

Mini Project

Big Data Analytics and Visualization Using AWS and PySpark

Submitted by: Varunkumar Sonawane (vsonawa@iu.edu)

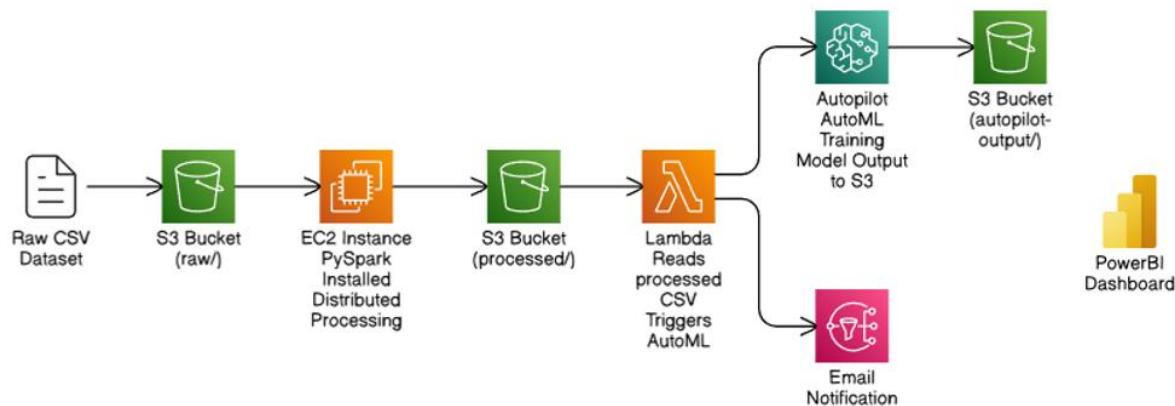
Introduction

This project focuses on building a complete big data analytics pipeline using AWS services and a Linux based PySpark environment. The **goal** of the project is to efficiently process large scale health data, engineer meaningful features, perform distributed analysis, and automate machine learning model training using cloud-based tools. The objective is to demonstrate practical data engineering skills, scalable processing, and automated predictive analytics in a real-world setting.

The **dataset** used in this project contains one hundred thousand records with twenty-one health and lifestyle attributes, including BMI, cholesterol levels, physical activity, sleep duration, blood pressure, calorie intake, and various risk related behaviors. The diversity of the data makes it suitable for cleaning, transformation, aggregation, SQL based exploration, and disease risk prediction.

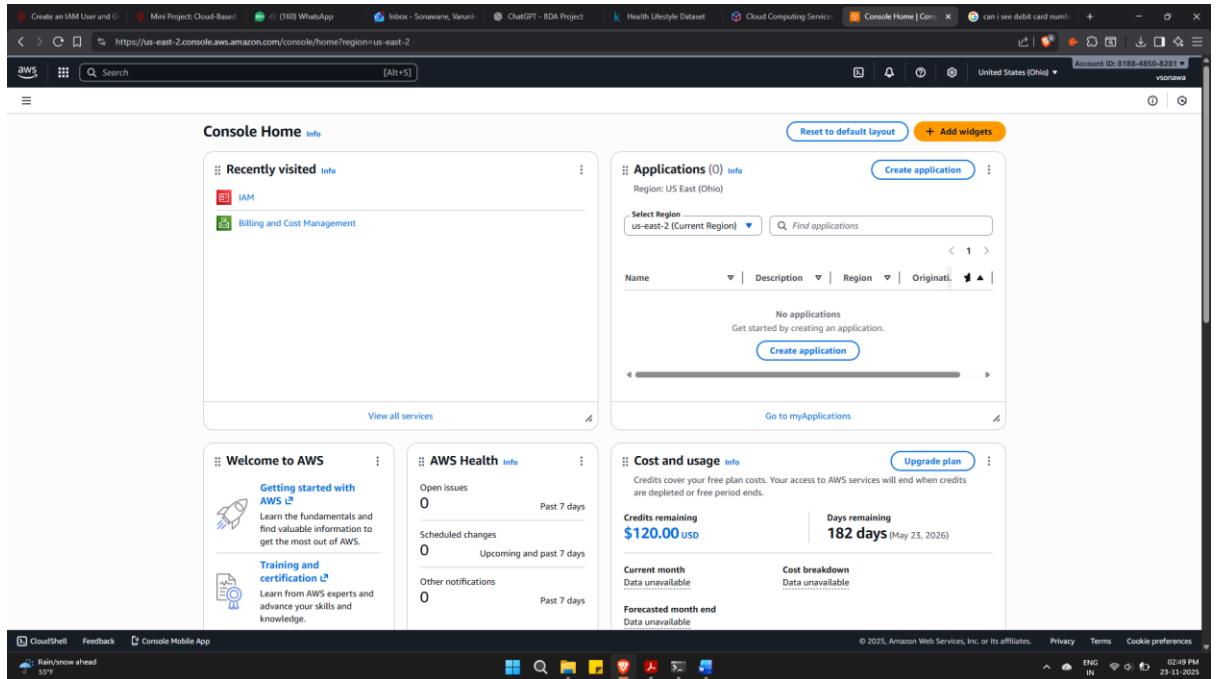
Amazon S3 is used for storing raw and processed data, while an EC2 instance running PySpark handles ingestion and distributed processing. Feature engineering and health metric calculations are performed using PySpark, followed by machine learning with AWS SageMaker Autopilot to automatically build and evaluate models. Power BI Desktop is used to create an interactive dashboard for visualization. The project also includes a fully automated serverless pipeline using S3 triggers, AWS Lambda, SageMaker, and SNS for continuous data processing and AutoML.

The **complete architecture diagram**, showing both the PySpark workflow and the automated pipeline, should be placed **immediately after this introduction** to provide a clear visual overview of the system before moving into the methodology section.



Methodology

1.1 Creating AWS account



1.2 Installing and Configuring AWS CLI

```
Command Prompt
Microsoft Windows [Version 10.0.26200.7171]
(c) Microsoft Corporation. All rights reserved.

C:\Users\VARUN>aws --version
aws-cli/2.0.30 Python/3.7.7 Windows/10 botocore/2.0.0dev34

C:\Users\VARUN>aws configure
AWS Access Key ID [*****KX67]: AKIA35JZMVF4SIHQXJHJ
AWS Secret Access Key [*****2R66]: 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
Default region name [us-east-1]:
Default output format [json]

C:\Users\VARUN>aws s3 ls

C:\Users\VARUN>
```

1.3 Creating Billing alert (Billing and Cost Management)

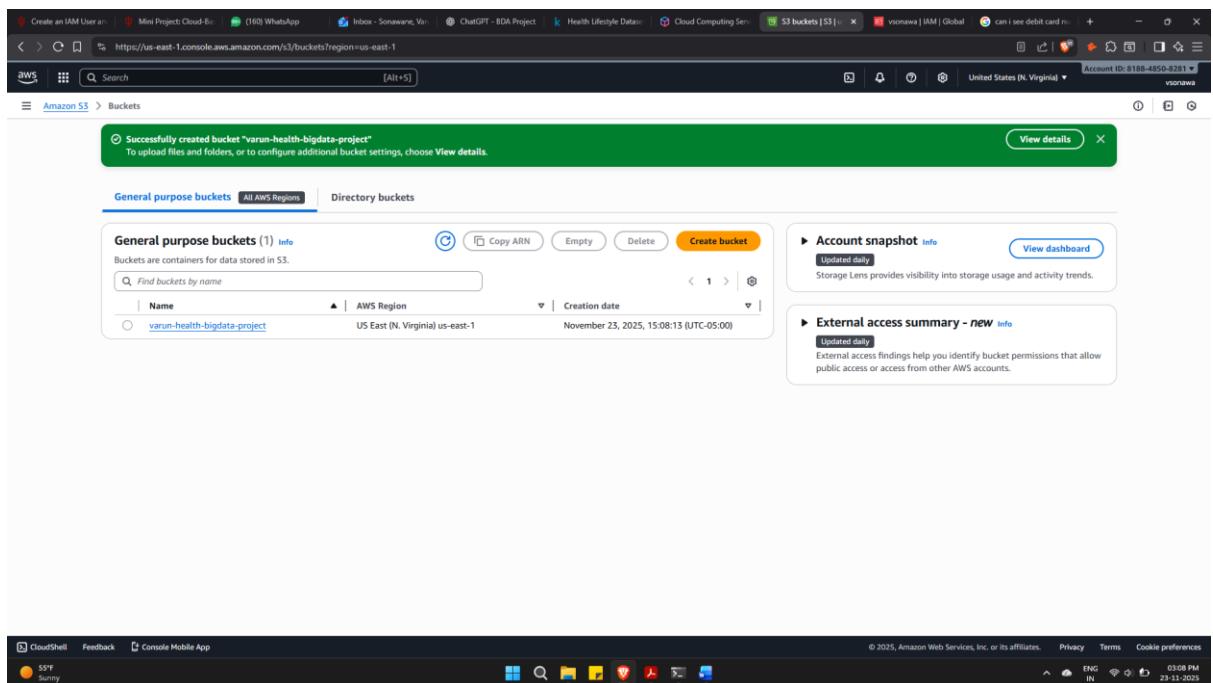
The screenshot shows the AWS Billing and Cost Management console. The left sidebar is titled 'Billing and Cost Management' and includes sections for Home, Getting Started, Dashboards, Billing and Payments, Bills, Payments, Credits, Purchase Orders, Cost and Usage Analysis, Cost Explorer, Cost Explorer Saved Reports, Cost Anomaly Detection, Free Tier, Data Exports, Customer Carbon Footprint Tool, Cost Organization, Cost Categories, Cost Allocation Tags, Billing Conductor, Budgets and Planning, and Budgets Reports. The main content area is titled 'My Zero-Spend Budget' and shows a summary of budget health, current vs. budgeted spending (0.00%), and forecasted vs. budgeted MTD (0.00%). It also displays budget details like budget amount (\$1.00), period (Monthly), start date (2025-11-01), and end date (2025-12-31). The 'Budget history' tab is selected, showing a history section with a link to 'View in AWS Cost Explorer'.

