Mini project

1. Dataset Selection

Project Goal

In a world where decisions need data, organizations require efficient and scalable systems that can process and analyze large datasets. The project aims at designing and implementing a resilient big data pipeline using AWS for scalable cloud-based data storage and visualization combined with a PySpark environment on Linux to enable distributed data processing in an efficient manner.

The dataset to be used for this task contains IoT sensor data (smoke detection). This data set is chosen because it has appropriate complexity and is very instrumental in demonstrating big data pipelines in dealing with real scenarios.

The project, therefore, emphasizes considerations that go beyond technical implementation, including data bias and data privacy. These are very critical issues in ensuring fairness, security, and responsible use of Big Data analytics. With the solving of these challenges, this project aims to provide better skills in data engineering, big data analytics, and cloud-based distributed systems, preparing students to be better prepared for real-world tasks on how to manage and analyze Big Data.

Dataset overview

Dataset Name: ‘smoke_detection_iot.csv’

Source: IoT-based environmental sensors dataset designed to detect fire hazards.

Dataset Size: Approximately 6MB

Collection of training data is done with the help of IOT devices since the goal is to develop an AI-based smoke detector device. For a good dataset of training, many different environments and fire sources have to be sampled. A short list of different scenarios which are captured:

Normal indoor, Normal outdoor, Indoor wood fire, firefighter training area, Indoor gas fire, firefighter training area, Outdoor wood, coal, and gas grill, Outdoor high humidity, etc.

The dataset is almost 60.000 readings long. All sensors have a sample rate of 1Hz. To keep track of the data, every sensor reading has been augmented with a UTC timestamp.

Purpose: The dataset enables the identification of fire alarms based on environmental metrics such as temperature, humidity, and particulate levels.

Key Dataset Attributes

The dataset contains the following attributes:

1. UTC: Unix Timestamp for recording data points.

2. Temperature[C]: Measured in degrees Celsius.

3. Humidity[%]: Relative humidity percentage.

4. TVOC[ppb]: Total Volatile Organic Compounds in parts per billion.

5. eCO2[ppm]: Equivalent CO2 concentration in parts per million.

6. Raw H2: Concentration of hydrogen gas (raw sensor data).

7. Raw Ethanol: Concentration of ethanol gas (raw sensor data).

8. Pressure[hPa]: Atmospheric pressure in hectopascals.

9. PM1.0, PM2.5: Particulate matter concentrations in microns.

10. Fire Alarm: Binary indicator for fire alarm status (0 = No Alarm, 1 = Alarm Triggered).

Use Cases

The dataset is well-suited for:

- Real-Time Monitoring: Enabling real-time analysis of environmental conditions for early fire detection.
- Trend Analysis: Exploring long-term trends in air quality and safety metrics.
- Machine Learning: Training predictive models for fire detection using historical patterns.

2. Environment Setup

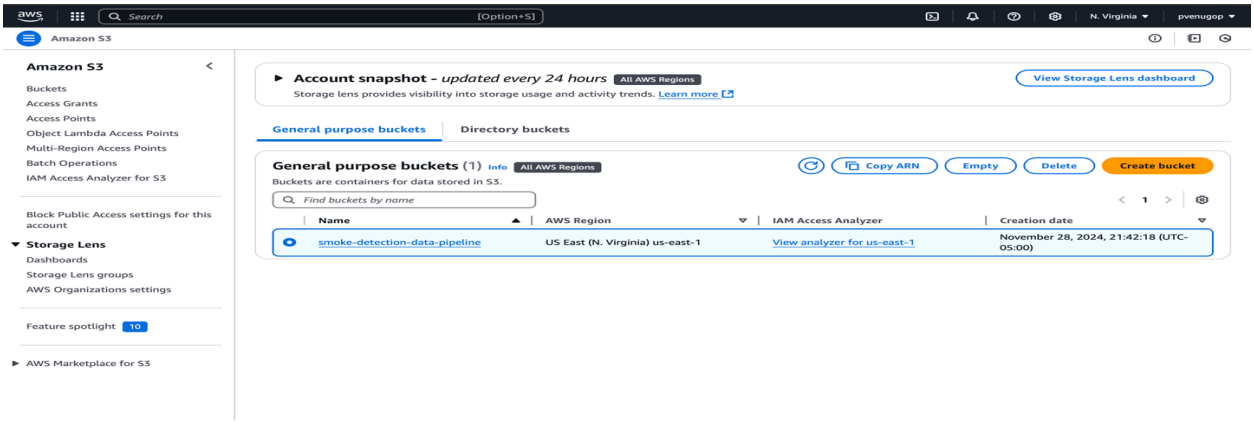
2.1 AWS S3 for Data Storage

a. Creating an S3 Bucket

To facilitate data storage and processing, an S3 bucket named ‘smoke-detection-data-pipeline’ was created using the AWS Management Console. This bucket will store:

- Raw Data: Unprocessed datasets uploaded directly from the source.
- Processed Data: Transformed and aggregated data outputs after PySpark processing.

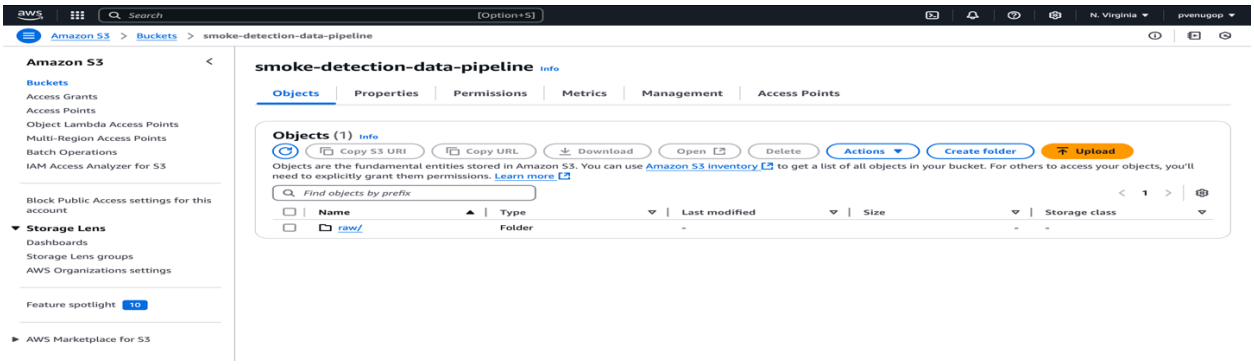
```
(base) prajwalv@Prajwals-MacBook-Pro .aws % aws s3 mb s3://smoke-detection-data-pipeline

make_bucket: smoke-detection-data-pipeline
```

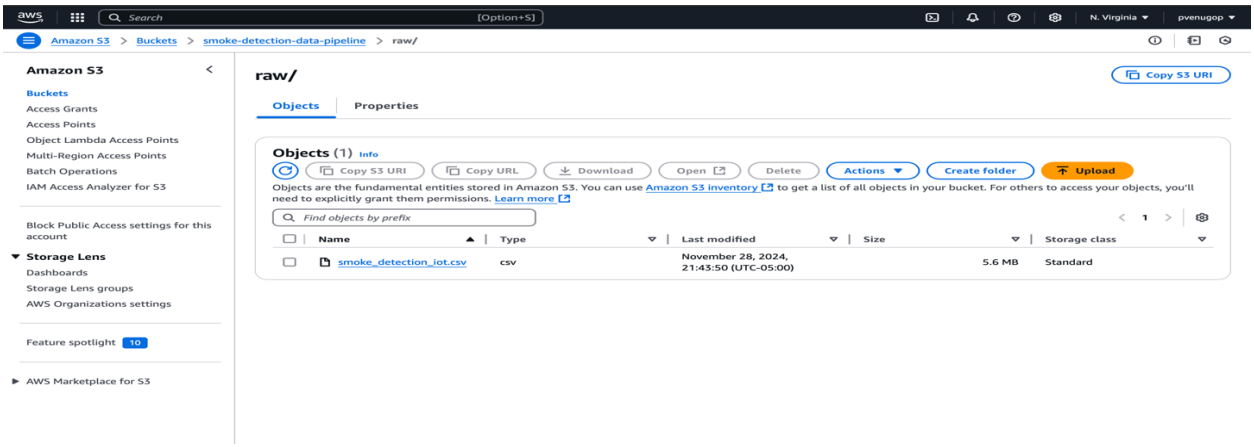


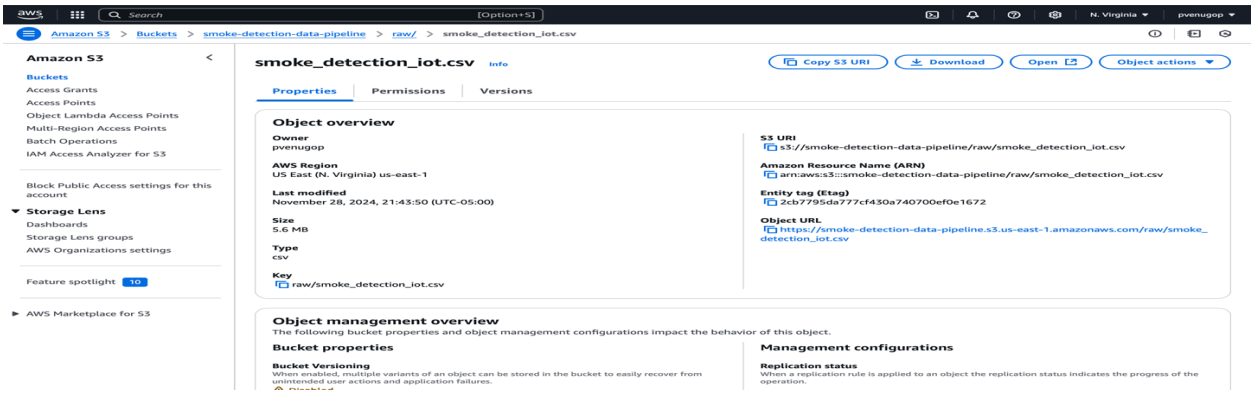
b. Uploading the Raw Dataset

The raw dataset ‘smoke_detection_iot.csv’ was uploaded to the ‘raw/’ folder in the bucket using the AWS Command Line Interface (CLI). The following command was executed for the upload:

```
(base) prajwalv@Prajwals-MacBook-Pro .aws % aws s3 cp ~/Downloads/smoke_detection_iot.csv s3://smoke-detection-data-pipeline/raw/

upload: ../Downloads/smoke_detection_iot.csv to s3://smoke-detection-data-pipeline/raw/smoke_detection_iot.csv
(base) prajwalv@Prajwals-MacBook-Pro .aws %
```





