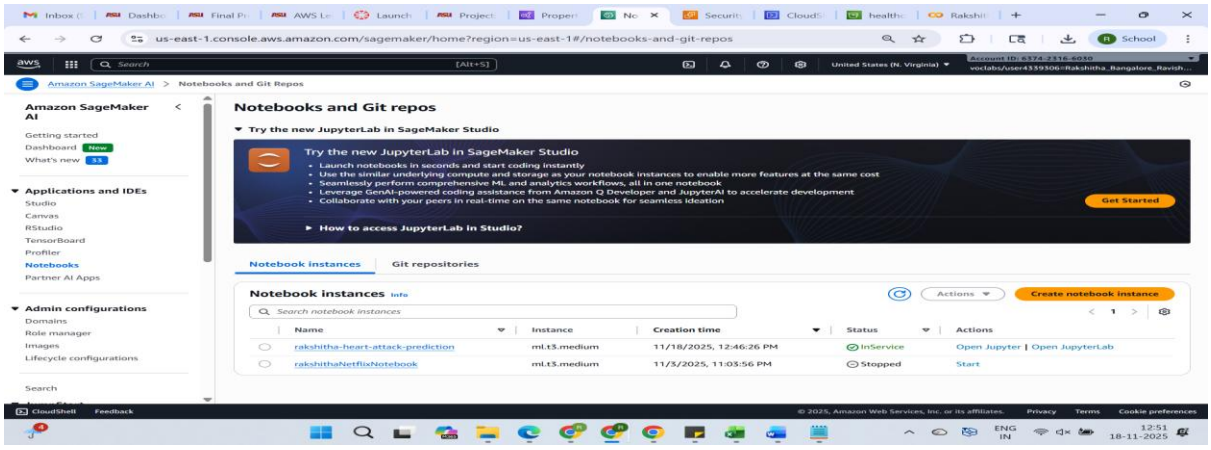

us-east-1.console.aws.amazon.com/cloudshell/home?region=us-east-1#
CloudShell
us-east-1

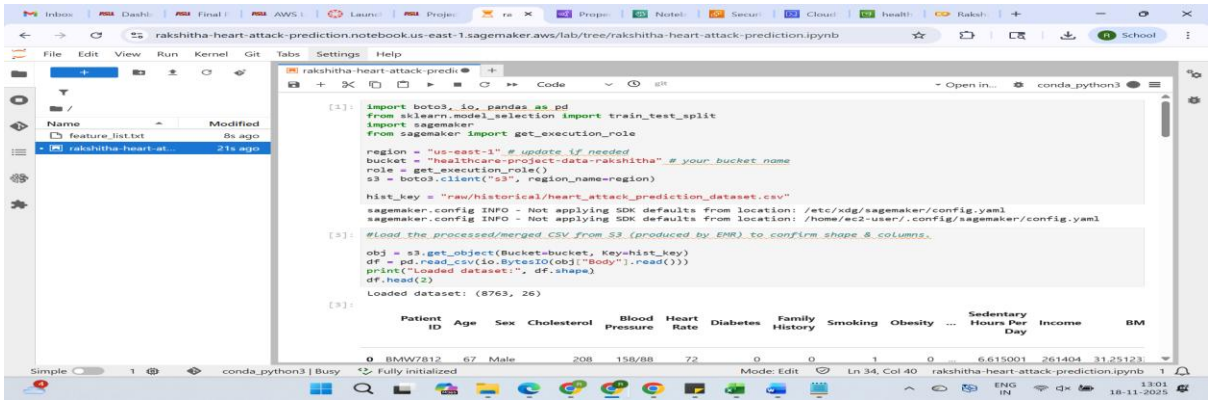

25/11/18 19:42:09 INFO MetricsSystemImpl: s3a-file-system metrics system stopped.
25/11/18 19:42:09 INFO MetricsSystemImpl: s3a-file-system metrics system shutdown complete.
[hadoop@ip-172-31-64-200 ~]$ cat log.txt
Final dataset written successfully to: s3://healthcare-project-data-rakshitha/processed/final_health_dataset_csv/
Columns in output: ['Patient ID', 'Heart Rate', 'Sleep Hours Per Day', 'Physical Activity Days Per Week', 'Blood Pressure', 'Age', 'Sex', 'Cholesterol', 'Diabetes', 'Family History', 'Smoking', 'Obesity', 'Alcohol Consumption', 'Exercise Hours Per Week', 'Diet', 'Previous Heart Problems', 'Medication Use', 'Stress Level', 'Sedentary Hours Per Day', 'Income', 'BMI', 'Triglycerides', 'Country', 'Continent', 'Hemisphere']
[hadoop@ip-172-31-64-200 ~]$

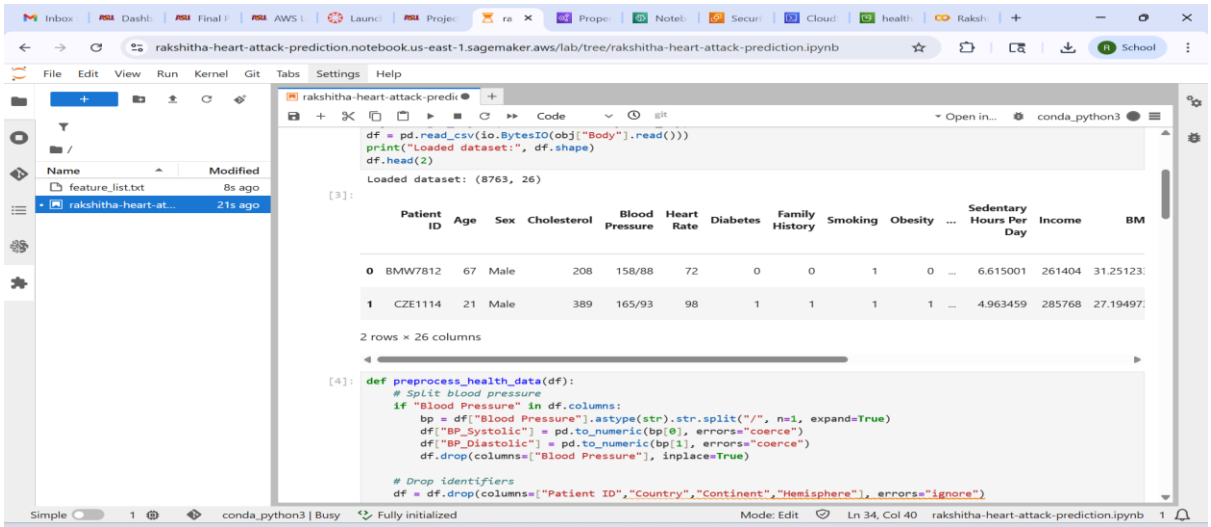


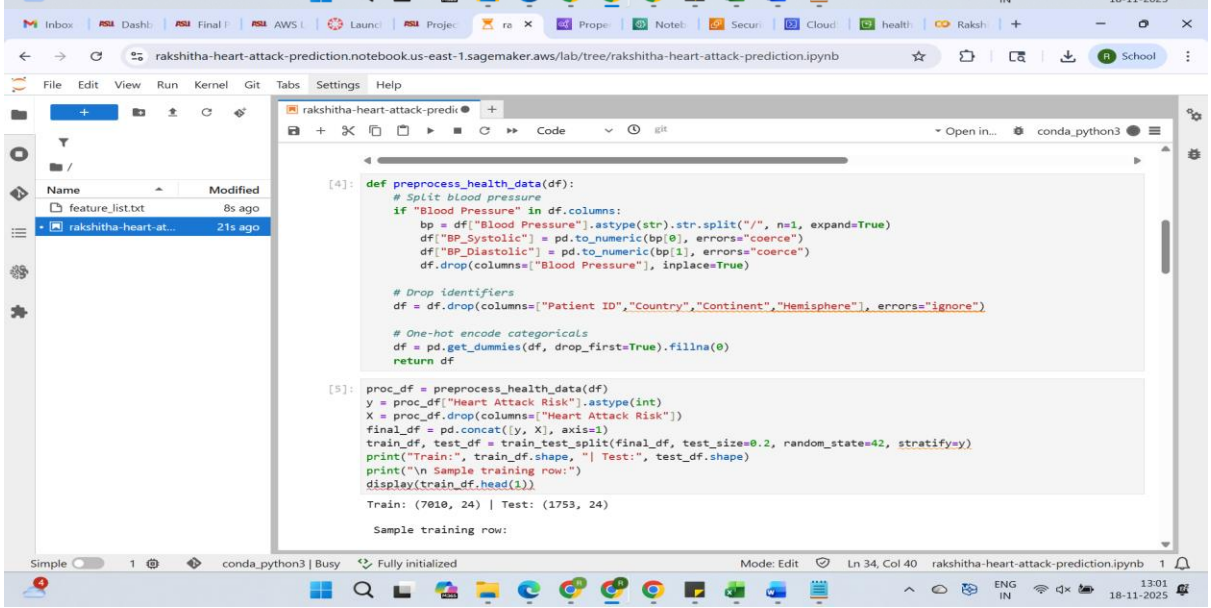


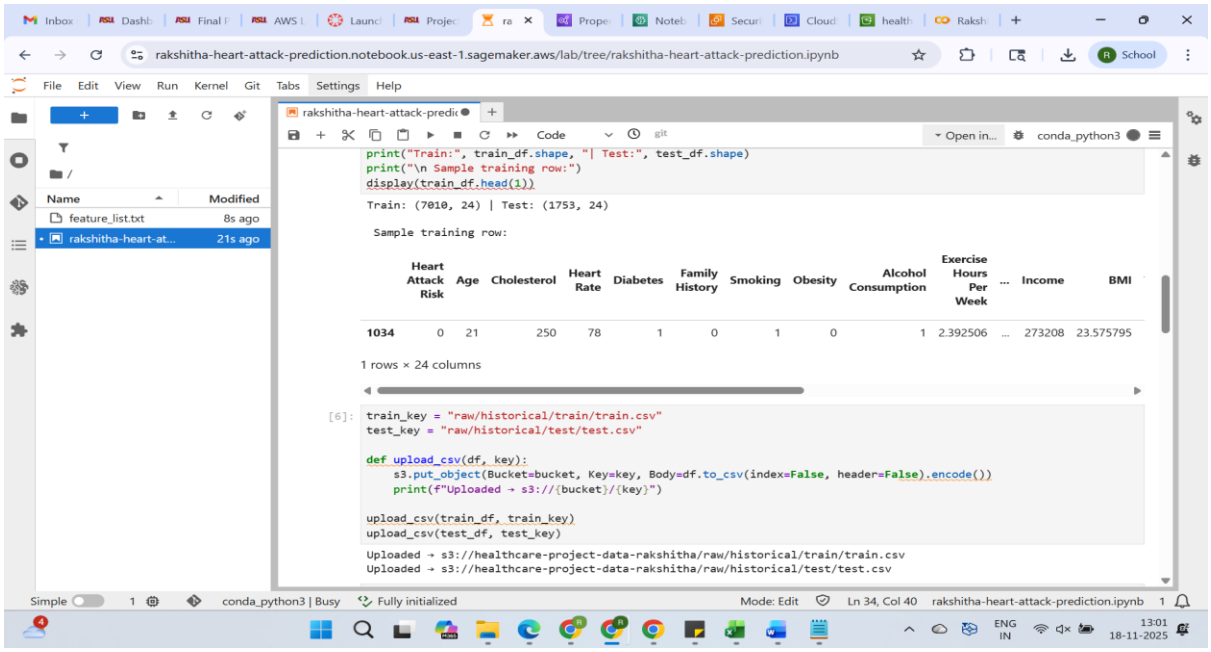
Phase 3: Train & Deploy Model in Amazon SageMaker




rakshitha-heart-attack-prediction.notebook.us-east-1.sagemaker.aws/lab/tree/rakshitha-heart-attack-prediction.ipynb
import boto3, io, pandas as pd
from sklearn.model_selection import train_test_split
import sagemaker
from sagemaker import get_execution_role
region = "us-east-1" # update if needed
bucket = "healthcare-project-data-rakshitha" # your bucket name
role = get_execution_role()
s3 = boto3.client("s3", region_name=region)
hist_key = "raw/historical/heart_attack_prediction_dataset.csv"
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
#Load the processed/merged CSV from S3 (produced by EMR) to confirm shape & columns.
obj = s3.get_object(Bucket=bucket, Key=hist_key)
df = pd.read_csv(io.BytesIO(obj["Body"].read()))
print("Loaded dataset:", df.shape)
df.head(2)
Loaded dataset: (8763, 26)
Patient ID
Age
Sex
Cholesterol
Blood Pressure
Heart Rate
Diabetes
Family History
Smoking
Obesity
Sedentary Hours Per Day
Income
BMW7812
67
Male
208
158/88
72
6.615001
261404


rakshitha-heart-attack-prediction.notebook.us-east-1.sagemaker.aws/lab/tree/rakshitha-heart-attack-prediction.ipynb
df = pd.read_csv(io.BytesIO(obj["Body"].read()))
print("Loaded dataset:", df.shape)
df.head(2)
Loaded dataset: (8763, 26)
Patient ID
Age
Sex
Cholesterol
Blood Pressure
Heart Rate
Diabetes
Family History
Smoking
Obesity
Sedentary Hours Per Day
Income
BMW7812
67
Male
208
158/88
72
6.615001
261404
CZE1114
21
Male
389
165/93
98
4.963459
285768
2 rows × 26 columns
def preprocess_health_data(df):
# Split blood pressure
if "Blood Pressure" in df.columns:
bp = df["Blood Pressure"].astype(str).str.split("/", n=1, expand=True)
df["BP_Systolic"] = pd.to_numeric(bp[0], errors="coerce")
df["BP_Diastolic"] = pd.to_numeric(bp[1], errors="coerce")
df.drop(columns=["Blood Pressure"], inplace=True)
# Drop identifiers
df = df.drop(columns=["Patient ID","Country","Continent","Hemisphere"], errors="ignore")


rakshitha-heart-attack-prediction.notebook.us-east-1.sagemaker.aws/lab/tree/rakshitha-heart-attack-prediction.ipynb
def preprocess_health_data(df):
# Split blood pressure
if "Blood Pressure" in df.columns:
bp = df["Blood Pressure"].astype(str).str.split("/", n=1, expand=True)
df["BP_Systolic"] = pd.to_numeric(bp[0], errors="coerce")
df["BP_Diastolic"] = pd.to_numeric(bp[1], errors="coerce")
df.drop(columns=["Blood Pressure"], inplace=True)
# Drop identifiers
df = df.drop(columns=["Patient ID","Country","Continent","Hemisphere"], errors="ignore")
# One-hot encode categoricals
df = pd.get_dummies(df, drop_first=True).fillna(0)
return df
proc_df = preprocess_health_data(df)
y = proc_df["Heart Attack Risk"].astype(int)
X = proc_df.drop(columns=["Heart Attack Risk"])
final_df = pd.concat([y, X], axis=1)
train_df, test_df = train_test_split(final_df, test_size=0.2, random_state=42, stratify=y)
print("Train:", train_df.shape, "| Test:", test_df.shape)
print("\n Sample training row:")
display(train_df.head(1))
Train: (7010, 24) | Test: (1753, 24)
Sample training row:



```
print("Train:", train_df.shape, "| Test:", test_df.shape)
print("\n Sample training row:")
display(train_df.head(1))
```

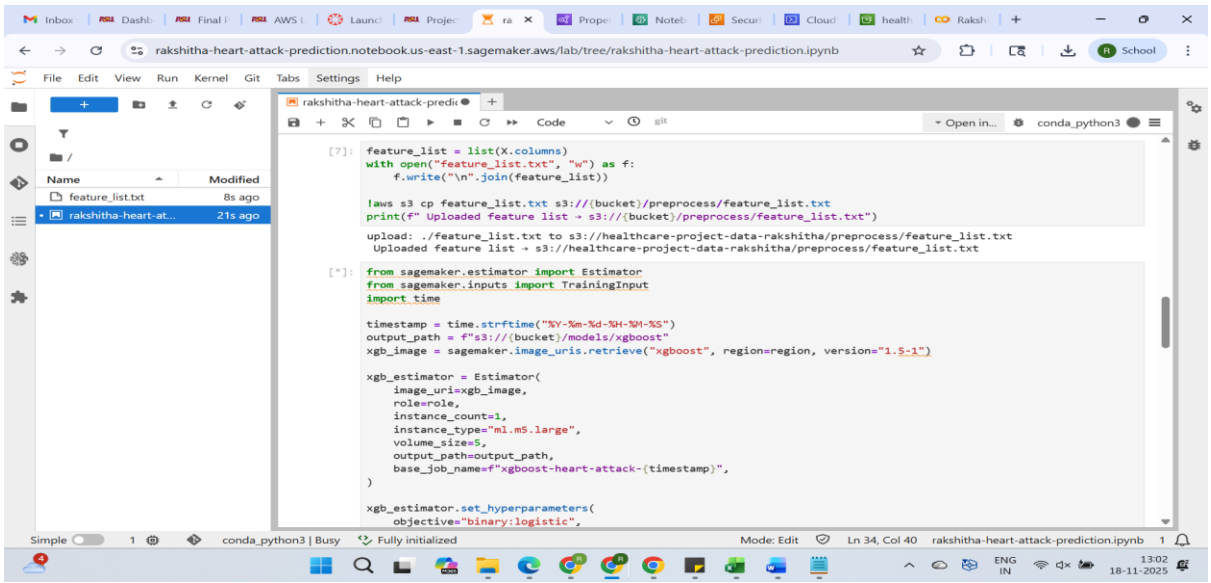
```
Train: (7010, 24) | Test: (1753, 24)

 Sample training row:
```

	Heart Attack Risk	Age	Cholesterol	Heart Rate	Diabetes	Family History	Smoking	Obesity	Alcohol Consumption	Exercise Hours Per Week	...	Income	BMI
1034	0	21	250	78	1	0	1	0	1	2.392506	...	273208	23.575795

1 rows × 24 columns

```
[6]: train_key = "raw/historical/train/train.csv"
     test_key = "raw/historical/test/test.csv"

     def upload_csv(df, key):
         s3.put_object(Bucket=bucket, Key=key, Body=df.to_csv(index=False, header=False).encode())
         print(f"Uploaded → s3://{bucket}/{key}")

     upload_csv(train_df, train_key)
     upload_csv(test_df, test_key)
```