1.4 Creating IAM user (vsonawa) and User group(admins)

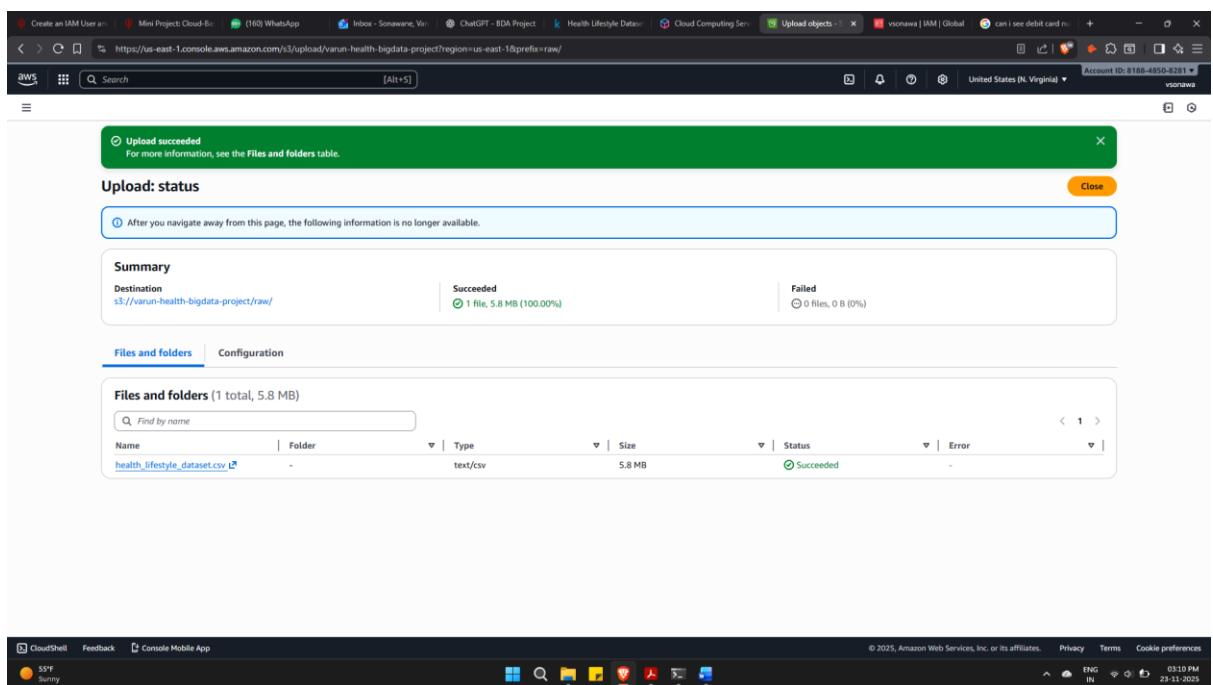
The screenshot shows the AWS IAM console. The left sidebar is titled 'Identity and Access Management (IAM)' and includes sections for Dashboard, Access management (User groups, Users, Roles, Policies, Identity providers, Account settings, Root access management, Temporary delegation requests), Access reports (Access Analyzer, Resource analysis, Unused access, Analyzer settings, Credential report, Organization activity, Service control policies, Resource control policies), and IAM Identity Center (AWS Organizations). The main content area is titled 'vsonawa' and shows a summary of the user's ARN (arn:aws:iam:818848508281:user/vsonawa), creation date (November 23, 2025, 14:59 (UTC-05:00)), and console access status (Enabled without MFA). It also shows the user's access key (Access key 1, Create access key). The 'Permissions' tab is selected, showing a list of attached policies (AdministratorAccess) and a 'Generate policy based on CloudTrail events' section. The bottom of the screen shows the Windows taskbar with various pinned icons.

Environment Setup

- 1. AWS S3 for Data Storage**
- a. Step 1: Create an S3 bucket to store both raw and processed data.**



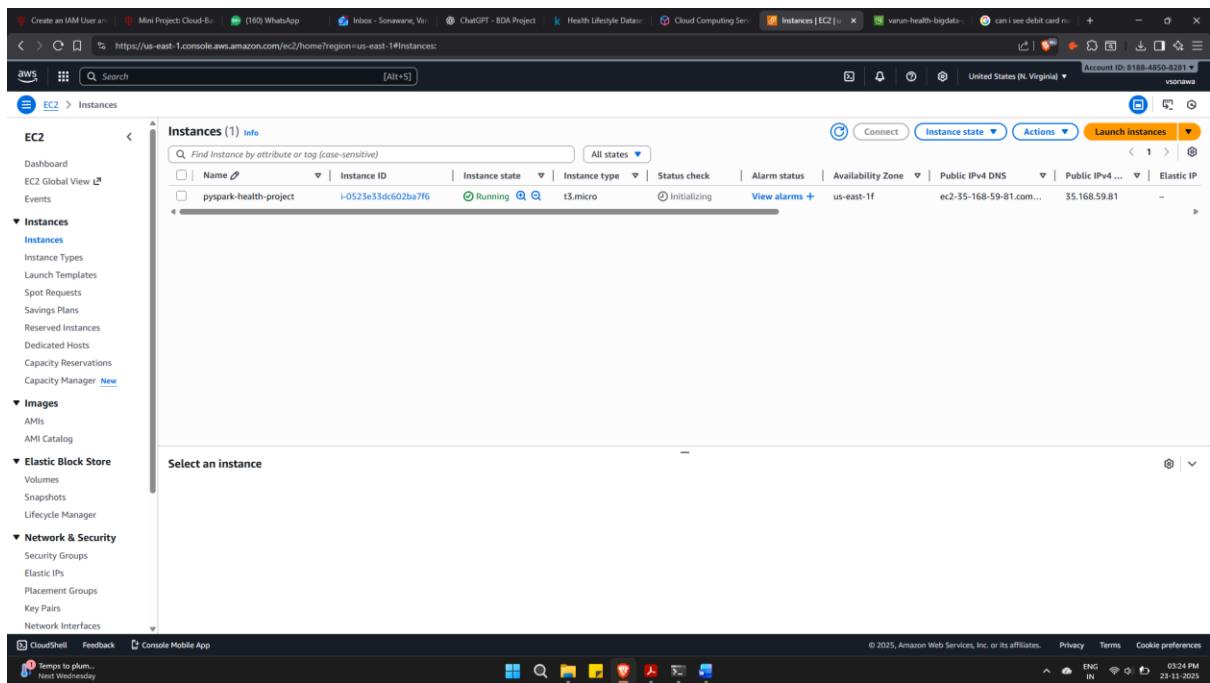
b. Step 2: Upload the raw dataset to the S3 bucket.



2. Linux Environment with PySpark

a. Step 1: Set up a Linux-based environment, either locally or using an AWS EC2 instance.

(instance name : pyspark-health-project)



b. Step 2: Install PySpark for distributed data processing.

1. Installing Java (required for spark)

2. Installing Python, pip

Last metadata expiration check: 0:02:30 ago on Sun Nov 23 20:34:17 2025.
Package python3-3.9.24-1.amzn2023.0.4.x86_64 is already installed.
Dependencies resolved.

Package	Architecture	Version	Repository	Size
Installing:				
python3-pip	x86_64	21.3.1-2.amzn2023.0.14	amazonlinux	1.8 M
Installing weak dependencies:				
libcrypt-compat	x86_64	4.4.33-7.amzn2023	amazonlinux	92 k

Transaction Summary

Install 2 Packages

Total download size: 1.9 M
Installed size: 11 M
Downloading Packages:
(1/2) : libcrypt-compat-4.4.33-7.amzn2023.x86_64.rpm 2.3 Mb/s | 34 kB/s | 1.8 MB 00:00
(2/2) : python3-pip-21.3.1-2.amzn2023.0.14.noarch.rpm 2.3 Mb/s | 1.9 MB 00:00

Total
Running transaction check
Pre-transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing: 1/2
Installing : libcrypt-compat-4.4.33-7.amzn2023.x86_64 2/2
Installing : python3-pip-21.3.1-2.amzn2023.0.14.noarch 2/2
Running scriptlet: python3-pip-21.3.1-2.amzn2023.0.14.noarch 1/2
Verifying : libcrypt-compat-4.4.33-7.amzn2023.x86_64 2/2
Verifying : python3-pip-21.3.1-2.amzn2023.0.14.noarch 2/2
Installed: libcrypt-compat-4.4.33-7.amzn2023.x86_64 python3-pip-21.3.1-2.amzn2023.0.14.noarch
Complete!
[ec2-user@ip-172-31-70-86 ~]\$ python3 --version
python3.9
[ec2-user@ip-172-31-70-86 ~]\$ pip --version
pip 21.3.1 from /usr/lib/python3.9/site-packages/pip (python 3.9)
[ec2-user@ip-172-31-70-86 ~]\$

i-0523e33dc602ba7f6 (pyspark-health-project)

PublicIP: 35.168.59.81 PrivateIP: 172.31.70.86

CloudShell Feedback Console Mobile App © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 03:38 PM 23-11-2025

3. Installing Pyspark

Step 1: Install wget and tar if missing

```
sudo yum install wget -y
```

Step 2: Download Apache Spark

```
wget https://archive.apache.org/dist/spark/spark-3.4.1/spark-3.4.1-bin-hadoop3.tgz
```

Step 3: Extract the file

```
tar xvf spark-3.4.1-bin-hadoop3.tgz
```

Step 4: Move Spark to a permanent location

```
sudo mv spark-3.4.1-bin-hadoop3 /opt/spark
```

Step 5: Add Spark to PATH

```
echo 'export SPARK_HOME=/opt/spark' >> ~/.bashrc  
echo 'export PATH=$SPARK_HOME/bin:$PATH' >> ~/.bashrc  
echo 'export PATH=$SPARK_HOME/sbin:$PATH' >> ~/.bashrc  
source ~/.bashrc
```

```

AWS Lambda
LambdaFunction
Overview Configuration Events Monitoring
[Alt+S]
Search
[Alt+S]

https://us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1&connType=standard&instanceId=1-Oad716e635ec0b5c4&hostUser=ec2-user&sshPort=22&addressFamily=ipv4
Account ID: 8188-4510-3231
Region: United States (N. Virginia)
vsvsaws

spark-3.4.1-bin-hadoop3/jars/spark-launcher_2.12-3.4.1.jar
spark-3.4.1-bin-hadoop3/jars/spark-kvstore_2.12-3.4.1.jar
spark-3.4.1-bin-hadoop3/jars/spark-kubernetes_2.12-3.4.1.jar
spark-3.4.1-bin-hadoop3/jars/spark-hive_2.12-3.4.1.jar
spark-3.4.1-bin-hadoop3/jars/spark-hive_2.12-3.4.1.jar
spark-3.4.1-bin-hadoop3/jars/spark-hive_2.12-3.4.1.jar
spark-3.4.1-bin-hadoop3/jars/spark-core_2.12-3.4.1.jar
spark-3.4.1-bin-hadoop3/jars/spark-catalyst_2.12-3.4.1.jar
spark-3.4.1-bin-hadoop3/jars/snappy-pipeline_2.12-3.4.1.jar
spark-3.4.1-bin-hadoop3/jars/zstd_1.33.jar
spark-3.4.1-bin-hadoop3/jars/api_2.0.6.jar
spark-3.4.1-bin-hadoop3/jars/shims_0.9.39.jar
spark-3.4.1-bin-hadoop3/jars-scala_2.12-2.1.0.jar
spark-3.4.1-bin-hadoop3/jars-scala-reflects_2.12-2.1.0.jar
spark-3.4.1-bin-hadoop3/jars-scala-parser-combinators_2.12-2.1.1.jar
spark-3.4.1-bin-hadoop3/jars-scala-library_2.12-2.17.jar
spark-3.4.1-bin-hadoop3/jars-scala-compiler_2.12-2.17.jar
spark-3.4.1-bin-hadoop3/jars-scala-library-collection-compat_2.12-2.7.0.jar
spark-3.4.1-bin-hadoop3/RELEASE
[ec2-user@ip-172-31-65-103 ~]$ sudo mv spark-3.4.1-bin-hadoop3 /opt/spark
[ec2-user@ip-172-31-65-103 ~]$ echo 'export SPARK_HOME="/opt/spark"' >> ~/.bashrc
[ec2-user@ip-172-31-65-103 ~]$ export PATH=$SPARK_HOME/bin:$PATH >> ~/.bashrc
[ec2-user@ip-172-31-65-103 ~]$ source ~/.bashrc
[ec2-user@ip-172-31-65-103 ~]$ spark-shell
Setting default log level to "WARN".
Consider setting the log level in spark-env.sh instead of using -Dlog4j.rootCategory.
5/11/24 03:45:06 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
spark context Web UI available at http://ip-172-31-65-103.ec2.internal:4040
spark context available as 'sc' (master = local[*], app id = local-1763955508298).
spark session available as 'spark'.
Welcome to
[ec2-user@ip-172-31-65-103 ~]$ 