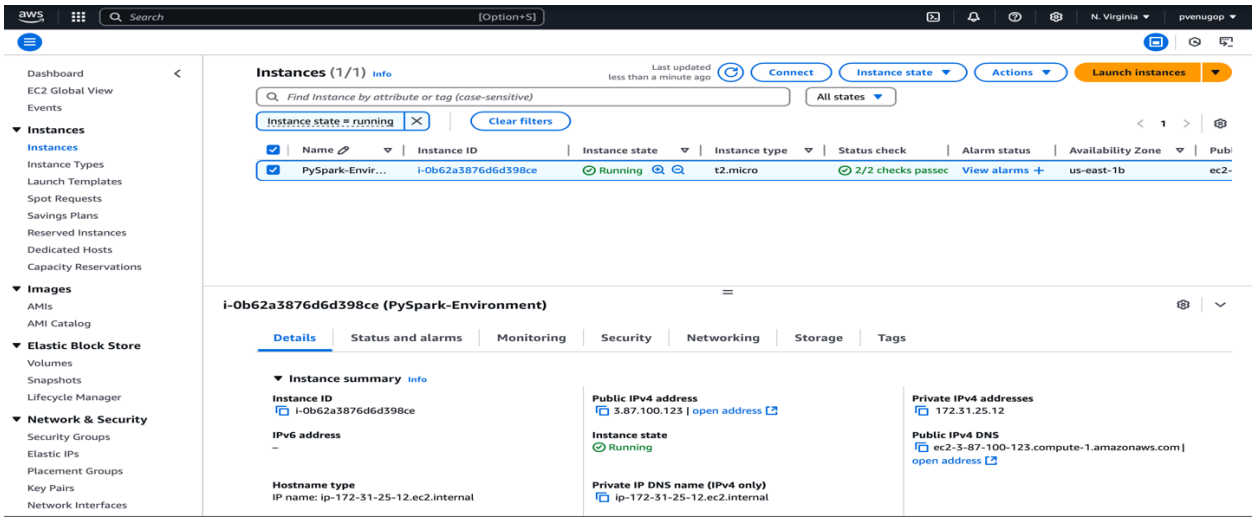


2.2 Linux Environment with PySpark

a. Setting Up a Linux-Based Environment

To ensure efficient distributed processing and seamless integration with AWS services, a Linux-based PySpark environment named ‘PySpark-Environment’ was set up on an AWS EC2 instance. This setup was chosen for:

- Scalability: EC2 provides flexibility in scaling resources based on processing needs.
- AWS Integration: Seamless connectivity with other AWS services like S3 and SageMaker.




```
(base) prajwalv@Prajwals-MacBook-Pro .aws % ssh -i ~/.ssh/myKeyPair.pem ec2-user@3.87.100.123

The authenticity of host '3.87.100.123 (3.87.100.123)' can't be established.
ED25519 key fingerprint is SHA256:YH2Dgp5BMGf5J+agPcMzQEUyzn0gAMO/vxKFQNp8P4A.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:7: 3.89.223.116
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '3.87.100.123' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
```

b. Installing PySpark

To enable distributed data processing, PySpark was installed in the PySpark-Environment using the pip package manager. The following command was executed:

```
[ec2-user@ip-172-31-25-12 ~]$ pip install pyspark
Defaulting to user installation because normal site-packages is not writeable
Collecting pyspark
  Downloading pyspark-3.5.3.tar.gz (317.3 MB)
     |████████████████████████████████| 317.3 MB 17 kB/s
  Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9.7
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
     |████████████████████████████████| 200 kB 71.3 MB/s
Using legacy 'setup.py install' for pyspark, since package 'wheel' is not installed.
Installing collected packages: py4j, pyspark
    Running setup.py install for pyspark ... done
Successfully installed py4j-0.10.9.7 pyspark-3.5.3
```

c. Configuring AWS CLI

The AWS CLI was configured to interact with the S3 bucket. This enables:

- Fetching raw datasets from the bucket.
- Uploading processed datasets back to the S3 bucket.

```
[[ec2-user@ip-172-31-25-12 ~]$ aws configure
[AWS Access Key ID [None]: AKIA2S2Y4ISCXGKFQ52T
[AWS Secret Access Key [None]: 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
[Default region name [None]: us-east-1
[Default output format [None]: json
[[ec2-user@ip-172-31-25-12 ~]$ aws s3 ls
 2024-11-29 02:42:18 smoke-detection-data-pipeline
 [ec2-user@ip-172-31-25-12 ~]$
```

Note: Hope you don't misuse these credentials

3. Data Pipeline Tasks

Task 1: Data Ingestion from S3

To pull raw data from an Amazon S3 bucket into the PySpark environment, ensuring the dataset is successfully loaded and ready for further analysis

Steps

1. Setup AWS CLI

- The AWS Command Line Interface (CLI) was used to interact with the S3 bucket and retrieve the dataset.
- The following commands were executed to browse the contents of the S3 bucket and download the required file:

```
[ec2-user@ip-172-31-25-12 ~]$ aws s3 ls
2024-11-29 02:42:18 smoke-detection-data-pipeline
[ec2-user@ip-172-31-25-12 ~]$ aws s3 ls s3://smoke-detection-data-pipeline/raw/
2024-11-29 02:43:50    5834376 smoke_detection_iot.csv
[ec2-user@ip-172-31-25-12 ~]$ aws s3 cp s3://smoke-detection-data-pipeline/raw/smoke_detection_iot.csv /home/ec2-user/
download: s3://smoke-detection-data-pipeline/raw/smoke_detection_iot.csv to ./smoke_detection_iot.csv
[ec2-user@ip-172-31-25-12 ~]$ ls /home/ec2-user/
Untitled.ipynb  pip_temp  smoke_detection_iot.csv
```

2. Initialize PySpark and Configure S3 Access

A PySpark session was initialized with the necessary configurations to handle S3 and AWS credentials using the ‘hadoop-aws’ library:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .master("local[*]") \
    .appName("smokeDetecton") \
    .config("spark.jars.packages", "org.apache.hadoop:hadoop-aws:3.3.4")\
    .config("fs.s3a.aws.credentials.provider", "com.amazonaws.auth.DefaultAWSCredentialsProviderChain")\
    .getOrCreate()

spark.conf.set("fs.s3a.aws.credentials.provider", "com.amazonaws.auth.DefaultAWSCredentialsProviderChain")
```

3. Read the Dataset into PySpark

The downloaded file was ingested into PySpark using the ‘spark.read.csv()’ function. The dataset's schema was inferred, and headers were extracted.

```
# Path to the downloaded file
file_path = "/home/ec2-user/smoke_detection_iot.csv"

# Load the dataset into PySpark
raw_data = spark.read.csv(file_path, header=True, inferSchema=True)

# Drop the unnecessary first column
cleaned_data = raw_data.drop("_c0")

# Showing the cleaned data
cleaned_data.show(5)
cleaned_data.printSchema()
```

4. Confirmation of Successful Ingestion

The ‘show()’ function displayed the first 5 rows of the dataset, while the ‘printSchema()’ function confirmed the successful inference of column data types.

```
+----------+--------------+-----------+---------+---------+------+-----------+-------------+-----+-----+-----+-----+-----+---+----------+
|       UTC|Temperature[C]|Humidity[%]|TVOC[ppb]|eCO2[ppm]|Raw H2|Raw Ethanol|Pressure[hPa]|PM1.0|PM2.5|NC0.5|NC1.0|NC2.5|CNT|Fire Alarm|
+----------+--------------+-----------+---------+---------+------+-----------+-------------+-----+-----+-----+-----+-----+---+----------+
|1654733331|          20.0|      57.36|        0|      400| 12306|      18520|      939.735|  0.0|  0.0|  0.0|  0.0|  0.0|  0|         0|
|1654733332|        20.015|      56.67|        0|      400| 12345|      18651|      939.744|  0.0|  0.0|  0.0|  0.0|  0.0|  1|         0|
|1654733333|        20.029|      55.96|        0|      400| 12374|      18764|      939.738|  0.0|  0.0|  0.0|  0.0|  0.0|  2|         0|
|1654733334|        20.044|      55.28|        0|      400| 12390|      18849|      939.736|  0.0|  0.0|  0.0|  0.0|  0.0|  3|         0|
|1654733335|        20.059|      54.69|        0|      400| 12403|      18921|      939.744|  0.0|  0.0|  0.0|  0.0|  0.0|  4|         0|
+----------+--------------+-----------+---------+---------+------+-----------+-------------+-----+-----+-----+-----+-----+---+----------+
only showing top 5 rows

root
 |-- UTC: integer (nullable = true)
 |-- Temperature[C]: double (nullable = true)
 |-- Humidity[%]: double (nullable = true)
 |-- TVOC[ppb]: integer (nullable = true)
 |-- eCO2[ppm]: integer (nullable = true)
 |-- Raw H2: integer (nullable = true)
 |-- Raw Ethanol: integer (nullable = true)
 |-- Pressure[hPa]: double (nullable = true)
 |-- PM1.0: double (nullable = true)
 |-- PM2.5: double (nullable = true)
 |-- NC0.5: double (nullable = true)
 |-- NC1.0: double (nullable = true)
 |-- NC2.5: double (nullable = true)
 |-- CNT: integer (nullable = true)
 |-- Fire Alarm: integer (nullable = true)
```

Task 2: Data Processing with PySpark

a. Data Transformation

Objective: Add meaningful columns to enable deeper analysis.

Convert UTC to Timestamp:

- Used the ‘from_unixtime’ function to convert ‘UTC’ into a readable timestamp format.
- Added a new column called ‘Timestamp’.

Extract Year and Month:

- Created two additional columns (‘Year’, ‘Month’) from the ‘Timestamp’ column to support temporal analysis.

```
from pyspark.sql.functions import from_unixtime, month, year
import pandas as pd

# Convert UTC to timestamp format
data_with_datetime = cleaned_data.withColumn("Timestamp", from_unixtime("UTC"))

# Extract Year and Month from the Timestamp column
data_with_datetime = data_with_datetime.withColumn("Year", year("Timestamp")) \
                                       .withColumn("Month", month("Timestamp"))

# Convert the PySpark DataFrame to a Pandas DataFrame for better presentation
pandas_data = data_with_datetime.toPandas()

# Display the data in a clear format using Pandas
print("\nTransformed Data (with Year and Month columns):")
print(pandas_data.head())
```

```
Transformed Data (with Year and Month columns):
          UTC  Temperature[C]  Humidity[%]  TVOC[ppb]  eCO2[ppm]  Raw H2  \
0  1654733331          20.000        57.36          0        400   12306
1  1654733332          20.015        56.67          0        400   12345
2  1654733333          20.029        55.96          0        400   12374
3  1654733334          20.044        55.28          0        400   12390
4  1654733335          20.059        54.69          0        400   12403

   Raw Ethanol  Pressure[hPa]  PM1.0  PM2.5  NC0.5  NC1.0  NC2.5  CNT  \
0        18520        939.735    0.0    0.0    0.0    0.0    0.0    0
1        18651        939.744    0.0    0.0    0.0    0.0    0.0    1
2        18764        939.738    0.0    0.0    0.0    0.0    0.0    2
3        18849        939.736    0.0    0.0    0.0    0.0    0.0    3
4        18921        939.744    0.0    0.0    0.0    0.0    0.0    4

   Fire Alarm           Timestamp  Year  Month
0           0 2022-06-09 00:08:51  2022      6
1           0 2022-06-09 00:08:52  2022      6
2           0 2022-06-09 00:08:53  2022      6
3           0 2022-06-09 00:08:54  2022      6
4           0 2022-06-09 00:08:55  2022      6
```

Rename Columns:

Standardized column names by removing spaces and special characters to ensure compatibility with PySpark and for better readability:

Example: ‘"Raw H2"‘ → ‘Raw_H2’, ‘PM2.5’ → ‘PM2_5’

```
# Rename columns to remove spaces or special characters
data_with_datetime = data_with_datetime.withColumnRenamed("Raw H2", "Raw_H2") \
                                       .withColumnRenamed("Raw Ethanol", "Raw_Ethanol") \
                                       .withColumnRenamed("TVOC[ppb]", "TVOC_ppb") \
                                       .withColumnRenamed("Pressure[hPa]", "Pressure_hPa") \
                                       .withColumnRenamed("PM2.5", "PM2_5") \
                                       .withColumnRenamed("Fire Alarm", "Fire_Alarm")
```

b. Data Aggregation

1. Monthly Chemical Emissions

Metric: Total ‘Raw_Ethanol’ and ‘TVOC_ppb’ per month.

Tracks the monthly release of volatile organic compounds (TVOC) and ethanol levels. Useful for monitoring air quality and industrial emissions trends. Helps assess compliance with environmental regulations.

Use Case: If emissions are above permissible limits, actions can be taken to control them. Monthly tracking helps identify consistent patterns or anomalies.

Interpretation:

- In June 2022, a significant amount of Raw Ethanol (1.2 billion) and TVOC (121 million) was observed.
- This indicates high levels of emissions, potentially due to seasonal activities or specific industrial processes during this period.

