

“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.

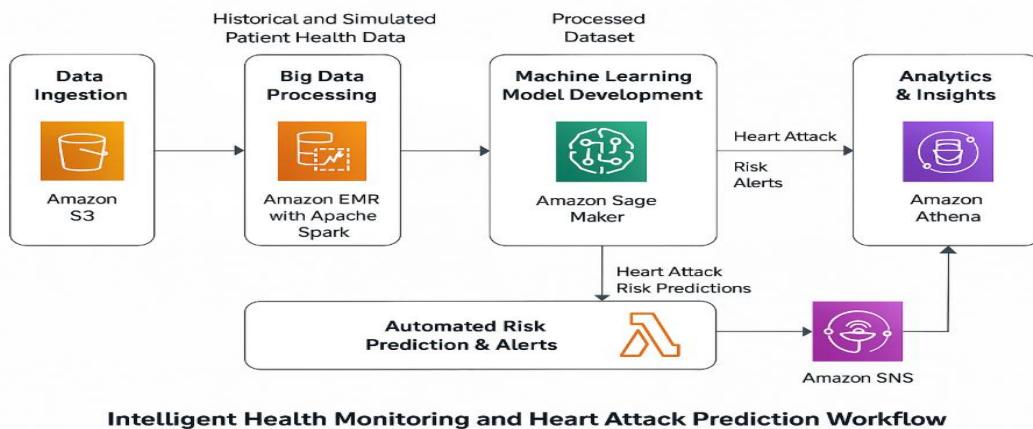
When a patient surpasses a predetermined risk threshold, an AWS Lambda function accesses the SageMaker endpoint for fresh patient data, assesses risk levels, and sends out Amazon

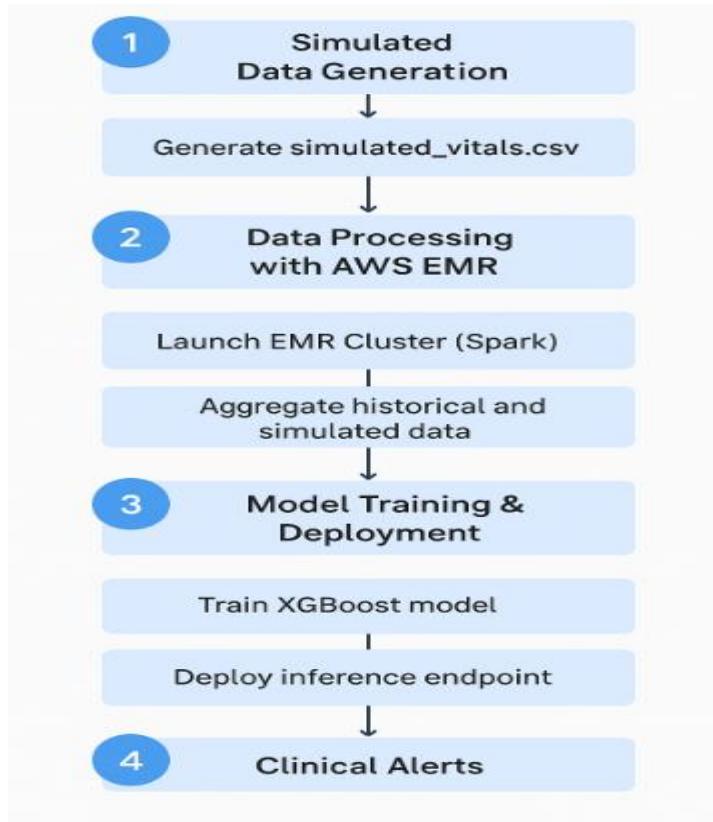
SNS (Simple Notification Service) notifications to healthcare providers in order to automate reactions. Lastly, Amazon Athena gives clinicians data-driven insights for long-term health monitoring and preventive care by enabling SQL-based querying and visualization of past predictions and trends straight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





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/
```

Screenshot of a web browser showing the Kaggle dataset page for the Heart Attack Risk Prediction Dataset. The page includes a search bar, a sidebar with navigation links like Home, Competitions, Datasets, Models, Benchmarks, Game Arena, Code, Discussions, Learn, and More, and a main content area with a title "Heart Attack Risk Prediction Dataset", a description, and a "Download" button. Below the main content is a Microsoft Excel spreadsheet titled "heart_attack_prediction_dataset.csv" showing a sample of data with columns such as Patient ID, Age, Sex, Cholesterol, Blood Pres, Heart Rate, Diabetes, Family His Smoking, Obesity, Alcohol, Cc, Exercise, H Diet, Previous H, Medication, Stress Lev, Sedentary Income, BMI, and Tri.

Screenshot of a web browser showing the AWS S3 console interface for creating a new bucket. The "Create bucket" wizard is displayed, with steps for General configuration, Object ownership, and Block Public Access settings. The "General configuration" step shows fields for Bucket name (set to "Rakshitha_notebook"), Region (set to "us-east-1"), and Bucket type (set to "Standard"). The "Object ownership" step shows options for "ACLS disabled (recommended)" or "ACLS enabled". The "Block Public Access settings for this bucket" step shows options for "Block public access to buckets and objects granted through user access control lists (ACLs)", "Block public access to buckets and objects granted through any access control lists (ACLs)", "Block public access to buckets and objects granted through any new public bucket or access point policies", and "Block public and cross-account access to buckets and objects through any public bucket or access point policies". The browser status bar at the bottom indicates the date and time as 18-11-2025 11:47.

Bucket Versioning
Versioning allows you to store multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

Tags - optional (0)
You can use bucket tags to track storage costs and organize buckets. [Learn more](#)

Default encryption
Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption type
 Server-side encryption with Amazon S3 managed keys (SSE-S3)
 Dual-layer server-side encryption with AWS Key Management Service keys (DSS-KMS)

Bucket key
Bucket key bucket key for SSE-KMS reduces encryption costs by lowering costs to AWS KMS. S3 bucket keys aren't supported for SSE-KMS. [Learn more](#)

Advanced settings

After creating the bucket, you can upload files and folders to the bucket, and configure additional bucket settings.

Amazon S3 > Buckets

Successfully created bucket "healthcare-project-data-rakshitha"
To upload files and folders, or to configure additional bucket settings, choose [View details](#).

General purpose buckets (1) [Info](#)

Name	AWS Region	Creation date
healthcare-project-data-rakshitha	US East (N. Virginia) us-east-1	November 18, 2025, 11:48:26 (UTC-07:00)

Account snapshot [Info](#)
Updated daily
[View dashboard](#)

Storage Lens provides visibility into storage usage and activity trends.

External access summary - new [Info](#)
Updated daily
External access findings help you identify bucket permissions that allow public access or access from other AWS accounts.

CloudShell Feedback

Amazon S3 > Buckets > healthcare-project-data-rakshitha > raw/ > historical/

historical/

Objects Properties

Objects (1)

Name	Type	Last modified	Size	Storage class
heart_attack_prediction_data_et.csv	csv	November 18, 2025, 11:49:56 (UTC-07:00)	1.4 MB	Standard

Phase 1 . Prepare Input Data

```

import pandas as pd
import random
from datetime import datetime, timedelta

patient_ids = [
    "BMW7812", "CZE1114", "BN19906", "T183459", "G08847", "2007941",
    "MVW7812", "XCE00022", "XCQ5937", "F135456", "H50228", "YSP9027",
    "PFS4415", "VWJ0565", "VM99065", "DCY3282", "DXB2434", "COP9566",
    "ABJ0932", "RQD3211"
]

records = []
for pid in patient_ids:
    for i in range(2):
        record = {
            "Patient ID": pid,
            "Heart Rate": random.randint(60, 110),
            "BP_Systolic": random.randint(100, 170),
            "BP_Diastolic": random.randint(60, 100),
            "Sleep Hours Per Day": round(random.uniform(3, 9), 1),
            "Physical Activity Per Day": random.randint(0, 15),
            "Timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        }
        records.append(record)