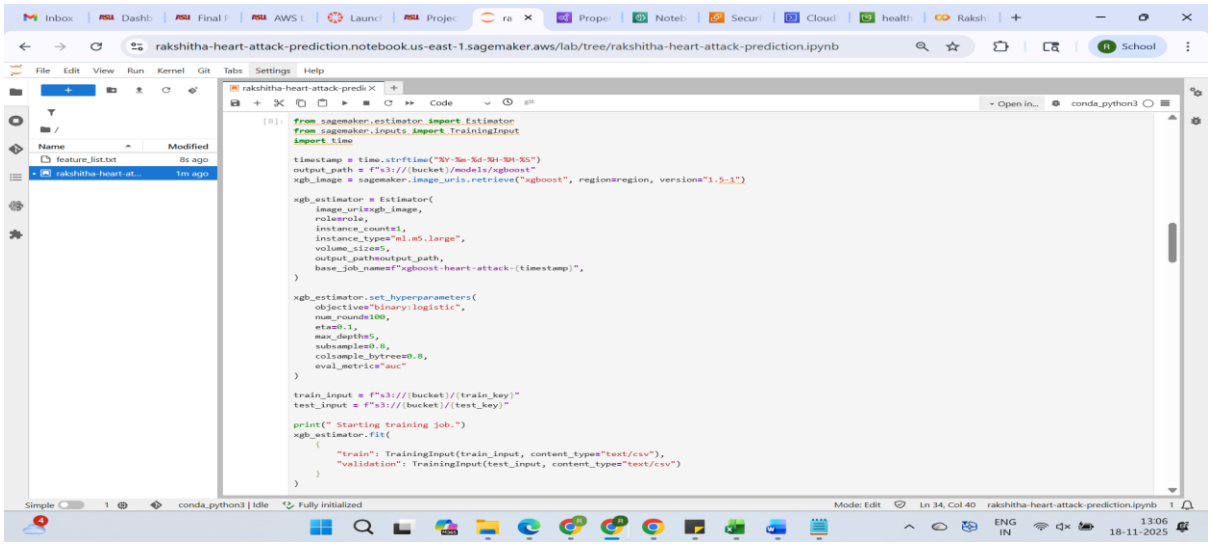
```
Uploaded → s3://healthcare-project-data-rakshitha/raw/historical/train/train.csv
Uploaded → s3://healthcare-project-data-rakshitha/raw/historical/test/test.csv
```

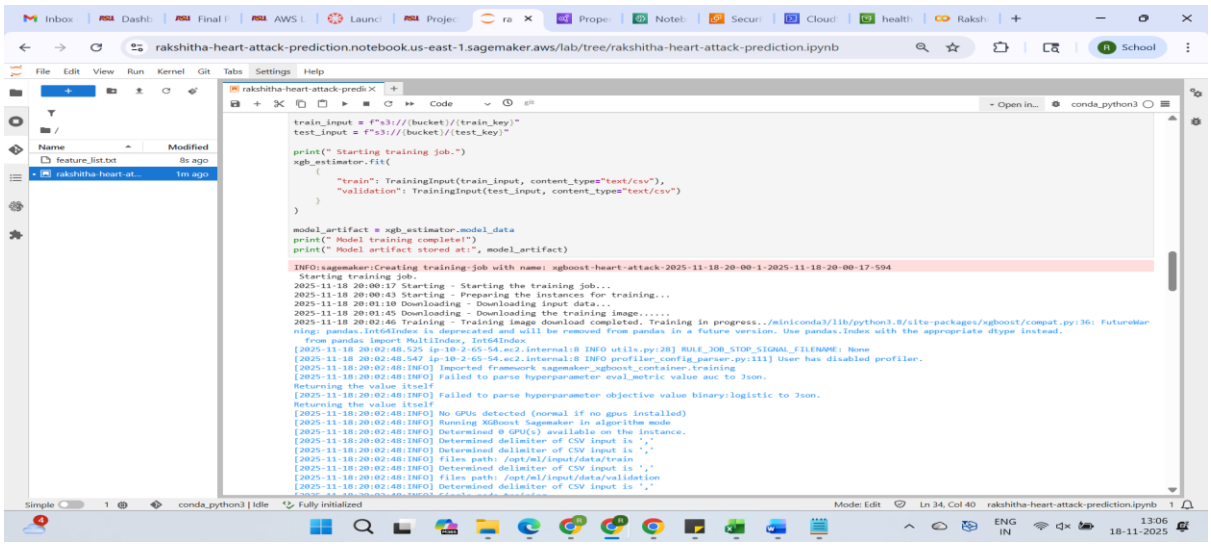


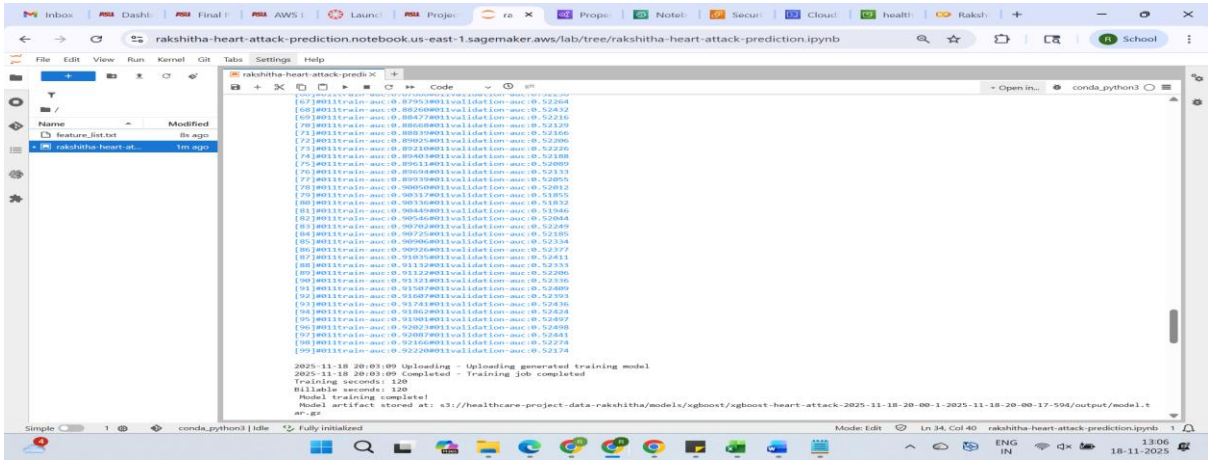
```
[7]: feature_list = list(X.columns)
     with open("feature_list.txt", "w") as f:
         f.write("\n".join(feature_list))

     !aws s3 cp feature_list.txt s3://{bucket}/preprocess/feature_list.txt
     print(f" Uploaded feature list → s3://{bucket}/preprocess/feature_list.txt")
```

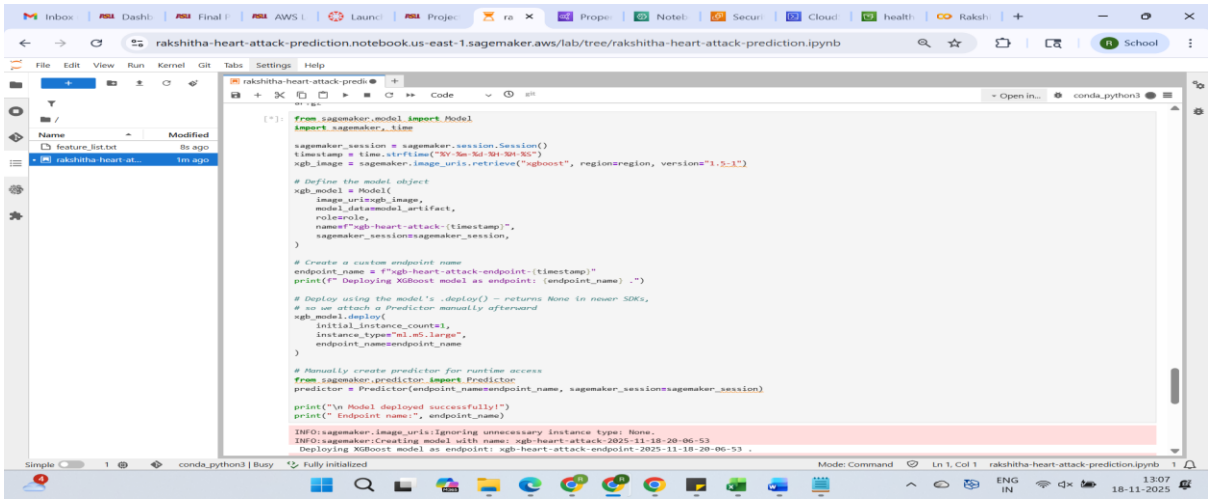
```
upload: ./feature_list.txt to s3://healthcare-project-data-rakshitha/preprocess/feature_list.txt
 Uploaded feature list → s3://healthcare-project-data-rakshitha/preprocess/feature_list.txt
```

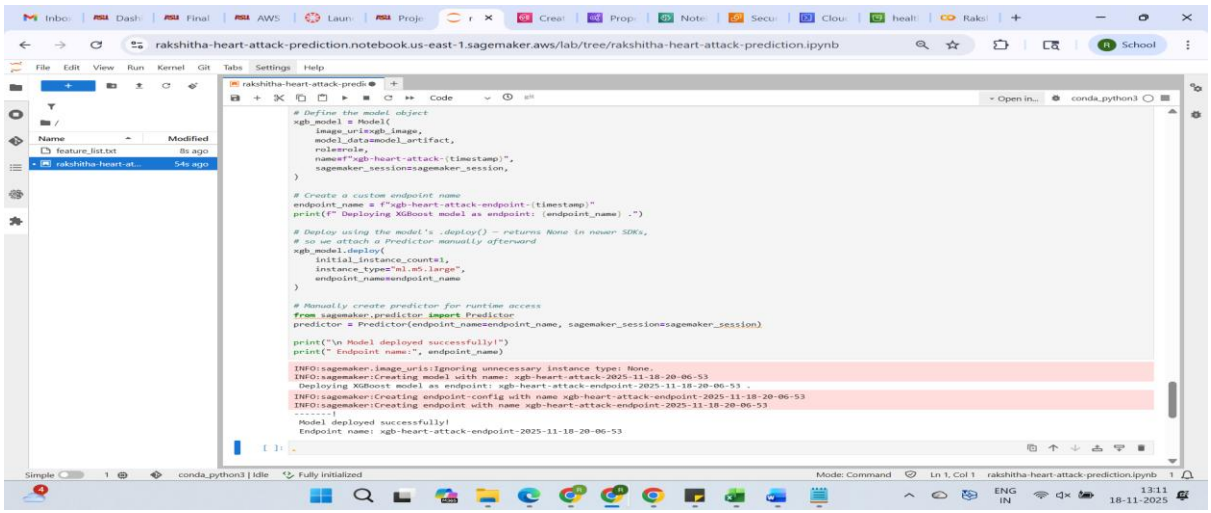
```
[*]: from sagemaker.estimator import Estimator
     from sagemaker.inputs import TrainingInput
     import time

     timestamp = time.strftime("%Y-%m-%d-%H-%M-%S")
     output_path = f"s3://{bucket}/models/xgboost"
     xgb_image = sagemaker.image_uris.retrieve("xgboost", region=region, version="1.5-1")

     xgb_estimator = Estimator(
         image_uri=xgb_image,
         role=role,
         instance_count=1,
         instance_type="ml.m5.large",
         volume_size=5,
         output_path=output_path,
         base_job_name=f"xgboost-heart-attack-{timestamp}",
     )

     xgb_estimator.set_hyperparameters(
         objective="binary:logistic",
```


rakshitha-heart-attack-prediction.notebook.us-east-1.sagemaker.aws/lab/tree/rakshitha-heart-attack-prediction.ipynb
from sagemaker.estimator import Estimator
from sagemaker.inputs import TrainingInput
import time
timestamp = time.strftime("%Y-%m-%d-%H-%M-%S")
output_path = f"s3://{bucket}/models/xgboost"
xgb_image = sagemaker.image_uris.retrieve("xgboost", region=region, version="1.5-1")
xgb_estimator = Estimator(
image_uri=xgb_image,
role=role,
instance_count=1,
instance_type="ml.m5.large",
volume_size=5,
output_path=output_path,
base_job_name=f"xgboost-heart-attack-{timestamp}",
)
xgb_estimator.set_hyperparameters(
objective="binary:logistic",
num_round=100,
eta=0.1,
max_depth=5,
subsample=0.8,
colsample_bytree=0.8,
eval_metric="auc"
)
train_input = f"s3://{bucket}/{train_key}"
test_input = f"s3://{bucket}/{test_key}"
print(" Starting training job.")
xgb_estimator.fit(
{
"train": TrainingInput(train_input, content_type="text/csv"),
"validation": TrainingInput(test_input, content_type="text/csv")
}
)


rakshitha-heart-attack-prediction.notebook.us-east-1.sagemaker.aws/lab/tree/rakshitha-heart-attack-prediction.ipynb
model_artifact = xgb_estimator.model_data
print(" Model training complete!")
print(" Model artifact stored at:", model_artifact)
INFO:sagemaker:Creating training-job with name: xgboost-heart-attack-2025-11-18-20-00-1-2025-11-18-20-00-17-594
Starting training job.
2025-11-18 20:00:17 Starting - Starting the training job...
2025-11-18 20:00:43 Starting - Preparing the instances for training...
2025-11-18 20:01:10 Downloading - Downloading input data...
2025-11-18 20:01:45 Downloading - Downloading the training image.....
2025-11-18 20:02:46 Training - Training image download completed. Training in progress..
[2025-11-18 20:02:48.525 ip-10-2-65-54.ec2.internal:8 INFO utils.py:28] RULE_JOB_STOP_SIGNAL_FILENAME: None
[2025-11-18 20:02:48.547 ip-10-2-65-54.ec2.internal:8 INFO profiler_config_parser.py:111] User has disabled profiler.
[2025-11-18 20:02:48:INFO] Imported framework sagemaker_xgboost_container.training
[2025-11-18 20:02:48:INFO] Failed to parse hyperparameter eval_metric value auc to Json.
Returning the value itself
[2025-11-18 20:02:48:INFO] Failed to parse hyperparameter objective value binary:logistic to Json.
Returning the value itself
[2025-11-18 20:02:48:INFO] No GPUs detected (normal if no gpus installed)
[2025-11-18 20:02:48:INFO] Running XGBoost Sagemaker in algorithm mode
[2025-11-18 20:02:48:INFO] Determined 0 GPU(s) available on the instance.
[2025-11-18 20:02:48:INFO] Determined delimiter of CSV input is ','
[2025-11-18 20:02:48:INFO] Determined delimiter of CSV input is ','
[2025-11-18 20:02:48:INFO] files path: /opt/ml/input/data/train
[2025-11-18 20:02:48:INFO] Determined delimiter of CSV input is ','
[2025-11-18 20:02:48:INFO] files path: /opt/ml/input/data/validation
[2025-11-18 20:02:48:INFO] Determined delimiter of CSV input is ','


rakshitha-heart-attack-prediction.notebook.us-east-1.sagemaker.aws/lab/tree/rakshitha-heart-attack-prediction.ipynb
[67]#011train-auc:0.87953#011validation-auc:0.52264
[68]#011train-auc:0.88260#011validation-auc:0.52432
[69]#011train-auc:0.88477#011validation-auc:0.52216
[70]#011train-auc:0.88668#011validation-auc:0.52329
[71]#011train-auc:0.88839#011validation-auc:0.52166
[72]#011train-auc:0.89025#011validation-auc:0.52206
[73]#011train-auc:0.89213#011validation-auc:0.52226
[74]#011train-auc:0.89403#011validation-auc:0.52188
[75]#011train-auc:0.89611#011validation-auc:0.52089
[76]#011train-auc:0.89694#011validation-auc:0.52133
[77]#011train-auc:0.89939#011validation-auc:0.52055
[78]#011train-auc:0.90050#011validation-auc:0.52012
[79]#011train-auc:0.90317#011validation-auc:0.51855
[80]#011train-auc:0.90336#011validation-auc:0.51832
[81]#011train-auc:0.90445#011validation-auc:0.51946
[82]#011train-auc:0.90546#011validation-auc:0.52044
[83]#011train-auc:0.90702#011validation-auc:0.52249
[84]#011train-auc:0.90725#011validation-auc:0.52185
[85]#011train-auc:0.90906#011validation-auc:0.52334
[86]#011train-auc:0.90926#011validation-auc:0.52377
[87]#011train-auc:0.91035#011validation-auc:0.52411
[88]#011train-auc:0.91132#011validation-auc:0.52353
[89]#011train-auc:0.91122#011validation-auc:0.52206
[90]#011train-auc:0.91218#011validation-auc:0.52336
[91]#011train-auc:0.91507#011validation-auc:0.52489
[92]#011train-auc:0.91607#011validation-auc:0.52393
[93]#011train-auc:0.91741#011validation-auc:0.52436
[94]#011train-auc:0.91862#011validation-auc:0.52424
[95]#011train-auc:0.91901#011validation-auc:0.52497
[96]#011train-auc:0.92023#011validation-auc:0.52498
[97]#011train-auc:0.92087#011validation-auc:0.52441
[98]#011train-auc:0.92166#011validation-auc:0.52274
[99]#011train-auc:0.92220#011validation-auc:0.52174
2025-11-18 20:03:09 Uploading - Uploading generated training model
2025-11-18 20:03:09 Completed - Training job completed
Training seconds: 120
Billable seconds: 120
Model training complete!
Model artifact stored at: s3://healthcare-project-data-rakshitha/models/xgboost/xgboost-heart-attack-2025-11-18-20-00-1-2025-11-18-20-00-17-594/output/model.tar.gz





Phase 4: Set Up Automated Alerts Using Lambda + SNS

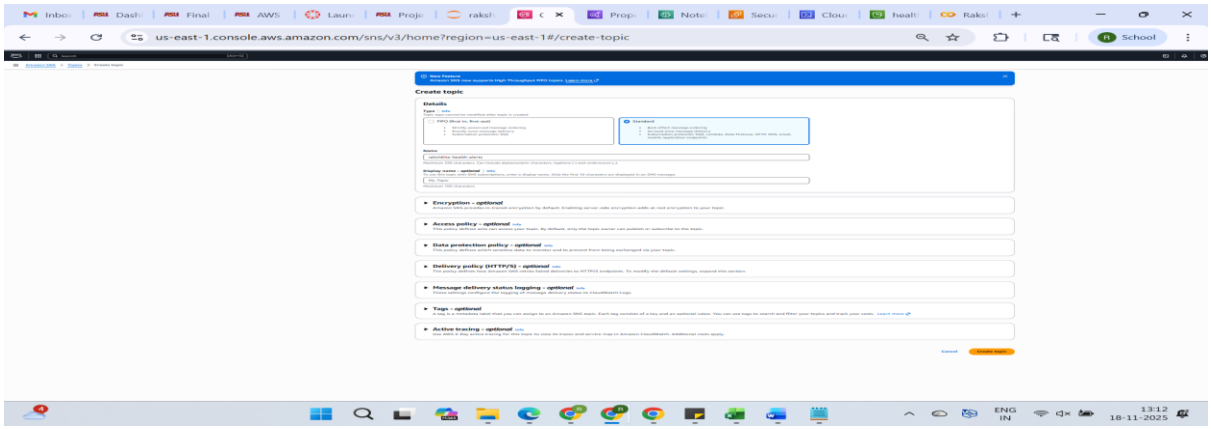
CODE:

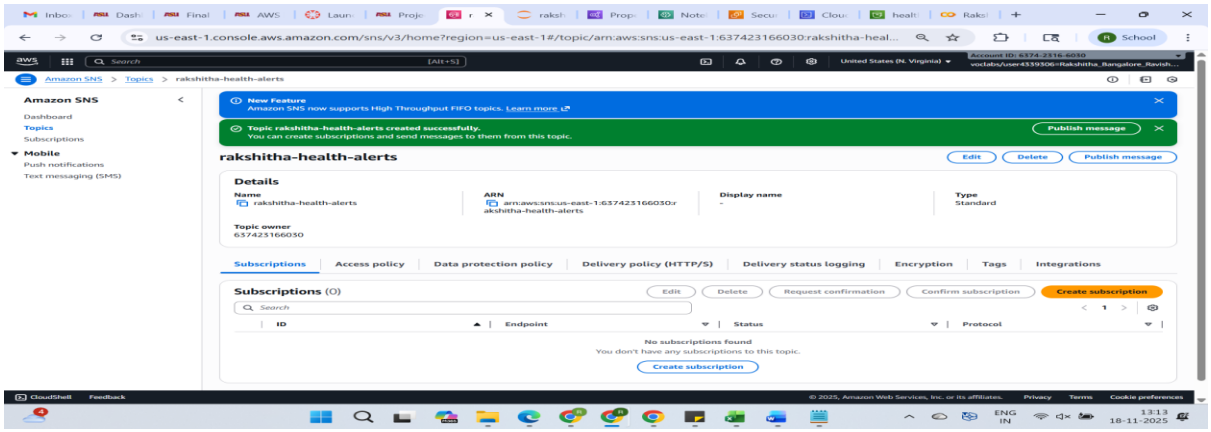
```
import boto3
import csv
import io
import pandas as pd
import json
from datetime import datetime

runtime = boto3.client("sagemaker-runtime")
s3 = boto3.client("s3")
sns = boto3.client("sns")

ENDPOINT_NAME = "xgb-heart-attack-endpoint-2025-10-29-01-07-38"  # Replace
BUCKET_NAME = "healthcare-project-data-rakshitha"
PROCESSED_PREFIX = "processed/final_health_dataset_csv/"
FEATURE_LIST_KEY = "preprocess/feature_list.txt"
```

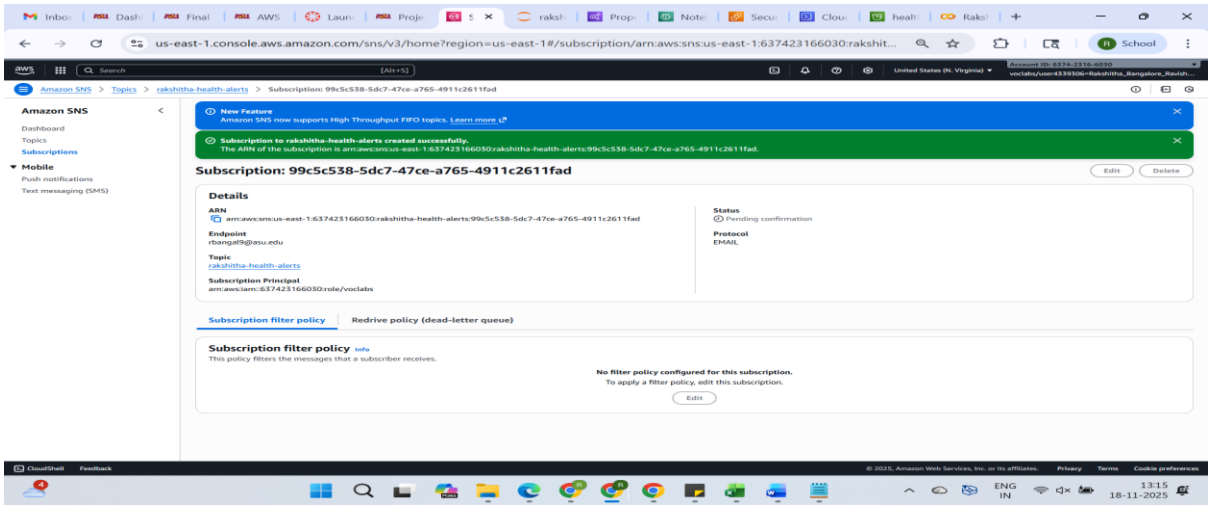
```
ALERT_TOPIC_ARN = "arn:aws:sns:us-east-1:445152320876:rakshitha-health-alerts"  # Replace

def load_feature_list():
    """Load expected feature list from S3."""
    obj = s3.get_object(Bucket=BUCKET_NAME, Key=FEATURE_LIST_KEY)
    features = obj["Body"].read().decode("utf-8").splitlines()
    features = [f.strip() for f in features if f.strip()]
    if "Heart Attack Risk" in features:
        features.remove("Heart Attack Risk")
    return features

def preprocess_row(df_row, expected_features):
    """Preprocess one row and align with training feature set."""
    df = pd.DataFrame([df_row])

    # Split Blood Pressure
    if "Blood Pressure" in df.columns:
        bp = df["Blood Pressure"].astype(str).str.split("/", n=1, expand=True)
        df["BP_Systolic"] = pd.to_numeric(bp[0], errors="coerce")
        df["BP_Diastolic"] = pd.to_numeric(bp[1], errors="coerce")
        df.drop(columns=["Blood Pressure"], inplace=True)

    # Drop non-feature columns
    df = df.drop(columns=["Patient ID", "Country", "Continent", "Hemisphere"], errors="ignore")

    # Manual encoding for categorical columns
    df["Sex_Male"] = (df["Sex"].astype(str).str.lower() == "male").astype(int)
    df["Diet_Healthy"] = (df["Diet"].astype(str).str.lower() == "healthy").astype(int)
    df["Diet_Unhealthy"] = (df["Diet"].astype(str).str.lower() == "unhealthy").astype(int)

    # Drop originals
    df = df.drop(columns=["Sex", "Diet"], errors="ignore")

    # Convert numeric & fill missing
    df = df.apply(pd.to_numeric, errors="coerce").fillna(0)

    # Align with expected features
    for col in expected_features:
        if col not in df.columns:
            df[col] = 0
    df = df[expected_features]

    return df

def lambda_handler(event, context):
    print("Lambda invoked. Checking for processed CSV file in S3...")
    expected_features = load_feature_list()
    print(f"Loaded {len(expected_features)} expected features from training.")

    # Get latest processed CSV
    objects = s3.list_objects_v2(Bucket=BUCKET_NAME, Prefix=PROCESSED_PREFIX)
    csv_files = [obj["Key"] for obj in objects.get("Contents", []) if
```

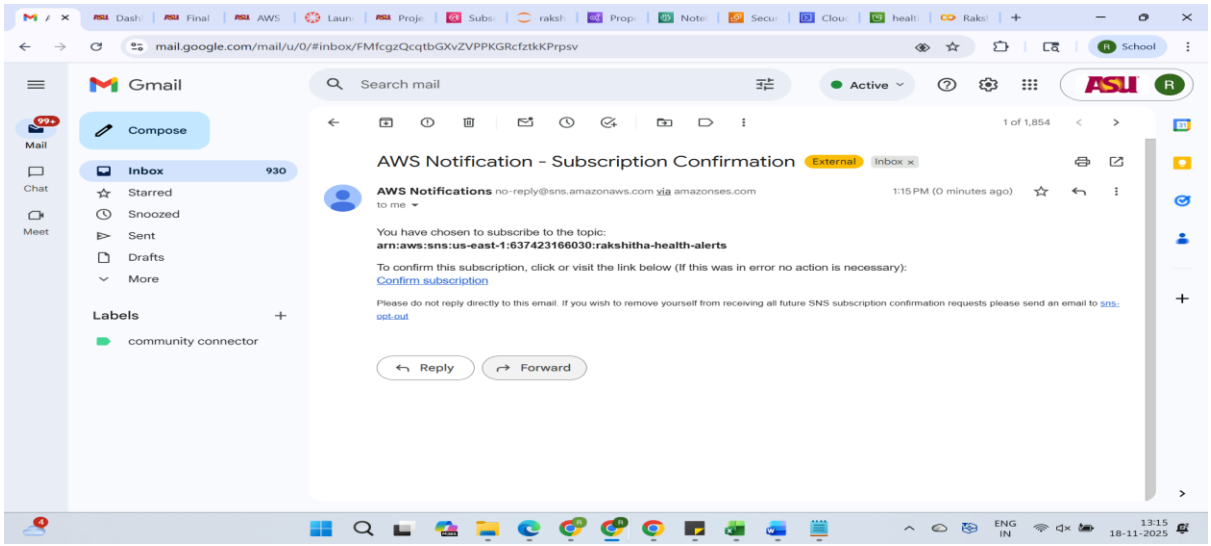
```
obj["Key"].endswith(".csv")]
    if not csv_files:
        return {"statusCode": 404, "body": json.dumps({"error": "No CSV file found."})}

    csv_key = csv_files[0]
    print(f"Using file: {csv_key}")

    # Read CSV
    obj = s3.get_object(Bucket=BUCKET_NAME, Key=csv_key)
    data = obj["Body"].read().decode("utf-8").splitlines()
    reader = csv.DictReader(data)
    rows = list(reader)
    print(f" Columns detected ({len(reader.fieldnames)}): {reader.fieldnames}")

    results, alerts, alert_details = [], 0, []
    predictions = []

    for i, row in enumerate(rows[:50]):  # process up to 50 rows
        patient_id = row.get("Patient ID", f"Row{i+1}")
        df_row = preprocess_row(row, expected_features)
        csv_payload = df_row.to_csv(index=False, header=False)

        response = runtime.invoke_endpoint(
            EndpointName=ENDPOINT_NAME,
            ContentType="text/csv",
            Body=csv_payload
        )
        raw_result = response["Body"].read().decode("utf-8").strip()

        if raw_result:
            score = float(raw_result)
            results.append(score)
            print(f"Risk score for {patient_id}: {score:.3f}")

            predictions.append({
                "Patient ID": patient_id,
                "Heart Attack Risk": round(score, 6),
                "Risk_Status": "HIGH_RISK" if score > 0.45 else "LOW_RISK",
                "ScoredAt": datetime.utcnow().isoformat() + "Z"
            })

            if score > 0.45:  # Alert threshold
                alerts += 1
                alert_details.append({
                    "patient_id": patient_id,
                    "risk_score": round(score, 3),
                    "status": "HIGH_RISK"
                })
                print(f"High-risk alert triggered for {patient_id} (score={score:.3f})")

    # Save predictions summary to S3 (without touching original data)
    timestamp = datetime.utcnow().strftime("%Y%m%d_%H%M%S")
    pred_key = f"predictions/heart_attack_predictions_{timestamp}.csv"
```

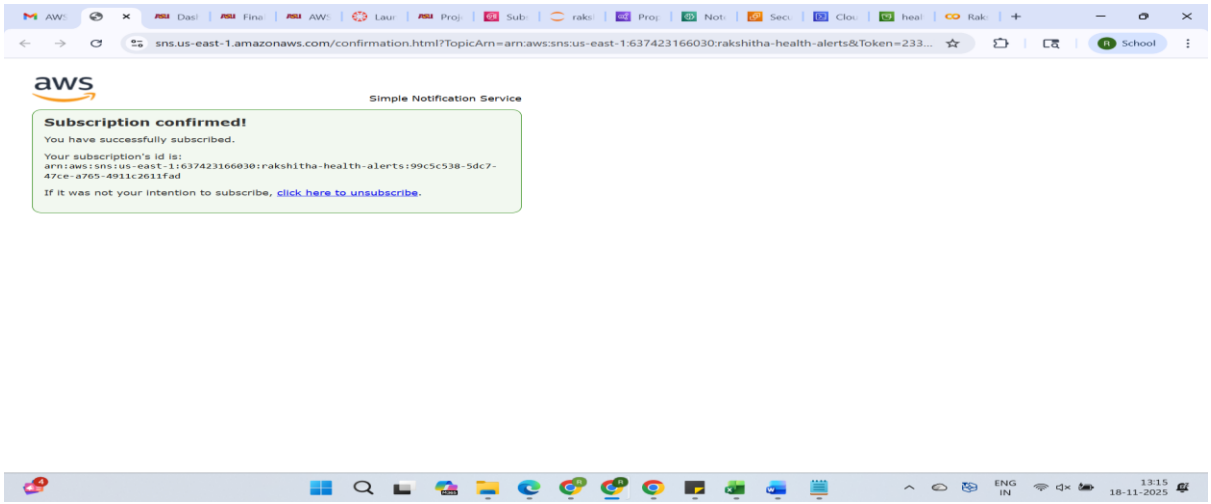
```
    if predictions:
        df_pred = pd.DataFrame(predictions)
        csv_buf = io.StringIO()
        df_pred.to_csv(csv_buf, index=False)
        s3.put_object(Bucket=BUCKET_NAME, Key=pred_key,
Body=csv_buf.getvalue().encode("utf-8"))
        print(f"Predictions saved to s3://{BUCKET_NAME}/{pred_key}")

    # SNS notification
    if alerts > 0:
        try:
            msg = "\n".join([f"{a['patient_id']}: {a['risk_score']}" for a in alert_details])
            sns.publish(
                TopicArn=ALERT_TOPIC_ARN,
                Message=f" {alerts} High-Risk Patients Detected:\n{msg}",
                Subject=" Heart Health Alert Triggered"
            )
            print(" SNS notification sent.")
        except Exception as e:
            print(f"SNS notification failed: {e}")

    print(f"Processed {len(rows)} rows, triggered {alerts} alerts.")

    return {
        "statusCode": 200,
        "body": json.dumps({
            "processed_rows": len(rows),
            "alerts_triggered": alerts,
            "alert_details": alert_details,
            "all_scores": results
        })
    }
```

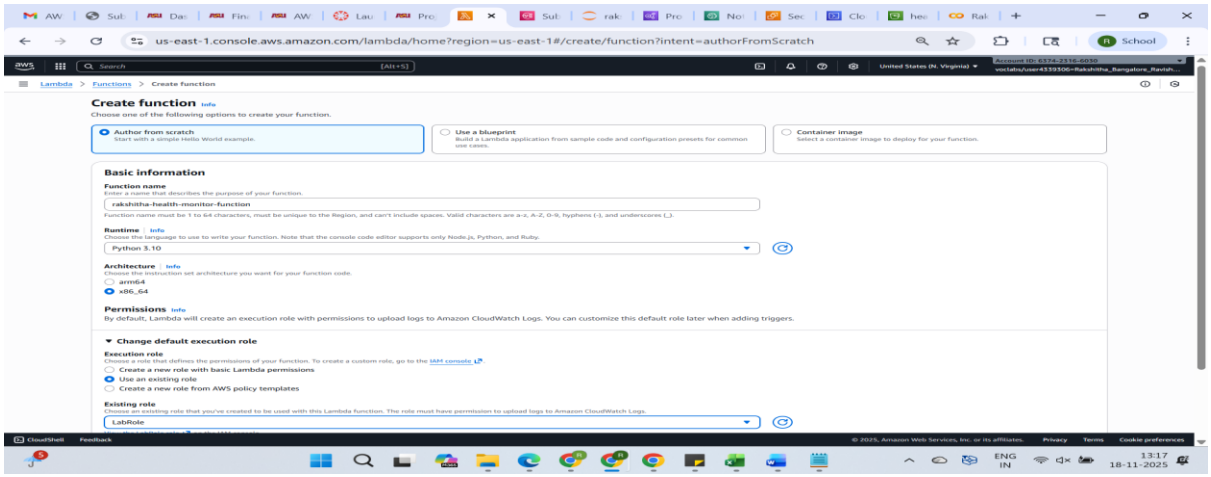


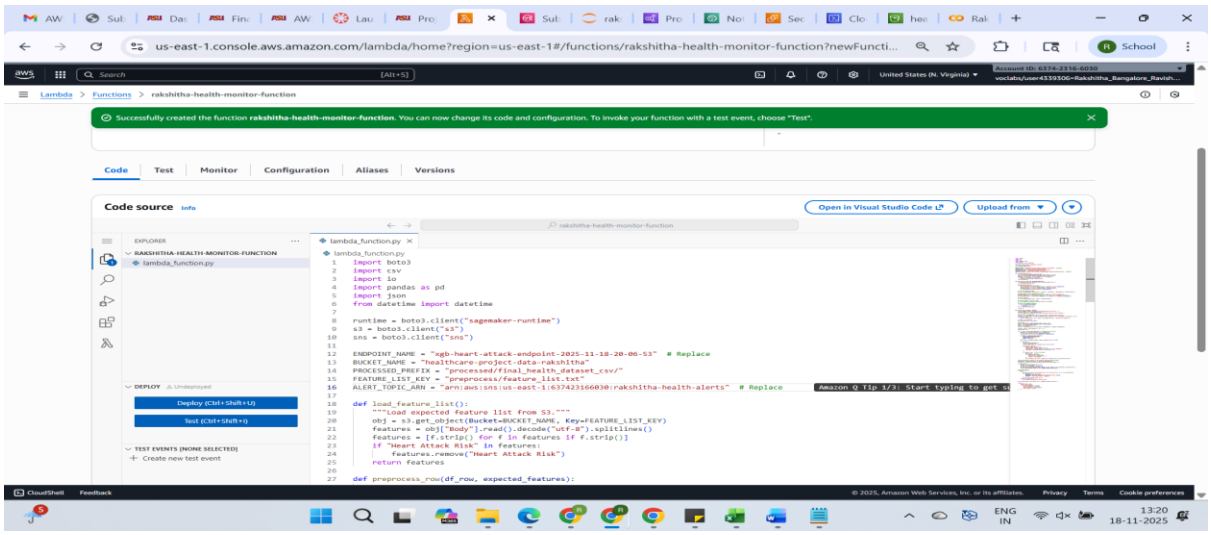

Amazon SNS
rakshitha-health-alerts created successfully.
rakshitha-health-alerts
Details
Subscriptions (0)
No subscriptions found

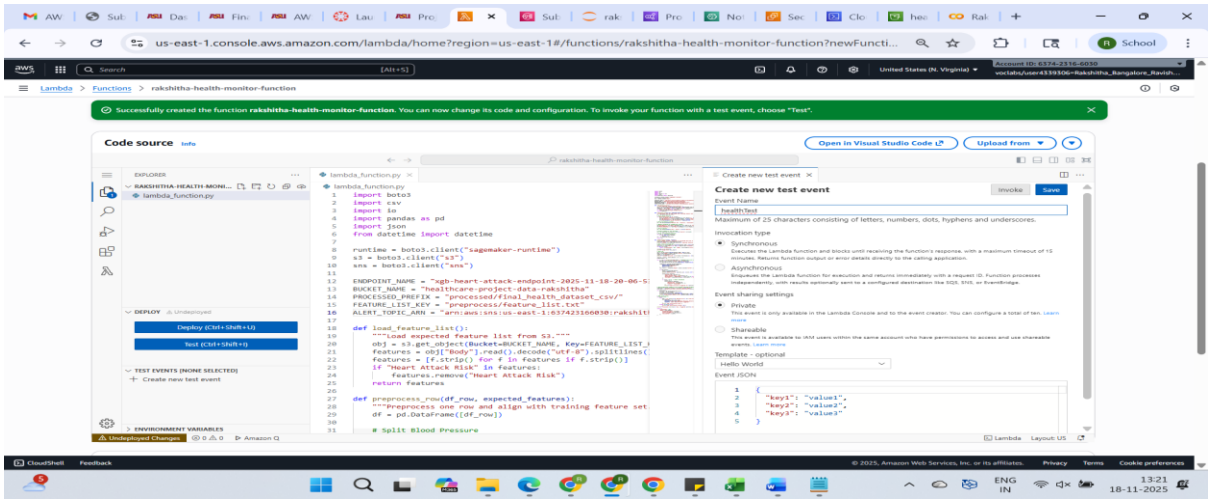

Subscription to rakshitha-health-alerts created successfully.
Subscription: 99c5c538-5dc7-47ce-a765-4911c2611fad
Subscription filter policy
No filter policy configured for this subscription.