```

i-0ad716e635ec0b5c4 (pyspark-health-project)
PublicIP: 44.192.67.57 PrivateIP: 172.31.65.103

c. Step 3: Configure AWS CLI to interact with S3 buckets

```

[ec2-user@ip-172-31-73-97 ~]$ sudo yum install awscli -y
Last metadata expiration check: 0:03:03 ago on Mon Nov 24 04:10:44 2025.
Package awscli-2-2.30.4-1.amzn2023.0.1.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-73-97 ~]$ aws configure
AWS Access Key ID [None]: AKIA35JZMVF4SIHQXJHJ
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-1
Default output format [None]: json
[ec2-user@ip-172-31-73-97 ~]$ 

```

Data Pipeline Tasks

Task 1: Data Ingestion from S3

Steps:

1. Use AWS CLI or PySpark's built-in S3 support to load the dataset directly.

```

[ec2-user@ip-172-31-73-97 ~]$ mkdir ~/health_bigdata_project
cd ~/health_bigdata_project
[ec2-user@ip-172-31-73-97 health_bigdata_project]$ aws s3 cp s3://varun-health-bigdata-project/raw/health_lifestyle_dataset.csv .
download: s3://varun-health-bigdata-project/raw/health_lifestyle_dataset.csv to ./health_lifestyle_dataset.csv
[ec2-user@ip-172-31-73-97 health_bigdata_project]$ ls
health_lifestyle_dataset.csv

```

2. Confirm successful ingestion by inspecting the dataset.

```
aws Search [Alt+S] United Sta

Welcome to
version 3.4.1

Using Python version 3.9.24 (main, Oct 20 2025 00:00:00)
Spark context Web UI available at http://ip-172-31-73-97.ec2.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1763957919914).
SparkSession available as 'spark'.
>>> df = spark.read.csv("health_lifestyle_dataset.csv", header=True, inferSchema=True)

>>>
>>> df.show(5)
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| id|age|gender|bmi|daily_steps|sleep_hours|water_intake_l|calories_consumed|smoker|alcohol|resting_hr|systolic_bp|diastolic_bp|cholesterol|family_history|disease_risk|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 1| 56| Male|[20.5]| 4198| 3.9| 3.4| 1602| 0| 0| 97| 161| 111| 240| 0| 0|
| 2| 69| Female|[33.3]| 14359| 9.0| 4.7| 2346| 0| 1| 68| 116| 65| 207| 0| 0|
| 3| 46| Male|[31.6]| 18172| 6.6| 4.2| 1643| 0| 1| 90| 123| 99| 296| 0| 0|
| 4| 32| Female|[38.2]| 15772| 3.6| 2.0| 2460| 0| 0| 71| 165| 95| 175| 0| 0|
| 5| 60| Female|[33.6]| 6037| 3.8| 4.0| 3756| 0| 1| 98| 139| 61| 294| 0| 0|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
only showing top 5 rows

>>> df.printSchema()
root
|-- id: integer (nullable = true)
|-- age: integer (nullable = true)
|-- gender: string (nullable = true)
|-- bmi: double (nullable = true)
|-- daily_steps: integer (nullable = true)
|-- sleep_hours: double (nullable = true)
|-- water_intake_l: double (nullable = true)
|-- calories_consumed: integer (nullable = true)
|-- smoker: Integer (nullable = true)
|-- alcohol: integer (nullable = true)
|-- resting_hr: integer (nullable = true)
|-- systolic_bp: integer (nullable = true)
|-- diastolic_bp: integer (nullable = true)
|-- cholesterol: integer (nullable = true)
|-- family_history: integer (nullable = true)
|-- disease_risk: integer (nullable = true)

>>> df.count()
100000
>>> |
```

Task 2: Data Processing with PySpark

1. **Data Transformation:** Create at least 2 new columns (e.g., `Year` , `Month`) to aid in analysis.

PySpark preprocessing code

```
from pyspark.sql.functions import when, col  
# BMI Category  
df2 = df.withColumn(  
    "bmi_category",  
    when(col("bmi") < 18.5, "Underweight")  
    .when((col("bmi") >= 18.5) & (col("bmi") < 25), "Healthy")  
    .when((col("bmi") >= 25) & (col("bmi") < 30), "Overweight")  
    .otherwise("Obese")  
)
```

```

# High blood pressure column
df2 = df2.withColumn(
  "high_blood_pressure",
  when((col("systolic_bp") >= 140) | (col("diastolic_bp") >= 90), 1).otherwise(0)
)

# Low sleep indicator
df2 = df2.withColumn(
  "low_sleep",
  when(col("sleep_hours") < 7, 1).otherwise(0)
)

# Calorie surplus feature
df2 = df2.withColumn(
  "calorie_surplus",
  when(col("calories_consumed") > 2000, 1).otherwise(0)
)

# Combined risk score
df2 = df2.withColumn(
  "risk_score",
  col("smoker") + col("alcohol") + col("high_blood_pressure") + col("low_sleep") +
  col("calorie_surplus")
)

```

```

| 2| 69|Female|[33,3] | 14359| 9.0| 4.7| 2346| 0| 1| 68| 116| 65| 207| 0| 0| Obese| 0|
| 3| 46| Male|[31,4] | 1817| 6.6| 4.2| 1643| 0| 1| 90| 123| 99| 296| 0| 0| Obese| 1|
| 4| 32|Female|[38,2] | 15772| 3.6| 2.0| 2460| 0| 0| 71| 165| 95| 175| 0| 0| Obese| 1|
| 5| 60|Female|[33,6] | 6037| 3.8| 4.0| 3756| 0| 1| 98| 139| 61| 294| 0| 0| Obese| 0|
| 6| 25| Male|[37,3] | 13495| 5.0| 4.4| 1301| 0| 1| 73| 107| 65| 284| 0| 0| Overweight| 0|
| 7| 78|Female|[37,1] | 16739| 9.5| 4.2| 3478| 1| 0| 90| 110| 102| 201| 0| 0| Obese| 1|
| 8| 38|Female|[18,9] | 1726| 4.8| 1.7| 3212| 0| 1| 64| 113| 109| 197| 0| 0| Healthy| 1|
| 9| 56|Female|[30,2] | 17641| 5.1| 1.5| 3740| 0| 1| 91| 112| 61| 237| 1| 0| Underweight| 0|
| 10| 75| Male|[23,5] | 9730| 4.5| 0.7| 3571| 0| 1| 54| 177| 90| 157| 1| 0| Healthy| 1|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
only showing top 10 rows

```

```

>>> df2.printSchema()
root
|-- id: integer (nullable = true)
|-- age: integer (nullable = true)
|-- gender: string (nullable = true)
|-- bmi: double (nullable = true)
|-- daily_steps: integer (nullable = true)
|-- resting_bp: integer (nullable = true)
|-- water_intake_l: double (nullable = true)
|-- calories_consumed: integer (nullable = true)
|-- alcohol_intake_ml: integer (nullable = true)
|-- resting_hr: integer (nullable = true)
|-- systolic_bp: integer (nullable = true)
|-- diastolic_bp: integer (nullable = true)
|-- cholesterol: integer (nullable = true)
|-- total_blood_pressure_mmhg: integer (nullable = true)
|-- disease_risk: integer (nullable = true)
|-- bmi_category: string (nullable = false)
|-- high_blood_pressure: boolean (nullable = false)
|-- low_sleep_hours: integer (nullable = false)
|-- calorie_surplus: integer (nullable = false)
|-- risk_score: integer (nullable = true)

```

2. Data Aggregation: Compute at least 5 key metrics, such as Total revenue by region, Monthly spending trends, Top 10 customers by transaction value (These metrics may vary depending on the specifics of your dataset, but the goal is to aggregate the data in ways that enable meaningful analysis and decision-making.)

Aggregation 1: Average BMI by gender

This tells us which gender has higher BMI on average.

```
df2.groupBy("gender").avg("bmi").show()
```

```

>>> df2.groupBy("gender").avg("bmi").show()
+-----+-----+
|gender|          avg(bmi)|
+-----+-----+
|Female| 29.027203818080036|
|  Male| 29.02238889332155|
+-----+-----+


>>> 

```

Aggregation 2: Average daily steps by age group

Create age bands and find physical activity patterns.

```
from pyspark.sql.functions import when
```

```
df3 = df2.withColumn(
    "age_group",
    when(col("age") < 30, "Under 30")
    .when(col("age") < 50, "30 to 49")
    .when(col("age") < 65, "50 to 64")
    .otherwise("65 plus")
)
```

```
df3.groupBy("age_group").avg("daily_steps").orderBy("age_group").show()
```

```
>>> from pyspark.sql.functions import when
>>>
>>> df3 = df2.withColumn(
...     "age_group",
...     when(col("age") < 30, "Under 30")
...     .when(col("age") < 50, "30 to 49")
...     .when(col("age") < 65, "50 to 64")
...     .otherwise("65 plus")
... )
>>>
>>> df3.groupBy("age_group").avg("daily_steps").orderBy("age_group").show()
+-----+-----+
|age_group| avg(daily_steps) |
+-----+-----+
| 30 to 49|10474.393635972725|
| 50 to 64|10508.087927657623|
| 65 plus |10509.322770484381|
| Under 30|10416.436905881124|
+-----+-----+
```

Aggregation 3: Percentage of high blood pressure cases

Useful health metric.

```
df2.groupBy("high_blood_pressure").count().show()
```

```
>>> df2.groupBy("high_blood_pressure").count().show()
+-----+-----+
|high_blood_pressure|count|
+-----+-----+
| 1 | 72264 |
| 0 | 27736 |
+-----+-----+
```

Aggregation 4: Average sleep hours by disease risk

Shows correlation between sleep and disease.

```
df2.groupBy("disease_risk").avg("sleep_hours").show()
```

```
>>> df2.groupBy("disease_risk").avg("sleep_hours").show()
+-----+-----+
|disease_risk| avg(sleep_hours) |
+-----+-----+
| 1 | 6.500870230852913 |
| 0 | 6.4887841019432635 |
+-----+-----+
```

Aggregation 5: Average cholesterol by BMI category

This is meaningful for medical analysis.

```
df2.groupBy("bmi_category").avg("cholesterol").show()
```

```
>>> df2.groupBy("bmi_category").avg("cholesterol").show()
+-----+-----+
|bmi_category| avg(cholesterol) |
+-----+-----+
| Overweight| 224.0055787392928|
| Underweight| 223.36733668341708|
|       Obese| 224.44890304008374|
|   Healthy| 224.3610288863938|
+-----+-----+
```

Aggregation 6: Top 10 highest risk individuals

```
df2.orderBy(col("risk_score").desc()).show(10)
```