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

```

Generated simulated_vitals.csv

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

Patient ID	Heart Rate	BP_Systolic	BP_Diastolic	Sleep Hours	Physical Activity	Timestamp
BMW7812	100	82	71	0	1.76E+09	
BMW7812	74	134	82	0	1.76E+09	
BMW7812	72	138	85	6.7	1	1.76E+09
BMW7812	107	140	104	5.8	1	1.76E+09
BMW7812	85	101	91	8.4	1	1.76E+09
BMW7812	60	138	99	3.9	1	1.76E+09
BMW7812	66	123	64	7.1	0	1.76E+09
CZE1114	102	102	81	4.9	1	1.76E+09
CZE1114	91	100	87	5.5	0	1.76E+09
CZE1114	70	134	63	6.9	0	1.76E+09
CZE1114	63	104	63	3.9	1	1.76E+09
CZE1114	71	145	81	7.6	0	1.76E+09
CZE1114	68	113	81	7	0	1.76E+09
CZE1114	63	131	91	5.9	1	1.76E+09
BN19906	73	162	98	8.4	0	1.76E+09
BN19906	74	111	65	3.3	0	1.76E+09
BN19906	66	143	100	6.2	0	1.76E+09
BN19906	106	111	65	6.6	1	1.76E+09
BN19906	65	113	114	3.4	0	1.76E+09
BN19906	86	105	88	7.1	0	1.76E+09
BN19906	63	114	115	3.2	1	1.76E+09

Successfully created folder "simulated".

raw/

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 Inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Name	Type	Last modified	Size	Storage class
historical/	Folder	-	-	-
simulated/	Folder	-	-	-

The screenshot shows the AWS S3 console interface. At the top, there are several tabs including 'Inbox (929) - rbang', 'Dashboard', 'Launch AWS Academy', 'Project: Building an...', 'Upload objects - S3', and 'Rakshitha_notebook'. The active tab is 'Upload objects - S3'. The URL in the address bar is 'us-east-1.console.aws.amazon.com/s3/upload/healthcare-project-data-rakshitha?region=us-east-1&prefix=raw/simulated/'. The top navigation bar includes 'aws' logo, search bar, account ID '6374-2316-6030', and dropdown for 'United States (N. Virginia)'.

Summary

Destination: <s3://healthcare-project-data-rakshitha/raw/simulated/>

Succeeded	Failed
1 file, 5.0 KB (100.00%)	0 files, 0 B (0%)

Files and folders (1 total, 5.0 KB)

Name	Folder	Type	Size	Status	Error
simulated_vitals.csv	-	text/csv	5.0 KB	Succeeded	-

At the bottom, there are links for 'CloudShell', 'Feedback', and 'Cookie preferences'. The status bar shows '© 2025, Amazon Web Services, Inc. or its affiliates.', 'Privacy', 'Terms', 'Cookie preferences', and system information like 'ENG IN', '11:53', and '18-11-2025'.

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

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

Private key file format: .pem

Tags - optional:

Create key pair

CloudShell Feedback

Today

- emr-processing.pem
- simulated_vitals
- heart_attack_prediction_dataset

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

AWS Glue Data Catalog settings:

- Use the AWS Glue Data Catalog to provide an external metastore for your application.
- Use for Hive table metadata
- Use for Spark table metadata

Summary

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: Choose EC2 instance type: m5.2xlarge
- Core: Choose EC2 instance type: m5.2xlarge
- Task: Choose EC2 instance type: m5.2xlarge

Node configuration - optional:

Core: Choose EC2 instance type: m5.2xlarge

Cluster scaling and provisioning - required

Provisioning configuration: Core size: 1 instance

Create cluster

Screenshot 1: Initial Cluster Configuration

- Name:** Task - 1
- Choose EC2 instance type:** m5.large (4 vCore, 16 GB memory, EBS only storage, On-Demand price: Lowest bid price)
- Node configuration - optional:** Add task instance group (You can add up to 47 more task instance groups)
- EBS root volume:** Size (GiB): 15, IOPS: 8000, Throughput (MiB/s): 125
- Cluster scaling and provisioning - required:** Choose how Amazon EMR should size your cluster. Core size: 1 instance.

Screenshot 2: Advanced Cluster Configuration

- Cluster logs:** Publish cluster-specific logs to Amazon S3 (Format: Use s3://bucket/prefix, Encrypt cluster-specific logs, Publish cluster-specific logs to Amazon CloudWatch)
- Tags:** Use tags to search and filter for resources, and track AWS costs associated with your cluster.
- Software settings:** Override the default configurations for specific applications on your cluster.
- Security configuration and EC2 key pair:** Choose a security configuration or create a new one that you can reuse with other clusters.
- Identity and Access Management (IAM) roles - required:** Choose or create a service role and instance profile for the EC2 instances in your cluster.

Screenshot 3: Final IAM and Instance Profile Configuration

- Amazon EC2 key pair for SSH to the cluster:** Name: emr-processing
- Identity and Access Management (IAM) roles - required:**
 - Amazon EMR service role:** Choose an existing service role (EMR_DefaultRole) or Create a service role (Create a new service role so that Amazon EMR can grant and restrict access to resources in other AWS services).
 - EC2 instance profile for Amazon EMR:** Choose an existing instance profile (EMR_EC2_DefaultRole) or Create an instance profile (Let Amazon EMR create a new instance profile so that you can specify a custom set of resources for it to access in Amazon S3).

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.

Create a sharing service role
Select a default service role or a custom role with IAM policies attached so that your cluster can interact with other AWS services.

Create a service role
Let Amazon EMR create a new service role so that you can grant and restrict access to resources in other AWS services.

Service role
EMR_DefaultRole

EC2 instance profile for Amazon EMR
When you launch an 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
Select a default role or a custom instance profile with IAM policies attached so that your cluster can interact with your resources in Amazon S3.

Create an instance profile
Let Amazon EMR create a new instance profile so that you can specify a custom set of resources for it to access in Amazon S3.

Instance profile
EMR_EC2_DefaultRole

Custom automatic scaling role - optional
When a custom automatic scaling rule triggers, Amazon EMR assumes this role to add and terminate EC2 instances.

[Learn more](#)

Custom automatic scaling role
EMR_AutoScaling_DefaultRole

[Create IAM role](#)

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 3.5.0...)

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](#)

Your cluster "rakshitha-health-data-processing-cluster" has been successfully created.

rakshitha-health-data-processing-cluster

Summary

Cluster info

- Cluster ID: j-2P2IXN2NB4Q3
- Cluster ARN: arn:aws:elasticmapreduce:us-east-1:63742316603:cluster/j-2P2IXN2NB4Q3
- Cluster configuration: Instance groups
- Capacity: 1 Primary | 1 Core | 1 Task

Properties **Bootstrap actions** **Instances (Hardware)** **Steps** **Applications** **Configurations** **Monitoring** **Events** **Tags (0)**

Cluster logs **Info**

- Archive log files to Amazon S3
- Turned on
- Amazon S3 location: s3://healthcare-project-data-rakshitha/logs/

Encryption for logs Turned off

Cluster management

- Log destination in Amazon S3: healthcare-project-data-rakshitha/logs
- Log destination in Amazon CloudWatch: /aws/emr/j-2P2IXN2NB4Q3
- Primary node public DNS: -

Status and time

- Status: Starting
- Creation time: November 18, 2025, 12:07 (UTC-07:00)
- Elapsed time: 1 second

Cluster termination and node replacement **Info**

- Termination option: Automatically terminate cluster after idle time
- Idle time: 1 hour
- Termination protection: Off
- Unhealthy node replacement: On

Network and security **Info**

Clusters (3) Info

Cluster ID	Cluster name	Status	Creation time (UTC-07:00)	Elapsed time	Normalized
j-2P2IXN2NB4Q3	rakshitha-health-data-processing-cluster	Waiting	November 18, 2025, 12:07	5 minutes, 25 seconds	0
j-1HII9JQYBXAGO	rakshithaDemoCluster	Terminated	November 16, 2025, 14:02	46 minutes, 3 seconds	24
j-MM0A1BPZHF22	RakshithademoCluster	Terminated	November 12, 2025, 19:06	1 hour, 17 minutes	48

[View details](#) [Terminate](#) [Clone](#) [Create cluster](#)

[Filter clusters by creation date-time](#)

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

The screenshot shows three consecutive screenshots of the AWS CloudShell interface.

Screenshot 1: The CloudShell interface with the message "File upload successful" indicating the file was uploaded to the directory /home/cloudshell.