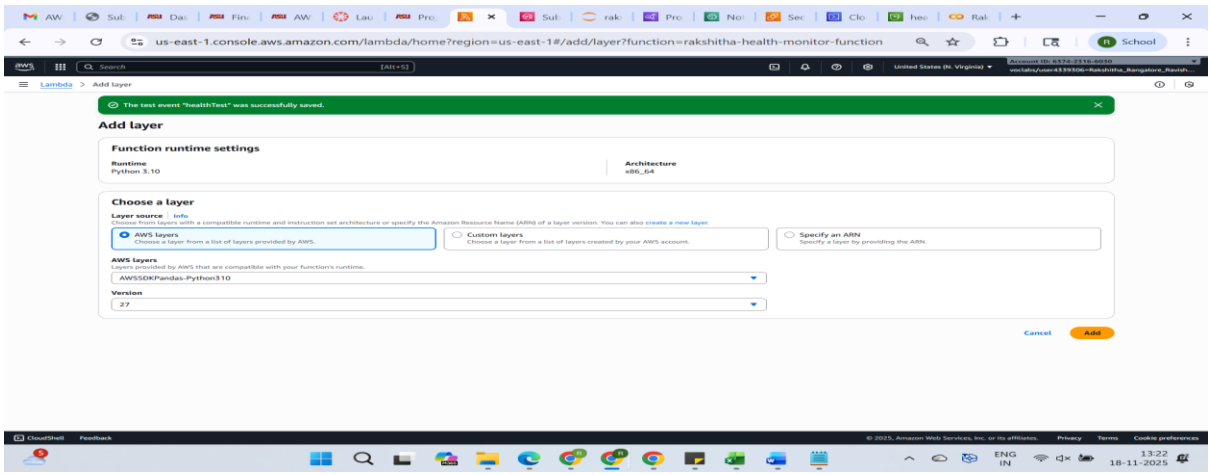

AWS Notification - Subscription Confirmation
AWS Notifications no-reply@sns.amazonaws.com via amazonses.com
You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:637423166030:rakshitha-health-alerts
Confirm subscription
Reply
Forward

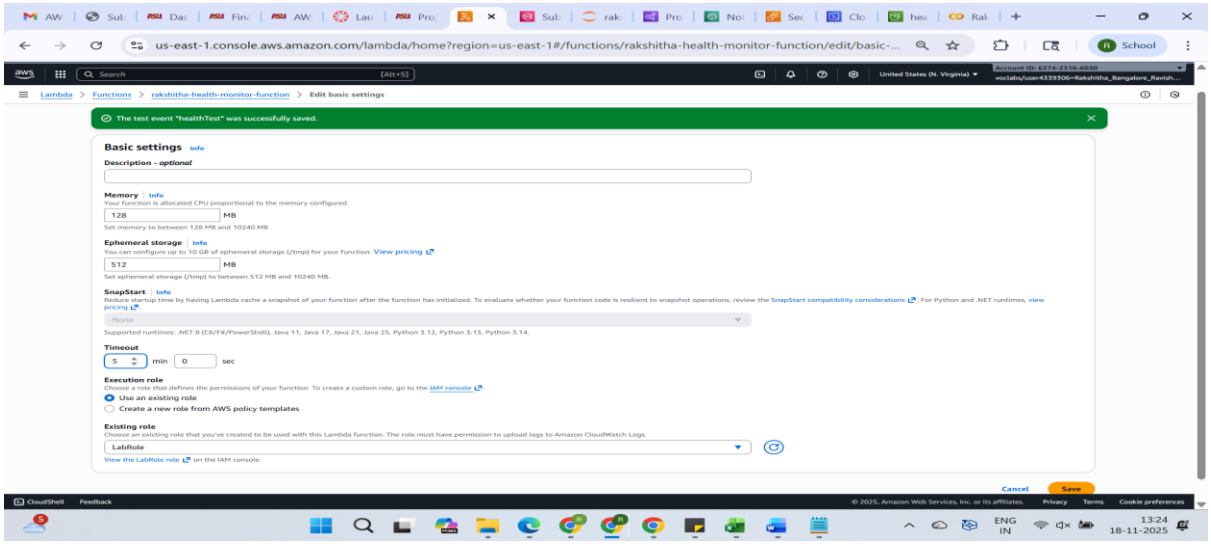

sns.us-east-1.amazonaws.com/confirmation.html?TopicArn=arn:aws:sns:us-east-1:637423166030:rakshitha-health-alerts&Token=233...
aws
Simple Notification Service
Subscription confirmed!
You have successfully subscribed.
Your subscription's id is:
arn:aws:sns:us-east-1:637423166030:rakshitha-health-alerts:99c5c538-5dc7-47ce-a765-4911c2611fad
If it was not your intention to subscribe, click here to unsubscribe.

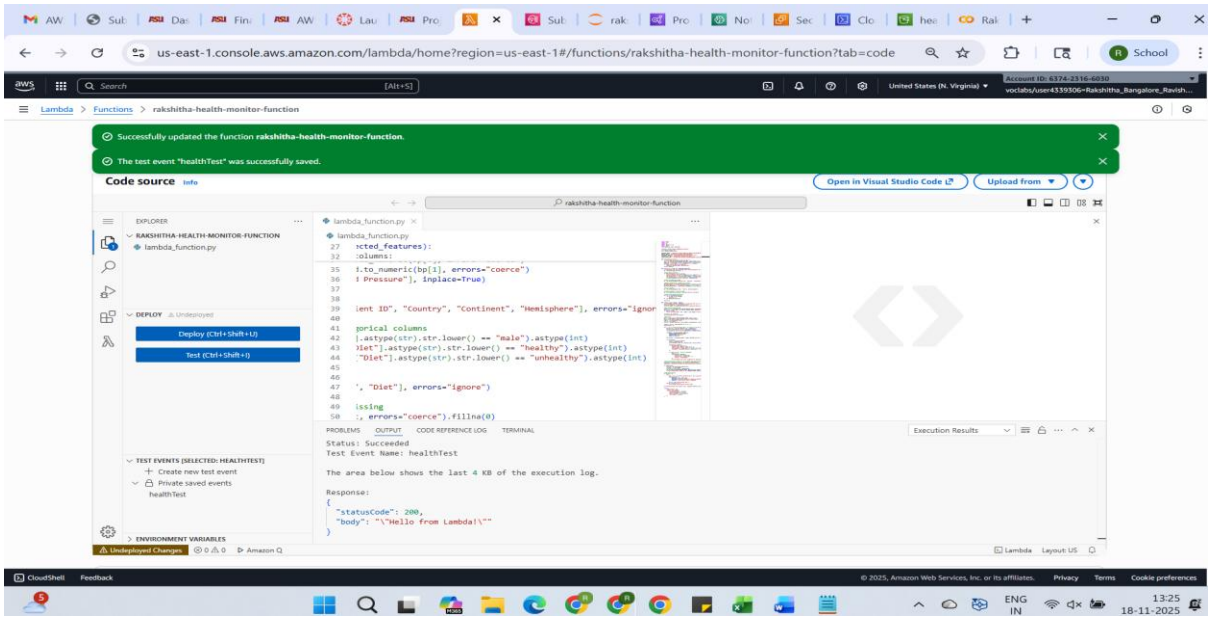

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/function?intent=authorFromScratch
Create function
Author from scratch
Use a blueprint
Container image
Basic information
Function name
rakshitha-health-monitor-function
Runtime
Python 3.10
Architecture
arm64
x86_64
Permissions
Change default execution role
Execution role
Use an existing role
Existing role
LabRole

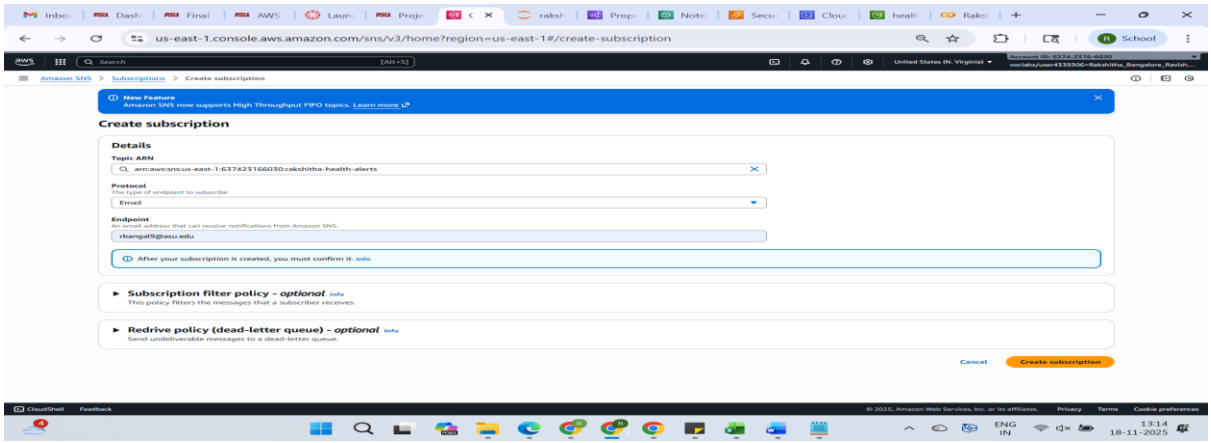

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/rakshitha-health-monitor-function?newFuncti...
Successfully created the function rakshitha-health-monitor-function. You can now change its code and configuration. To invoke your function with a test event, choose "Test".
Code
Test
Monitor
Configuration
Aliases
Versions
Code source
Open in Visual Studio Code
Upload from
lambda_function.py
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)

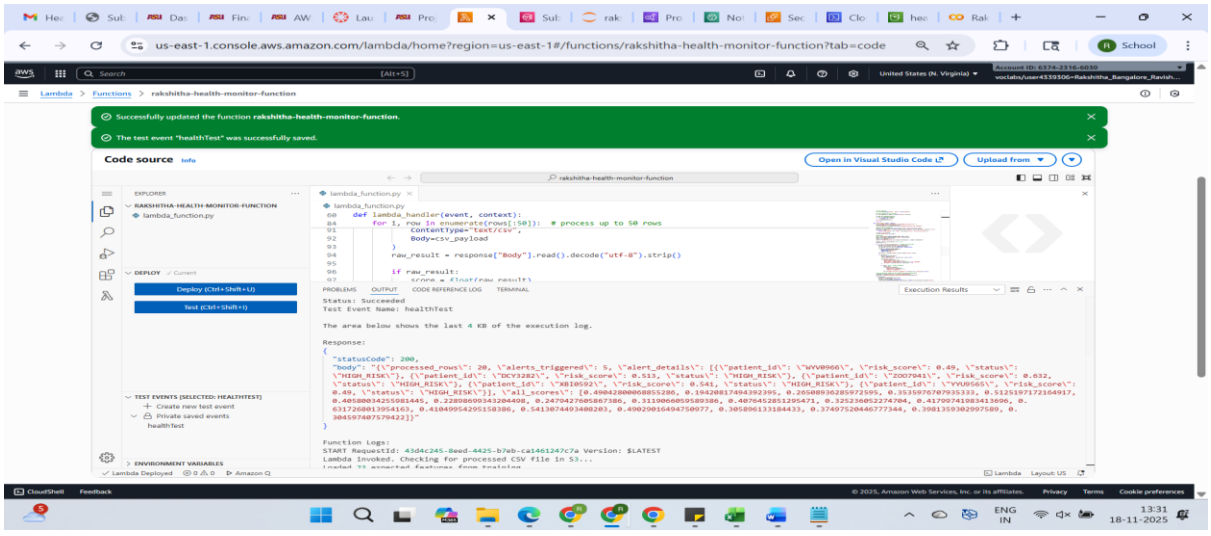

Code source
Open in Visual Studio Code
Upload from
Create new test event
Create new test event
Event Name
healthTest
Event JSON
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)

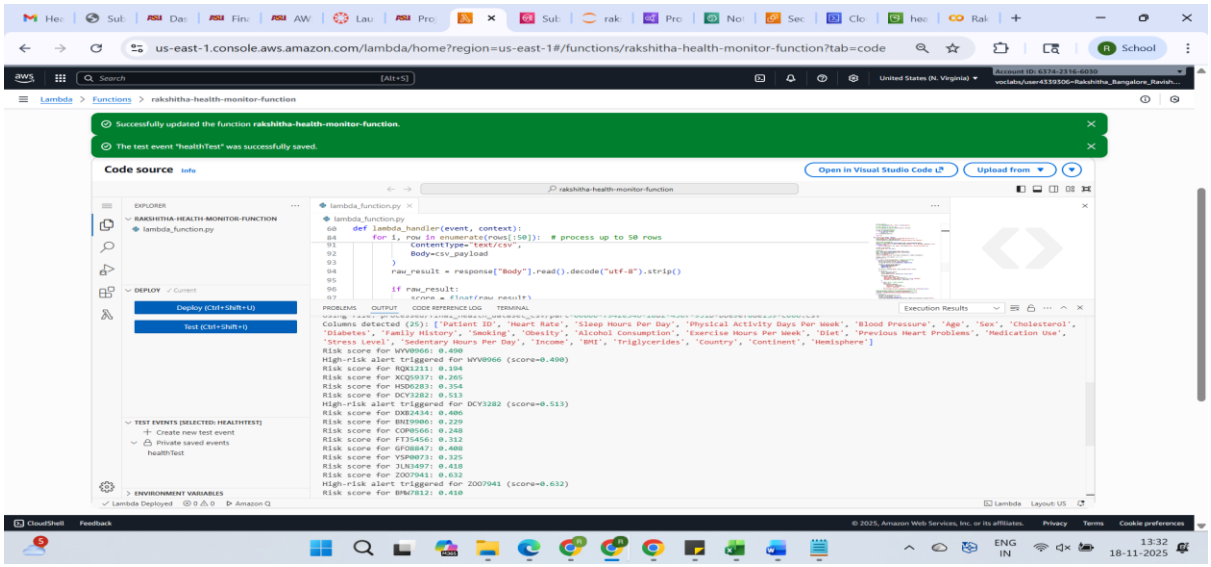

The test event "healthTest" was successfully saved.
Add layer
Function runtime settings
Runtime
Python 3.10
Architecture
x86_64
Choose a layer
Layer source
AWS layers
Custom layers
Specify an ARN
AWS layers
AWSSDKPandas-Python310
Version
27
Cancel
Add

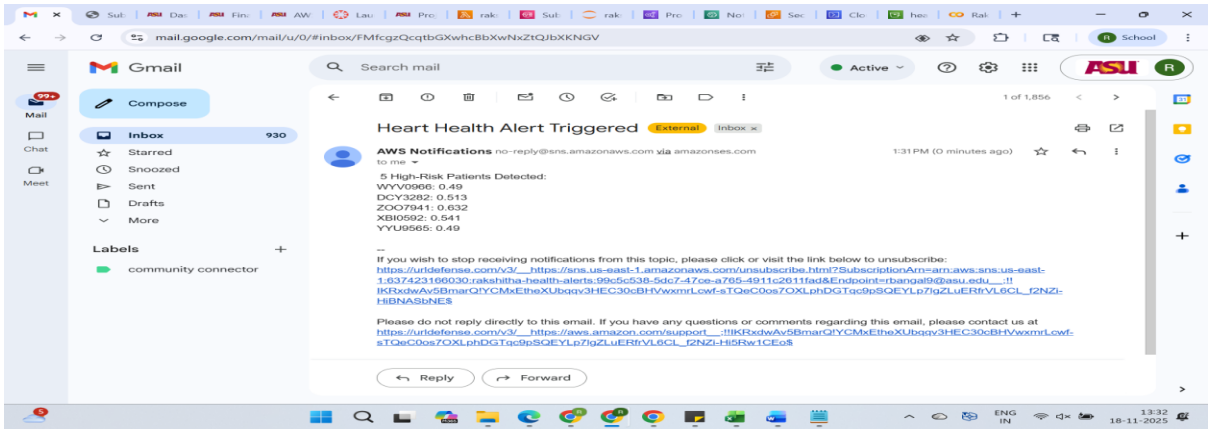

The test event "healthTest" was successfully saved.
Basic settings
Description - optional
Memory
128
MB
Ephemeral storage
512
MB
SnapStart
Timeout
Execution role
Use an existing role
Create a new role from AWS policy templates
Existing role
LabRole
Cancel
Save