```python
# Compute Total Raw Ethanol and TVOC per Month
from pyspark.sql import functions as F

monthly_emissions = data_with_datetime.groupBy("Year", "Month") \
                                      .agg(
                                          F.sum("Raw_Ethanol").alias("Total_Raw_Ethanol"),
                                          F.sum("TVOC_ppb").alias("Total_TVOC")
                                      )
monthly_emissions.show()
```

```
[Stage 4:============================>                            (1 + 1) / 2]
+----+-----+-----------------+----------+
|Year|Month|Total_Raw_Ethanol|Total_TVOC|
+----+-----+-----------------+----------+
|2022|    6|       1237209173| 121631063|
+----+-----+-----------------+----------+
```

2. Monthly Environmental Trends

Metric: Average ‘Temperature[C]’, ‘Humidity[%]’, and ‘Pressure_hPa’ per month.

Provides an understanding of the environmental conditions during the data collection period. Correlating these metrics with pollutant levels can help identify seasonal or weather-related trends.

Use Case: If high TVOC or PM2.5 levels correlate with temperature and humidity changes, preventive measures (e.g., better ventilation or filters) can be implemented during such conditions.

Interpretation:

- The average temperature in June 2022 was 15.97°C, with an average humidity of 48.54% and pressure of 938.63 hPa.
- These conditions suggest a moderate climate, though relatively low pressure might indicate weather instability, which could influence emissions.

```
# Compute Average Temperature, Humidity, and Pressure per Month
monthly_environmental_trends = data_with_datetime.groupBy("Year", "Month") \
                                                  .agg(
                                                      F.avg("Temperature[C]").alias("Avg_Temperature"),
                                                      F.avg("Humidity[%]").alias("Avg_Humidity"),
                                                      F.avg("Pressure_hPa").alias("Avg_Pressure")
                                                  )
monthly_environmental_trends.show()
```

```
[Stage 7:============================>                            (1 + 1) / 2]
+----+-----+----------------+----------------+-----------------+
|Year|Month| Avg_Temperature|    Avg_Humidity|     Avg_Pressure|
+----+-----+----------------+----------------+-----------------+
|2022|    6|15.9704235829474|48.5394994411625|938.6276494651149|
+----+-----+----------------+----------------+-----------------+
```

3. Top 10 Days with Highest TVOC Levels

Metric: Days with the highest levels of 'TVOC_ppb'.

Identifies peak emission events that may warrant further investigation. It helps correlate spikes with specific activities or incidents, such as factory operations or nearby fires.

Use Case: Once identified, these peak days can be cross-checked with external events (e.g., factory malfunctions, festivals, or climatic conditions) to pinpoint sources of pollution.

Interpretation:

- The highest TVOC levels occurred on days with extremely high temperatures (~43°C) and relatively low humidity (~32%).
- These conditions could amplify the release of volatile compounds, suggesting environmental or industrial factors causing emissions spikes

```
# Top 10 Days with Highest TVOC Levels
top_10_tvoc_days = data_with_datetime.orderBy(F.desc("TVOC_ppb")).limit(10)
top_10_tvoc_days.show()

# Top 10 Days with Highest Raw Ethanol Levels
top_10_ethanol_days = data_with_datetime.orderBy(F.desc("Raw_Ethanol")).limit(10)
top_10_ethanol_days.show()
```

UTC	Temperature[C]	Humidity[%]	TVOC_ppb	eCO2[ppm]	Raw_H2	Raw_Ethanol	Pressure_hPa	PM1.0	PM2_5	NC0.5	NC1.0	NC2.5	CNT	Fire_Alarm	Timestamp	Year	Month
1654717046	43.81	32.2	60000	1354	12321	18192	936.858	83.3	86.54	573.3	89.399	2.019	4859	0	2022-06-08 19:37:26	2022	6
1654717055	43.8	32.29	60000	1480	12256	18096	936.873	221.58	230.21	1525.05	237.813	5.371	4868	0	2022-06-08 19:37:35	2022	6
1654717047	43.78	32.38	60000	1363	12318	18181	936.867	69.22	71.91	476.39	74.287	1.678	4860	0	2022-06-08 19:37:27	2022	6
1654717044	43.88	31.61	60000	1294	12343	18210	936.842	95.5	99.22	657.26	102.492	2.315	4857	0	2022-06-08 19:37:24	2022	6
1654717048	43.77	32.56	60000	1418	12281	18151	936.87	45.53	47.3	313.36	48.864	1.104	4861	0	2022-06-08 19:37:28	2022	6
1654717050	43.71	32.54	60000	1586	12207	18042	936.868	30.87	32.07	212.48	33.133	0.748	4863	0	2022-06-08 19:37:30	2022	6
1654717054	43.68	32.34	60000	1546	12218	18048	936.877	178.63	185.59	1229.45	191.718	4.33	4867	0	2022-06-08 19:37:34	2022	6
1654717051	43.73	32.53	60000	1614	12202	18041	936.87	27.45	28.52	188.94	29.464	0.665	4864	0	2022-06-08 19:37:31	2022	6
1654717045	43.85	32.0	60000	1297	12351	18216	936.844	99.95	103.84	687.91	107.272	2.423	4858	0	2022-06-08 19:37:25	2022	6
1654717052	43.67	32.52	60000	1593	12207	18039	936.892	34.55	35.9	237.79	37.08	0.837	4865	0	2022-06-08 19:37:32	2022	6

4. Top 10 Days with Highest Raw Ethanol Levels

Metric: Days with the highest levels of ‘Raw_Ethanol’.

Highlights days of abnormally high ethanol levels, possibly due to spills, leaks, or overproduction. Pinpoints safety risks associated with flammable substances.

Use Case: Enables targeted safety inspections and root cause analysis to prevent similar incidents.

Interpretation:

- The highest Raw Ethanol emissions align with high-temperature and low-humidity days, similar to the TVOC trend.
- This correlation indicates possible emissions from heat-sensitive sources like agriculture or chemical processes.

```
+----------+-------------+----------+--------+--------+------+-----------+-----------+-----+-----+-----+-----+-----+----+----------+-----
-------------+----+-----+
|       UTC|Temperature[C]|Humidity[%]|TVOC_ppb|eCO2[ppm]|Raw_H2|Raw_Ethanol|Pressure_hPa|PM1.0|PM2_5|NC0.5|NC1.0|NC2.5| CNT|Fire_Alarm|
Timestamp|Year|Month|
+----------+-------------+----------+--------+--------+------+-----------+-----------+-----+-----+-----+-----+-----+----+----------+-----
-------------+----+-----+
|1654716977|         42.96|      18.48|       0|      400| 13658|      21410|      936.82| 0.87|  0.9| 5.96| 0.93|0.021|4790|         0|2022-
06-08 19:36:17|2022|    6|
|1654716978|         42.58|      18.49|       0|      400| 13659|      21410|      936.82| 0.85| 0.89| 5.88|0.916|0.021|4791|         0|2022-
06-08 19:36:18|2022|    6|
|1655129098|         5.512|      18.48|       0|      400| 13658|      21410|      936.82| 0.87|  0.9| 5.96| 0.93|0.021|4790|         0|2022-
06-13 14:04:58|2022|    6|
|1655129099|         5.443|      18.49|       0|      400| 13659|      21410|      936.82| 0.85| 0.89| 5.88|0.916|0.021|4791|         0|2022-
06-13 14:04:59|2022|    6|
|1654716979|         41.46|      18.77|       0|      400| 13646|      21402|     936.826| 0.85| 0.88| 5.82|0.908| 0.02|4792|         0|2022-
06-08 19:36:19|2022|    6|
|1655129100|         5.374|      18.77|       0|      400| 13646|      21402|     936.826| 0.85| 0.88| 5.82|0.908| 0.02|4792|         0|2022-
06-13 14:05:00|2022|    6|
|1654716012|         27.32|      39.16|       0|      400| 13430|      21401|      937.4| 1.48| 1.54|10.18|1.587|0.036|3825|         0|2022-
06-08 19:20:12|2022|    6|
|1654716976|         42.92|      18.51|       0|      400| 13655|      21401|     936.825| 0.88| 0.91| 6.02|0.939|0.021|4789|         0|2022-
06-08 19:36:16|2022|    6|
|1655128133|        13.196|      39.16|       0|      400| 13430|      21401|      937.4| 1.48| 1.54|10.18|1.587|0.036|3825|         0|2022-
06-13 13:48:53|2022|    6|
|1655129097|         5.582|      18.51|       0|      400| 13655|      21401|     936.825| 0.88| 0.91| 6.02|0.939|0.021|4789|         0|2022-
06-13 14:04:57|2022|    6|
+----------+-------------+----------+--------+--------+------+-----------+-----------+-----+-----+-----+-----+-----+----+----------+-----
-------------+----+-----+
```

5. Average PM2.5 Levels per Month

Metric: Monthly average of fine particulate matter (‘PM2.5’).

Tracks air quality by measuring the concentration of harmful fine particles. Aids in understanding the health impact of air pollution over time.

Use Case: High PM2.5 levels can trigger public health advisories or policy interventions to reduce emissions from vehicles or industrial activities.

Interpretation:

The average PM2.5 level for June 2022 is 184.47 µg/m³, indicating significant air pollution during this period. This suggests potential environmental or human activities contributing to high particulate matter concentrations, warranting further investigation into pollution sources and their impacts.

```
# Compute Average PM2.5 per Month
avg_pm25_per_month = data_with_datetime.groupBy("Year", "Month") \
                                       .agg(F.avg("PM2_5").alias("Avg_PM2_5"))
avg_pm25_per_month.show()
```

```
+----+-----+------------------+
|Year|Month|         Avg_PM2_5|
+----+-----+------------------+
|2022|    6|184.46777023790509|
+----+-----+------------------+
```

6. Total Fire Alarms per Month

Metric: Monthly count of fire alarms (‘Fire_Alarm’).

Indicates fire risks or incidents in the monitored area.Helps identify relationships between environmental conditions (e.g., high TVOC, high temperature) and fire risks.

Use Case: If certain thresholds (e.g., high TVOC or ethanol) frequently precede fire alarms, automated alerts can be set up to prevent fires.

Interpretation:
A total of 44,757 fire alarms were triggered in June 2022. This high count may indicate frequent fire-related incidents or potential false alarms, which could be correlated with the elevated PM2.5 levels during this period. This correlation should be further analyzed to identify causation.