```
File upload successful
emr-processing.pem was successfully uploaded to the following directory: /home/cloudshell
user.
```

Screenshot 2: The CloudShell interface showing the execution of the command "main.py". The output shows the creation of an RSA key pair and its addition to the known hosts file.

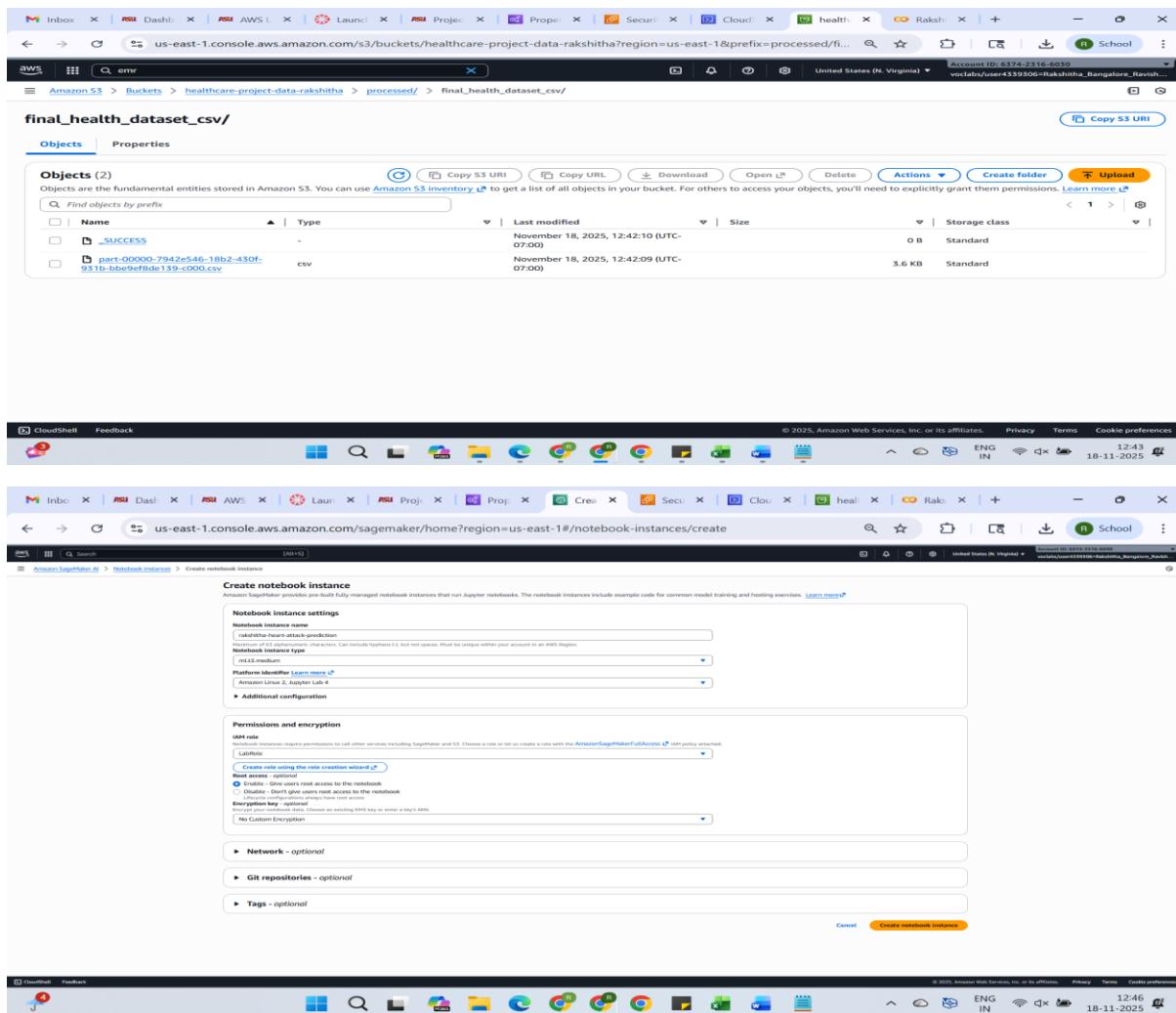
```
# CloudShell
us-east-1
$ cd /home/cloudshell
$ chmod 400 emr-processing.pem
$ ssh-keygen -t rsa -b 2048 -f /home/cloudshell/.ssh/id_rsa -q -N ""
Generating public/private rsa key pair.
The key fingerprint is:
SHA256:JzQHkZLjwvXWzCnDfVdIwvOoKJyPmBxg /home/cloudshell/.ssh/id_rsa
The key is saved in /home/cloudshell/.ssh/id_rsa.
Warning: Permanently added "ec2-3-236-151-46.compute-1.amazonaws.com" (RSA) to the list of known hosts.
Amazon Linux 2023
[ec2-user@ip-172-31-64-200 ~]
```

Screenshot 3: The CloudShell interface showing the execution of "main.py". The output shows the creation of an RSA key pair and its addition to the known hosts file.

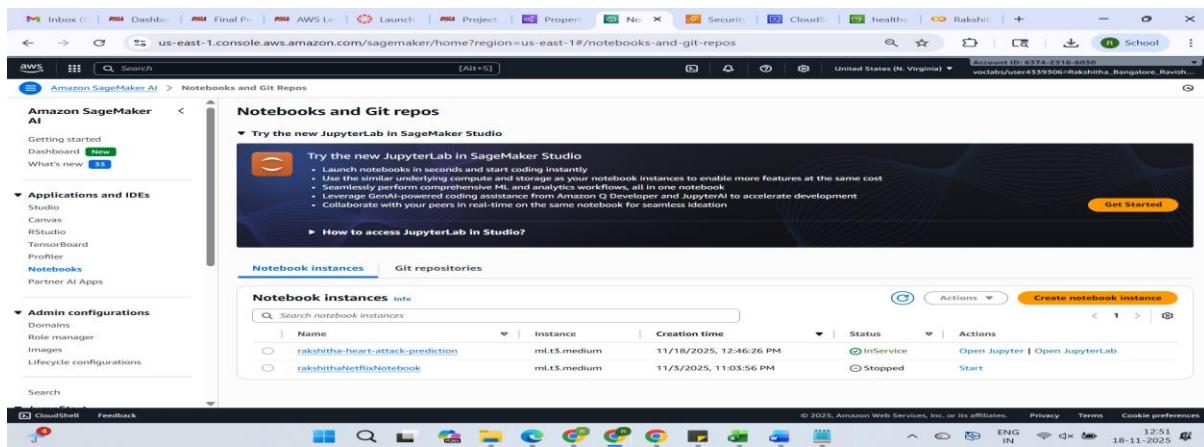
```
Last login: Sun Nov 18 21:21:22 2025
$ cd /home/cloudshell
$ chmod 400 emr-processing.pem
$ ssh-keygen -t rsa -b 2048 -f /home/cloudshell/.ssh/id_rsa -q -N ""
Generating public/private rsa key pair.
The key fingerprint is:
SHA256:JzQHkZLjwvXWzCnDfVdIwvOoKJyPmBxg /home/cloudshell/.ssh/id_rsa
The key is saved in /home/cloudshell/.ssh/id_rsa.
Warning: Permanently added "ec2-3-236-151-46.compute-1.amazonaws.com" (RSA) to the list of known hosts.
Amazon Linux 2023
[ec2-user@ip-172-31-64-200 ~]
```

```
[root@ip-172-31-64-200 ~]# nano main.py
[root@ip-172-31-64-200 ~]# spark-submit main.py > log.txt
20/11/18 10:43:17 INFO SparkContext: Using Spark version 3.0.0-RC1
20/11/18 10:43:17 INFO SparkContext: spark://ip-172-31-64-200.mzn2023.vsg6.64.ams64
20/11/18 10:43:17 INFO SparkContext: Java version 17.0_17
20/11/18 10:43:17 INFO SparkContext: Scala version 2.13.1
20/11/18 10:43:17 INFO SparkContext: Hadoop version 3.2.1
20/11/18 10:43:17 INFO ResourceUtils: No custom resources configured for spark.driver.
20/11/18 10:43:17 INFO SparkContext: Submitted application: AggregateVitalsJournal
20/11/18 10:43:17 INFO SparkContext: Application arguments: /tmp/vitals/journal, /tmp/vitals/journal, Map[resources: Map[executorType -> name: executorType, amount: 1, script: vendor], cores -> name: cores, amount: 1, memory -> name: memory, diskSpace -> name: diskSpace, network -> name: network, tasks: Map[taskType -> name: taskType, amount: 1, script: vendor], taskResources: Map[cpus -> name: cpus, amount: 1.0)]
20/11/18 10:43:17 INFO ResourceProfileManager: Adding ResourceProfile for created, executor resources: Map[executorType -> name: executorType, amount: 1, script: vendor], cores -> name: cores, amount: 1.0)
20/11/18 10:43:17 INFO ResourceProfileManager: Listing Resources in cluster: 4 tasks per executor
20/11/18 10:43:17 INFO ResourceProfileManager: Limiting resource is cpus at 4 tasks per executor
20/11/18 10:43:17 INFO SecurityManager: Changing view acls to: hadoop
20/11/18 10:43:17 INFO SecurityManager: Changing view acls groups to: hadoop
20/11/18 10:43:17 INFO SecurityManager: Changing view acls groups to: hadoop
20/11/18 10:43:17 INFO SecurityManager: authentication disabled; ui acl disabled; users with view permissions: hadoop; groups with view permissions: EMPTY; users with modify permissions: hadoop; groups with modify permissions: hadoop
20/11/18 10:43:18 INFO Utils: Successfully started service 'sparkDriver' on port 45001.
20/11/18 10:43:18 INFO SparkEnv: Registering BlockManagerMasterEndpoint
20/11/18 10:43:18 INFO SparkEnv: Registering spark.storage.StorageDefaultTopologyMapper for getting topology information
20/11/18 10:43:18 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
20/11/18 10:43:18 INFO DiskBlockManager: Created local directory at /tmp/tmp/blockmgr-d2eac01a-49e1-45b0-bc3e-1bb667d1ff6
20/11/18 10:43:18 INFO DiskBlockManager: DiskBlockManager initialized
20/11/18 10:43:18 INFO SparkEnv: Registering OutputCommitCoordinator
20/11/18 10:43:18 INFO SparkEnv: Authentication disabled.
20/11/18 10:43:18 INFO Utils: Starting successfully, trying service 'sparkUI' on port 4040.
20/11/18 10:43:18 INFO Utils: spark.dynamicalAllocation.preallocateExecutors to "false" disable executor preallocation.
20/11/18 10:43:18 INFO DefaultZNodeWatcherProxyProvider: Connecting to resource manager at ip-172-31-64-200.ec2.internal/172.31.64.200:8080
20/11/18 10:43:18 INFO ResourceUtils: Unable to find resourceTypes.wcl
20/11/18 10:43:18 INFO Client: Total memory allocated more than the maximum memory Capability of the cluster (12206 MB per container)
20/11/18 10:43:18 INFO Client: Will allocate AM container, with the MB memory including 384 MB overhead
20/11/18 10:43:18 INFO Client: Setting up the launch environment for our AM container
20/11/18 10:43:18 INFO Client: Setting up the launch environment for our AM containers
20/11/18 10:43:18 INFO Client: Launching AM Container, User class specified as: /tmp/vitals/journal
20/11/18 10:43:18 INFO Client: User class specified as: /tmp/vitals/journal
```

25/11/18 10:42:08 INFO MetricsSystemImpl: s3a-file-system metrics system stopped.
25/11/18 10:42:08 INFO MetricsSystemImpl: s3a-file-system metrics system shutdown complete.
DataNode[http://192.168.1.11:50070/] - 11 nodes
Final output written successfully at: /user/hadoop/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-172-31-64-298 ~]\$ |



Phase 3: Train & Deploy Model in Amazon SageMaker