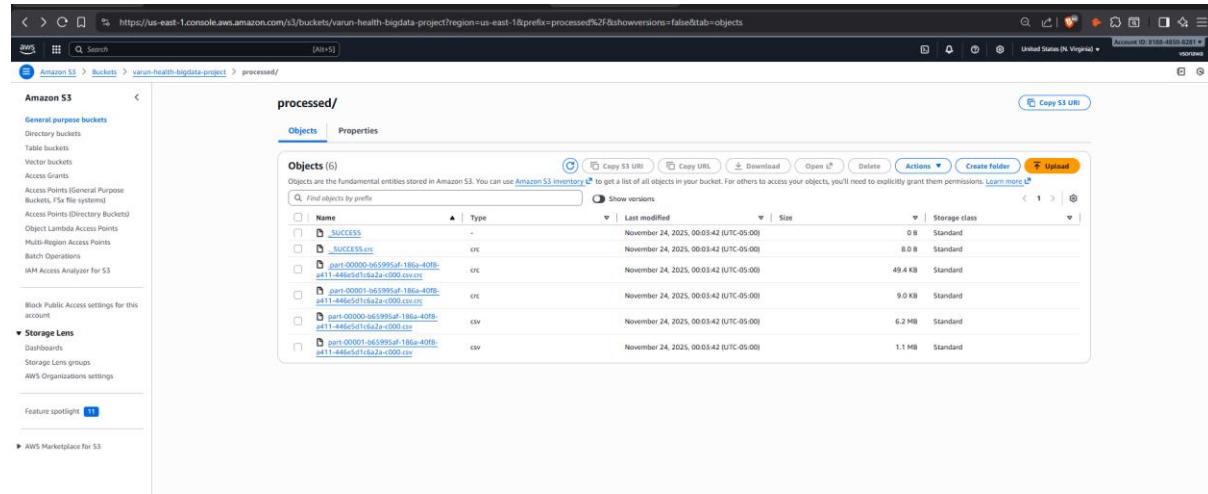
Task 3: Store Processed Data Back to S3

Steps: 1. Export data in CSV or Parquet format.

2. Upload the processed data to a designated S3 location for easy access.

```
>>> df2.write.mode("overwrite").option("header", "true").csv("processed_output")
>>> exit()
[ec2-user@ip-172-31-73-97 health_bigdata_project]$ aws s3 cp processed_output s3://varun-health-bigdata-project/processed/ --recursive
upload: processed_output/_SUCCESScrc to s3://varun-health-bigdata-project/processed/_SUCCESScrc
upload: processed_output/_SUCCESS to s3://varun-health-bigdata-project/processed/_SUCCESS
upload: processed_output/_part-00001-b65995af-186a-40f8-a411-446e5d1c6a2a-c000.csv.crc
upload: processed_output/_part-00000-b65995af-186a-40f8-a411-446e5d1c6a2a-c000.csv.crc
upload: processed_output/_part-00000-b65995af-186a-40f8-a411-446e5d1c6a2a-c000.csv
upload: processed_output/_part-00000-b65995af-186a-40f8-a411-446e5d1c6a2a-c000.csv
[ec2-user@ip-172-31-73-97 health_bigdata_project]$ 

>>> df2.coalesce(1).write.mode("overwrite").option("header", "true").csv("permanent_processed")
>>> exit()
[ec2-user@ip-172-31-73-97 health bigdata_project]$ cd ~/health_bigdata_project/permanent_processed
[ec2-user@ip-172-31-73-97 permanent_processed]$ ls
_SUCCESS part-00000-103fc401-4100-4ee9-b43b-aa75bb29aedf-c000.csv
[ec2-user@ip-172-31-73-97 permanent_processed]$
```



Task 4: Data Analysis Using Spark SQL

Objective: Use SQL to derive insights (Atleast 5 Queries).

Query 1: Average BMI by gender

```
spark.sql("""  
SELECT gender, AVG(bmi) AS avg_bmi  
FROM health  
GROUP BY gender  
""").show()
```

Insight: Helps understand BMI difference across genders.

```
>>> spark.sql("""  
... SELECT gender, AVG(bmi) AS avg_bmi  
... FROM health  
... GROUP BY gender  
... """).show()  
+-----+  
|gender|      avg_bmi|  
+-----+  
|Female|29.027203818080036|  
|  Male| 29.02238889332155|  
+-----+
```

Query 2: Average daily steps by age group

First create an age group:

```
spark.sql("""  
SELECT  
CASE  
    WHEN age < 30 THEN 'Under 30'  
    WHEN age BETWEEN 30 AND 49 THEN '30 to 49'  
    WHEN age BETWEEN 50 AND 64 THEN '50 to 64'  
    ELSE '65 plus'  
END AS age_group,  
AVG(daily_steps) AS avg_steps  
FROM health  
GROUP BY age_group  
ORDER BY age_group  
""").show()
```

Insight: Shows physical activity trends.

```

>>> spark.sql("""
... SELECT
...     CASE
...         WHEN age < 30 THEN 'Under 30'
...         WHEN age BETWEEN 30 AND 49 THEN '30 to 49'
...         WHEN age BETWEEN 50 AND 64 THEN '50 to 64'
...         ELSE '65 plus'
...     END AS age_group,
...     AVG(daily_steps) AS avg_steps
... FROM health
... GROUP BY age_group
... ORDER BY age_group
... """).show()
+-----+
|age_group|      avg_steps|
+-----+
| 30 to 49|10474.393635972725|
| 50 to 64|10508.087927657623|
| 65 plus|10509.322770484381|
| Under 30|10416.436905881124|
+-----+

```

Query 3: Count of high blood pressure cases

```

spark.sql("""
SELECT high_blood_pressure, COUNT(*) AS total_people
FROM health
GROUP BY high_blood_pressure
""").show()

```

Insight: How common high BP is in the dataset.

```

>>> spark.sql("""
... SELECT high_blood_pressure, COUNT(*) AS total_people
... FROM health
... GROUP BY high_blood_pressure
... """).show()
+-----+
|high_blood_pressure|total_people|
+-----+
|                  1|        72264|
|                  0|        27736|
+-----+

```

Query 4: Average sleep hours by disease risk

```

spark.sql("""
SELECT disease_risk, AVG(sleep_hours) AS avg_sleep
FROM health
GROUP BY disease_risk
""").show()

```

Insight: Poor sleep often correlates with higher disease risk.

```

>>> spark.sql("""
... SELECT disease_risk, AVG(sleep_hours) AS avg_sleep
... FROM health
... GROUP BY disease_risk
... """).show()
+-----+-----+
|disease_risk| avg_sleep|
+-----+-----+
| 1 | 6.500870230852913|
| 0 | 6.4887841019432635|
+-----+-----+

```

Query 5: Average cholesterol by BMI category

```

spark.sql("""
SELECT bmi_category, AVG(cholesterol) AS avg_cholesterol
FROM health
GROUP BY bmi_category
ORDER BY avg_cholesterol DESC
""").show()

```

Insight: High BMI categories often have higher cholesterol.

```

>>> spark.sql("""
... SELECT bmi_category, AVG(cholesterol) AS avg_cholesterol
... FROM health
... GROUP BY bmi_category
... ORDER BY avg_cholesterol DESC
... """).show()
+-----+-----+
|bmi_category| avg_cholesterol|
+-----+-----+
| Obese | 224.44890304008374 |
| Healthy | 224.3610288863938 |
| Overweight | 224.0055787392928 |
| Underweight | 223.36733668341708 |
+-----+-----+

```

Query 6: Top 10 highest lifestyle risk individuals

```

spark.sql("""
SELECT id, age, gender, risk_score
FROM health
ORDER BY risk_score DESC
LIMIT 10
""").show()

```

```

>>> spark.sql("""
...  SELECT id, age, gender, risk_score
...  FROM health
...  ORDER BY risk_score DESC
...  LIMIT 10
... """).show()
+---+---+---+-----+
| id|age|gender|risk_score|
+---+---+---+-----+
|84831| 39| Male|      5|
| 161| 65| Male|      5|
|85011| 20| Male|      5|
| 779| 76|Female|      5|
|84837| 42|Female|      5|
| 229| 47|Female|      5|
|84619| 41| Male|      5|
| 48| 64|Female|      5|
|84839| 63| Male|      5|
| 327| 20| Male|      5|
+---+---+---+-----+

```

Query 7: Correlation check between activity and BMI

```

spark.sql("""
SELECT CORR(daily_steps, bmi) AS steps_bmi_correlation
FROM health
""").show()

```

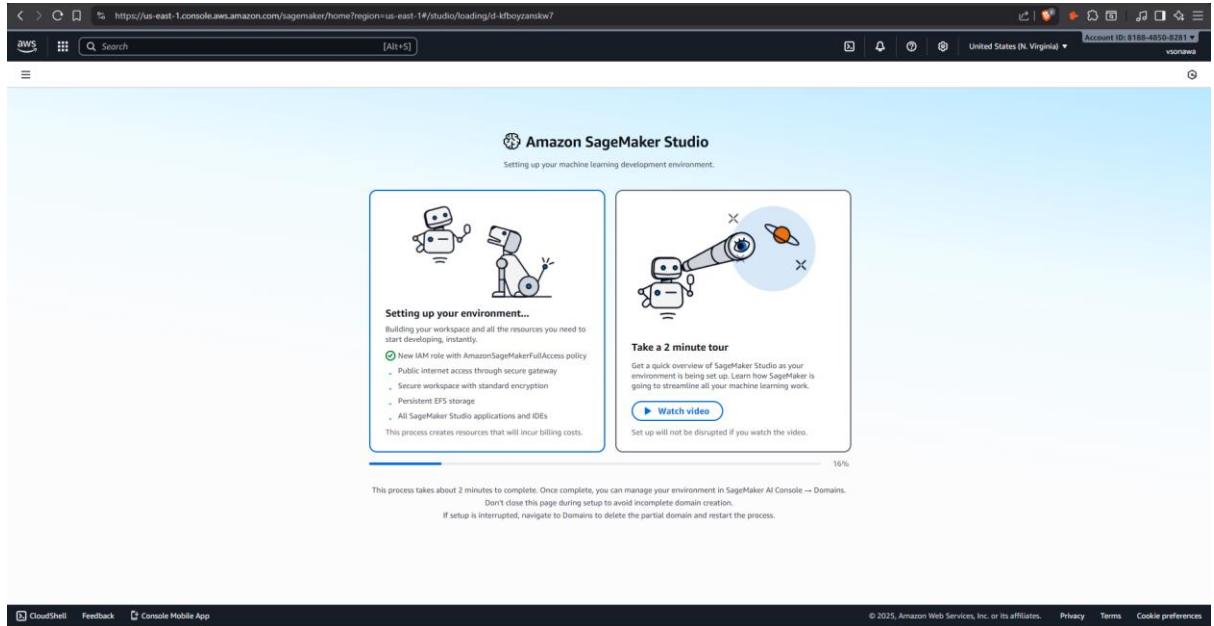
```

>>> spark.sql("""
...  SELECT CORR(daily_steps, bmi) AS steps_bmi_correlation
...  FROM health
... """).show()
+-----+
|steps_bmi_correlation|
+-----+
| 0.001814009171528951|
+-----+

```

Task 5: Machine Learning with AWS SageMaker Autopilot

1. Creating a Domain in Sagemaker



2. Opening Canvas

A screenshot of the Amazon SageMaker AI console. On the left, there's a sidebar with navigation links like 'Dashboard', 'What's new', 'Environment configuration', 'Applications and IDEs' (which is currently selected), 'Model training & customization', 'Training plans', and 'Deployments & inference'. The main content area has a dark background with white text. It features a large heading 'SageMaker Canvas' and the subtext 'Generate accurate machine learning predictions — no code required'. To the right of this, there's a 'Get Started' box with dropdown menus for 'Select Domain' (set to 'QuickSetupDomain-20251127T194533') and 'Select user profile' (set to 'default-20251127T194533'), and a prominent orange 'Open Canvas' button. Below this, there are two boxes: 'Free Demo' (with a 'Demo' button) and 'Free Course' (with a 'Course' button). Both boxes contain brief descriptions and duration information: 'Try free demo' (5 min) and 'Launch Course' (75 min).