```
# Compute Total Fire Alarms per Month
fire_alarm_count = data_with_datetime.groupBy("Year", "Month") \
                                     .agg(F.sum("Fire_Alarm").alias("Total_Fire_Alarms"))
fire_alarm_count.show()
```

```
+----+-----+-----------------+
|Year|Month|Total_Fire_Alarms|
+----+-----+-----------------+
|2022|    6|            44757|
+----+-----+-----------------+
```

7.PM2.5 Outlier Detection

Metric: Identification of outliers where ‘PM2.5’ levels exceed the mean by 3 standard deviations.

Flags extreme pollution events, which could be isolated incidents or signal systemic issues. Helps isolate and study specific events to understand their cause and impact.

Use Case: Provides actionable insights to investigate specific time periods, locations, or activities that led to extreme pollution.

Interpretation:

Outliers are defined as PM2.5 values exceeding mean + 3 × standard deviation, representing exceptionally high pollution levels. These extreme cases may result from specific incidents, such as wildfires, industrial accidents, or sensor malfunctions. Further analysis of these timestamps could reveal underlying causes and patterns.

```
# Compute Mean and Standard Deviation of PM2.5
pm25_stats = data_with_datetime.agg(
    F.mean("PM2_5").alias("Mean_PM2_5"),
    F.stddev("PM2_5").alias("StdDev_PM2_5")
).collect()

mean_pm25 = pm25_stats[0]["Mean_PM2_5"]
stddev_pm25 = pm25_stats[0]["StdDev_PM2_5"]

# Filter Outliers (greater than mean + 3*stddev)
outliers_pm25 = data_with_datetime.filter(
    F.col("PM2_5") > mean_pm25 + 3 * stddev_pm25
)
outliers_pm25.show()
```

```
+----------+--------------+-----------+--------+---------+------+-----------+------------+--------+--------+--------+---------+---------+---+
----------+------------------+----+-----+
|       UTC|Temperature[C]|Humidity[%]|TVOC_ppb|eCO2[ppm]|Raw_H2|Raw_Ethanol|Pressure_hPa|   PM1.0|   PM2_5|   NC0.5|    NC1.0|    NC2.5|CNT|
Fire_Alarm|          Timestamp|Year|Month|
+----------+--------------+-----------+--------+---------+------+-----------+------------+--------+--------+--------+---------+---------+---+
----------+------------------+----+-----+
|1654903053|         22.11|      47.94|       0|     1405| 12961|      20049|     931.186| 7914.14| 9019.41| 52467.6| 9447.012|  965.387| 48|
1|2022-06-10 23:17:33|2022|    6|
|1654903054|         21.94|      50.24|       0|     1402| 12962|      20043|     931.195| 9608.86| 12702.7|59301.19| 13565.18| 2872.672| 49|
1|2022-06-10 23:17:34|2022|    6|
|1654903055|         21.81|      52.44|       0|     1448| 12942|      20036|     931.187|10356.99|15075.03|60442.71|16275.758| 4439.094| 50|
1|2022-06-10 23:17:35|2022|    6|
|1654903056|         21.73|      53.86|       0|     1550| 12884|      20004|     931.192|11433.76|19884.37|58580.91|21845.273| 8047.641| 51|
1|2022-06-10 23:17:36|2022|    6|
|1654903057|         21.69|      55.34|       0|     1517| 12908|      20007|     931.196|11722.13|21849.34| 56381.3|24146.516| 9671.195| 52|
1|2022-06-10 23:17:37|2022|    6|
|1654903058|         21.66|      56.72|       0|     1498| 12909|      19994|     931.208|12107.31|24590.06|53151.68|27359.121|11952.469| 53|
1|2022-06-10 23:17:38|2022|    6|
|1654903059|          21.6|      58.16|       0|     1539| 12889|      19974|     931.203|12158.49|25434.94|51514.74|28360.902|12722.195| 54|
1|2022-06-10 23:17:39|2022|    6|
|1654903060|         21.56|      59.54|       0|     1516| 12899|      19965|     931.208|12266.83|27147.14|48241.31|30390.297|14277.578| 55|
1|2022-06-10 23:17:40|2022|    6|
|1654903061|         21.56|       60.5|       0|     1539| 12896|      19945|     931.219|12251.45|27710.73|46679.25|31066.941|14839.797| 56|
1|2022-06-10 23:17:41|2022|    6|
|1654903062|         21.52|      61.62|       0|     1534| 12894|      19928|     931.219|12705.84|30719.31|43433.61|34588.191|17312.953| 57|
1|2022-06-10 23:17:42|2022|    6|
|1654903063|         21.49|      63.47|       0|     1476| 12923|      19940|     931.219|12456.58|30870.15|40688.44|34810.785|17704.711| 58|
1|2022-06-10 23:17:43|2022|    6|
|1654903064|         21.52|      64.03|       5|     1487| 12917|      19917|     931.233|12874.66| 32943.8|39447.12| 37220.02|19306.086| 59|
1|2022-06-10 23:17:44|2022|    6|
|1654903065|         21.52|       64.6|      33|     1479| 12931|      19920|     931.237|13228.08|35020.92|37583.35|39644.367|20974.461| 60|
1|2022-06-10 23:17:45|2022|    6|
|1654903066|         21.47|      65.18|      41|     1445| 12937|      19908|     931.253|13252.56| 36124.9|35041.93|40960.555|22021.992| 61|
1|2022-06-10 23:17:46|2022|    6|
|1654903067|         21.49|       65.8|      25|     1400| 12960|      19909|     931.256|13315.76|36893.58|33710.61|41869.164|22705.938| 62|
1|2022-06-10 23:17:47|2022|    6|
|1654903068|         21.47|      66.37|      35|     1391| 12970|      19895|     931.262|13706.52|39270.41|31448.22|44645.625|24628.484| 63|
```

8. Correlation Matrix for Environmental Variables and Pollutants

```
# Assemble environmental variables and pollutants into a vector
vector_col = "features"
assembler = VectorAssembler(
    inputCols=["Temperature[C]", "Humidity[%]", "TVOC_ppb", "Raw_Ethanol"],
    outputCol=vector_col
)

# Transform data into vector format
data_vector = assembler.transform(data_with_datetime)

# Compute the correlation matrix for these features
correlation_matrix = Correlation.corr(data_vector, vector_col).head()[0]

# Display the correlation matrix
print(f"Correlation Matrix:\n{correlation_matrix}")
```

```
24/12/07 23:52:09 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
Correlation Matrix:
DenseMatrix([[ 1.        , -0.24398559,  0.08244183, -0.03734264],
             [-0.24398559,  1.        , -0.4888779 ,  0.06878211],
             [ 0.08244183, -0.4888779 ,  1.        , -0.67371463],
             [-0.03734264,  0.06878211, -0.67371463,  1.        ]])
```

Variables: ‘Temperature[C]’, ‘Humidity[%]’, ‘TVOC_ppb’, ‘Raw_Ethanol’

Interpretation:

- Weak Positive Correlation between Temperature and TVOC (‘0.082’). Suggests a negligible relationship, indicating that temperature changes alone do not significantly impact TVOC levels in this dataset.
- Negative Correlation between Humidity and TVOC (‘-0.488’): Indicates that higher humidity suppresses TVOC levels, potentially due to absorption or dispersion effects.
- Negative Correlation between TVOC and Raw Ethanol (‘-0.673’):
- Stronger negative relationship suggesting that these emissions are not co-occurring and may originate from independent sources.

Usefulness:

- Highlights key influencers of pollutant levels such as humidity.
- Differentiates sources of emissions, enabling targeted interventions.

9. Correlation Matrix with Fire Alarms

```
# Correlate Fire Alarms with environmental conditions and chemical emissions
assembler_fire_alarm = VectorAssembler(
    inputCols=["Temperature[C]", "Humidity[%]", "TVOC_ppb", "Raw_Ethanol", "PM2_5"],
    outputCol="fire_alarm_features"
)

# Transform data
data_with_fire_alarm_features = assembler_fire_alarm.transform(data_with_datetime)

# Calculate correlation matrix
correlation_matrix_fire_alarm = Correlation.corr(data_with_fire_alarm_features, "fire_alarm_features").head()[0]