```
[1]: import boto3, io, pandas as pd
from sagemaker.model_selection import train_test_split
from sagemaker import get_execution_role
region = "us-east-1"
bucket = "sagemaker-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

[2]: #load the processed/merged CSV from S3 (produced by EHR), 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)
```

```
[3]: 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 BM
0 BMW7812 67 Male 208 158/88 72 0 0 1 0 ... 6.615001 261404 31.25123
1 CZE1114 21 Male 389 165/93 98 1 1 1 1 ... 4.963459 285768 27.19497

2 rows × 26 columns
```

```
[4]: 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")
```



```
[4]: 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")
```



```
[5]: # One-hot encode categoricals
df = pd.get_dummies(df, drop_first=True).fillna(0)
return df
```



```
[5]: 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("\nSample training row:")
display(train_df.head(1))

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

Sample training row:
```

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** The title bar displays the notebook's name: "rakshitha-heart-attack-prediction.notebook.us-east-1.sagemaker.aws/lab/tree/rakshitha-heart-attack-prediction.ipynb".
- File Explorer:** On the left, there is a sidebar showing files: "feature_list.txt" (Modified 8s ago) and "rakshitha-heart-att..." (Modified 21s ago).
- Code Cell:** The main area contains Python code for data analysis and visualization.

```
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 Age Cholesterol Heart
   Risk 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 x 24 columns
```
- Output:** The output of the code cell shows the shape of the datasets (Train: 7010 rows, 24 columns; Test: 1753 rows, 24 columns), a sample training row, and the first row of the dataset.
- Code Cell 6:** The next code cell (index [6]) contains code for uploading CSV files to S3.

```
[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 {key} to {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
```
- Bottom Status Bar:** The status bar indicates "Fully initialized", "Mode: Edit", "Ln 34, Col 40", and the file name "rakshitha-heart-attack-prediction.ipynb". It also shows a "conda_python3" icon and a "Busy" status.

```
[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("Uploaded feature list to s3://(bucket)/preprocess/feature_list.txt")
Upload: ./feature_list.txt to s3://healthcare-project-data-rakshitha/preprocess/feature_list.txt
Uploaded feature list to 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",
)
```

```
File Edit View Run Kernel Git Tabs Settings Help
+ rakeshitha-heart-attack-prediction.ipynb + Open in... conda_python3
Name Modified
feature_list.txt 8s ago
rakeshitha-heart-attack-prediction.ipynb 1m ago
[1]: 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//nodev/sghost"
xgb_image = sagemaker.Image_uris.retrieve("xgboost", regions=region, versions="1.5-3")

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

xgb_estimator.set_hyperparameters(
    objective="binary:logistic",
    num_rounds=100,
    eta=0.1,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    eval_metric="auc"
)

train_input = "s3://bucket//train_key"
test_input = "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")}
)

```

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** 'rakshitha-heart-attack-prediction.notebook.us-east-1.sagemaker.aws/lab/tree/rakshitha-heart-attack-prediction.ipynb'
- File List:** 'feature_list.txt' (modified 8s ago) and 'rakshitha-heart-attack-prediction.ipynb' (modified 1m ago).
- Code Cell:** Contains Python code for training an XGBoost estimator. The code includes imports for pandas, numpy, and xgboost, defines training and test inputs, and fits the estimator with specified parameters.
- Output Log:** Shows the execution log for the training job, including starting the job, preparing instances, downloading training images, and training progress. It also includes several warnings from the xgboost profiler.
- Bottom Status:** 'Fully initialized' and 'Mode: Edit'.

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** rakshitha-heart-attack-prediction.ipynb
- File List:** feature_list.txt, rakshitha-heart-attack-prediction.ipynb
- Content:** A list of training runs (labeled 0 to 99) with their corresponding AUC values:
 - Run 0: 0.52236
 - Run 1: 0.52236
 - Run 2: 0.52236
 - Run 3: 0.52236
 - Run 4: 0.52236
 - Run 5: 0.52236
 - Run 6: 0.52236
 - Run 7: 0.52236
 - Run 8: 0.52236
 - Run 9: 0.52236
 - Run 10: 0.52236
 - Run 11: 0.52236
 - Run 12: 0.52236
 - Run 13: 0.52236
 - Run 14: 0.52236
 - Run 15: 0.52236
 - Run 16: 0.52236
 - Run 17: 0.52236
 - Run 18: 0.52236
 - Run 19: 0.52236
 - Run 20: 0.52236
 - Run 21: 0.52236
 - Run 22: 0.52236
 - Run 23: 0.52236
 - Run 24: 0.52236
 - Run 25: 0.52236
 - Run 26: 0.52236
 - Run 27: 0.52236
 - Run 28: 0.52236
 - Run 29: 0.52236
 - Run 30: 0.52236
 - Run 31: 0.52236
 - Run 32: 0.52236
 - Run 33: 0.52236
 - Run 34: 0.52236
 - Run 35: 0.52236
 - Run 36: 0.52236
 - Run 37: 0.52236
 - Run 38: 0.52236
 - Run 39: 0.52236
 - Run 40: 0.52236
 - Run 41: 0.52236
 - Run 42: 0.52236
 - Run 43: 0.52236
 - Run 44: 0.52236
 - Run 45: 0.52236
 - Run 46: 0.52236
 - Run 47: 0.52236
 - Run 48: 0.52236
 - Run 49: 0.52236
 - Run 50: 0.52236
 - Run 51: 0.52236
 - Run 52: 0.52236
 - Run 53: 0.52236
 - Run 54: 0.52236
 - Run 55: 0.52236
 - Run 56: 0.52236
 - Run 57: 0.52236
 - Run 58: 0.52236
 - Run 59: 0.52236
 - Run 60: 0.52236
 - Run 61: 0.52236
 - Run 62: 0.52236
 - Run 63: 0.52236
 - Run 64: 0.52236
 - Run 65: 0.52236
 - Run 66: 0.52236
 - Run 67: 0.52236
 - Run 68: 0.52236
 - Run 69: 0.52236
 - Run 70: 0.52236
 - Run 71: 0.52236
 - Run 72: 0.52236
 - Run 73: 0.52236
 - Run 74: 0.52236
 - Run 75: 0.52236
 - Run 76: 0.52236
 - Run 77: 0.52236
 - Run 78: 0.52236
 - Run 79: 0.52236
 - Run 80: 0.52236
 - Run 81: 0.52236
 - Run 82: 0.52236
 - Run 83: 0.52236
 - Run 84: 0.52236
 - Run 85: 0.52236
 - Run 86: 0.52236
 - Run 87: 0.52236
 - Run 88: 0.52236
 - Run 89: 0.52236
 - Run 90: 0.52236
 - Run 91: 0.52236
 - Run 92: 0.52236
 - Run 93: 0.52236
 - Run 94: 0.52236
 - Run 95: 0.52236
 - Run 96: 0.52236
 - Run 97: 0.52236
 - Run 98: 0.52236
 - Run 99: 0.52236
- Log Output:**

```
2025-11-18 20:03:09 Uploading ... Uploading generated training model
2025-11-18 20:03:09 Training completed! Training job completed
Training seconds: 120
Killable 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.t
[...]
```

```

  from sagemaker.model import Model
  import sagemaker.session
  sagemaker_session = sagemaker.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=xgb_image,
      model_data=model_artifact,
      role=role,
      name="xgb-heart-attack-(timestamp)",
      sagemaker_session=sagemaker_session,
  )