3. Canvas Dashboard

4. Uploaded data from S3

Name	Dataset type	Source	Files	Cells (Columns x Rows)	Last updated	Status
bda_health_project_data	V1 Tabular	S3	1	1,600,000 (16 x 100,000)	11/27/2025 8:11 PM	Ready
canva-sample-housing.csv	V1 Tabular	S3	1	10,000 (10 x 1,000)	11/27/2025 7:55 PM	Ready
canva-sample-loans-part-1.csv	V1 Tabular	S3	1	19,000 (19 x 1,000)	11/27/2025 7:55 PM	Ready
canva-sample-loans-part-2.csv	V1 Tabular	S3	1	5,000 (5 x 1,000)	11/27/2025 7:55 PM	Ready
canva-sample-databricks-dolly-15k.csv	V1 Tabular	S3	1	21,758 (2 x 10,879)	11/27/2025 7:55 PM	Ready
canva-sample-diabetic-readmission.csv	V1 Tabular	S3	1	16,000 (16 x 1,000)	11/27/2025 7:55 PM	Ready
canva-sample-retail-electronics-forecasting.csv	V1 Tabular	S3	1	243,000 (6 x 40,500)	11/27/2025 7:56 PM	Ready
canva-sample-shipping-logs.csv	V1 Tabular	S3	1	12,000 (12 x 1,000)	11/27/2025 7:56 PM	Ready
canva-sample-product-descriptions.csv	V1 Tabular	S3	1	600 (5 x 120)	11/27/2025 7:55 PM	Ready
canva-sample-maintenance.csv	V1 Tabular	S3	1	9,000 (9 x 1,000)	11/27/2025 7:55 PM	Ready

5. Creating a new model in canvas autopilot

New! Amazon SageMaker Canvas supports comprehensive data preparation including a conversational interface for data transformation, Gen AI capabilities to access, evaluate, and fine-tune LLMs, configuration of parameters to build models, and the ability to directly deploy models to real-time endpoints. [Learn more](#)

My models

Grid List

Filter by problem type: 2 category prediction

In draft

BDA_health_project

Versions 1

Target disease_risk

Problem type 2 category prediction

Updated 2025-11-27 8:13:43 PM

[View](#)

Resources

6. Selecting target column disease_risk

My models > BDA_health_project > Version 1

+ Create new version

Select Build Analyze Predict Deploy

Select a column to predict

Choose the target column. The model that you build predicts values for the column that you select.

Target column disease_risk

Value distribution

Model type

SageMaker Canvas automatically recommends the appropriate model type for your analysis.

2 category prediction

Your model classifies disease_risk into two categories.

Configure model

Standard build

Preview model

bda_health.project_data

Random sample: 20.0k rows

Manage columns Manage rows Time series View all

Data visualizer

Column name	Data type	Feature type	Missing	Mismatched	Unique	Mode
water_intake_l	123 Numeric	-	0.00% (0)	0.00% (0)	46	2.6
systolic_bp	123 Numeric	-	0.00% (0)	0.00% (0)	90	162
smoker	123 Numeric	Binary	0.00% (0)	0.00% (0)	2	0
sleep_hours	123 Numeric	-	0.00% (0)	0.00% (0)	71	6.5
resting_hr	123 Numeric	-	0.00% (0)	0.00% (0)	50	91
id	123 Numeric	-	0.00% (0)	0.00% (0)	20000	1
gender	A Text	Binary	0.00% (0)	0.00% (0)	2	Female
family_history	123 Numeric	Binary	0.00% (0)	0.00% (0)	2	0
disease_risk	123 Numeric	Binary	0.00% (0)	0.00% (0)	2	0

Total columns: 16 Total rows: 100,000 Total cells: 1,600,000 Show dropped columns

7. Building Model

My models > bda-health-project > Version 1

+ Create new version ⚙️ ⚙️

Select Build Analyze Predict Deploy

Model overview

Your model is being created. Standard build usually takes between 2–4 hours. You can now leave this view.

Time elapsed: 2 min 54 sec | Expected build time: 45 min | Build type: Standard build | Detailed progress: Training models



Legend: bda_health_project_data (blue bar), Total columns: 21 (green bar), Total rows: 100,000 (orange bar), Total cells: 2,100,000 (yellow bar), disease_risk (red bar), 2 category prediction (purple bar)

8. Analysing model

Model Performance Summary

The SageMaker Autopilot experiment completed successfully using a Standard Build.

The best model achieved:

- Accuracy: 50.23 percent
 - F1 Score: 0.339

This indicates **moderate predictive performance** on the disease_risk classification task, suggesting the dataset may need additional feature engineering or balancing.

9. Model Leaderboard

The screenshot shows the AWS SageMaker console under the 'My models > bda-health-project > Version 1' path. The 'Analyze' tab is selected. A search bar at the top right says 'Search leaderboard'. Below it is a table titled 'Model leaderboard' with columns: Model name, F1 | Optimization, Accuracy, AUC, Balanced Accuracy, Precision, Recall, Log Loss, and Inference latency (seconds). The table lists several models, with the first one being the 'Default model'. The first few rows of the table are:

Model name	F1 Optimization	Accuracy	AUC	Balanced Accuracy	Precision	Recall	Log Loss	Inference latency (seconds)
ULL-t10818848508281Canvas1764295526416	33.898%	50.237%	0.504	50.627%	25.287%	51.400%	0.693	0.102
FULL-t8818848508281Canvas1764295526416	33.925%	50.297%	0.504	50.667%	25.317%	51.400%	0.693	0.104
FULL-t7818848508281Canvas1764295526416	33.925%	50.297%	0.504	50.667%	25.317%	51.400%	0.693	0.104
FULL-t6818848508281Canvas1764295526416	33.925%	50.297%	0.504	50.667%	25.317%	51.400%	0.693	0.105
FULL-t5818848508281Canvas1764295526416	33.196%	52.247%	0.505	50.756%	25.429%	47.795%	0.693	0.114
FULL-t4818848508281Canvas1764295526416	33.196%	52.247%	0.505	50.756%	25.429%	47.795%	0.693	0.143
FULL-t3818848508281Canvas1764295526416	33.196%	52.247%	0.505	50.756%	25.429%	47.795%	0.693	0.116
FULL-t2818848508281Canvas1764295526416	33.196%	52.247%	0.505	50.756%	25.429%	47.795%	0.693	0.127
FULL-t1818848508281Canvas1764295526416	33.925%	50.297%	0.504	50.667%	25.317%	51.400%	0.693	0.105
-LI-FULL-t9818848508281Canvas1764295526416	11.189%	72.061%	0.504	50.303%	26.526%	7.090%	0.661	0.141

Model Leaderboard Findings

Autopilot evaluated multiple algorithms and displayed them on the Model Leaderboard. The best-performing model was automatically selected based on the optimization metric (F1 score). The leaderboard provided a comparison of candidate models along with their key metrics, allowing quick identification of the most effective model for this classification task.

10. Matrix Table

The screenshot shows the AWS SageMaker console under the 'My models > bda-health-project > Version 1' path. The 'Analyze' tab is selected. At the top, it shows 'Model status' as 'Standard build' and 'Accuracy' as 50.237% and 'F1' as 0.339. A note below says 'The model predicts the correct Disease_risk 50.237% of the time.' To the right are 'Predict' and 'Deploy' buttons. Below this is a table with tabs for 'Overview', 'Scoring', and 'Advanced metrics'. The 'Advanced metrics' tab is selected, showing values for Positive Class (1), F1 (33.302%), Accuracy (50.33%), Precision (24.96%), Recall (50.02%), and AUC-ROC (0.501). Further down are sections for 'Metrics table' and 'Confusion matrix'.

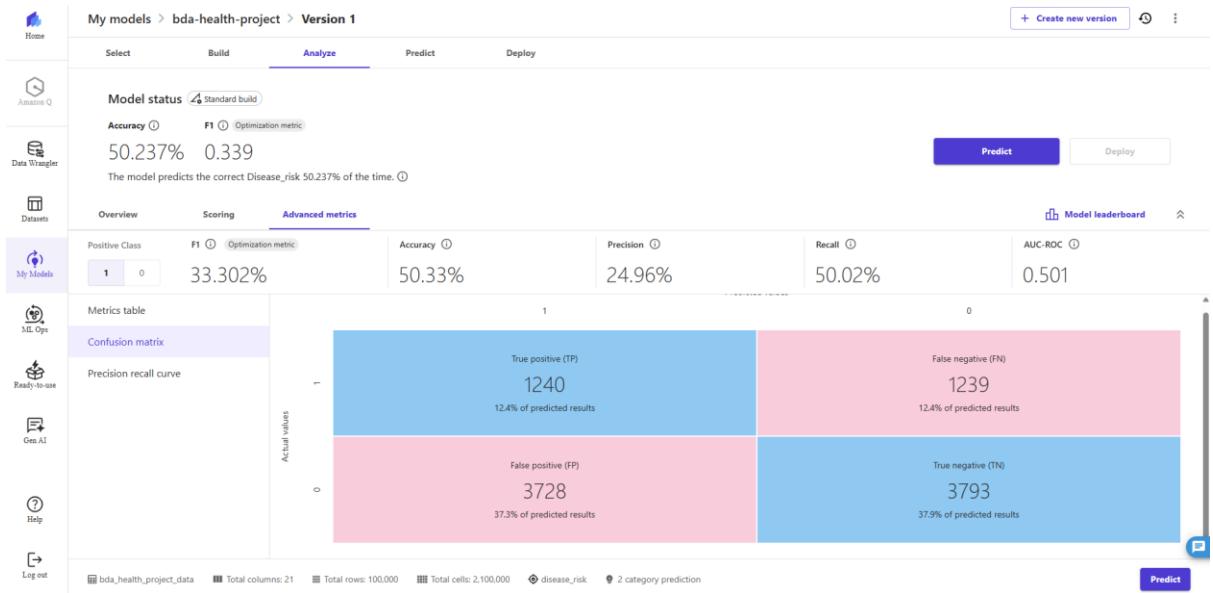
Positive Class	F1	Accuracy	Precision	Recall	AUC-ROC
1	33.302%	50.33%	24.96%	50.02%	0.501