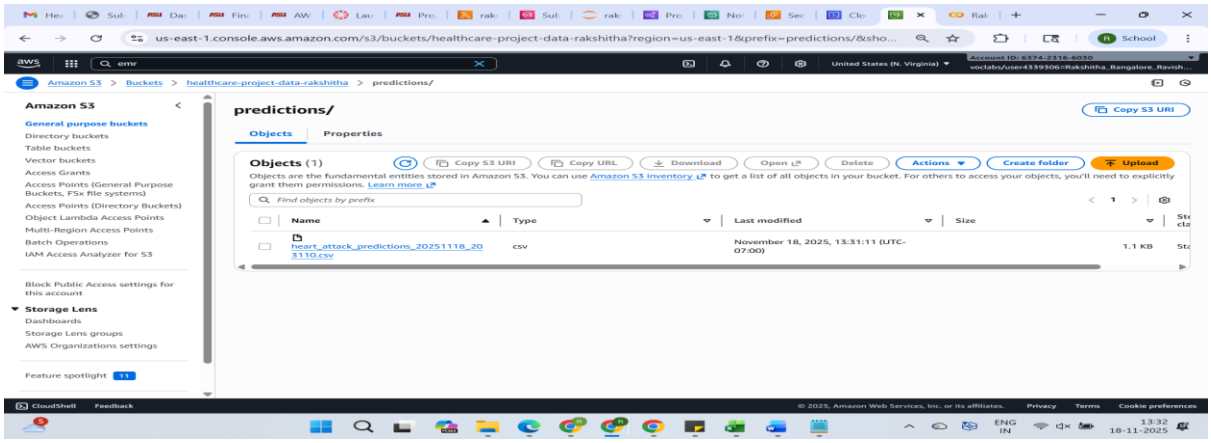

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/rakshitha-health-monitor-function?tab=code
Successfully updated the function rakshitha-health-monitor-function.
The test event "healthTest" was successfully saved.
Code source
Open in Visual Studio Code
Upload from
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
Status: Succeeded
Test Event Name: healthTest
The area below shows the last 4 KB of the execution log.
Response:
{
"statusCode": 200,
"body": "\"Hello from Lambda!\""
}


us-east-1.console.aws.amazon.com/sns/v3/home?region=us-east-1#/create-subscription
Create subscription
Details
Topic ARN
Protocol
Email
Endpoint
rbangal9@asu.edu
Subscription filter policy - optional
Redrive policy (dead-letter queue) - optional
Cancel
Create subscription

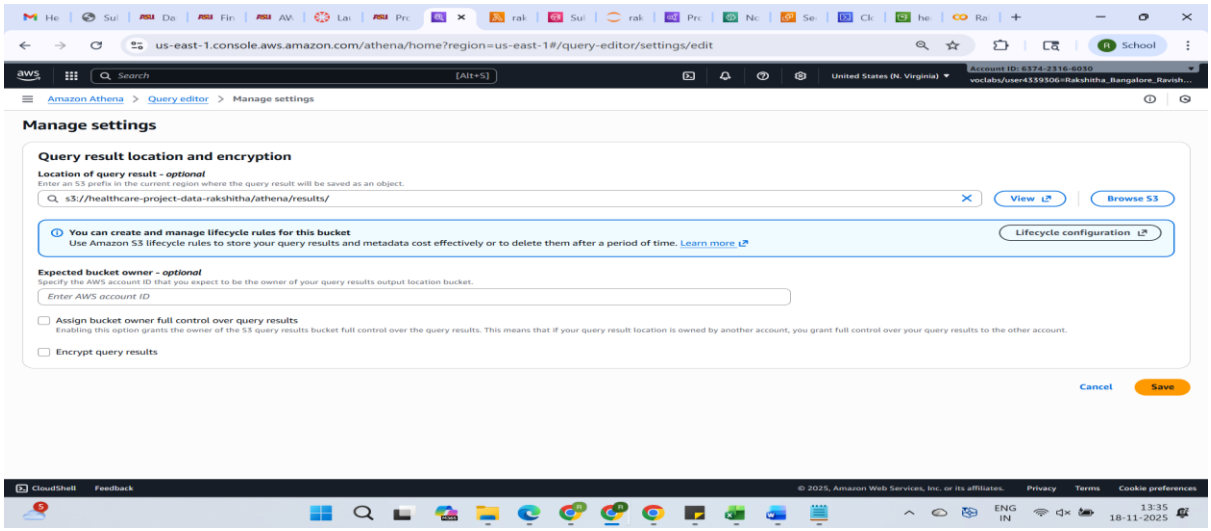

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/rakshitha-health-monitor-function?tab=code
Successfully updated the function rakshitha-health-monitor-function.
The test event "healthTest" was successfully saved.
Code source
Open in Visual Studio Code
Upload from
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
Status: Succeeded
Test Event Name: healthTest
The area below shows the last 4 KB of the execution log.
Response:
{
"statusCode": 200,
"body": "{\"processed_rows\": 20, \"alerts_triggered\": 5, \"alert_details\": [{\"patient_id\": \"WYV0966\", \"risk_score\": 0.49, \"status\": \"HIGH_RISK\"}, {\"patient_id\": \"DCY3282\", \"risk_score\": 0.513, \"status\": \"HIGH_RISK\"}, {\"patient_id\": \"ZOO7941\", \"risk_score\": 0.632, \"status\": \"HIGH_RISK\"}, {\"patient_id\": \"XBI0592\", \"risk_score\": 0.541, \"status\": \"HIGH_RISK\"}, {\"patient_id\": \"YVU9565\", \"risk_score\": 0.49, \"status\": \"HIGH_RISK\"}], \"all_scores\": [0.4904280000885286, 0.19420017494392395, 0.2650893628972595, 0.3535976707935333, 0.5125197172164917, 0.4058003425598144 5, 0.2280869934204408, 0.24794276058673 86, 0.3119066059589386, 0.4076452851205471, 0.3252360522747044, 0.41709741985413606, 0.6317260013954163, 0.4104995429515 8386, 0.5413074493408203, 0.4902901649475 0977, 0.30589613318 4433, 0.3749752044677344, 0.3981359302997589, 0.3045974075794223]}"
}
Function Logs:
START RequestId: 43d4c245-8eed-4425-b7eb-ca1461247c7a Version: $LATEST
Lambda Invoked. Checking for processed CSV file in S3...

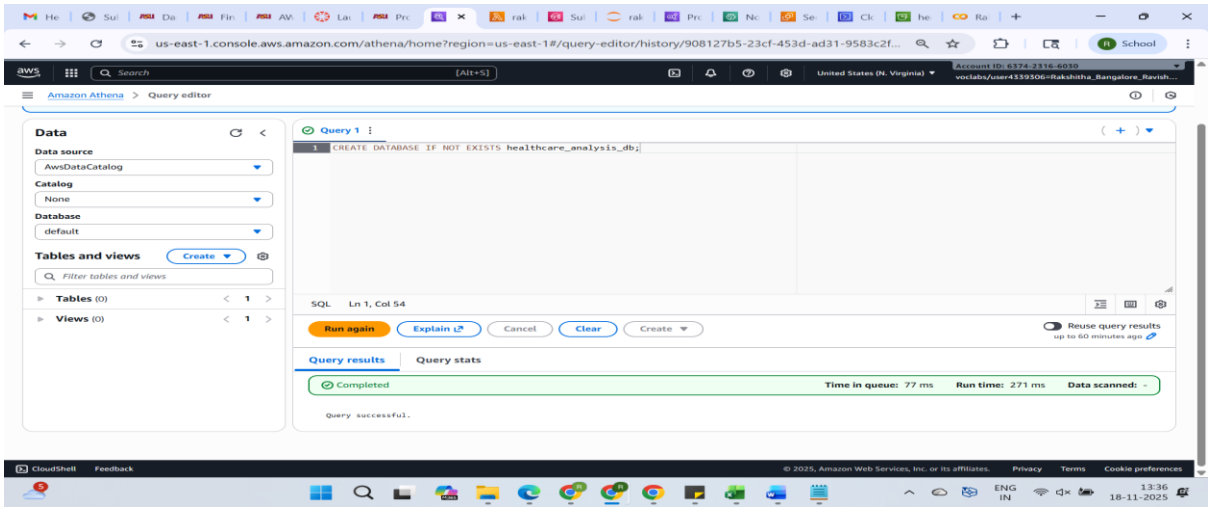






Phase 5: Perform Data Analytics in Amazon Athena

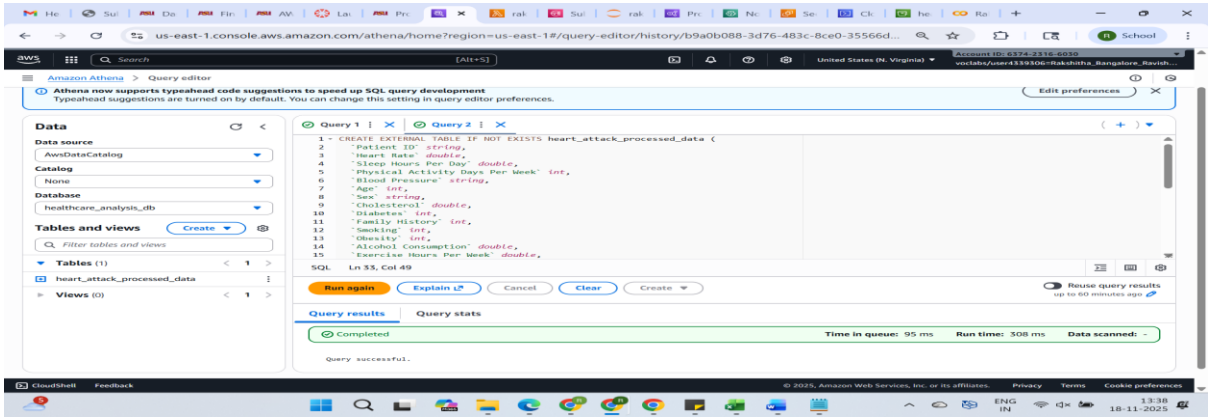


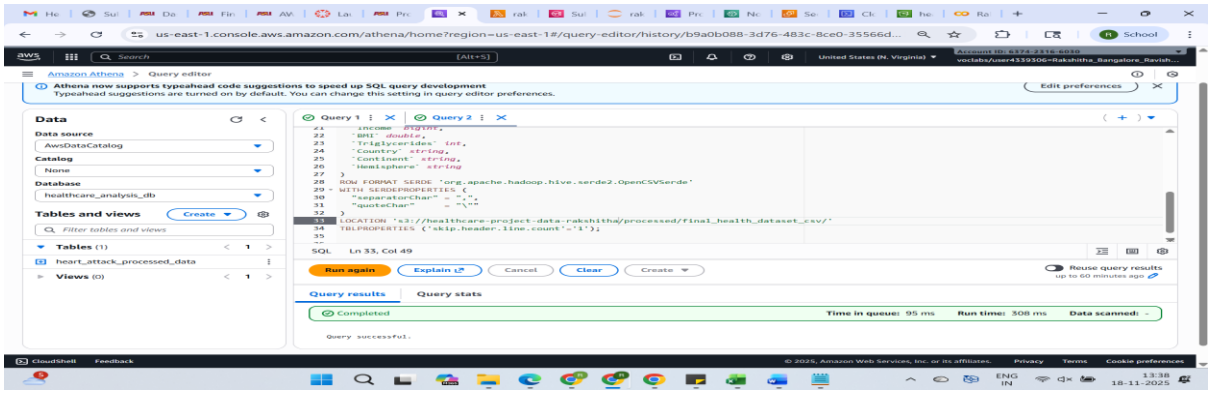


5.2 : Create Table: Processed EMR Output (Vitals data & predictions data)

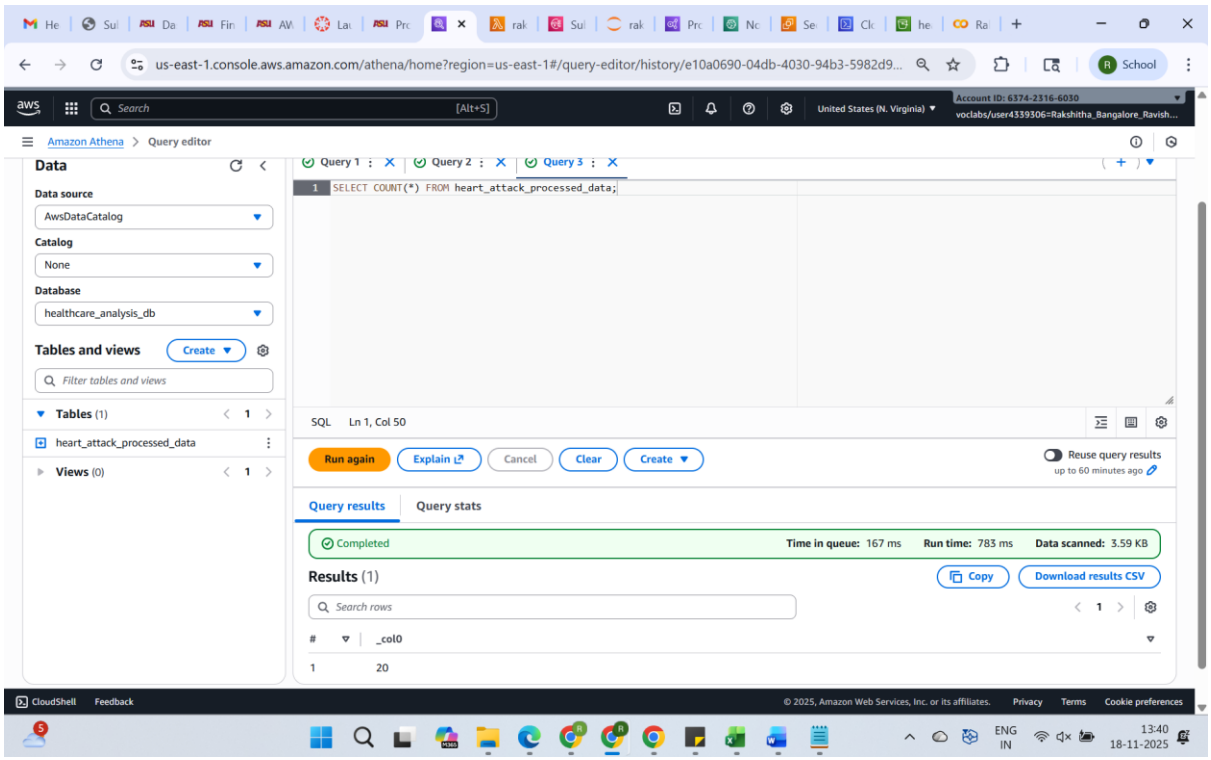
This is to make your EMR output queryable in Athena so you can join your vitals data with prediction scores.

DDL for Processed vitals data:

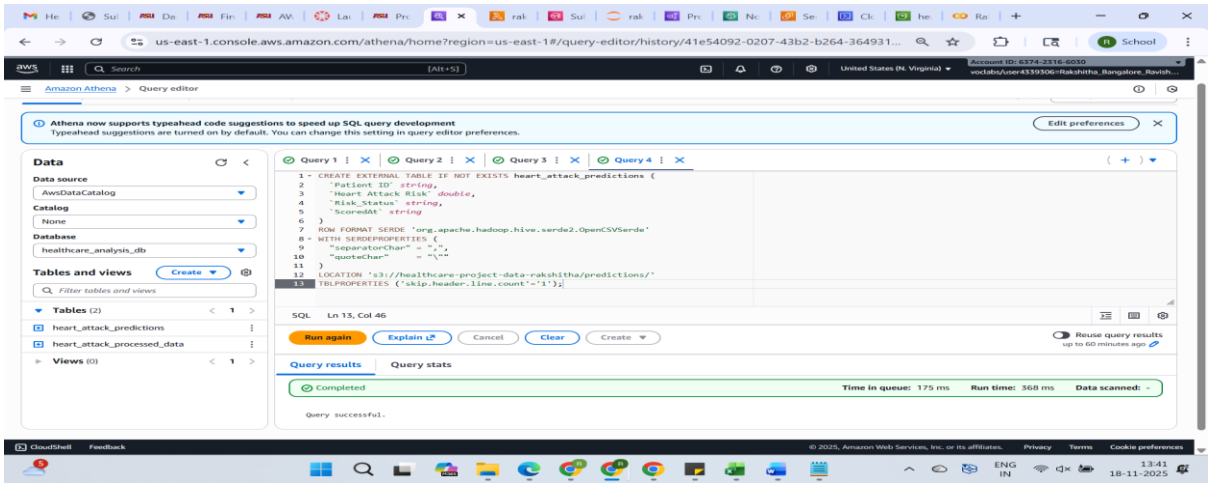




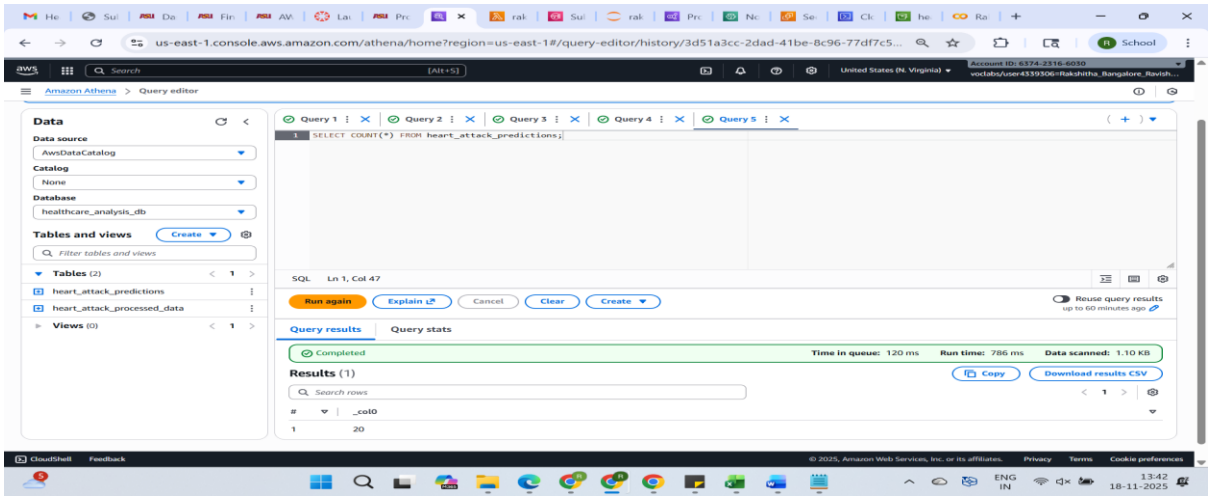
a. Verify the table creation:



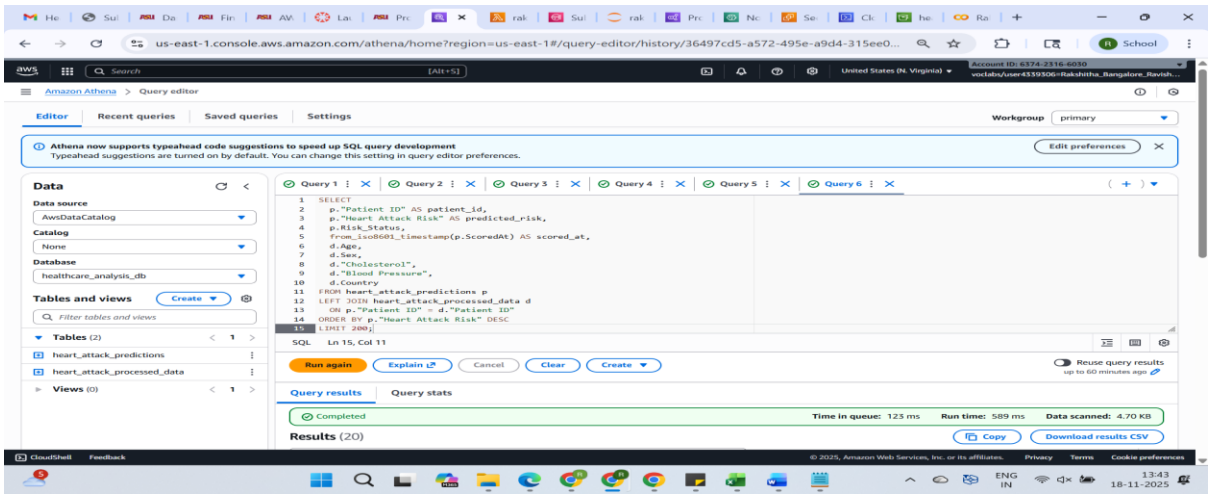
b. DDL for predictions data:

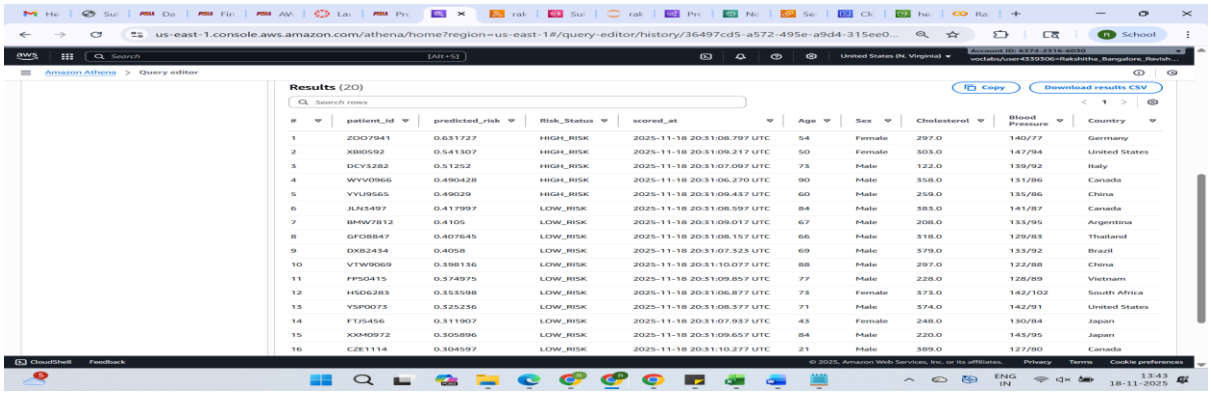


c. Verify the table creation:



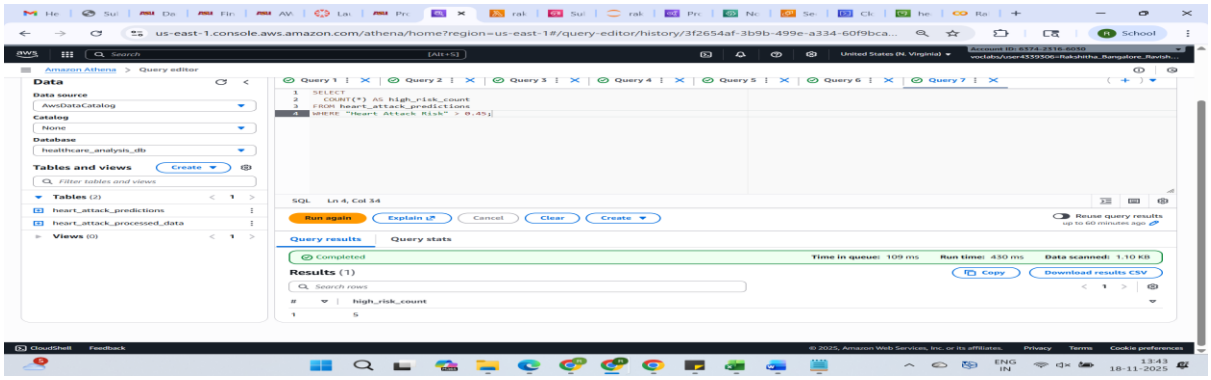
d. Who are the highest-risk patients and what are their other features?



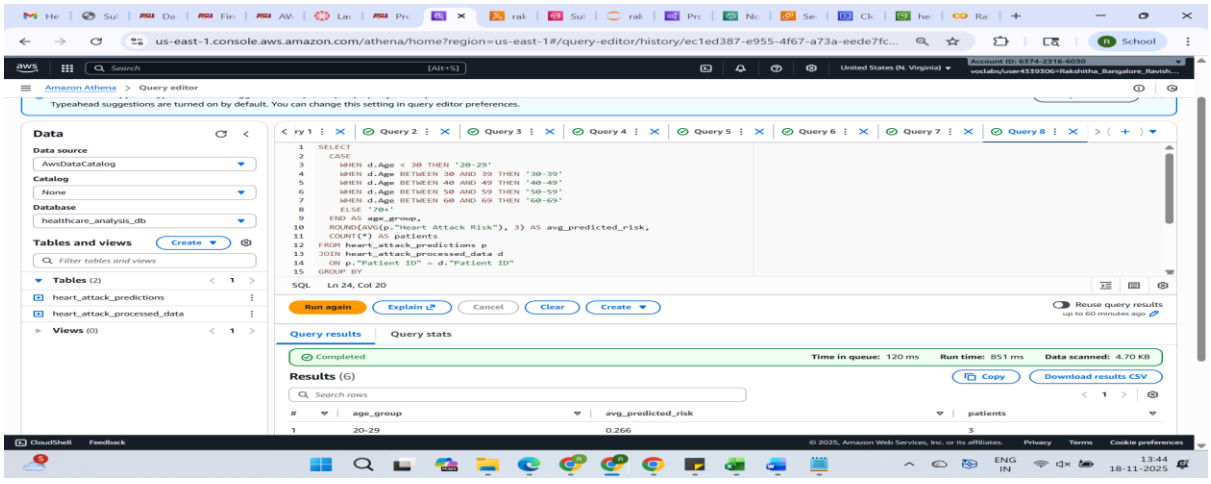


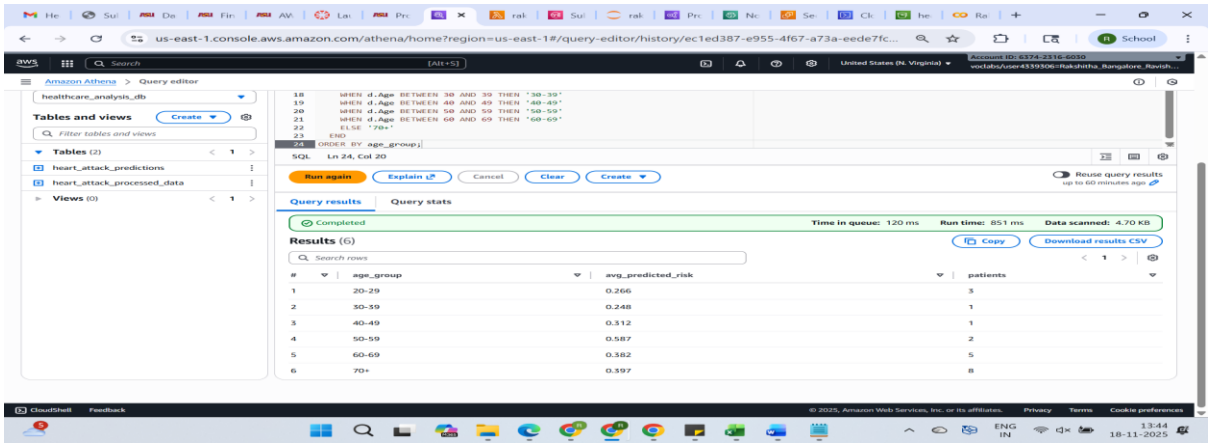
e. How many patients exceed the alert threshold (> 0.45)?

The below Query Counts the number of prediction records with probability above 0.7. This Gives a quick estimate of how many patients currently require urgent follow-up under the chosen threshold.

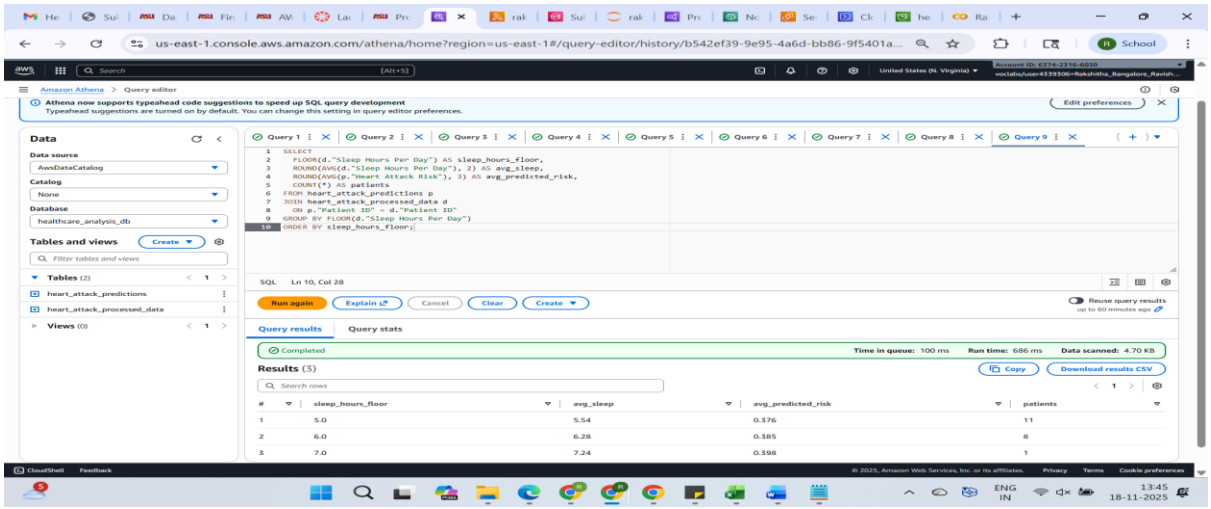


f. Which age groups have higher average predicted risk?

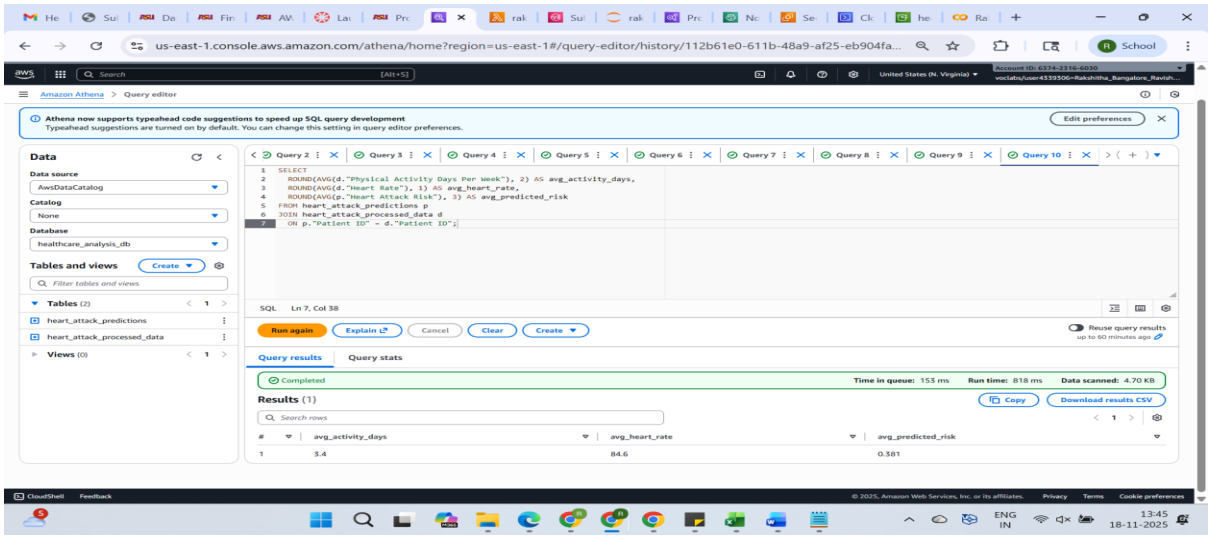


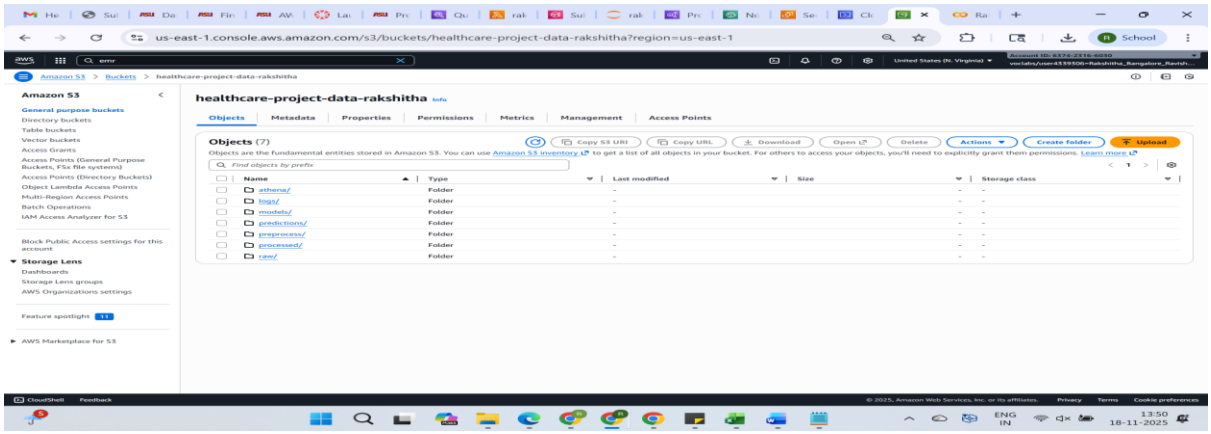


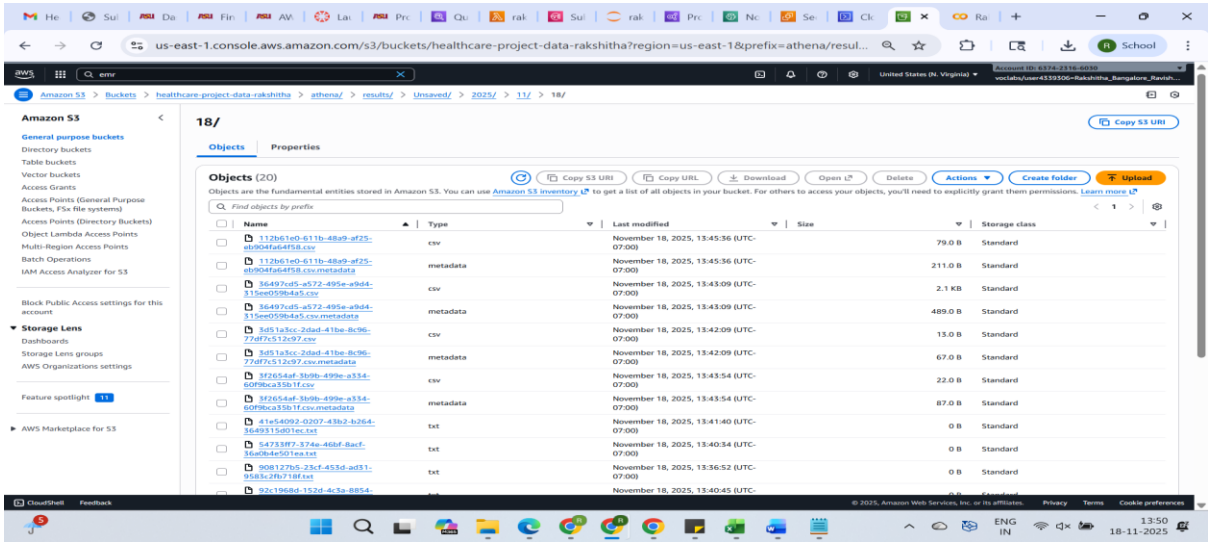
g. Is shorter sleep associated with higher predicted risk?



h. How does physical activity correlate with heart rate and predicted risk?







3. Answer following questions: first write question then answer

Phase 1 Script – Simulated Data Generation

CODE :

```
import pandas as pd
import random
import time


patient_ids = [
    "BMW7812","CZE1114","BNI9906","JLN3497","GFO8847","ZOO7941",
    "WYV0966","XXM0972","XCQ5937","FTJ5456","HSD6283","YSP0073",
    "FPS0415","YYU9565","VTW9069","DCY3282","DXB2434","COP0566",
    "XBI0592","RQX1211"
]
```

```
records = []
for pid in patient_ids:
    for i in range(7):
        record = {
            "Patient ID": pid,
            "Heart Rate": random.randint(60, 110),
            "BP_Systolic": random.randint(100, 170),
            "BP_Diastolic": random.randint(60, 120),
            "Sleep Hours Per Day": round(random.uniform(3, 9), 1),
            "Physical Activity Per day": random.randint(0, 1),
            "Timestamp": int(time.time()) + i
        }
        records.append(record)

df = pd.DataFrame(records)
df.to_csv("simulated_vitals.csv", index=False)
print("Generated simulated_vitals.csv")
```

1. **What is the main purpose of generating simulated_vitals.csv in Phase 1, and how does this file mimic real patient monitoring data?**

Answer:

Creating a realistic dataset that mimics the kind of continuous health monitoring data usually gathered by wearable devices like fitness trackers or smartwatches is the main goal of creating the simulated_vitals.csv file. This file provides dynamic input data for the prediction model by simulating daily patient vitals over a predetermined period of time, including heart rate, blood pressure, sleep duration, and physical activity. It allows the research to show how, even in situations where real-world medical data is unavailable owing to privacy limitations, an intelligent monitoring system can process and evaluate near-real-time patient health data in a secure, cloud-based environment.