  # Create a custom endpoint name
  endpoint_name = "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(f" Endpoint name: {endpoint_name}")

INFO:sagemaker:image_uris:Ignoring unnecessary instance type: None.
INFO:sagemaker:create_endpoint:Creating model with name: xgb-heart-attack-2025-11-18-20-06-53
Deploying XGBoost model as endpoint: xgb-heart-attack-endpoint-2025-11-18-20-06-53 .
INFO:sagemaker:create_endpoint:Creating endpoint with name: xgb-heart-attack-endpoint-2025-11-18-20-06-53
INFO:sagemaker:create_endpoint:Model deployed successfully!
Endpoint name: xgb-heart-attack-endpoint-2025-11-18-20-06-53
  
```

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"
```

```

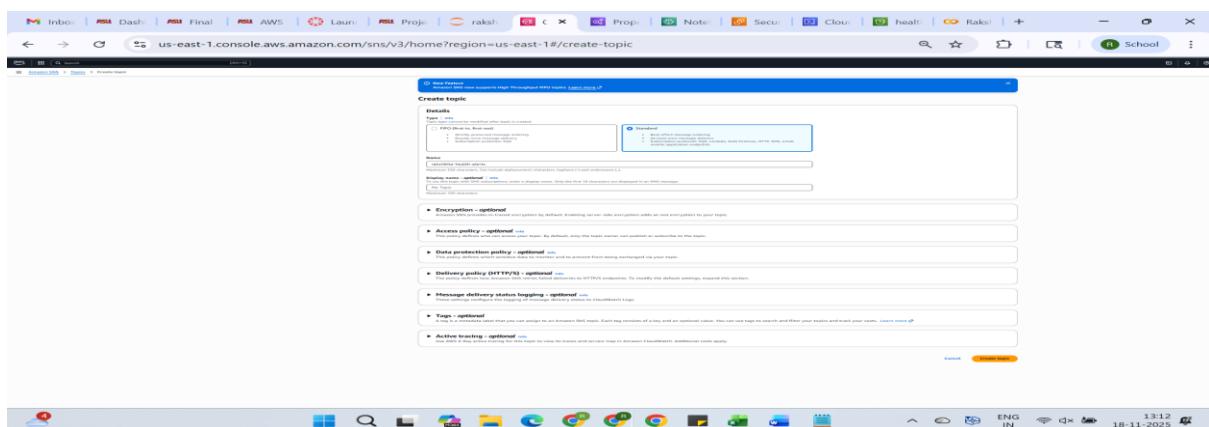
if predictions:
    df_pred = pd.DataFrame(predictions)
    csv_buf = io.StringIO()
    df_pred.to_csv(csv_buf, index=False)
    s3.put_object(Bucket=BUCKET_NAME, Key=pred_key,
    Body=csv_buf.getvalue().encode("utf-8"))
    print(f'Predictions saved to s3://{BUCKET_NAME}/{pred_key}')

# 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
    })
}

```



New Feature
Amazon SNS now supports High Throughput FIFO topics. Learn more [Learn more](#)

rakshitha-health-alerts

Details

Name: [rakshitha-health-alerts](#)

ARN: [arn:aws:sns:us-east-1:637423166030:rakshitha-health-alerts](#)

Topic owner: 637423166030

Type: Standard

Subscriptions

No subscriptions found.

You don't have any subscriptions to this topic.

[Create subscription](#)

New Feature
Amazon SNS now supports High Throughput FIFO topics. Learn more [Learn more](#)

Subscription: 99c5c538-5dc7-47ce-a765-4911c2611fad

Details

ARN: [arn:aws:sns:us-east-1:637423166030:rakshitha-health-alerts:99c5c538-5dc7-47ce-a765-4911c2611fad](#)

Endpoint: rbangar9@asu.edu

Topic: [rakshitha-health-alerts](#)

Subscription Principal: arn:aws:iam::637423166030:role/vocabs

Status: Pending confirmation

Protocol: EMAIL

Subscription filter policy [Redrive policy \(dead-letter queue\)](#)

Subscription filter policy [Info](#)

This policy filters the messages that a subscriber receives.

No filter policy configured for this subscription.

To apply a filter policy, edit this subscription.

[Edit](#)

Gmail

Compose

Inbox 930

Starred Snoozed Sent Drafts More

Labels community connector

Search mail

AWS Notification - Subscription Confirmation [External](#) [Inbox](#)

AWS Notifications no-reply@sns.amazonaws.com via amazones.com to me 1:15 PM (0 minutes ago)

You have chosen to subscribe to the topic: arn:aws:sns:us-east-1:637423166030:rakshitha-health-alerts

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):

[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#).

[Reply](#) [Forward](#)

Screenshot of a web browser showing the AWS Simple Notification Service (SNS) confirmation page. The URL is sns.us-east-1.amazonaws.com/confirmation.html?TopicArn=arn:aws:sns:us-east-1:637423166030:rakshitha-health-alerts&Token=233.... The page displays a green box with the message "Subscription confirmed! You have successfully subscribed." and the topic ARN: arn:aws:sns:us-east-1:637423166030:rakshitha-health-alerts:99c5c538-5dc7-47ce-a765-4911c261fad.

Below the confirmation message is a link: "If it was not your intention to subscribe, [click here to unsubscribe](#)".

The browser's address bar shows the URL: sns.us-east-1.amazonaws.com/confirmation.html?TopicArn=arn:aws:sns:us-east-1:637423166030:rakshitha-health-alerts&Token=233...

The system tray at the bottom right of the screen shows the date and time: 18-11-2025 13:15.

Next screenshot: AWS Lambda function creation page. The URL is us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/function?intent=authorFromScratch. The page title is "Create function". It shows three options for creating a function: "Author from scratch" (selected), "Use a blueprint", and "Container Image".

Basic information

- Function name:** rakshitha-health-monitor-function
- Runtime:** Python 3.10
- Architecture:** arm64
- Permissions:** Create a new role from AWS policy templates

Execution role: Choose an existing role that you've created to be used with this Lambda function. The role must have permissions to upload logs to Amazon CloudWatch Logs.

Existing roles: Choose an existing role that you've created to be used with this Lambda function. The role must have permissions to upload logs to Amazon CloudWatch Logs.

Code Editor: The code editor shows the following Python script:

```

# This function takes a CSV file as input and processes it to identify heart attack risk.

import boto3
import csv
import json
import pandas as pd
import s3fs
import datetime

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

ENDPOINT_NAME = "gb-heart-attack-endpoint-2023-11-18-20-00-53" # Replace
PROJECT_NAME = "heart-attack-project-data-neptune"
PROCESSED_PREFIX = "processed/final_health_dataset"
FEATURES_LIST_KEY = "preprocess/features.list.txt"
ALERT_TOPIC_ARN = "arn:aws:sns:us-east-1:637423166030:rakshitha-health-alerts" # Replace

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

def preprocess_row(df_row, expected_features):
    ...

```

The browser's address bar shows the URL: us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/function?intent=authorFromScratch

The system tray at the bottom right of the screen shows the date and time: 18-11-2025 13:17.

Final screenshot: AWS Lambda function configuration page for "rakshitha-health-monitor-function". The URL is us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/rakshitha-health-monitor-function?newFunction=true. A green success message at the top says: "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*."

The page shows tabs for "Code", "Test", "Monitor", "Configuration", "Aliases", and "Versions". The "Code" tab is selected, displaying the code editor with the same Python script as before.

The browser's address bar shows the URL: us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/rakshitha-health-monitor-function?newFunction=true

The system tray at the bottom right of the screen shows the date and time: 18-11-2025 13:20.

The screenshot shows the AWS Lambda console interface. A green success message at the top states: "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 source: Info

- Deployment: RAKSHITHA-HEALTH-MONITOR-FUNCTION
- File: lambda_function.py

```

    1 import boto3
    2 import json
    3 import os
    4 import pandas as pd
    5 import requests
    6 from datetime import datetime
    7
    8 runtime = boto3.client("sagemaker-runtime")
    9 s3 = boto3.client("s3")
    10 sns = boto3.client("sns")
    11
    12 ENDPOINT_NAME = "gb-heart-attack-endpoint-2025-11-18-20-06-5"
    13 BUCKET_NAME = "healthcare-project-data-rakshitha"
    14 FILENAME_PHRASE = "heart-attack-risk-score.csv"
    15 FEATURE_LIST_KEY = "preprocess/feature_list.txt"
    16 ALERTEVENT_ID = "lambda.rakshitha.health-monitor.function"
    17
    18 def load_feature_list():
    19     """Load expected feature list from S3"""
    20     features = []
    21     FeatureListObj = runtime.read().decode("utf-8").splitlines()
    22     for f in FeatureListObj:
    23         if f.strip() != "" and f not in features:
    24             features.append(f.strip())
    25
    26     return features
    27
    28 def preprocess(row, expected_features):
    29     """Preprocess one row and align with training feature set
    30     df = pd.DataFrame([row])
    31
    32     # Split Blood Pressure
    
```

Deploy: Deploy (Ctrl+Shift+U) Not (Ctrl+Shift+R)

TEST EVENTS (NONE SELECTED)

Create new test event

Open in Visual Studio Code **Upload from**

Create new test event

Event name: healthTest

Invocation type: Synchronous

Event sharing settings: Private

Template - optional: Hello World

Event JSON:

```

1 {
2     "key1": "value1",
3     "key2": "value2",
4     "key3": "value3"
5 }

```

Lambda Layout: US

The screenshot shows the 'Add layer' dialog. A green success message at the top says: "The test event "healthTest" was successfully saved."