Metrics table

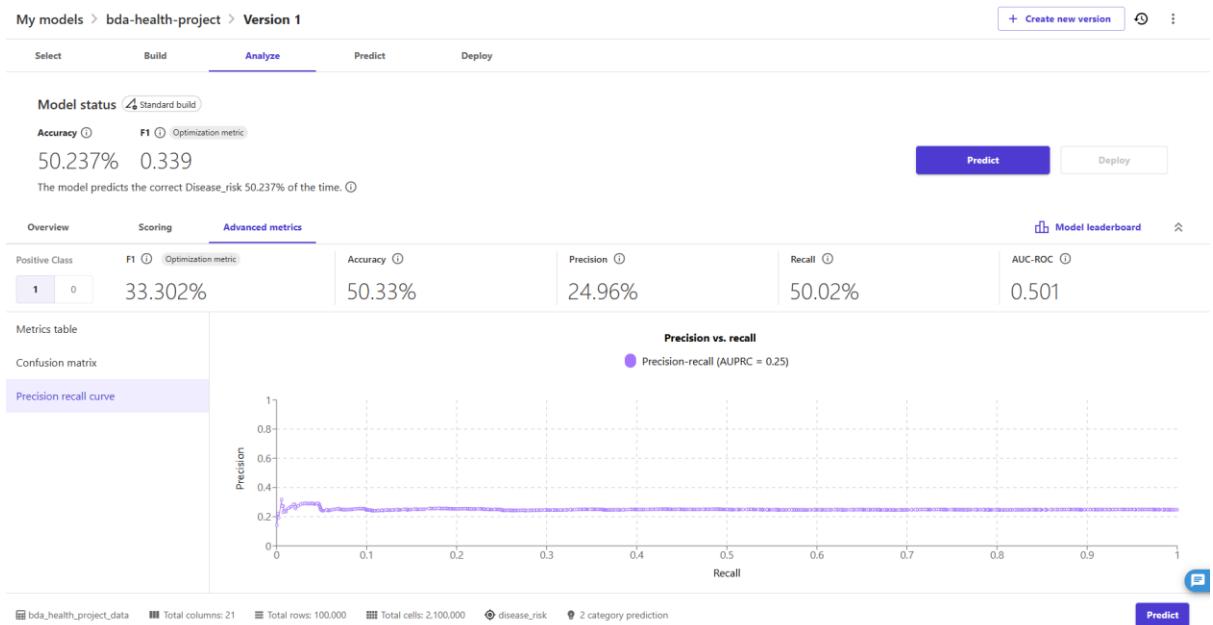
Metric name	Value
precision	0.250
recall	0.500
accuracy	0.503
f1	0.333
auc	0.501

bda_health_project_data Total columns: 21 Total rows: 100,000 Total cells: 2,100,000 disease_risk 2 category prediction

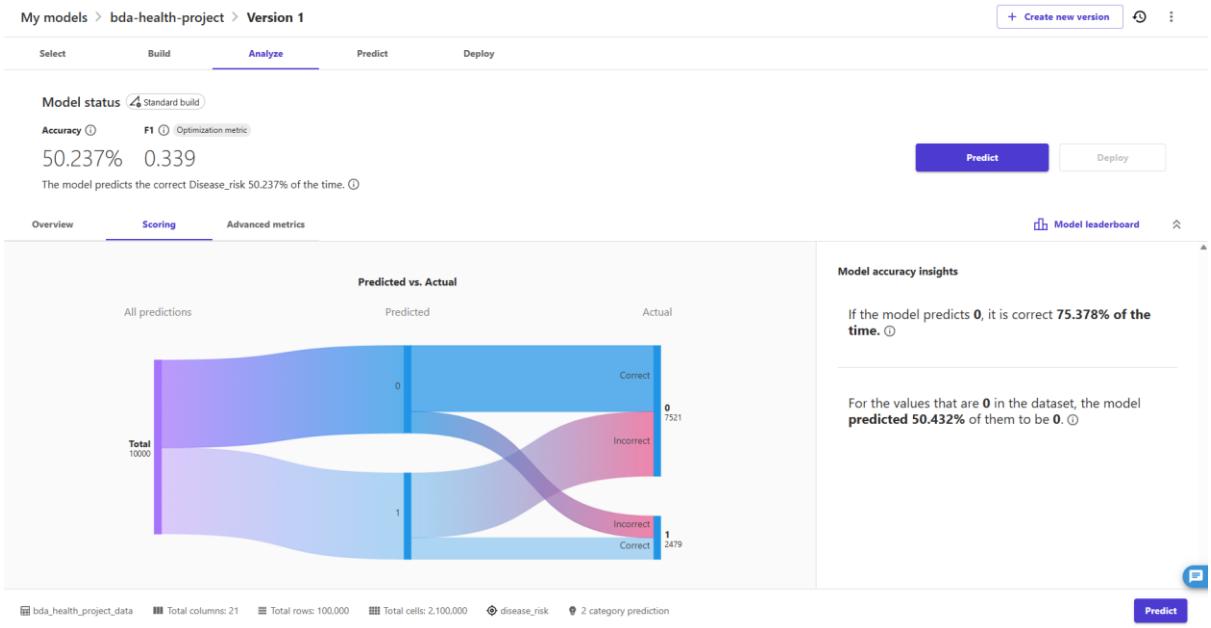
11. Confusion Matrix



12. Precision Recall



13. Scoring



Advanced Evaluation Metrics

The confusion matrix and ROC curve helped assess the model's classification quality. The confusion matrix revealed the distribution of correct and incorrect predictions, highlighting areas where the model struggled to differentiate between risk classes. The ROC curve indicated the trade-off between true positive and false positive rates, offering additional insights into overall model performance beyond simple accuracy.

Ethical Considerations

A. Bias in Training Data

- The dataset includes sensitive attributes like **gender**, **family_history**, **smoker**, and **sleep patterns**, which may introduce bias.
- If these categories are **not evenly distributed**, the model may give unfair predictions for underrepresented groups.
- Strongly correlated features (such as BMI or smoking status) can cause the model to **over-emphasize certain lifestyle habits**, reducing fairness.
- The model accuracy and F1 score indicate that additional checks (class balancing, feature selection) may be needed to **prevent biased outcomes**.

B. Privacy Concerns

- Health and lifestyle data is sensitive and must be stored securely in **Amazon S3 with encryption**.
- Access should be restricted using **IAM least-privilege permissions** to avoid unauthorized viewing of personal data.
- Identifiable fields (like **ID**) should be removed or anonymized to protect individuals' privacy.
- All processing in SageMaker should follow standard **health data privacy practices**, ensuring no personal identity or confidential information is exposed.

Visualization

Visualization 1: Average of risk_score by gender and disease_risk

Chart Type: Clustered Column Chart

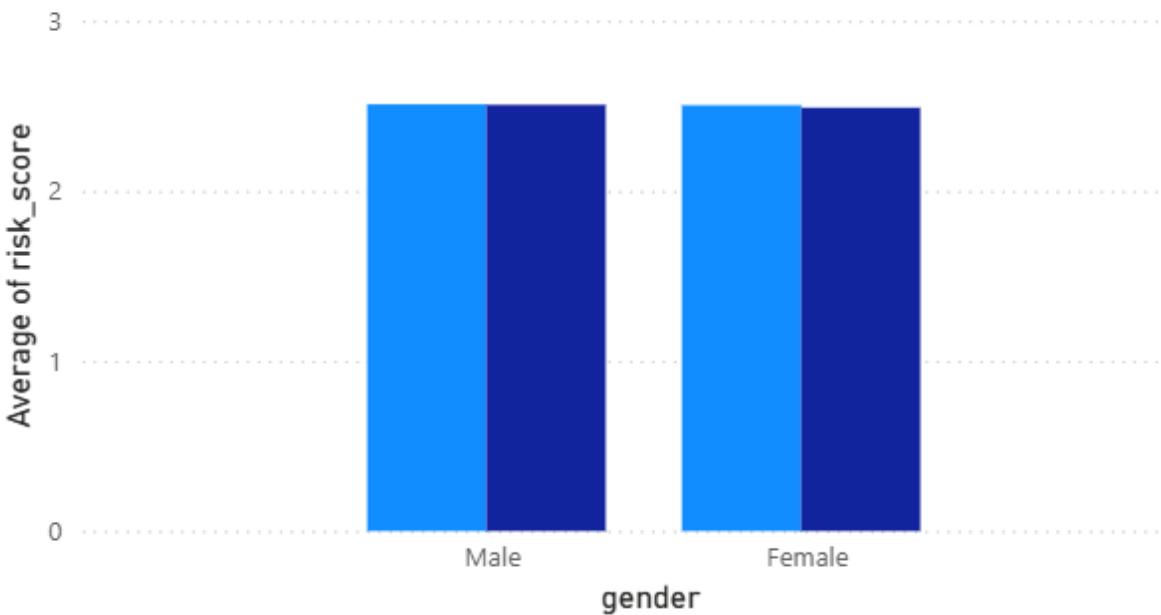
Description: This visualization displays the average risk score segmented by gender (Male and Female) and disease risk status (0 = No disease risk, 1 = Has disease risk). The chart shows two bars for each gender - a light blue bar representing individuals without disease risk (0) and a dark blue bar representing those with disease risk (1). Both males and females show similar average risk scores of approximately 2.5 across both disease risk categories, indicating consistent risk distribution regardless of gender.

Key Insights:

- Males and females have nearly identical average risk scores
- Disease risk status shows minimal variation in risk scores between the two groups
- Both genders show average risk scores between 2-3 on the scale

Average of risk_score by gender and disease_risk

disease_risk ● 0 ● 1



Visualization 2: Count of risk_score and Average of cholesterol by age

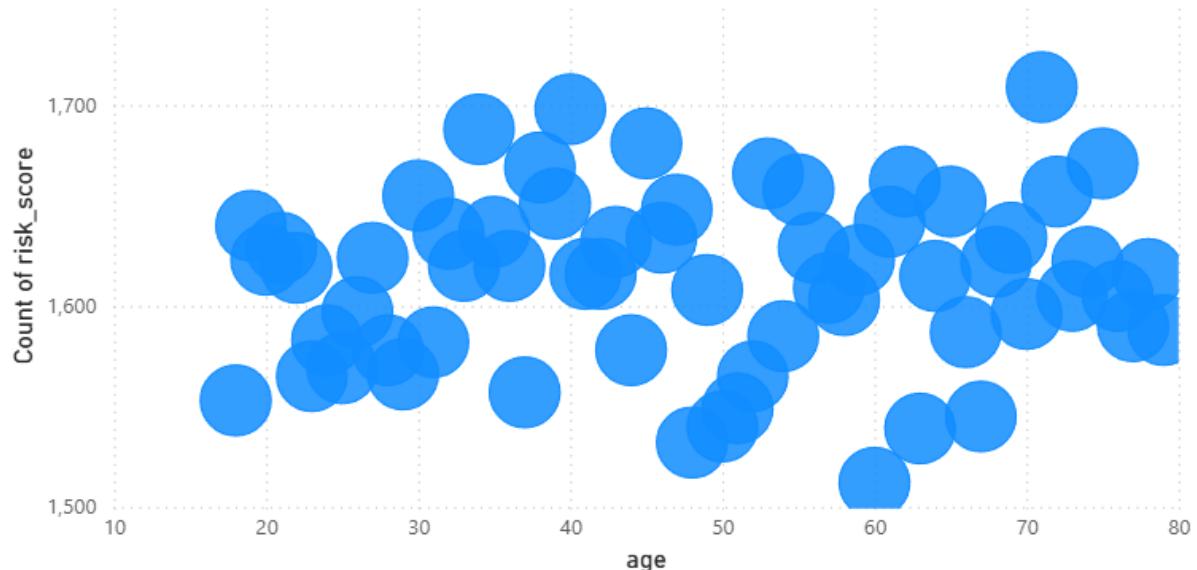
Chart Type: Scatter Plot (Bubble Chart)

Description: This scatter plot illustrates the relationship between age (X-axis), count of risk scores (Y-axis represented by bubble size), and average cholesterol levels. Each blue bubble represents an age group, with the size indicating the count of individuals at that age. The visualization spans ages from approximately 10 to 80 years, with count values ranging from 1,500 to 1,700. The bubble distribution shows relatively even representation across all age groups.