# Display the correlation matrix
print(f"Correlation Matrix with Fire Alarms:\n{correlation_matrix_fire_alarm}")
```

```
Correlation Matrix with Fire Alarms:
DenseMatrix([[ 1.        , -0.24398559,  0.08244183, -0.03734264,  0.03208418],
             [-0.24398559,  1.        , -0.4888779 ,  0.06878211, -0.17888169],
             [ 0.08244183, -0.4888779 ,  1.        , -0.67371463,  0.47742447],
             [-0.03734264,  0.06878211, -0.67371463,  1.        , -0.39319215],
             [ 0.03208418, -0.17888169,  0.47742447, -0.39319215,  1.        ]])
```

Variables: ‘Temperature[C]’, ‘Humidity[%]’, ‘TVOC_ppb’, ‘Raw_Ethanol’, ‘PM2.5’

Interpretation:

- Positive Correlation between PM2.5 and TVOC (‘0.477’):
- Indicates that these two pollutants often increase together, potentially sharing a common source or being influenced by similar environmental conditions.
- Negative Correlation between PM2.5 and Raw Ethanol (‘-0.393’):
- Suggests that PM2.5 levels are higher when ethanol emissions are lower, again pointing to distinct emission sources.
- Weak Correlation between Fire Alarms and Variables (e.g., TVOC: ‘0.032’, PM2.5: ‘0.032’):
- No strong dependency on any specific variable, suggesting that fire alarms may be influenced by additional unmeasured factors.

Usefulness:

- Helps refine fire risk models by identifying the limited direct role of pollutants like TVOC and PM2.5 in triggering alarms.

- Guides the integration of additional variables (e.g., wind speed, human activity) into risk prediction models.

Task 3: Store Processed Data Back to S3

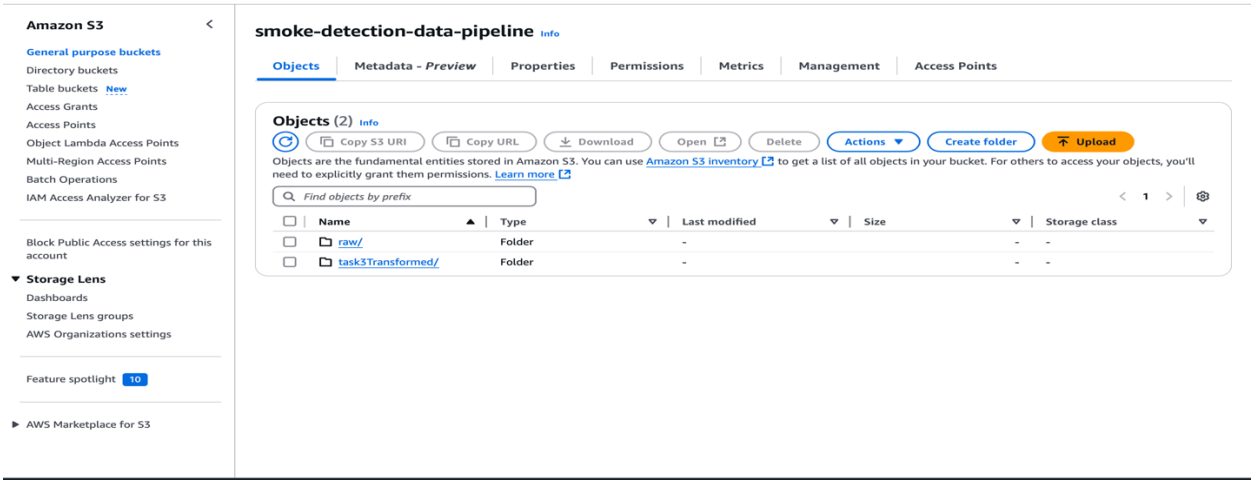
Implementation Steps

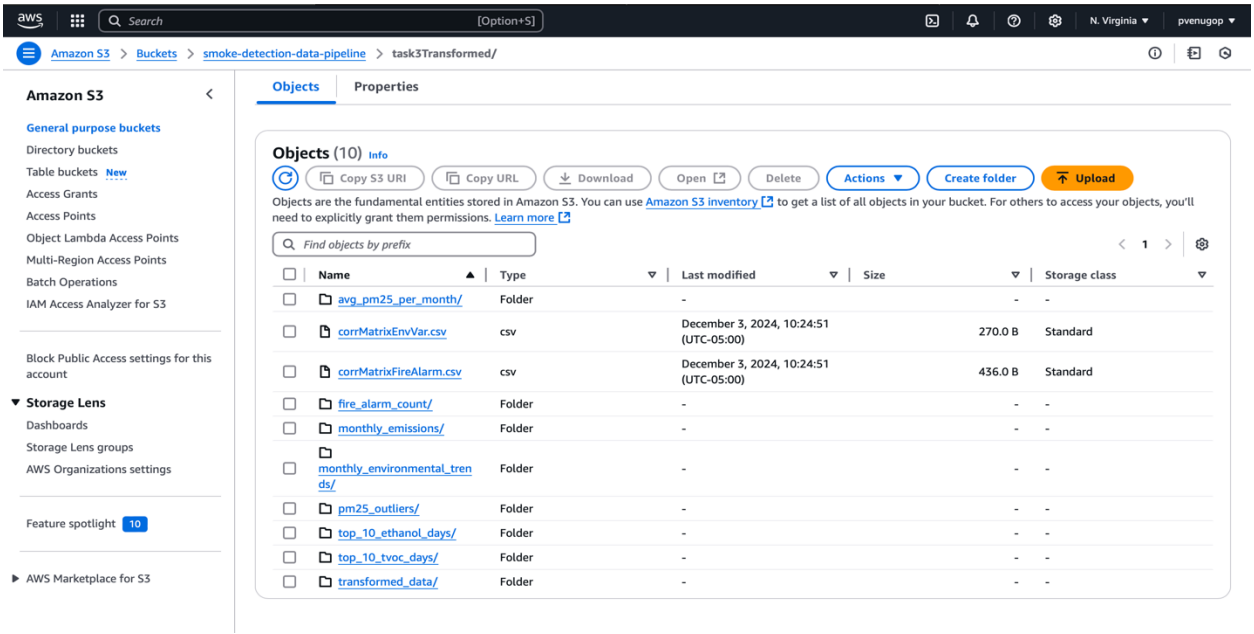
1. Export the Data: Each dataset is exported into CSV format using the 'coalesce(1)' method for combining partitions where necessary, ensuring that the output is a single CSV file.

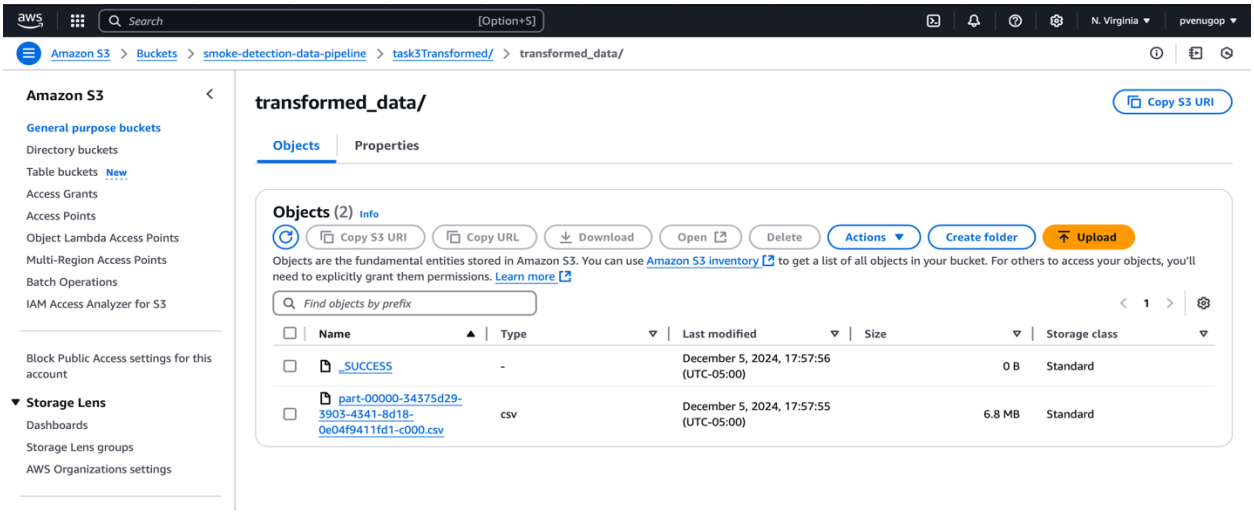
2. Upload to S3 Bucket: The data is uploaded to designated folders within the S3 bucket ('s3a://smoke-detection-data-pipeline/task3Transformed/'). The bucket and folder paths ensure organization and accessibility for future processing or analysis.

3. Correlation Matrix Handling: For correlation matrices, the processed results are converted to pandas DataFrames and exported as CSV files directly to the S3 bucket.

```
# Define the S3 path for processed data
s3_bucket = "s3a://smoke-detection-data-pipeline/task3Transformed/"

# Save Transformed Data (with Year and Month columns)
data_with_datetime.coalesce(1).write.csv(f"{s3_bucket}transformed_data/", header=True, mode="overwrite")

# Save Aggregated Monthly Emissions Data
monthly_emissions.write.csv(f"{s3_bucket}monthly_emissions/", header=True, mode="overwrite")

# Save Monthly Environmental Trends
monthly_environmental_trends.write.csv(f"{s3_bucket}monthly_environmental_trends/", header=True, mode="overwrite")

# Save Top 10 Days with Highest TVOC Levels
top_10_tvoc_days.write.csv(f"{s3_bucket}top_10_tvoc_days/", header=True, mode="overwrite")

# Save Top 10 Days with Highest Raw Ethanol Levels
top_10_ethanol_days.write.csv(f"{s3_bucket}top_10_ethanol_days/", header=True, mode="overwrite")

# Save Average PM2.5 Levels Per Month
avg_pm25_per_month.write.csv(f"{s3_bucket}avg_pm25_per_month/", header=True, mode="overwrite")

# Save Total Fire Alarms Per Month
fire_alarm_count.write.csv(f"{s3_bucket}fire_alarm_count/", header=True, mode="overwrite")

# Save PM2.5 Outliers Data
outliers_pm25.write.csv(f"{s3_bucket}pm25_outliers/", header=True, mode="overwrite")

# Convert Correlation Matrices to Pandas DataFrame
correlation_matrix_pd = pd.DataFrame(correlation_matrix.toArray())
correlation_matrix_fire_alarm_pd = pd.DataFrame(correlation_matrix_fire_alarm.toArray())

# Save Correlation Matrices as CSV Files in S3
correlation_matrix_pd.to_csv(f"{s3_bucket}corrMatrixEnvVar.csv", index=False, header=True, storage_options={"anon": False})
correlation_matrix_fire_alarm_pd.to_csv(f"{s3_bucket}corrMatrixFireAlarm.csv", index=False, header=True, storage_options={"anon": False})
```


Amazon S3
General purpose buckets
Directory buckets
Table buckets New
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight 10
AWS Marketplace for S3
smoke-detection-data-pipeline Info
Objects
Metadata - Preview
Properties
Permissions
Metrics
Management
Access Points
Objects (2) Info
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
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
Find objects by prefix
Name
Type
Last modified
Size
Storage class
raw/
Folder
task3Transformed/
Folder


Search
[Option+S]
N. Virginia
pvenugop
Amazon S3 > Buckets > smoke-detection-data-pipeline > task3Transformed/
Amazon S3
General purpose buckets
Directory buckets
Table buckets New
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight 10
AWS Marketplace for S3
Objects
Properties
Objects (10) Info
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
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
Find objects by prefix
Name
Type
Last modified
Size
Storage class
avg_pm25_per_month/
Folder
corrMatrixEnvVar.csv
csv
December 3, 2024, 10:24:51 (UTC-05:00)
270.0 B
Standard
corrMatrixFireAlarm.csv
csv
December 3, 2024, 10:24:51 (UTC-05:00)
436.0 B
Standard
fire_alarm_count/
Folder
monthly_emissions/
Folder
monthly_environmental_trends/
Folder
pm25_outliers/
Folder
top_10_ethanol_days/
Folder
top_10_tvoc_days/
Folder
transformed_data/
Folder


Search
[Option+S]
N. Virginia
pvenugop
Amazon S3 > Buckets > smoke-detection-data-pipeline > task3Transformed/ > transformed_data/
Amazon S3
General purpose buckets
Directory buckets
Table buckets New
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
transformed_data/
Copy S3 URI
Objects
Properties
Objects (2) Info
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
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
Find objects by prefix
Name
Type
Last modified
Size
Storage class
_SUCCESS
-
December 5, 2024, 17:57:56 (UTC-05:00)
0 B
Standard
part-00000-34375d29-3903-4341-8d18-0e04f9411fd1-c000.csv
csv
December 5, 2024, 17:57:55 (UTC-05:00)
6.8 MB
Standard

Task 4: Data Analysis Using Spark SQL

To derive insights from the processed data using Spark SQL. This task includes running at least five SQL queries on the data to extract meaningful patterns, trends, and insights.

Steps Performed:

1. Register DataFrames as SQL Views

The processed and aggregated data were registered as temporary SQL views in Spark to enable SQL-style querying.

```
# Register the DataFrames as SQL views
data_with_datetime.createOrReplaceTempView("transformed_data")
monthly_emissions.createOrReplaceTempView("monthly_emissions")
monthly_environmental_trends.createOrReplaceTempView("monthly_environmental_trends")
top_10_tvoc_days.createOrReplaceTempView("top_10_tvoc_days")
top_10_ethanol_days.createOrReplaceTempView("top_10_ethanol_days")
fire_alarm_count.createOrReplaceTempView("fire_alarm_count")
```

2. SQL Queries and Analysis

The following queries were executed:

Query 1: Identify Days with the Highest Total TVOC and Raw Ethanol Emissions

Objective: To find the top 5 days with the highest Total Volatile Organic Compounds (TVOC) and raw ethanol levels.