Add layer

Function runtime settings: Runtime Python 3.10

Choose a layer

- Layer source**: AWS layers
- Custom layers**: Choose a layer from a list of layers created by your AWS account.
- Specify an ARN**: Specify a layer by providing the ARN.

AWS layers: AWSLambdaLayerProviderLayer

Version: 27

Add

CloudShell Feedback

13:21 18-11-2025

The screenshot shows the 'Edit basic settings' dialog. A green success message at the top says: "The test event "healthTest" was successfully saved."

Basic settings

Description - optional:

Memory: 128 MB

Ephemeral storage: 512 MB

SnapStart: None

Timeout: 5 sec

Execution role: Use an existing role or Create a new role from AWS policy templates

Existing role: LabRole

Save

CloudShell Feedback

13:22 18-11-2025

The screenshot shows the AWS Lambda console interface. A green success message at the top states: "Successfully updated the function rakshitha-health-monitor-function." and "The test event 'healthTest' was successfully saved." Below this, the "Code source" tab is selected, showing the contents of `lambda_function.py`. The code reads a CSV file and processes its rows. The deployment stage dropdown is set to "Untested". On the left, there's a sidebar for "TEST EVENTS (SELECTED: healthTest)" and "ENVIRONMENT VARIABLES". At the bottom, the status bar indicates "Status: Succeeded" and "Execution results" are shown.

This screenshot shows the AWS SNS "Create subscription" page. It has sections for "Details" (Topic ARN: `arn:aws:sns:us-east-1:6374231660:rakshitha-health-alerts`), "Protocol" (Email), and "Endpoint" (Email address: `rakshitha@auqa.edu`). Below these, there are optional sections for "Subscription filter policy" and "Dead-letter policy". At the bottom right, there are "Cancel" and "Create subscription" buttons.

This screenshot shows the AWS Lambda console after the function has been deployed. A green success message at the top says: "Successfully updated the function rakshitha-health-monitor-function." and "The test event 'healthTest' was successfully saved." The "Code source" tab is selected, showing the updated code. The deployment stage dropdown is now set to "Current". The status bar at the bottom indicates "Status: Succeeded" and "Lambda invoked. Checking for processed CSV file in S3...".

The screenshot shows the AWS Lambda console interface. The top navigation bar includes links for Home, Sub, AWS Data, AWS Fin, AWS AW, Lau, AWS Pro, Sub, rak, Proc, Noi, Sec, CLO, hee, CO, Rak, and School. The main title is "us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/rakshitha-health-monitor-function?tab=code". The status bar at the bottom right shows the date as 18-11-2025.

Code source (info)

EXPLORER

- RAKSHITHA-HEALTH-MONITOR-FUNCTION
 - lambda_function.py

DEPLOY (Ctrl+Shift+U) Test (Ctrl+Shift+T)

PREDICTIONS OUTPUT CODE SERVING TERMINAL

TEST EVENTS (SELECTED: HEALTHTEST)

- + Create new test event
- Private saved events
- HealthTest

ENVIRONMENT VARIABLES

Lambda Deployed 0.0 Amazon Q

Execution Results

Output from the Lambda function execution:

```
rakshitha-health-monitor-function

lambda_function.py
def lambda_handler(event, context):
    for i, row in enumerate(event['rows'][50:]): # process up to 50 rows
        if row['Content-Type'] == 'text/csv':
            Body = row['Body']
            raw_result = response["Body"].read().decode("utf-8").strip()
            if raw_result:
                score = float(raw_result)
            else:
                score = None
    return {
        "Rows": event["Rows"]
    }

Columns detected (28): ["Patient ID", "Heart Rate", "Sleep Hours Per Day", "Physical Activity Days Per Week", "Blood Pressure", "Age", "Sex", "Cholesterol", "Smoker", "Diet", "Exercise", "Alcohol", "Hypertension", "Diabetes", "Obesity", "Previous Heart Problems", "Medication Use", "Stress Level", "Sedentary Hours Per Day", "Income", "BHT", "Triglycerides", "Country", "Continent", "Hemisphere"]

Risk score for WVW0060: 0.406
Risk score for RDX1211: 0.194
Risk score for HSD0001: 0.182
Risk score for HSD0083: 0.154
Risk score for DCV3282: 0.153
Risk score for HSD0002: 0.153
Risk score for DCV3282: 0.153
Risk score for DXZ2434: 0.406
Risk score for BHT0001: 0.229
Risk score for HSD0001: 0.182
Risk score for FTJ3545: 0.312
Risk score for YSPW073: 0.325
Risk score for JLN3497: 0.458
Risk score for BHT0001: 0.223
High-risk alert triggered for 2007941 (score=0.612)
Risk score for BM7012: 0.418
```

The screenshot shows the AWS S3 console interface. The left sidebar is collapsed, and the main area displays the contents of a folder named 'predictions' within the bucket 'healthcare-project-data-rakshitha'. The 'Objects' tab is selected, showing one file: 'heart_attack_predictions_20251118_202110.csv' (Type: CSV). The file was last modified on November 18, 2025, at 13:31:11 (UTC-07:00) and is 1.1 KB in size. The 'Properties' tab is also visible.

Phase 5: Perform Data Analytics in Amazon Athena

The screenshot shows two consecutive screenshots of the AWS Athena Query Editor interface.

Top Screenshot (Manage settings):

- Query result location and encryption:**
 - Location of query result - optional:** Enter an S3 prefix in the current region where the query result will be saved as an object. (Input: s3://healthcare-project-data-rakshitha/athena/results/)
 - Lifecycle configuration:** You can create and manage lifecycle rules for this bucket. Use Amazon S3 lifecycle rules to store your query results and metadata cost effectively or to delete them after a period of time. (Link: Learn more)
- Expected bucket owner - optional:** Specify the AWS account ID that you expect to be the owner of your query results output location bucket. (Input: Enter AWS account ID)
- Assign bucket owner full control over query results:** Enabling this option grants the owner of the S3 query results bucket full control over the query results. This means that if your query result location is owned by another account, you grant full control over your query results to the other account. (checkbox)
- Encrypt query results:** Encrypts the query results. (checkbox)

Bottom Screenshot (History):

- Data:** Data source: AwsDataCatalog; Catalog: None; Database: default.
- Tables and views:** Tables (0) and Views (0).
- Query 1:** SQL command: CREATE DATABASE IF NOT EXISTS healthcare_analysis_db;
- Run again**, **Explain**, **Cancel**, **Clear**, **Create** buttons.
- Query results:** Completed. Time in queue: 77 ms, Run time: 271 ms, Data scanned: -.
- Completed** status message: Query successful.

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:

The screenshot shows the AWS Athena Query Editor interface with two tabs open: **Query 1** and **Query 2**.

Query 1: SQL command: CREATE EXTERNAL TABLE IF NOT EXISTS heart_attack_processed_data (

```

1 - CREATE EXTERNAL TABLE IF NOT EXISTS heart_attack_processed_data (
2 -   Patient_ID int,
3 -   Heart_Rate double,
4 -   Blood_Pressure double,
5 -   Physical_Activity_Days_Per_Week int,
6 -   Blood_Pressure string,
7 -   Age int,
8 -   Sex string,
9 -   BMI double,
10 -   Diabetes int,
11 -   Family_History int,
12 -   Smoking int,
13 -   Obesity int,
14 -   Total_Calories_Consumption double,
15 -   Exercise_Hours_Per_Week double,

```

SQL: Ln 5\$, Col 49

Run again, **Explain**, **Cancel**, **Clear**, **Create** buttons.

Query results: Completed. Time in queue: 95 ms, Run time: 508 ms, Data scanned: -.

Completed status message: Query successful.