Key Insights:

- Sample size is fairly consistent across all age groups
- Age distribution is well-balanced throughout the dataset
- Most age groups have between 1,550-1,700 records
- Cholesterol levels can be analyzed in relation to age through bubble positioning

Count of risk_score and Average of cholesterol by age



Visualization 3: Count of id by bmi_category

Chart Type: Donut Chart

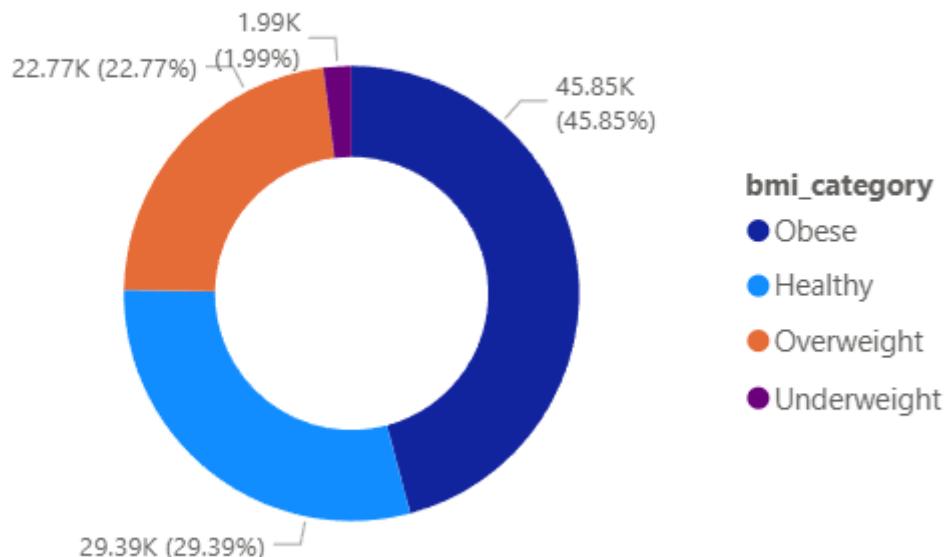
Description: This donut chart presents the distribution of the population across four BMI categories: Obese (dark blue), Healthy (light blue), Overweight (orange), and Underweight (purple). The chart shows that:

- **Obese:** 45,854 individuals (45.85%)
- **Healthy:** 29,391 individuals (29.39%)
- **Overweight:** 22,777 individuals (22.77%)
- **Underweight:** 1,990 individuals (1.99%)

Key Insights:

- Nearly half (45.85%) of the population falls into the obese category
- About one-third (29.39%) maintain a healthy BMI
- Underweight individuals represent less than 2% of the population
- Combined overweight and obese categories represent 68.62% of the population

Count of id by bmi_category



Visualization 4: BMI Category Matrix by Disease Risk

Chart Type: Matrix/Table

Description: This matrix visualization cross-tabulates BMI categories with disease risk status (0 = No disease risk, 1 = Has disease risk). The table displays:

- **Healthy:** 8,135 without disease risk, 21,256 with disease risk (Total: 29,391)
- **Obese:** 12,792 without disease risk, 33,062 with disease risk (Total: 45,854)
- **Overweight:** 6,250 without disease risk, 16,515 with disease risk (Total: 22,765)
- **Underweight:** 559 without disease risk, 1,431 with disease risk (Total: 1,990)
- **Overall Total:** 27,736 without disease risk, 72,264 with disease risk (100,000 total)

Key Insights:

- 72.26% of the population has disease risk (72,264 out of 100,000)
- Obese individuals have the highest disease risk count (33,062)
- Even healthy BMI individuals show significant disease risk (21,256)
- Disease risk is present across all BMI categories

bmi_category	0	1	Total
Healthy	8135	21256	29391
Obese	12792	33062	45854
Overweight	6250	16515	22765
Underweight	559	1431	1990
Total	27736	72264	100000

Visualization 5: Key Performance Indicator Cards

Chart Type: KPI Cards (4 cards)

Card 1 - Average of BMI: 29.02

- Displays the overall average BMI across the entire population
- A BMI of 29.02 falls into the "Overweight" category (25-29.9)
- Indicates the population generally trends toward higher BMI values

Card 2 - Average of daily_steps: 10.48K

- Shows the average daily step count is 10,480 steps
- Exceeds the commonly recommended 10,000 steps per day
- Suggests moderate to good physical activity levels in the population

Card 3 - Average of sleep_hours: 6.49

- Displays average sleep duration of 6.49 hours per night
- Falls below the recommended 7-9 hours of sleep for adults
- May indicate potential sleep deficiency across the population

Card 4 - Average of risk_score: 2.50

- Shows overall average health risk score of 2.50
- On a scale where higher values indicate greater risk
- Represents a moderate risk level across the population

Combined Insights from KPI Cards:

- Despite adequate physical activity (10,480 steps), the population shows elevated BMI (29.02)
- Insufficient sleep (6.49 hours) may contribute to overall health risk
- The moderate risk score (2.50) aligns with the high percentage of individuals with disease risk

29.02

Average of bmi

10.48K

Average of daily_steps

6.49

Average of sleep_hours

2.50

Average of risk_score

Overall Dashboard Summary

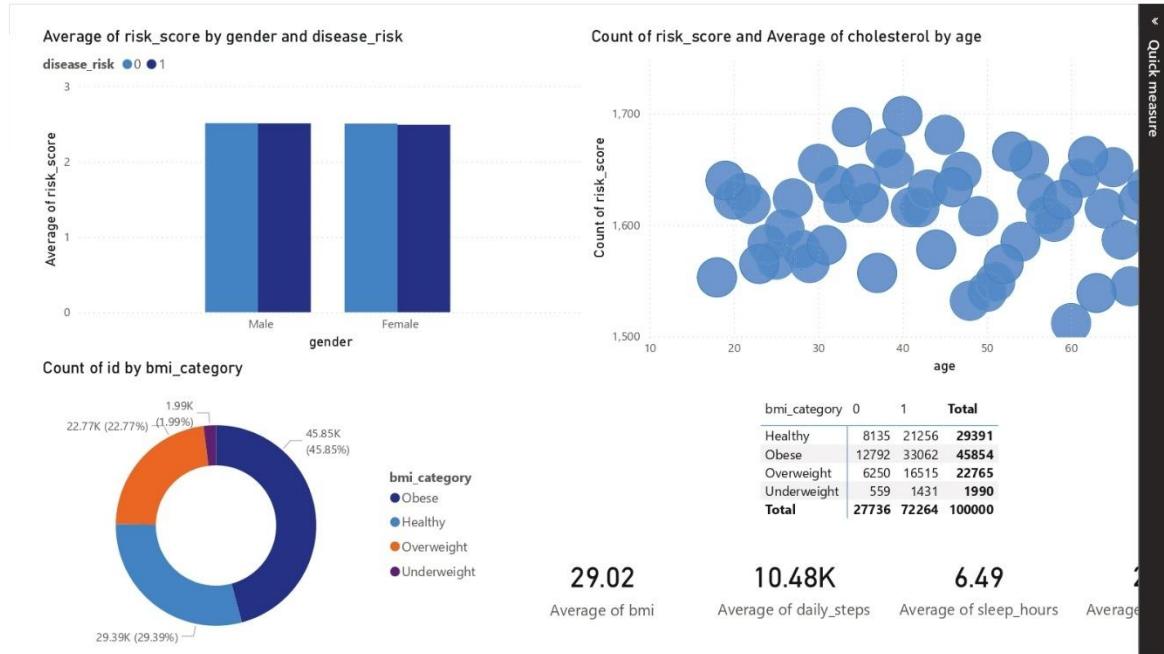
Dashboard Title: Health Risk Analysis Dashboard

Purpose: This interactive dashboard provides comprehensive insights into the health status and risk factors of a population of 100,000 individuals, analyzing the relationships between demographics, lifestyle factors, BMI categories, and disease risk.

Key Findings:

1. The population shows significant health concerns with 72% having disease risk
2. BMI distribution skews heavily toward overweight/obese categories (68.6%)
3. Physical activity levels are adequate, but sleep duration is below recommended levels
4. Disease risk is distributed across all BMI categories, not just obese individuals
5. Gender shows minimal impact on risk scores
6. Age distribution is balanced, allowing for fair comparison across age groups

Data Source: Combined health dataset (combine.csv) containing 100,000 records with 21 health and lifestyle variables



Technical Notes

Tools Used: Microsoft Power BI Desktop

Data Processing:

- Dataset imported from CSV format
- No data transformation required (data was pre-processed in PySpark)
- Total records: 100,000
- Total fields analyzed: 21 variables

Visualizations Created: 5 main visualizations plus 4 KPI cards

Automation

Creating an SNS topic pipeline-alerts and subscription to email in AWS Console

The screenshot shows the AWS SNS Topics page. A blue banner at the top indicates a new feature: "Amazon SNS now supports High Throughput FIFO topics. Learn more". Below this, a green banner says "Topic pipeline-alerts created successfully. You can create subscriptions and send messages to them from this topic." The main section is titled "pipeline-alerts" and contains a "Details" card with the following information:

- Name: pipeline-alerts
- ARN: arn:aws:sns:us-east-1:818848508281:pipeline-alerts
- Type: Standard
- Topic owner: 818848508281

Below the details card are tabs for "Subscriptions", "Access policy", "Data protection policy", "Delivery policy (HTTP/S)", "Delivery status logging", "Encryption", "Tags", and "Integrations". The "Subscriptions" tab is selected, showing a table with one row: "No subscriptions found. You don't have any subscriptions to this topic." There is a "Create subscription" button at the bottom of the table.

Creating IAM role for AWS Lambda

- **AmazonS3FullAccess**
(or AmazonS3ReadOnlyAccess + one custom PutObject policy)
- **AmazonSageMakerFullAccess**
(this includes CreateAutoMLJob)
- **AmazonSNSFullAccess**
(for publishing messages)
- **CloudWatchLogsFullAccess**
(Lambda log permissions)

The screenshot shows the AWS IAM Roles page. A green banner at the top says "Role lambda-role-bda-project created." The main section is titled "lambda-role-bda-project" and contains a "Summary" card with the following information:

- Creation date: November 29, 2025, 20:19 (UTC-05:00)
- Last activity: -
- ARN: arn:aws:iam::818848508281:role/lambda-role-bda-project
- Maximum session duration: 1 hour

Below the summary card are tabs for "Permissions", "Trust relationships", "Tags", "Last Accessed", and "Revoke sessions". The "Permissions" tab is selected, showing a table of attached policies:

Policy name	Type	Attached entities
AmazonS3FullAccess	AWS managed	2
AmazonSageMakerFullAccess	AWS managed	3
AmazonSNSFullAccess	AWS managed	1
CloudWatchLogsFullAccess	AWS managed	1

At the bottom of the "Permissions" section is a "Permissions boundary (not set)" card.

Also created lambda policy to pass the SageMaker execution role to Autopilot.