```
query1 = """
SELECT DAY(FROM_UNIXTIME(UTC)) AS Day,
       SUM(TVOC_ppb) AS Total_TVOC,
       SUM(Raw_Ethanol) AS Total_Raw_Ethanol
FROM transformed_data
GROUP BY Day
ORDER BY Total_TVOC DESC
LIMIT 5
"""
highest_emissions_days = spark.sql(query1)
highest_emissions_days.show()
```

```
+---+----------+-----------------+
|Day|Total_TVOC|Total_Raw_Ethanol|
+---+----------+-----------------+
|  8|  40947898|        115780034|
| 13|  40947898|        115780034|
|  9|  39054608|        982481786|
| 10|    680659|         23167319|
+---+----------+-----------------+
```

Insights:

Days 8, 13, 9, and 10 recorded the highest TVOC and raw ethanol emissions.

Query 2: Analyze Daily Temperature and Humidity Trends

Objective: To calculate the average daily temperature and humidity.

```
query2 = """
SELECT DAY(FROM_UNIXTIME(UTC)) AS Day,
       AVG(`Temperature[C]`) AS Avg_Temperature,
       AVG(`Humidity[%]`) AS Avg_Humidity
FROM transformed_data
GROUP BY Day
ORDER BY Avg_Temperature DESC
"""
daily_temp_humidity = spark.sql(query2)
daily_temp_humidity.show()
```

```
+---+------------------+------------------+
|Day|   Avg_Temperature|      Avg_Humidity|
+---+------------------+------------------+
| 10|  34.2332235701906|23.258587521663777|
|  8|  33.7831023676881| 38.12003830083569|
|  9|14.629292450187956| 51.51767304153025|
| 13| 6.160041434540405| 38.12003830083566|
+---+------------------+------------------+
```

Insights:

Day 10 had the highest average temperature of 34.23°C, while Day 9 exhibited the highest average humidity of 51.52%.

Query 3: Identify Days with the Most Fire Alarms

Objective: To find the top 5 days with the highest number of fire alarms triggered.

```
query3 = """
SELECT DAY(FROM_UNIXTIME(UTC)) AS Day,
       COUNT(Fire_Alarm) AS Total_Fire_Alarms
FROM transformed_data
WHERE Fire_Alarm = 1
GROUP BY Day
ORDER BY Total_Fire_Alarms DESC
LIMIT 5
"""
most_fire_alarms_days = spark.sql(query3)
most_fire_alarms_days.show()
```

```
+---+-----------------+
|Day|Total_Fire_Alarms|
+---+-----------------+
|  9|            43632|
| 10|             1121|
|  8|                2|
| 13|                2|
+---+-----------------+
```

Insights:

Day 9 recorded the most fire alarms (43,632), followed by Day 10.

Query 4: Identify Days with the Highest PM2.5 Levels

Objective: To find the top 5 days with the highest average PM2.5 concentrations.

```
query4 = """
SELECT DAY(FROM_UNIXTIME(UTC)) AS Day,
       AVG(PM2_5) AS Avg_PM2_5
FROM transformed_data
GROUP BY Day
ORDER BY Avg_PM2_5 DESC
LIMIT 5
"""
highest_pm25_days = spark.sql(query4)
highest_pm25_days.show()
```

```
[Stage 39:==============================>                          (1 + 1) / 2]
+---+------------------+
|Day|         Avg_PM2_5|
+---+------------------+
| 10|2973.7339428076302|
|  8| 699.6721552924782|
| 13| 699.6721552924782|
|  9|1.6742770264863744|
+---+------------------+
```

Insights:

Day 10 had the highest PM2.5 level with an average of 2973.73, indicating severe pollution.

Query 5: Explore Environmental Conditions During Fire Alarms

Objective: To analyze temperature, humidity, PM2.5 levels, and fire alarm counts on days when fire alarms were triggered.

```
query5 = """
SELECT DAY(FROM_UNIXTIME(UTC)) AS Day,
       AVG(`Temperature[C]`) AS Avg_Temperature,
       AVG(`Humidity[%]`) AS Avg_Humidity,
       AVG(PM2_5) AS Avg_PM2_5,
       COUNT(Fire_Alarm) AS Fire_Alarm_Count
FROM transformed_data
WHERE Fire_Alarm = 1
GROUP BY Day
ORDER BY Fire_Alarm_Count DESC
"""
fire_alarm_conditions = spark.sql(query5)
fire_alarm_conditions.show()
```