The screenshot shows the AWS CloudShell interface with the Athena Query Editor open. The sidebar on the left lists the Data source (AwsDataCatalog), Catalog (None), and Database (healthcare_analysis_db). Under Tables and views, there is one table named 'heart_attack_processed_data'. The main area displays the SQL code for creating the table:

```

Query 1 : 
Query 2 : 
Query 3 : 
1 CREATE TABLE `heart_attack_processed_data` (
  `age` int,
  `income` double,
  `BRCodes` double,
  `Country` string,
  `Gender` string,
  `Hemisphere` string
) ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  'separatorChar' = "\t",
  'quoteChar' = "\\");

SQL Ln 33, Col 49
Run again Explain Cancel Clear Create

```

The status bar at the bottom indicates: Time in queue: 95 ms, Run time: 308 ms, Data scanned: 0. The message "Query successful." is displayed.

a. Verify the table creation:

The screenshot shows the AWS CloudShell interface with the Athena Query Editor open. The sidebar on the left lists the Data source (AwsDataCatalog), Catalog (None), and Database (healthcare_analysis_db). Under Tables and views, there is one table named 'heart_attack_processed_data'. The main area displays the SQL code for counting rows in the table:

```

Query 1 : 
Query 2 : 
Query 3 : 
1 SELECT COUNT(*) FROM heart_attack_processed_data;

```

The status bar at the bottom indicates: Time in queue: 167 ms, Run time: 785 ms, Data scanned: 3.59 KB. The results section shows the output:

#	_col0
1	20

The message "Results (1)" is displayed above the table.

b. DDL for predictions data:

The screenshot shows the AWS Management Console with the URL us-east-1.console.aws.amazon.com/athena/home?region=us-east-1#/query-editor/history/41e54092-0207-43b2-b264-364931.... The query editor interface is displayed, showing the creation of a new external table:

```

CREATE EXTERNAL TABLE IF NOT EXISTS heart_attack_predictions (
    "Patient ID" string,
    "Heart Attack Risk" double,
    "Risk Status" string,
    "Scored At" string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
    "separatorChar" = ",",
    "quoteChar" = "\"
)
LOCATION 's3://Healthcare-project-data-rakshitha/predictions/'
TBLPROPERTIES ('skip.header.line.count'=1);
  
```

The table has been created successfully, as indicated by the message "Query successful." at the bottom of the editor.

c. Verify the table creation:

The screenshot shows the AWS Management Console with the URL us-east-1.console.aws.amazon.com/athena/home?region=us-east-1#/query-editor/history/3d51a3cc-2dad-41be-8c96-77df7c5.... A simple query is run to verify the table's contents:

```

SELECT COUNT(*) FROM heart_attack_predictions;
  
```

The results show there are 20 rows in the table.

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

The screenshot shows the AWS Management Console with the URL us-east-1.console.aws.amazon.com/athena/home?region=us-east-1#/query-editor/history/36497cd5-a572-495e-a9d4-315ee0.... A more complex query is run to identify the top patients based on heart attack risk:

```

SELECT
    p."Patient ID" AS patient_id,
    p."Heart Attack Risk" AS predicted_risk,
    d.Risk_Status,
    d.ScoredAt,
    timestamp(p.ScoredAt) AS scored_at,
    d.Age,
    d.Sex,
    d.Smoker,
    d.TobaccoLevel,
    d.Blood_Pressure,
    d.Blood_Sugar
FROM heart_attack_predictions p
LEFT JOIN heart_attack_processed_data d
    ON p."Patient ID" = d."Patient ID"
ORDER BY "Heart Attack Risk" DESC
LIMIT 2000;
  
```

The results show the top 20 patients with the highest predicted heart attack risk.

#	patient_id	predicted_risk	Risk_Status	scored_at	Age	Sex	Cholesterol	Blood Pressure	Country
1	ZOO7941	0.631727	HIGH_RISK	2025-11-18 20:31:08.797 UTC	54	Female	297.0	140/77	Germany
2	XB0592	0.541307	HIGH_RISK	2025-11-18 20:31:09.217 UTC	50	Female	303.0	147/94	United States
3	DCY5282	0.51252	HIGH_RISK	2025-11-18 20:31:07.097 UTC	75	Male	122.0	139/92	Italy
4	WVVO966	0.490428	HIGH_RISK	2025-11-18 20:31:06.270 UTC	90	Male	358.0	131/86	Canada
5	YYU9565	0.49022	HIGH_RISK	2025-11-18 20:31:09.437 UTC	60	Male	259.0	135/86	China
6	JLN3497	0.417997	LOW_RISK	2025-11-18 20:31:08.597 UTC	84	Male	385.0	141/87	Canada
7	BMW7812	0.4105	LOW_RISK	2025-11-18 20:31:09.017 UTC	67	Male	268.0	133/95	Argentina
8	GFOB847	0.407645	LOW_RISK	2025-11-18 20:31:08.157 UTC	56	Male	318.0	129/85	Thailand
9	DRB2434	0.4058	LOW_RISK	2025-11-18 20:31:07.523 UTC	69	Male	379.0	135/92	Brazil
10	VTW9069	0.398136	LOW_RISK	2025-11-18 20:31:10.077 UTC	81	Male	297.0	122/88	China
11	FPS0415	0.574975	LOW_RISK	2025-11-18 20:31:09.857 UTC	77	Male	228.0	128/89	Vietnam
12	HSDE288	0.353598	LOW_RISK	2025-11-18 20:31:06.877 UTC	75	Female	373.0	142/102	South Africa
13	YSPO075	0.525236	LOW_RISK	2025-11-18 20:31:08.577 UTC	71	Male	574.0	142/91	United States
14	FTJ5456	0.311907	LOW_RISK	2025-11-18 20:31:07.937 UTC	43	Female	248.0	130/84	Japan
15	XXM0972	0.305896	LOW_RISK	2025-11-18 20:31:09.657 UTC	84	Male	220.0	145/95	Japan
16	CZE1114	0.304597	LOW_RISK	2025-11-18 20:31:10.277 UTC	21	Male	389.0	127/80	Canada

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.

High_risk_count
5

f. Which age groups have higher average predicted risk?

age_group	avg_predicted_risk
20-29	0.266

The screenshot shows the Amazon Athena Query Editor interface. A complex SQL query is displayed in the editor, which includes multiple WHERE clauses and an ORDER BY clause. Below the editor, the 'Results' section displays a table with two columns: 'age_group' and 'avg_predicted_risk'. The table has six rows, with the last row being a summary row labeled '70+'. The table also includes a column for 'patients'.

#	age_group	avg_predicted_risk	patients
1	20-29	0.266	5
2	30-39	0.248	1
3	40-49	0.312	1
4	50-59	0.587	2
5	60-69	0.582	5
6	70+	0.597	8

g. Is shorter sleep associated with higher predicted risk?

The screenshot shows the Amazon Athena Query Editor interface. A complex SQL query is displayed in the editor, which includes multiple WHERE clauses and an ORDER BY clause. Below the editor, the 'Results' section displays a table with three columns: 'sleep_hours_floor', 'avg_sleep', and 'avg_predicted_risk'. The table has three rows, with the last row being a summary row labeled '7.0'. The table also includes a column for 'patients'.

#	sleep_hours_floor	avg_sleep	avg_predicted_risk	patients
1	5.0	5.54	0.376	11
2	6.0	6.28	0.385	8
3	7.0	7.24	0.398	1

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

The screenshot shows the Amazon Athena Query Editor interface. A complex SQL query is displayed in the editor, which includes multiple WHERE clauses and an ORDER BY clause. Below the editor, the 'Results' section displays a table with three columns: 'avg_activity_days', 'avg_heart_rate', and 'avg_predicted_risk'. The table has one row, with the value 3.4 for 'avg_activity_days'. The table also includes a column for 'avg_predicted_risk'.

#	avg_activity_days	avg_heart_rate	avg_predicted_risk
1	3.4	84.6	0.581

The screenshots show the AWS S3 console interface. The top screenshot displays the main bucket page for 'healthcare-project-data-rakshitha' with 7 objects listed. The bottom screenshot shows a detailed view of a specific folder path: 'athena/results/unsaved/2025/11/18/'. This folder contains 18 objects, primarily CSV files and some metadata files, all modified on November 18, 2025.

Name	Type	Last modified	Size	Storage class
112b61e0-611b-48a9-a72c-ab504f4d4f89.csv	csv	November 18, 2025, 13:45:36 (UTC-07:00)	79.0 B	Standard
112b61e0-611b-48a9-a72c-athena-metadata.csv	metadata	November 18, 2025, 13:45:36 (UTC-07:00)	211.0 B	Standard
36497d5f-a572-495e-a9d4-519ee059845.csv	csv	November 18, 2025, 13:43:09 (UTC-07:00)	2.1 kB	Standard
36497d5f-a572-495e-a9d4-519ee059845.csv.metadata	metadata	November 18, 2025, 13:43:09 (UTC-07:00)	489.0 B	Standard
5ab1a3cc-2d4d-41ba-8c96-73d7f512a97.csv	csv	November 18, 2025, 13:42:09 (UTC-07:00)	13.0 B	Standard
5ab1a3cc-2d4d-41ba-8c96-73d7f512a97.csv.metadata	metadata	November 18, 2025, 13:42:09 (UTC-07:00)	67.0 B	Standard
5f2054af-3fb0-499e-a134-50f98c353bf1.csv	csv	November 18, 2025, 13:43:54 (UTC-07:00)	22.0 B	Standard
5f2054af-3fb0-499e-a134-50f98c353bf1.csv.metadata	metadata	November 18, 2025, 13:43:54 (UTC-07:00)	87.0 B	Standard
41654092-0207-43b2-b264-3649315d01c.txt	txt	November 18, 2025, 13:41:40 (UTC-07:00)	0 B	Standard
54733f77-374e-46bf-bacf-500127b5-25cf-453d-a031-9183c2fb710f.txt	txt	November 18, 2025, 13:40:34 (UTC-07:00)	0 B	Standard
500127b5-25cf-453d-a031-9183c2fb710f.txt	txt	November 18, 2025, 13:35:52 (UTC-07:00)	0 B	Standard
92c1968d-152d-4c3a-8854-1b2533333333.txt	txt	November 18, 2025, 13:40:45 (UTC-07:00)	0 B	Standard

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 your 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

The screenshot shows the AWS S3 console with a green success message at the top: "Upload succeeded". Below it, a summary table shows one file uploaded successfully (Succeeded) and zero files failed (Failed). A detailed table below lists the uploaded file: "simulated_vitals.csv" (text/csv, 5.0 KB, Status: Succeeded).