The screenshot shows the AWS IAM 'Edit policy' page for 'lambda-policy-bda-project'. It displays a 'Policy editor' section with the following JSON code:

```
1 Version: "2012-10-17",
2 Statement: [
3     {
4         Effect: "Allow",
5         Action: "iam:PassRole",
6         Resource: "arn:aws:iam::818848508281:role/AmazonSageMaker-ExecutionRole-20251127T194534"
7     }
8 ]
9
10
```

The right side of the screen shows a 'Select a statement' dropdown and a '+ Add new statement' button.

Created a Lambda Function

The screenshot shows the AWS Lambda 'Function overview' page for 'bda-health-project-pipeline'. It displays the function name 'bda-health-project-pipeline' and its ARN. The 'Code' tab is selected, showing the function code in a code editor:

```
lambda_function.py
import os
import uuid
from datetime import datetime
s3 = boto3.client("s3")
sagemaker = boto3.client("sagemaker")
sns = boto3.client("sns")
```

The sidebar includes tabs for Code, Test, Monitor, Configuration, Aliases, and Versions.

Added a new Trigger for s3 (This allows code to read .csv file from S3 bucked /raw folder)

The screenshot shows the AWS Lambda 'Configuration' page for 'bda-health-project-pipeline'. The 'Triggers' section lists a single trigger:

S3: varun-health-bigdata-project
arnaws:s3::varun-health-bigdata-project
Event types: s3:ObjectCreated:
isComplexStatement: No
Notification name: c2f611db-0c2f-4308-a60a-267b879471cf
Prefix: raw/
Service principal: s3.amazonaws.com
Source account: 818848508281
Statement ID: lambda-608f4504-c28c-4385-82ee-663a93409a85
Suffix: .csv

Attached the new created IAM role for AWS Lambda

The screenshot shows the AWS Lambda function configuration page for 'bda-health-project-pipeline'. On the left, a sidebar lists various configuration options: General configuration, Triggers, Permissions (selected), Destinations, Function URL, Environment variables, Tags, VPC, RDS databases, Monitoring and operations tools, Concurrency and recursion detection, Asynchronous invocation, Code signing, File systems, and State machines. The main panel is titled 'Execution role' and shows the role name 'lambda-role-bda-project'. It includes a 'Resource summary' section with a dropdown menu set to 'AWS Application Auto Scaling' (10 actions, 1 resource). Below this are tabs for 'By action' and 'By resource', with 'By resource' selected. A table lists actions under 'All resources': Allow: application-autoscaling:RegisterScalableTarget, Allow: application-autoscaling:PutScalingPolicy, Allow: application-autoscaling:DescribeScalingActivities, Allow: application-autoscaling:DescribeScalingPolicies, Allow: application-autoscaling:DeleteScalingPolicy, Allow: application-autoscaling:PutScheduledAction, Allow: application-autoscaling:DeleteScheduledAction, Allow: application-autoscaling:DescribeScalableTargets, Allow: application-autoscaling:DescribeScheduledActions, and Allow: application-autoscaling:DeregisterScalableTarget. A note at the bottom states: 'Lambda obtained this information from the following policy statements: • Managed policy AmazonSageMakerFullAccess, statement AllowAWSServiceActions'.

Added Environment variables

The screenshot shows the AWS Lambda function configuration page for 'bda-health-project-pipeline'. The sidebar on the left is identical to the previous screenshot. The main panel is titled 'Configuration' and shows the 'Environment variables' section. It displays three environment variables: BUCKET_NAME (varun-health-bigdata-project), SAGEMAKER_ROLE (arn:aws:iam::818848508281:role/service-role/AmazonSageMaker-ExecutionRole-20251127T194534), and SNS_TOPIC (arn:aws:sns:us-east-1:818848508281:pipeline-alerts). A note above the table states: 'The environment variables below are encrypted at rest with the default Lambda service key.' An 'Edit' button is located in the top right corner of the environment variables table.

Wrote and Deployed the Python Script

The screenshot shows the AWS Lambda function editor. The left sidebar lists 'LAMBDA' and 'TEST EVENTS'. Under 'ENVIRONMENT VARIABLES', there are three entries: BUCKET_NAME, SAGEMAKER_ROLE, and SNS_TOPIC, each with a corresponding ARN value. The main area displays the code for 'lambda_function.py'. The code imports boto3, pandas, os, uuid, and datetime. It defines an S3 client (s3), a SageMaker client (sagemaker), and an SNS client (sns). It sets up constants for BUCKET, PROCESSED_PREFIX, SNS_TOPIC, and SAGEMAKER_ROLE. The script contains two main functions: 'send_notification' which publishes an SNS message, and 'process_dataframe' which applies a cut-off for BMI categories and adds columns for blood pressure ranges.

```
BUCKET = "varun-health-bigdata-project"
PROCESSED_PREFIX = "processed/"
SNS_TOPIC = "arn:aws:sns:us-east-1:818848508281:pipeline-alerts"
SAGEMAKER_ROLE = "arn:aws:iam::818848508281:role/service-role/AmazonSageMaker-ExecutionRole-20251127T194534"

def send_notification(subject, message):
    sns.publish(
        TopicArn=SNS_TOPIC,
        Subject=subject,
        Message=message
    )

def process_dataframe(df):
    df["bmi_category"] = pd.cut(
        df["bmi"],
        bins=[0, 18.5, 25, 30, 100],
        labels=["Underweight", "Healthy", "Overweight", "Obese"]
    )
    df["high blood pressure"] = ((df["systolic bp"] >= 140) | (df["diastolic bp"] >= 90)).astype(int)
```

What the Lambda script does

1. Gets triggered when a CSV is uploaded to raw/ in S3.
2. Downloads the raw CSV into Lambda.
3. Loads the data using Pandas.(instead of pyspark because for that I need to use AWS Glue instead of Lambda and it will cost a lot.)
4. Applies all preprocessing steps and creates new columns.
5. Saves the processed CSV to processed/ in S3.
6. Starts a SageMaker Autopilot job using the processed file.
7. Sends an SNS notification on success.
8. Sends an SNS notification on failure.

Lambda.function.py

```
import json
import boto3
import pandas as pd
import os
import uuid
from datetime import datetime

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

BUCKET = "varun-health-bigdata-project"
PROCESSED_PREFIX = "processed/"
SNS_TOPIC = "arn:aws:sns:us-east-1:818848508281:pipeline-alerts"
SAGEMAKER_ROLE = "arn:aws:iam::818848508281:role/service-role/AmazonSageMaker-ExecutionRole-20251127T194534"
```

```

def send_notification(subject, message):
    sns.publish(
        TopicArn=SNS_TOPIC,
        Subject=subject,
        Message=message
    )

def process_dataframe(df):
    df["bmi_category"] = pd.cut(
        df["bmi"],
        bins=[0, 18.5, 25, 30, 100],
        labels=["Underweight", "Healthy", "Overweight", "Obese"]
    )

    df["high_blood_pressure"] = ((df["systolic_bp"] >= 140) | (df["diastolic_bp"] >=
90)).astype(int)
    df["low_sleep"] = (df["sleep_hours"] < 7).astype(int)
    df["calorie_surplus"] = (df["calories_consumed"] > 2000).astype(int)
    df["risk_score"] = (
        df["smoker"] + df["alcohol"] + df["high_blood_pressure"] +
        df["low_sleep"] + df["calorie_surplus"]
    )

```

return df

```

def lambda_handler(event, context):
    try:
        # Extract S3 object info from event
        record = event["Records"][0]
        s3_bucket = record["s3"]["bucket"]["name"]
        s3_key = record["s3"]["object"]["key"]

        if not s3_key.startswith("raw/"):
            return {"status": "ignored"}

        # Download file to /tmp
        tmp_path = "/tmp/raw.csv"
        s3.download_file(s3_bucket, s3_key, tmp_path)

        # Load using pandas
        df = pd.read_csv(tmp_path)

        # Process
        df = process_dataframe(df)

        # Save processed file

```

```

processed_filename = f"processed_{uuid.uuid4()}.csv"
processed_path = "/tmp/" + processed_filename
df.to_csv(processed_path, index=False)

# Upload to processed folder
processed_s3_key = PROCESSED_PREFIX + processed_filename
s3.upload_file(processed_path, BUCKET, processed_s3_key)

processed_s3_uri = f"s3://{BUCKET}/{processed_s3_key}"

# Start SageMaker Autopilot job
job_name = "autopilot-" + datetime.now().strftime("%Y%m%d%H%M%S")

sagemaker.create_auto_ml_job(
    AutoMLJobName=job_name,
    InputDataConfig=[
        {
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": processed_s3_uri
                }
            },
            "TargetAttributeName": "disease_risk",
            "ContentType": "text/csv"
        }
    ],
    OutputDataConfig={"S3OutputPath": f"s3://{BUCKET}/autopilot-output/"},
    AutoMLJobConfig={
        "CompletionCriteria": {
            "MaxAutoMLJobRuntimeInSeconds": 3600,
            "MaxCandidates": 10
        }
    },
    RoleArn=SAGEMAKER_ROLE
)

send_notification(
    "Pipeline success",
    f"Processed file uploaded to {processed_s3_uri}\nAutopilot job: {job_name}"
)

return {"status": "success", "autopilot_job": job_name}

except Exception as e:
    send_notification("Pipeline failure", str(e))
    raise

```

PowerBI Automation:

The visualization component is not automated because my project uses PowerBI Desktop instead of AWS QuickSight. PowerBI Desktop is an offline tool, and it does not support automatic refresh through serverless workflows. Based on the professor's permission, the dashboard visualizations created in PowerBI have been included directly in the report, while the rest of the data pipeline is fully automated through AWS services.

Conclusions

This project successfully implemented an end to end big data pipeline using AWS and PySpark. Large scale health data was ingested from S3, processed in a distributed PySpark environment, and analyzed through feature engineering, aggregations, and SQL queries. AWS SageMaker Autopilot automated the machine learning process, providing model evaluations and highlighting areas for improvement such as feature enhancement and class balance.

The serverless automation pipeline using S3 triggers, Lambda, SageMaker, and SNS demonstrated how new data can be processed and modeled without manual steps, reflecting real world data engineering workflows. Power BI visualizations effectively summarized health patterns, BMI distribution, lifestyle risks, and disease likelihood across the population.

Overall, the project achieved its goals by integrating scalable processing, automated ML, and interactive visualization into a unified cloud based analytics solution.

References

- Amazon Web Services. “Amazon S3 Documentation.” AWS, <https://docs.aws.amazon.com/s3/>.
- Amazon Web Services. “Amazon EC2 Documentation.” AWS, <https://docs.aws.amazon.com/ec2/>.
- Amazon Web Services. “AWS Lambda Documentation.” AWS, <https://docs.aws.amazon.com/lambda/>.
- Amazon Web Services. “Amazon SageMaker Autopilot Documentation.” AWS, <https://docs.aws.amazon.com/sagemaker/latest/dg/autopilot-automate-model-development.html>.
- Apache Software Foundation. “PySpark Documentation.” <https://spark.apache.org/docs/latest/api/python/>.
- Microsoft Corporation. “Power BI Desktop Documentation.” <https://learn.microsoft.com/power-bi/>.
- Kaggle. “Dataset Source.” <https://www.kaggle.com/>.
- OpenAI. “ChatGPT.” Used as a support tool for understanding AWS setup, exploring debugging approaches, and clarifying implementation steps.