```
+---+------------------+-----------------+------------------+----------------+
|Day|   Avg_Temperature|     Avg_Humidity|         Avg_PM2_5|Fire_Alarm_Count|
+---+------------------+-----------------+------------------+----------------+
|  9|13.965375618811771| 51.5017766776682|1.8266909607627617|           43632|
| 10| 34.60313113291699|22.69262265834077|3059.8000267618245|            1121|
|  8|            27.295|            43.91|             2.335|               2|
| 13|           20.2225|            43.91|             2.335|               2|
+---+------------------+-----------------+------------------+----------------+
```

Insights:

Day 9 showed a significant correlation between high PM2.5 levels (1.82 on average) and a large number of fire alarms triggered (43,632).

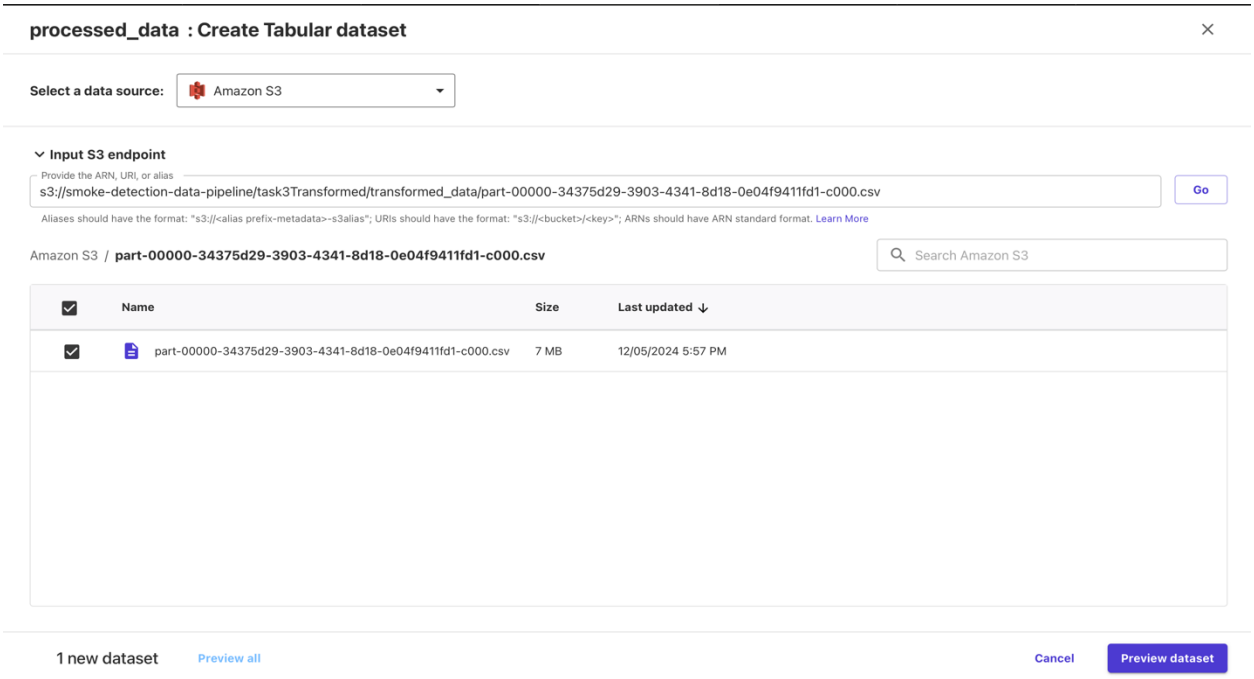
Day 10 had high PM2.5 levels (3059.8 on average) but fewer fire alarms compared to Day 9.

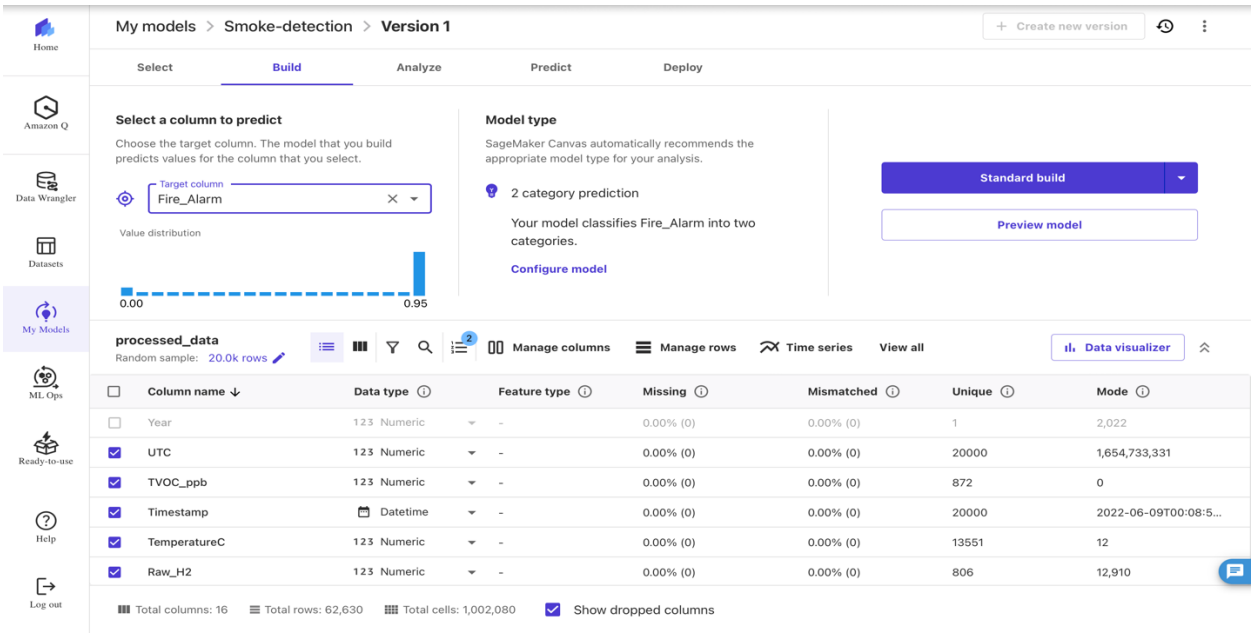
Task 5: Machine Learning with AWS SageMaker Autopilot

Steps

Running Autopilot Experiment

- The transformed dataset was loaded into SageMaker Autopilot.
- 'Fire_Alarm' was selected as the target column.
- Autopilot generated multiple candidate models with varying hyperparameters and feature engineering pipelines.

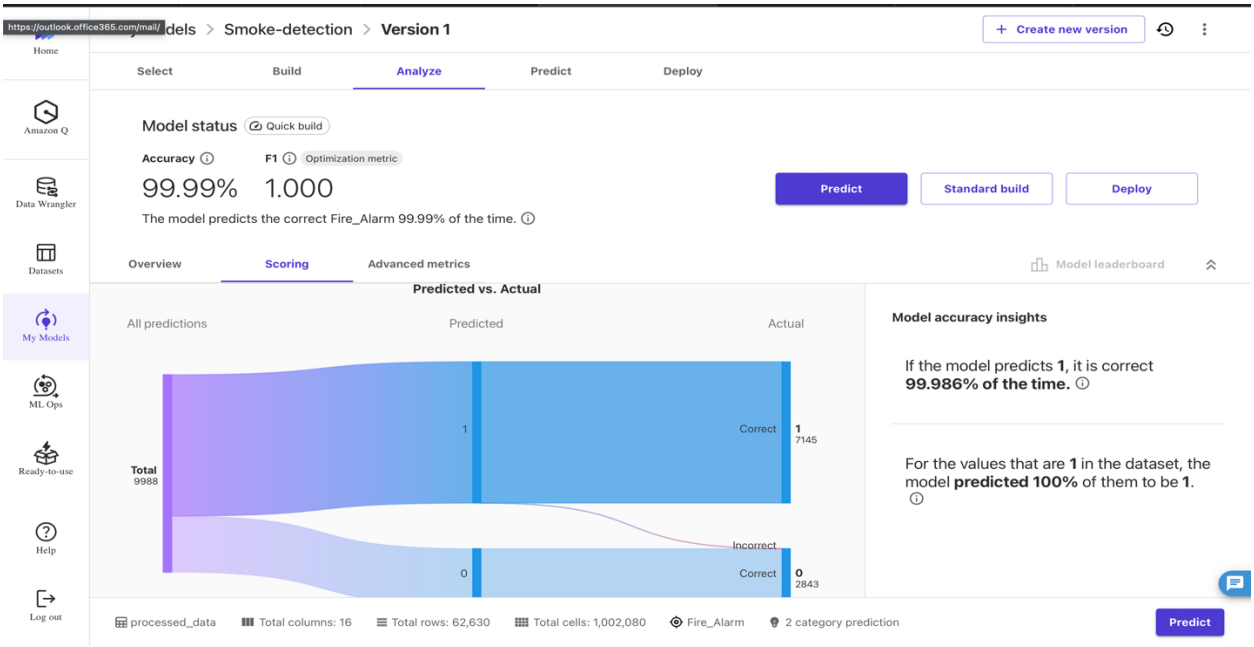




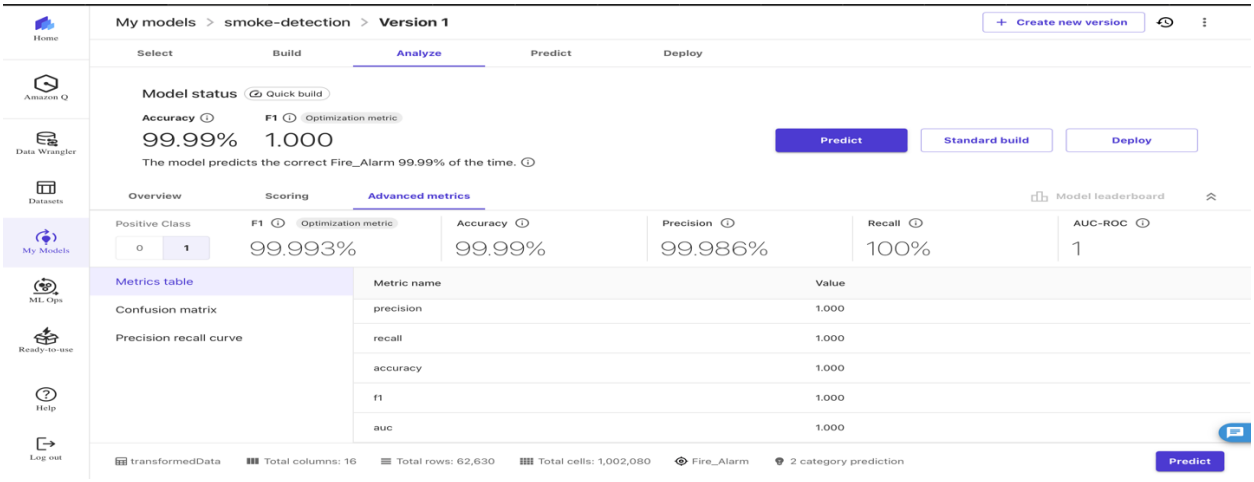
Model Performance Analysis

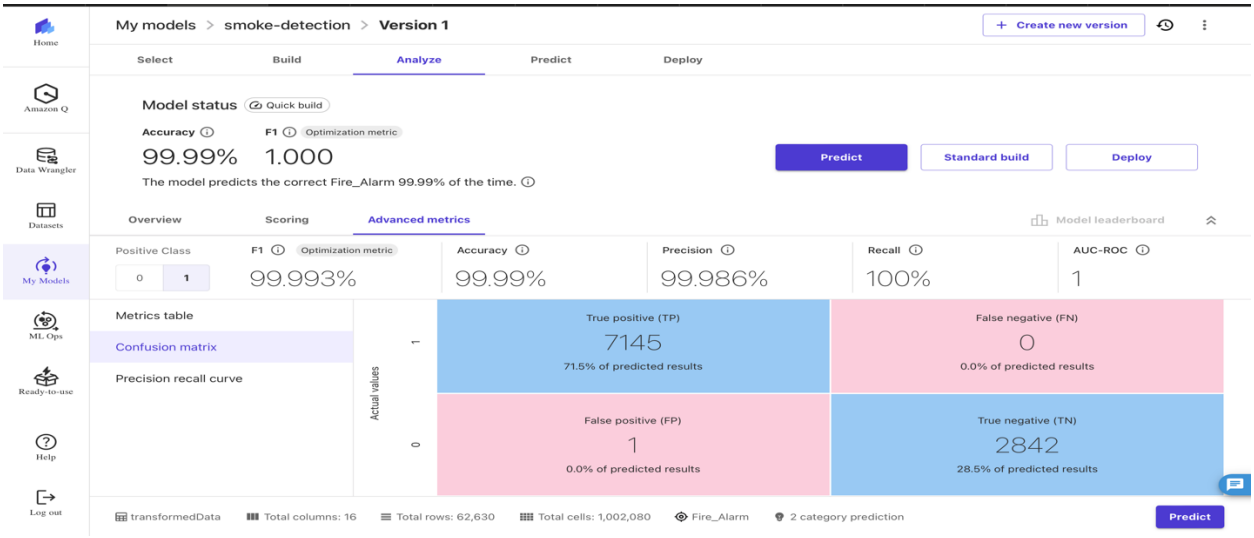
The best-performing model delivered the following results:

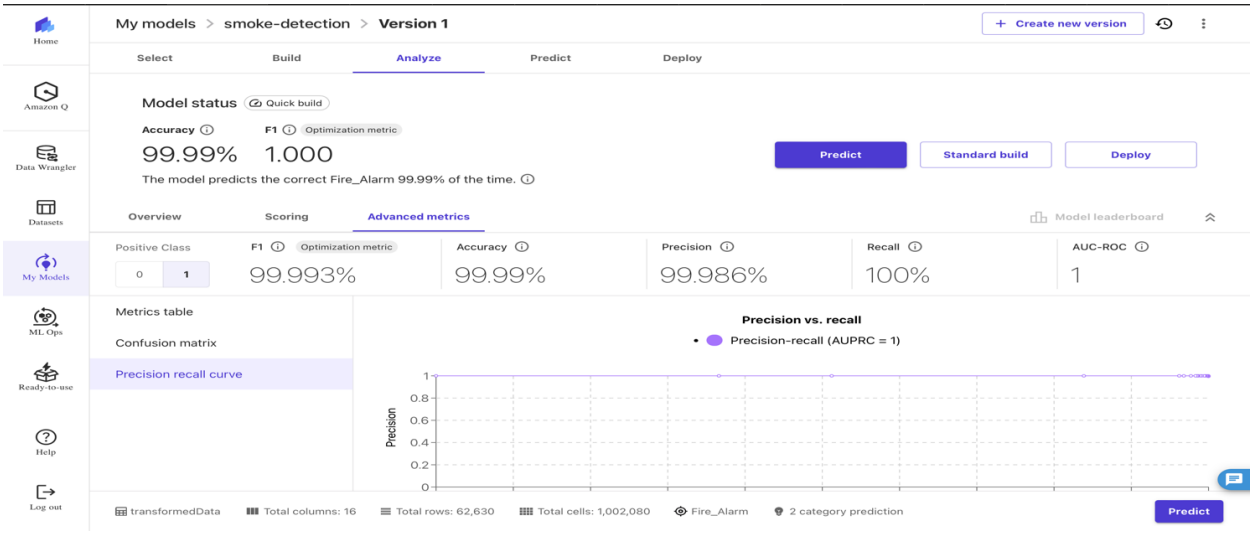
Overall Metrics



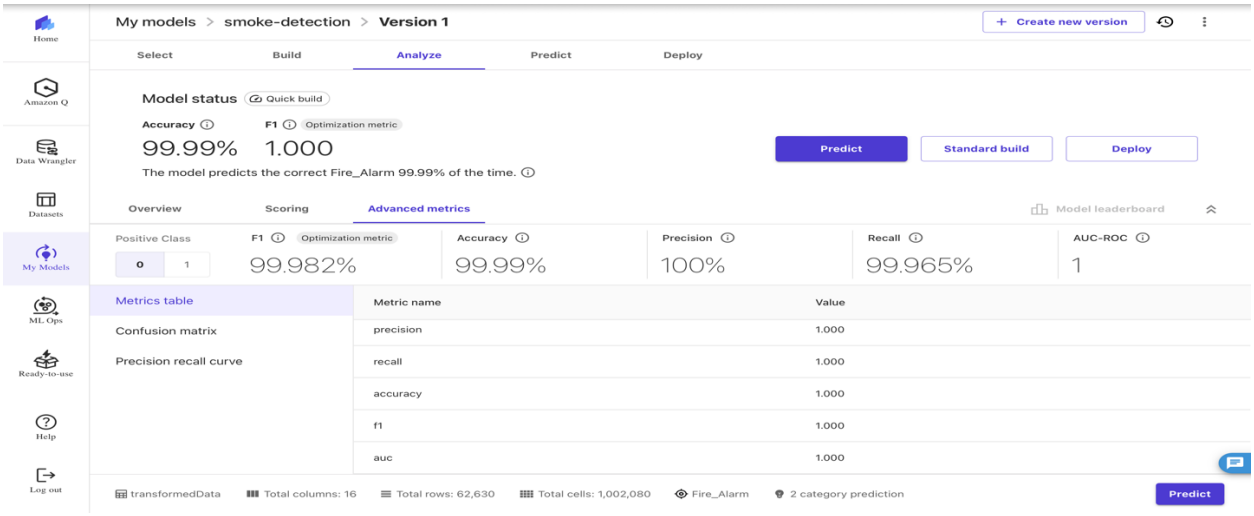
Per-class Metrics (1 - Fire Alarm)

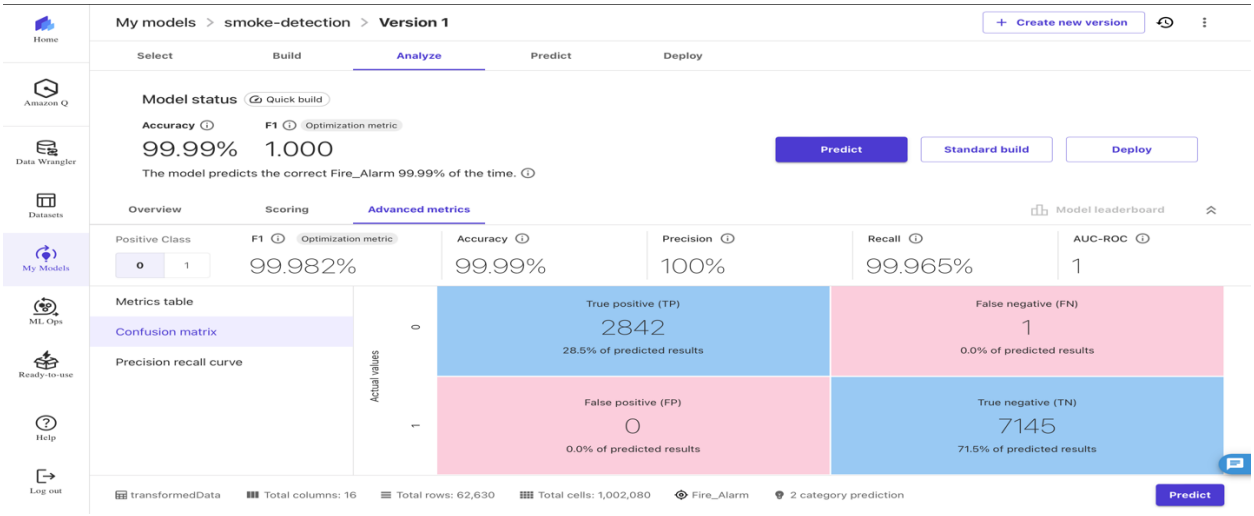


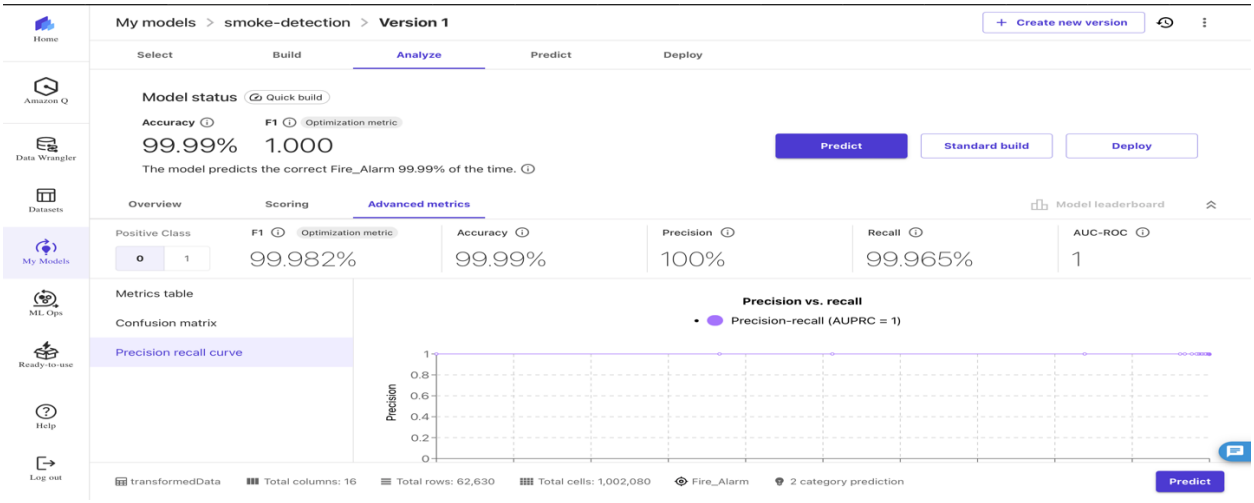




Per-class Metrics (0 - No Fire Alarm)







Classification Performance:

Class 1 (Fire Alarm):

The model achieved 99.99% accuracy, demonstrating exceptional capability in correctly identifying fire alarm instances while minimizing false negatives.

Precision: 99.986%, a minimal rate of false positives, indicating high reliability in detecting fire alarms.

Recall: 100% ensuring no fire alarms are missed, a critical factor for safety applications.