S2: Cluster Creation

The screenshot shows the AWS EMR console with a green success message: "Your cluster 'rakshitha-health-data-processing-cluster' has been successfully created." The cluster details page is displayed, showing the cluster configuration, applications (Amazon EMR version emr-7.12.0), and cluster management (log destination in Amazon S3). The status is "Starting".

This screenshot shows the cluster configuration section of the EMR console. It includes tabs for Properties, Bootstrap actions, Instances (Hardware), Steps, Applications, Configurations, Monitoring, Events, and Tags (0). Under Applications, it shows installed applications like Hadoop 3.4.1, Hive 3.1.3, JupyterEnterpriseGateway 2.6.0, Livy 0.8.0, and Spark 3.5.6.

This screenshot shows the cluster termination and node replacement settings. It includes options for termination option (Automatically terminate cluster after idle time), termination protection (Off), and unhealthy node replacement (On).

This screenshot shows the network and security settings for the cluster, including options for archive log files to Amazon S3 and encryption for logs.

This screenshot shows the status and time information for the cluster, indicating it was created on November 18, 2025, at 12:07 UTC, and has been running for 1 second.

This screenshot shows the list of clusters in the EMR console. It displays three clusters: "rakshitha-health-data-processing-cluster" (Status: Waiting), "rakshithaDemoCluster" (Status: Terminated), and "RakshithademoCluster" (Status: Terminated). The "Create cluster" button is visible at the top right.

```

aws CloudShell us-east-1
$ hadoop fs -ls emr-processing
$ curl -I "emr-processing.s3-226-155-48.compute-1.amazonaws.com"
HTTP/2.0 200 OK
Date: Mon, 18 Nov 2025 12:26:15 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 124
Last-Modified: Mon, 18 Nov 2025 12:26:15 GMT
ETag: "d23e3248013440107a700a40f9b7484c"
Warning: Permanently added "ec2-123-126-151-46.compute-1.amazonaws.com" (IP: 123.126.151.46) to the list of known hosts.
curl: (7) Failed connect to emr-processing.s3-226-155-48.compute-1.amazonaws.com:443; Connection refused
*   Trying 123.126.151.46...
* TCP_NODELAY set
* Connected to emr-processing.s3-226-155-48.compute-1.amazonaws.com (123.126.151.46) port 443 (#0)
* ALPN selected: h2
* TLSv1.3 (OUT), TLS handshake (14), length 144
* [https://aws.amazon.com/linux/amazon-linux-2023]
* Last login: Tue Nov 18 19:52:22 2025
* [root@ip-172-31-64-200 ~]#

```

S3: Processed data in bucket

final_health_dataset_csv/

Name	Type	Last modified	Size	Storage class
_SUCCESS	-	November 18, 2025, 12:42:10 (UTC-07:00)	0 B	Standard
part-00000-7942e546-18b2-430f-931b-bbe9ef8de139-c000.csv	csv	November 18, 2025, 12:42:09 (UTC-07:00)	3.6 KB	Standard

S4: Model training Endpoint

```

# Create a custom endpoint name
endpoint_name = "gb-heart-attack-endpoint-(timestamp)"
print(f"Creating endpoint with name: {endpoint_name} ...")
# Deploy using the model's .deploy() - returns None in newer SDKs,
# so we attach a Predictor manually afterward
sgeb_endpoint = sgeb_model.deploy(
    initial_instance_count=1,
    instance_type="ml.m5.large",
    endpoint_name=endpoint_name,
    sagemaker_session=sagemaker_session
)
# Manually create predictor for private access
# Predictor is required for creating Predictor
predictor = Predictor(endpoint_name=endpoint_name, sagemaker_session=sagemaker_session)
print("Model deployed successfully!")
print(predictor.endpoint_name)
INFO:sagemaker:Ignoring unnecessary instance type: None
INFO:sagemaker:(Creating endpoint with name: sgeb-heart-attack-endpoint-2025-11-18-20-06-53)
Deploying endpoint as an endpoint sgeb-heart-attack-endpoint-2025-11-18-20-06-53 ...
INFO:sagemaker:(Creating endpoint config with name: sgeb-heart-attack-endpoint-2025-11-18-20-06-53)
INFO:sagemaker:(Creating endpoint with name: sgeb-heart-attack-endpoint-2025-11-18-20-06-53)
Model deployed successfully!
Endpoint name: sgeb-heart-attack-endpoint-2025-11-18-20-06-53

```

```

from sagemaker.model import Model
import sagemaker.session
import time
# Define the model object
xgb_model = Model(
    image='449155534075.dkr.ecr.us-east-1.amazonaws.com/xgboost:1.5-1',
    model_data=model_artifact,
    role=role,
    name='xgb-heart-attack-' + timestamp,
    sagemaker_session=sagemaker_session
)

# Create a custom endpoint
xgb_endpoint = xgb_model.create_endpoint(endpoint_name='xgb-heart-attack-endpoint-' + timestamp)
print("Deploying XGBoost model as endpoint: " + endpoint_name)

# Deploy using the model's .deploy() - returns None in newer SDKs
# and returns a Future object which can be used to check status afterwards
xgb_model.deploy(initial_instance_count=1,
                  instance_type='ml.m5.large',
                  endpoint_name=endpoint_name)

# Manually create predictor for easier access
predictor = xgb_model.deployed_predictor(endpoint_name, sagemaker_session=sagemaker_session)
print("Model deployed successfully!")
print(predictor.endpoint)

INFO[sagemaker]: Image url:Ignoring unnecessary instance type: None
INFO[sagemaker]: Creating endpoint with name: xgb-heart-attack-endpoint-2023-11-19-20-06-53
INFO[sagemaker]: Endpoint created with endpoint_name: xgb-heart-attack-endpoint-2023-11-19-20-06-53 .

```

S5: SNS Alert topic

Topic rakshitha-health-alerts created successfully.

You can create subscriptions and send messages to them from this topic.

S6: SNS Email

Heart Health Alert Triggered

AWS Notifications no-reply@sns.amazonaws.com via amazonesons.com to me 1:31PM (0 minutes ago)

5 High-Risk Patients Detected:
WVY0966: 0.49
DCY3282: 0.513
ZOO7941: 0.632
XBI0592: 0.541
YYU9565: 0.49

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://uridefense.com/v3/_https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:637423166030:rakshitha-health-alerts-59c5b-5dc-747ce-a765-4911c281fa&EndpointArn=rakshitha@asu.edu...!IKRxwAvBmrQHCMxEtneXUbqpvJ3HEC30cBFVwxmrLcw-lsTQeC0os7OXLphDG7gZLuERfrVL6CL_f2NZl-HIBNASONES

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://uridefense.com/v3/_https://aws.amazon.com/support_!!IKRxwAvBmrQHCMxEtneXUbqpvJ3HEC30cBFVwxmrLcw-lsTQeC0os7OXLphDG7gZLuERfrVL6CL_f2NZl-HIBNASONES

S7: Athena query

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

The screenshot shows the AWS Lambda interface with the URL <https://us-east-1.console.aws.amazon.com/athena/home?region=us-east-1#/query-editor/history/112b61e0-611b-48a9-af25-eb904fa...>. The query editor displays a completed SQL query:

```

1 SELECT
2   ROUND(AVG(d."Physical Activity Days Per Week"), 2) AS avg_activity_days,
3   ROUND(AVG(d."Heart Rate"), 1) AS avg_heart_rate,
4   ROUND(AVG(d."Predicted Risk"), 3) AS avg_predicted_risk
5 FROM heart_attack_predictions
6 JOIN heart_attack_processed_data d
7 ON p."Patient ID" = d."Patient ID";
  
```

The results section shows one row of data:

#	avg_activity_days	avg_heart_rate	avg_predicted_risk
1	3.4	84.6	0.381

Time in queue: 153 ms Run time: 818 ms Data scanned: 4.70 KB

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-pxXWqmlD0va8QkRFPmXSSI3O8H_UXRK?usp=drive_link