2. **Why does the script assign random but realistic ranges for heart rate, blood pressure, and sleep hours?**

Answer:

In order to maintain medically believable values while simulating natural variability across patients, the script assigns random but clinically realistic ranges for each vital sign. For instance, blood pressure and sleep duration adhere to reasonable physiological parameters, whereas heart rate values are randomly generated between 60 and 110 beats per minute, representing typical resting and active conditions. This randomization maintains all readings within safe and medically reliable bounds while guaranteeing data diversity that reflects the daily fluctuations in patient vital signs. As a result, the dataset is still appropriate for precise model training and practical patient monitoring scenario simulation.

3. **How many days of vitals are simulated per patient, and why is this important for calculating weekly averages in Phase 2?**

Answer:

A complete week's worth of health monitoring data is represented by the seven days of recorded vitals for each patient in the simulated dataset. In Phase 2, where Apache Spark calculates weekly averages and activity summaries for every patient, this architectural decision enables meaningful aggregation. By averaging over a seven-day period, daily variations are lessened, data noise is decreased, and more consistent health indicators are produced. These weekly aggregates serve as the basis for spotting recurring health patterns and enhancing the precision of the machine learning model that predicts the risk of a heart attack.

Phase 2 Script – Data Processing with AWS EMR (Spark)

CODE:

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import avg, sum as _sum, round as _round, col, concat_ws

from pyspark.sql.types import IntegerType

# CONFIG

HIST_PATH = "s3://healthcare-project-data-rakshitha/raw/historical/heart_attack_prediction_dataset.csv"

SIM_PATH  = "s3://healthcare-project-data-rakshitha/raw/simulated/simulated_vitals.csv"

OUT_PATH  = "s3://healthcare-project-data-rakshitha/processed/final_health_dataset_csv/"

# Start Spark session

spark = SparkSession.builder.appName("AggregateVitalsJoinFinal").getOrCreate()

# Load datasets
```

```
hist = spark.read.option("header", True).csv(HIST_PATH, inferSchema=True)
sim  = spark.read.option("header", True).csv(SIM_PATH, inferSchema=True)
# Aggregate simulated vitals (7 days → weekly averages)
agg = sim.groupBy("Patient ID").agg(
    _round(avg(col("Heart Rate")), 0).alias("Heart Rate"),
    _round(avg(col("BP_Systolic")), 0).alias("AvgBP_Systolic"),
    _round(avg(col("BP_Diastolic")), 0).alias("AvgBP_Diastolic"),
    _round(avg(col("Sleep Hours Per Day")), 2).alias("Sleep Hours Per Day"),
    _sum(col("Physical Activity Per day")).alias("Physical Activity Days Per Week")
)
# Combine averaged systolic & diastolic into "Blood Pressure"
agg = agg.withColumn(
    "Blood Pressure",
    concat_ws("/", col("AvgBP_Systolic").cast(IntegerType()),
col("AvgBP_Diastolic").cast(IntegerType()))
).drop("AvgBP_Systolic", "AvgBP_Diastolic")
# Clean historical dataset (drop old vitals and risk column)
hist_clean = hist.drop(
    "Heart Rate",
    "Blood Pressure",
    "Sleep Hours Per Day",
    "Physical Activity Days Per Week",
    "Heart Attack Risk"
)
# Join historical demographics with aggregated vitals
final =agg.join(hist_clean, on="Patient ID", how="left")
# Write final dataset to CSV (ready for SageMaker)
final.coalesce(1).write.mode("overwrite").option("header", True).csv(OUT_PATH)
print("Final dataset written successfully to:", OUT_PATH)
print("Columns in output:", final.columns)
spark.stop()
```

4. **Describe how the Spark script aggregates the seven-day vitals for each Patient ID.**

Answer:

In order to process the simulated vitals information, the Spark script first groups all records according to the Patient ID field. Next, it computes summary statistics for every patient throughout the course of seven days of monitoring. In particular, it adds up the days of physical activity during the week and computes the average heart rate, average systolic and diastolic blood pressure, and average sleep length. By combining several daily inputs into a single representative record for each patient, this aggregation makes the data clear, organized, and appropriate for model training. This algorithm can scale effectively even for big healthcare datasets, maintaining both accuracy and performance, thanks to the use of Apache Spark on EMR.

5. **The historical dataset (heart_attack_prediction_dataset.csv) contains many more IDs than the simulated file.**
    - **What join strategy (left, right, inner) is used, and how does it determine which patients appear in the final dataset?**

Answer:

The Spark script uses a left join, with the historical patient dataset as the right table and the aggregated simulated vitals as the left table. Even if some of the simulated patients do not have matching entries in the historical dataset, this approach guarantees that they are all included in the final output. As a result, only patients who show up in the simulated data that is, patients who are currently under observation are kept in the final processed file. By adding available demographic and clinical history to the patient records, this method ensures that the final dataset concentrates on the most recent, actively watched patients.

6. **Explain how the combined “Blood Pressure” column is created and why it must later be split again in Phase 3.**

Answer:

Spark calculates each patient's average systolic and diastolic blood pressure values during data aggregation. After then, these two numbers are combined into a single text column named "Blood Pressure," which is formatted as Systolic/Diastolic (e.g., 120/80). The generated dataset is smaller for storage and easier to read thanks to this representation. To guarantee compatibility with the machine learning technique, this combined column is divided back into two distinct numerical fields in Phase 3: BP_Systolic and BP_Diastolic. Separating these values is crucial for efficient training

and prediction since the model needs numerical characteristics for precise computation.

7. **Why are we dropping – old vitals and risk column**

Answer:

Older vital signs and a specified "Heart Attack Risk" column are included in the history dataset, which, if left unaltered, could skew the machine learning model. In order to ensure that the model only learns from fresh simulated inputs rather than depending on previously labeled outcomes, these columns are removed to stop data leaking. A clean, consistent dataset centered on the most recent aggregated patient metrics is also maintained by eliminating redundant or out-of-date vitals. This step guarantees that the model's predictions are only based on new, pertinent data and are not impacted by previously calculated risks or out-of-date physiological data.

Phase 3 Script – Model Training & Deployment (SageMaker)

CODE:

```
import boto3, io, pandas as pd
from sklearn.model_selection import train_test_split
import sagemaker
from sagemaker import get_execution_role

region = "us-east-1" # update if needed
bucket = "healthcare-project-data-rakshitha" # your bucket name
role = get_execution_role()
s3 = boto3.client("s3", region_name=region)

hist_key = "raw/historical/heart_attack_prediction_dataset.csv"

#load the processed/merged CSV from S3 (produced by EMR) to confirm shape & columns.

obj = s3.get_object(Bucket=bucket, Key=hist_key)
```

```
df = pd.read_csv(io.BytesIO(obj["Body"].read()))
print("Loaded dataset:", df.shape)
df.head(2)


def preprocess_health_data(df):
    # Split blood pressure
    if "Blood Pressure" in df.columns:
        bp = df["Blood Pressure"].astype(str).str.split("/", n=1, expand=True)
        df["BP_Systolic"] = pd.to_numeric(bp[0], errors="coerce")
        df["BP_Diastolic"] = pd.to_numeric(bp[1], errors="coerce")
        df.drop(columns=["Blood Pressure"], inplace=True)


    # Drop identifiers
    df = df.drop(columns=["Patient ID","Country","Continent","Hemisphere"],
errors="ignore")


    # One-hot encode categoricals
    df = pd.get_dummies(df, drop_first=True).fillna(0)
    return df


proc_df = preprocess_health_data(df)
y = proc_df["Heart Attack Risk"].astype(int)
X = proc_df.drop(columns=["Heart Attack Risk"])
final_df = pd.concat([y, X], axis=1)
train_df, test_df = train_test_split(final_df, test_size=0.2, random_state=42, stratify=y)
print("Train:", train_df.shape, "| Test:", test_df.shape)
print("\n Sample training row:")
display(train_df.head(1))



train_key = "raw/historical/train/train.csv"
test_key = "raw/historical/test/test.csv"
```

```
def upload_csv(df, key):
    s3.put_object(Bucket=bucket, Key=key, Body=df.to_csv(index=False, header=False).encode())
    print(f"Uploaded → s3://{bucket}/{key}")

upload_csv(train_df, train_key)
upload_csv(test_df, test_key)

feature_list = list(X.columns)
with open("feature_list.txt", "w") as f:
    f.write("\n".join(feature_list))

!aws s3 cp feature_list.txt s3://{bucket}/preprocess/feature_list.txt
print(f" Uploaded feature list → s3://{bucket}/preprocess/feature_list.txt")

from sagemaker.estimator import Estimator
from sagemaker.inputs import TrainingInput
import time

timestamp = time.strftime("%Y-%m-%d-%H-%M-%S")
output_path = f"s3://{bucket}/models/xgboost"
xgb_image = sagemaker.image_uris.retrieve("xgboost", region=region, version="1.5-1")

xgb_estimator = Estimator(
    image_uri=xgb_image,
    role=role,
    instance_count=1,
    instance_type="ml.m5.large",
    volume_size=5,
    output_path=output_path,
    base_job_name=f"xgboost-heart-attack-{timestamp}",
```

```
)

xgb_estimator.set_hyperparameters(
    objective="binary:logistic",
    num_round=100,
    eta=0.1,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    eval_metric="auc"
)

train_input = f"s3://{bucket}/{train_key}"
test_input = f"s3://{bucket}/{test_key}"

print(" Starting training job.")
xgb_estimator.fit(
    {
        "train": TrainingInput(train_input, content_type="text/csv"),
        "validation": TrainingInput(test_input, content_type="text/csv")
    }
)

model_artifact = xgb_estimator.model_data
print(" Model training complete!")
print(" Model artifact stored at:", model_artifact)

from sagemaker.model import Model
import sagemaker, time

sagemaker_session = sagemaker.session.Session()
timestamp = time.strftime("%Y-%m-%d-%H-%M-%S")
```

```
xgb_image = sagemaker.image_uris.retrieve("xgboost", region=region, version="1.5-1")

# Define the model object
xgb_model = Model(
    image_uri=xgb_image,
    model_data=model_artifact,
    role=role,
    name=f"xgb-heart-attack-{timestamp}",
    sagemaker_session=sagemaker_session,
)

# Create a custom endpoint name
endpoint_name = f"xgb-heart-attack-endpoint-{timestamp}"
print(f" Deploying XGBoost model as endpoint: {endpoint_name} .")

# Deploy using the model's .deploy() — returns None in newer SDKs,
# so we attach a Predictor manually afterward
xgb_model.deploy(
    initial_instance_count=1,
    instance_type="ml.m5.large",
    endpoint_name=endpoint_name
)

# Manually create predictor for runtime access
from sagemaker.predictor import Predictor
predictor = Predictor(endpoint_name=endpoint_name,
sagemaker_session=sagemaker_session)

print("\n Model deployed successfully!")
print(" Endpoint name:", endpoint_name)
```

8. During preprocessing, what steps are performed to make the data compatible with XGBoost training?

Answer:

To make sure the dataset is clean, organized, and prepared for the XGBoost algorithm to consume, a number of preprocessing procedures are carried out prior to model training. To enable the model to analyze the combined "Blood Pressure" column as independent features, it is first divided into two distinct numerical columns, BP_Systolic and BP_Diastolic. In order to prevent bias and useless data, non-numeric and identifying columns such Patient ID, Country, Continent, and Hemisphere are then eliminated. One-hot encoding is used to convert categorical categories, such as Sex and Diet, into numerical form. To ensure data consistency, missing values are filled in with zeros. In order to accurately assess model performance, the dataset is finally divided into training (80%) and testing (20%) subsets. These preparation procedures guarantee that the data satisfies the stringent numerical input requirements of XGBoost and generates precise, objective forecasts.

9. What is saved in feature_list.txt, and how is this file later used by Lambda in Phase 4?

Answer:

After preprocessing, the ordered list of all input features required for model training is stored in the feature_list.txt file. During inference, this list is used as a guide to ensure constant feature alignment. To guarantee that each incoming record has the same format as the training data, the AWS Lambda function loads the feature_list.txt from S3 when it receives new patient data in Phase 4. The Lambda function automatically adds any missing features to the new data with a default value (often zero). In order to ensure accurate and dependable predictions, this phase avoids schema incompatibilities and ensures that the SageMaker endpoint receives data in the precise format it was trained on.

10. What metric (AUC) is used to evaluate the XGBoost model, and what does it indicate about model performance?

Answer:

The AUC (Area Under the Receiver Operating Characteristic Curve) metric is used to assess the model's performance. The model's accuracy in differentiating between high-risk and low-risk patients is represented by the AUC. A perfect model with perfect classification skill is shown by an AUC value of 1.0, whereas random guessing is implied by a score of 0.5. Even in cases when the dataset contains class imbalances, the use of AUC in this project guarantees a thorough evaluation of the

model's predictive accuracy. Strong discriminative power is indicated by a high AUC value, which indicates that the model successfully prioritizes people who are actually at risk of having a heart attack.

11. How does the endpoint_name generated in Phase 3 link directly to the inference step in the Lambda script?

Answer:

Once the XGBoost model has been trained and deployed, SageMaker gives it a distinct endpoint name. The live API access for real-time forecasts is provided by this endpoint. In Phase 4, the SageMaker model is invoked by the AWS Lambda function using this precise endpoint name (given as ENDPOINT_NAME) in its setup. The invoke_endpoint() method from the SageMaker runtime API is called by Lambda when processing fresh patient data. This method sends the input characteristics to the deployed model and returns the predicted heart-attack risk score. The model and the alerting system can communicate smoothly and automatically thanks to this direct relationship**.**

Integration & Reflection

19. How do Phases 1 – 4 together represent a complete data-driven pipeline—from data generation to real-time clinical alerts?

Answer:

The project's four stages work together to produce a smooth, intelligent, and completely automated data-driven healthcare pipeline. Phase 1 involves gathering or simulating patient health data to mimic real-world wearable device measurements including blood pressure, heart rate, and sleep duration. This information forms the basis of the system's data lake and is kept in Amazon S3. In phase two, the raw vitals are cleaned, aggregated, and transformed into organized, relevant data that can be utilized for machine learning using Amazon EMR and Apache Spark. An XGBoost model is trained and implemented using Amazon SageMaker in Phase 3 to forecast each patient's risk of having a heart attack. Lastly, Phase 4 automates the process of real-time risk assessment and alert creation using AWS Lambda and Amazon SNS, promptly alerting physicians when high-risk patients are identified. These four stages show how cloud technology can transform patient monitoring and preventative care by simulating a whole healthcare intelligence cycle, from data ingestion and analysis to prediction and actionable clinical alarms.

20. If model performance degrades over time, which phases would need to be revisited or retrained, and why?

Answer:

The phenomena known as data drift or concept drift usually implies that the data patterns have changed if the model's accuracy or dependability decreases over time. In order for the XGBoost model to adjust to evolving health trends, demographics, or habits, the model retraining procedure in Phase 3 needs to be repeated using freshly gathered and pertinent patient data. In order to ensure that the preprocessing and feature extraction procedures are consistent with current datasets, Phase 2 may also need to be modified if the structure, scale, or nature of incoming health data changes. Phase 1 could be modified to replicate more varied or exact vitals that mirror actual circumstances. Retraining and verifying the model on a regular basis guarantees that the system will continue to provide timely alarms and accurate predictions, sustaining its clinical dependability and utility over time.

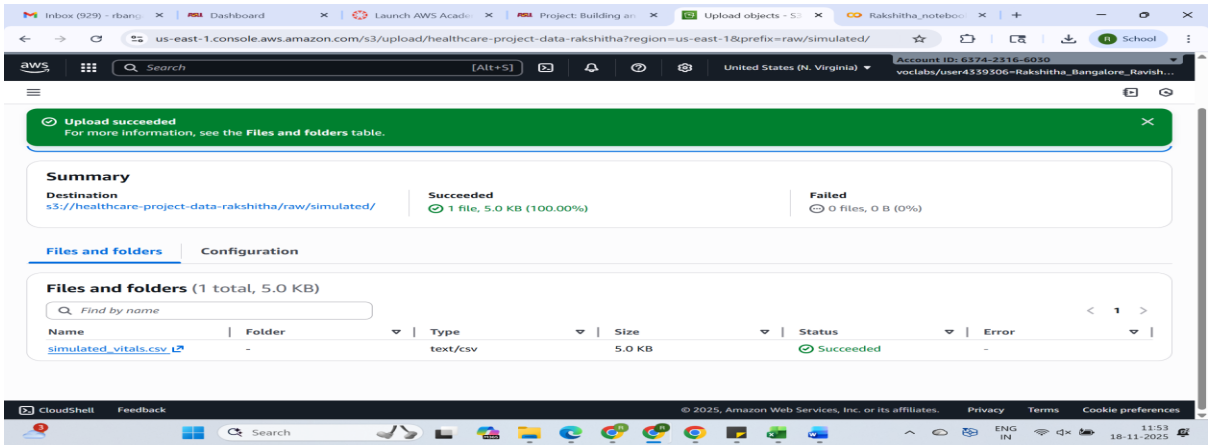
21. In the current setup of the Heart Health Alert project, will alerts be generated only for the 20 simulated ids, if tour answer is "yes" then explain" why" ?

Answer:

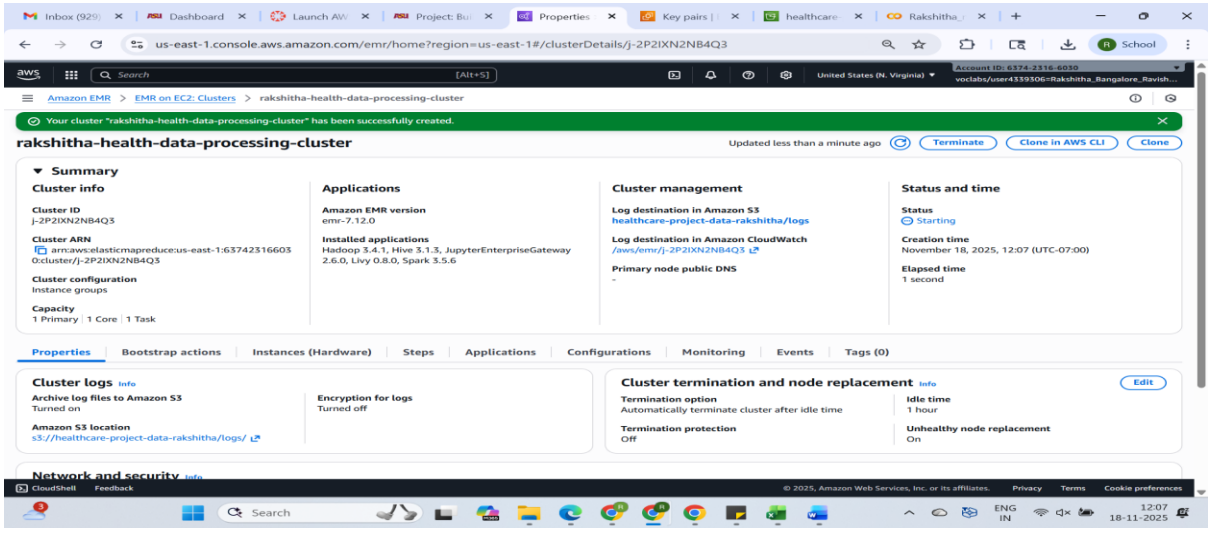
Yes, only the 20 simulated patient IDs found in the simulated_vitals.csv file will receive alerts under the present configuration. This is due to the fact that data from those simulated records is intentionally combined and aggregated during the EMR processing phase. Only those 20 patient IDs are carried over into the final processed dataset during the join procedure between simulated vitals and historical data. Because of this, only those 20 people can be evaluated and alerted by the AWS Lambda code that reads this processed file and calls the SageMaker API for predictions. However, with continuous, real-time data ingestion from wearable devices, this same approach could easily expand to thousands of patients in a real-world deployment, guaranteeing that all monitored patients receive timely alarms based on their unique heart health risk levels.

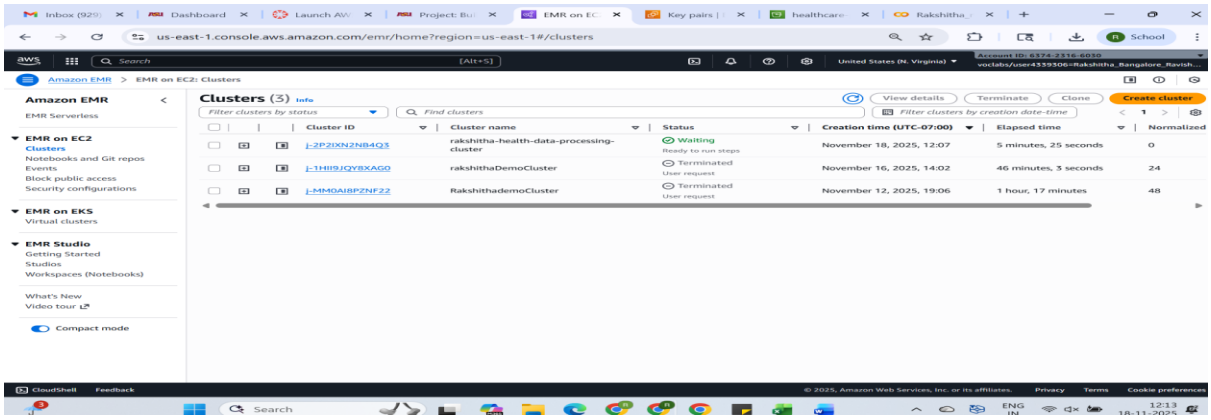
4. ScreenShots

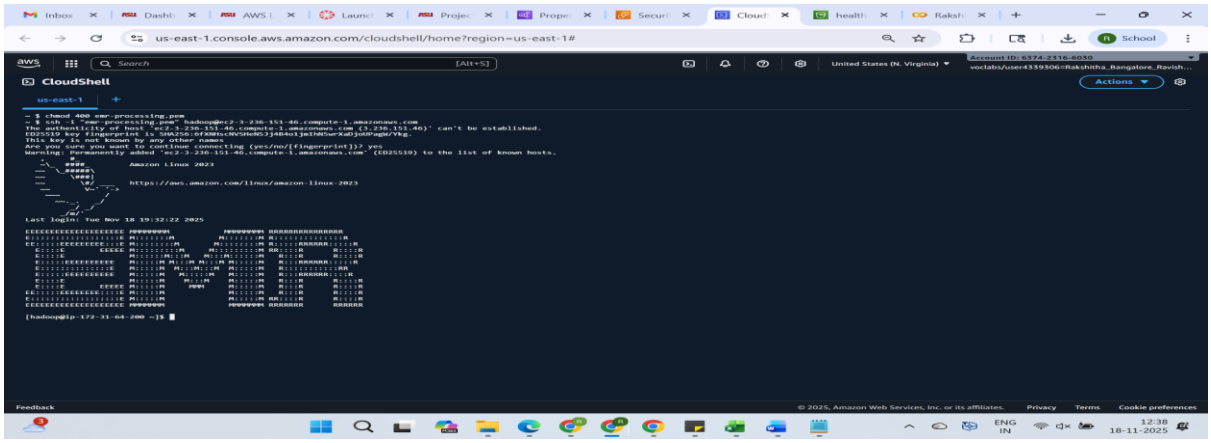
S1: Upload Simulated Data to S3



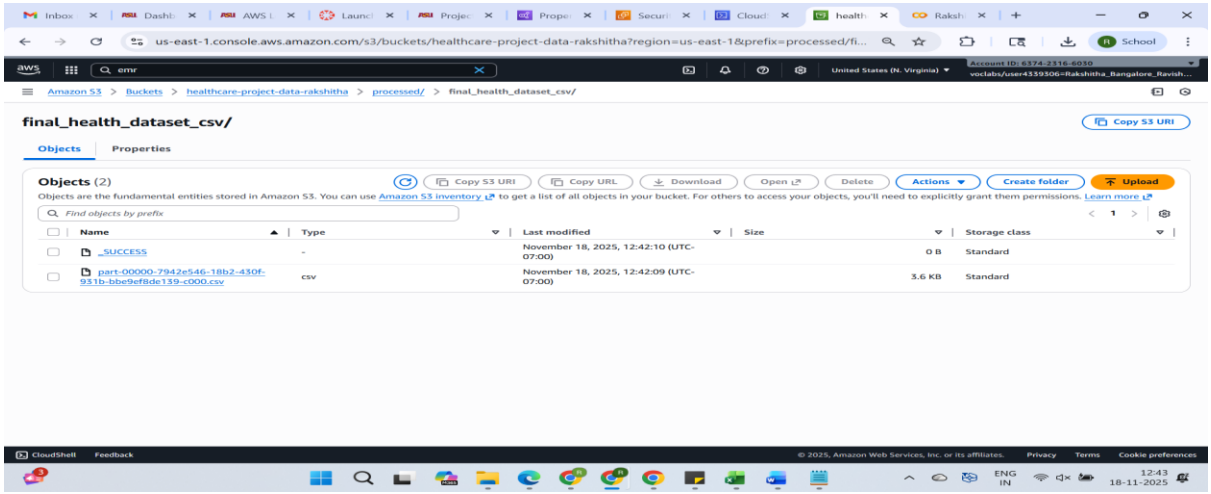
S2: Cluster Creation



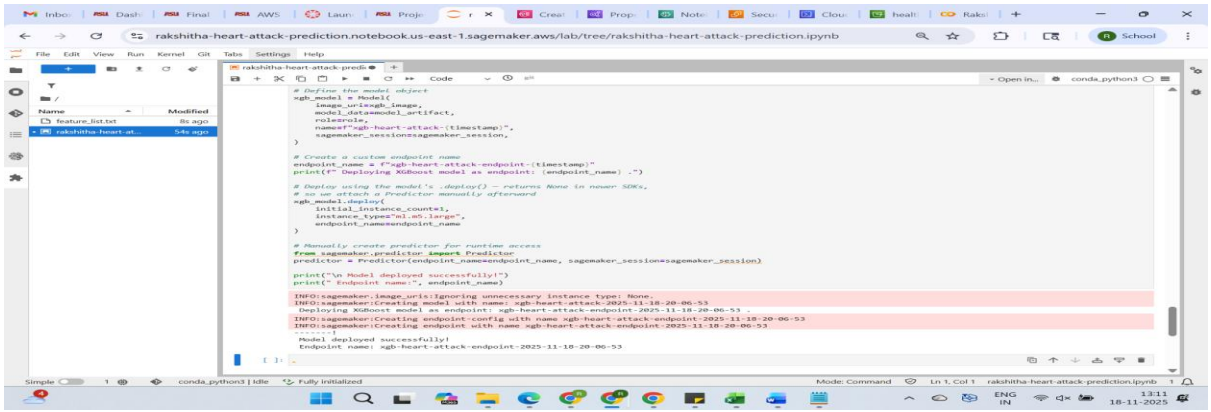


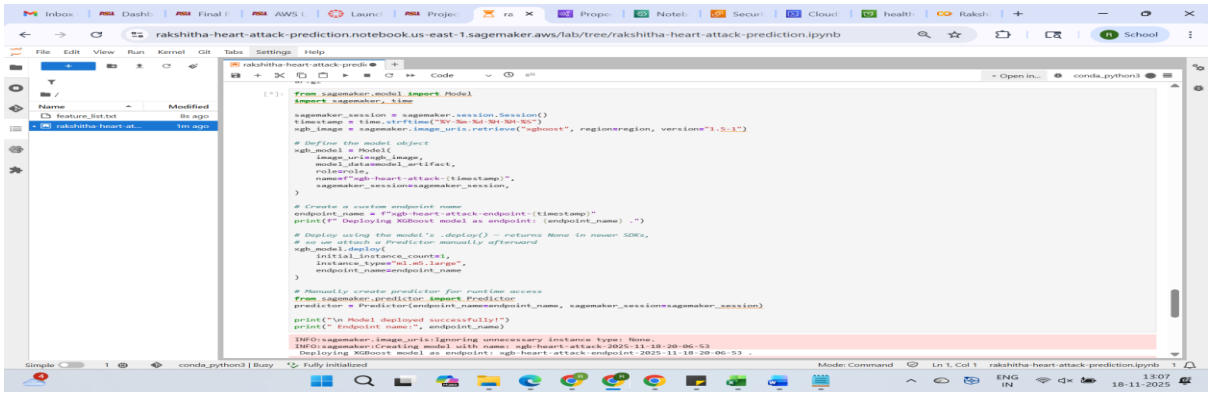


S3: Processed data in bucket

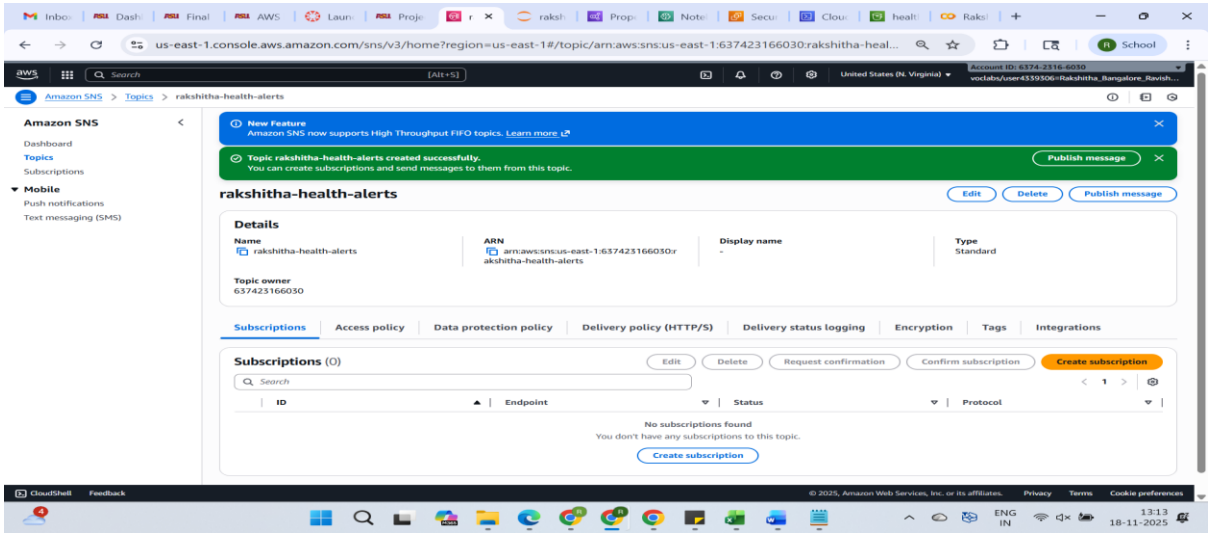


S4: Model training Endpoint

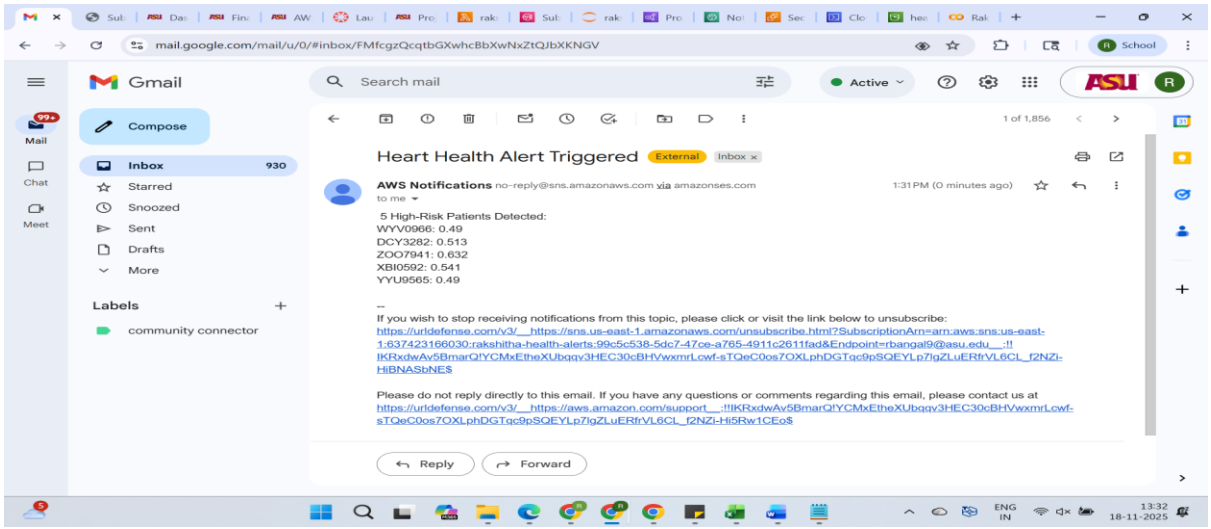




S5: SNS Alert topic

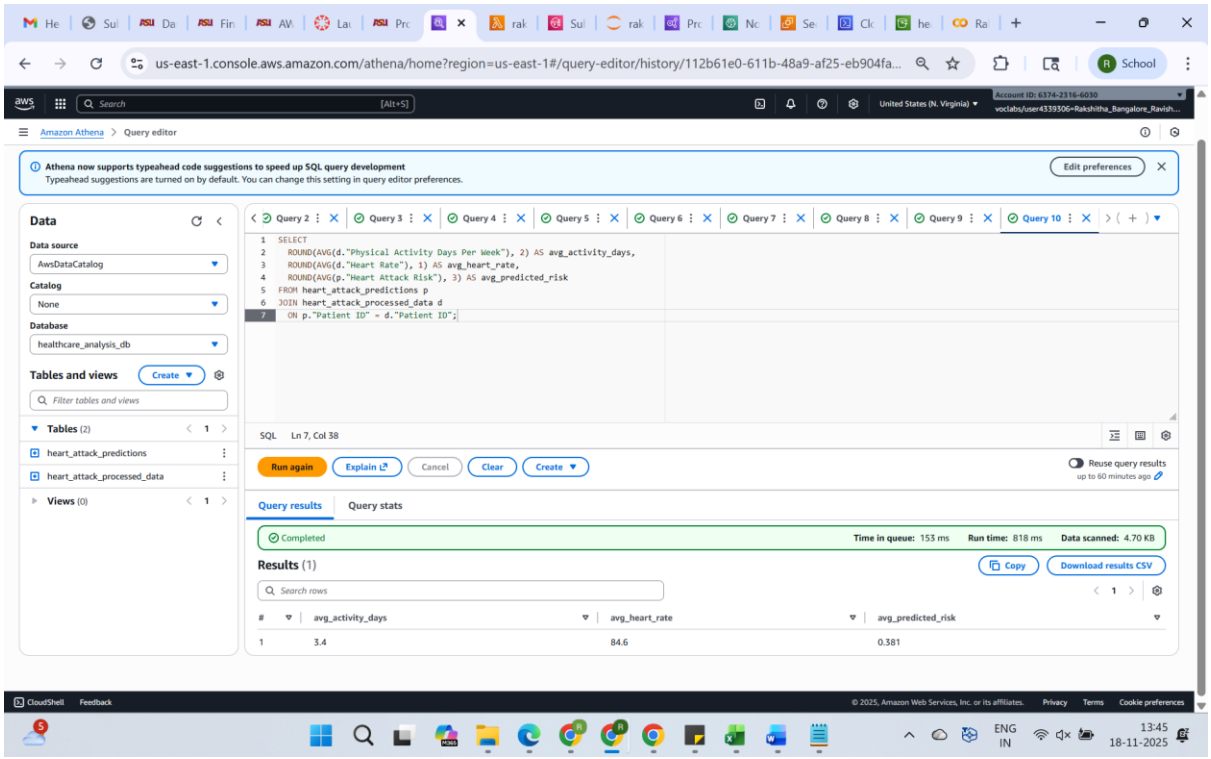


S6: SNS Email



S7: Athena query

h. How does physical activity correlate with heart rate and predicted risk?



4. Project Video:

- **Zoom Meeting:** https://asu.zoom.us/rec/share/CNNA5THdOmJnXrWfmU4v_whtD5202uanc_c61ab5QRm27kFf3OIVMoLuDscq0-c.OupZdJF23a7nR9er?startTime=1763868672000

  Passcode: Bmrw0+GC

- **Google Drive:** https://drive.google.com/drive/folders/1-pxXWqmlD0va8QkRFPmXSSl3O8H_UXRK?usp=drive_link