Class 0 (No Fire Alarm):

The model achieved 99.965% recall, indicating strong accuracy in identifying non-fire scenarios.

Precision: 100% no false alarms for class 0, reflecting the model's robustness in discerning normal conditions.

True Negative Rate (Specificity): 100% ensuring all actual no-fire instances are correctly classified.

Overall Performance:

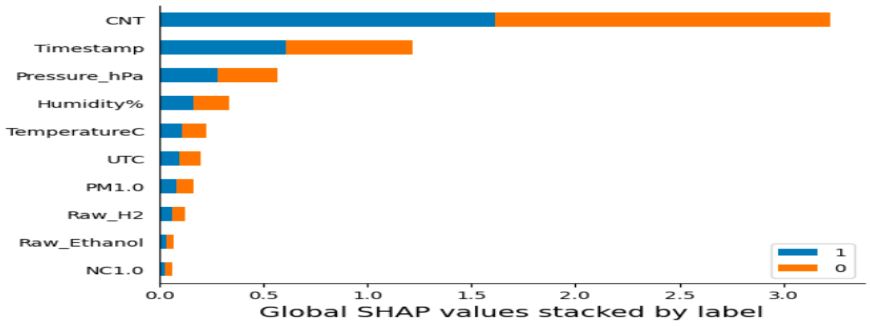
The model demonstrates a balanced and robust performance across both classes, with near-perfect precision, recall, and F1 scores. The AUC-ROC of 1.0 confirms the model's strong ability to differentiate between fire and no-fire scenarios.

SageMaker Analysis Report

We report the following SageMaker analysis.

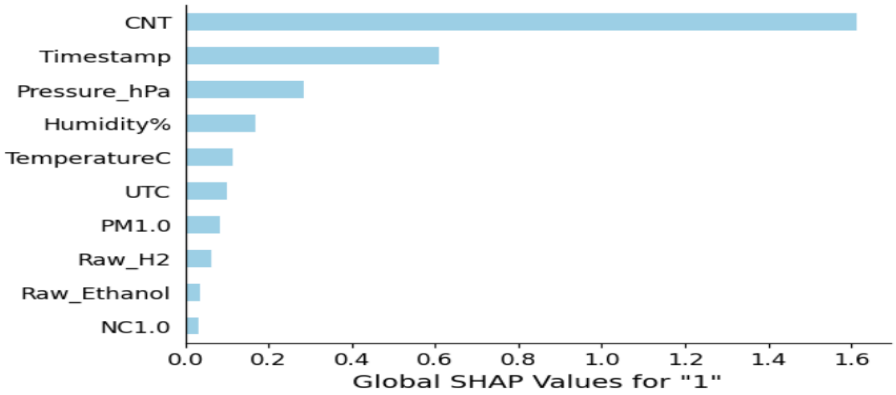
Explanations for all labels

The Model has `15` input features and `2` output labels. We computed KernelShap on the dataset and display the `10` features with the greatest feature attribution summed up over all the labels.



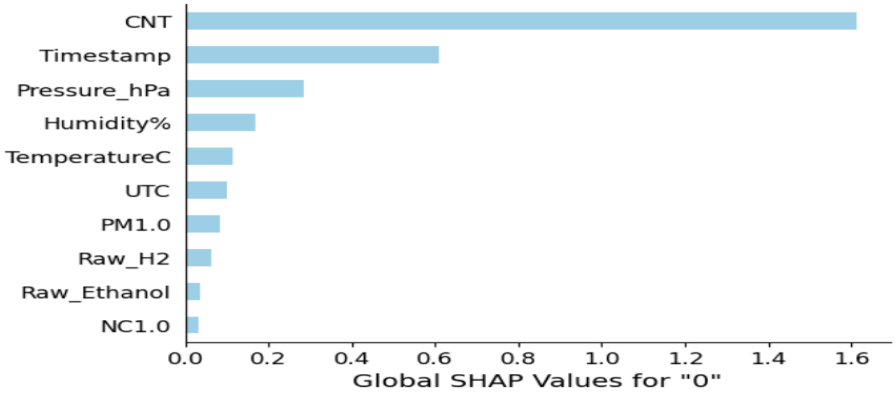
Explanations for "1"

The Model has `15` input features. We computed KernelShap on the dataset `dataset` and display the `10` features with the greatest feature attribution.



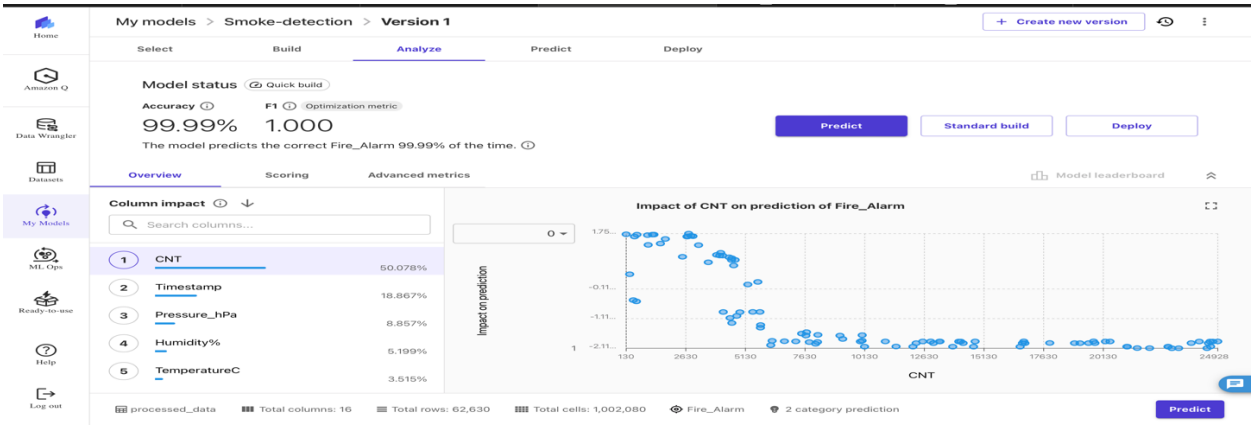
Explanations for "0"

The Model has `15` input features. We computed KernelShap on the dataset `dataset` and display the `10` features with the greatest feature attribution.



Feature Impact Analysis

SageMaker Autopilot highlighted the following features as most influential for predictions:



Key Observations:

- CNT (a sensor-derived metric) contributed to over 50% of the predictions, suggesting that the dataset relies heavily on sensor-based data.
- Timestamp indicated potential temporal dependencies, hinting at the need for temporal validation to ensure model generalization.
- Environmental factors like Pressure_hPa, Humidity[%], and Temperature[C] were moderately impactful.

Ethical Considerations and Limitations

Potential Biases

- Feature Dependency: The reliance on CNT (50.078%) may introduce bias due to skewed sensor data.

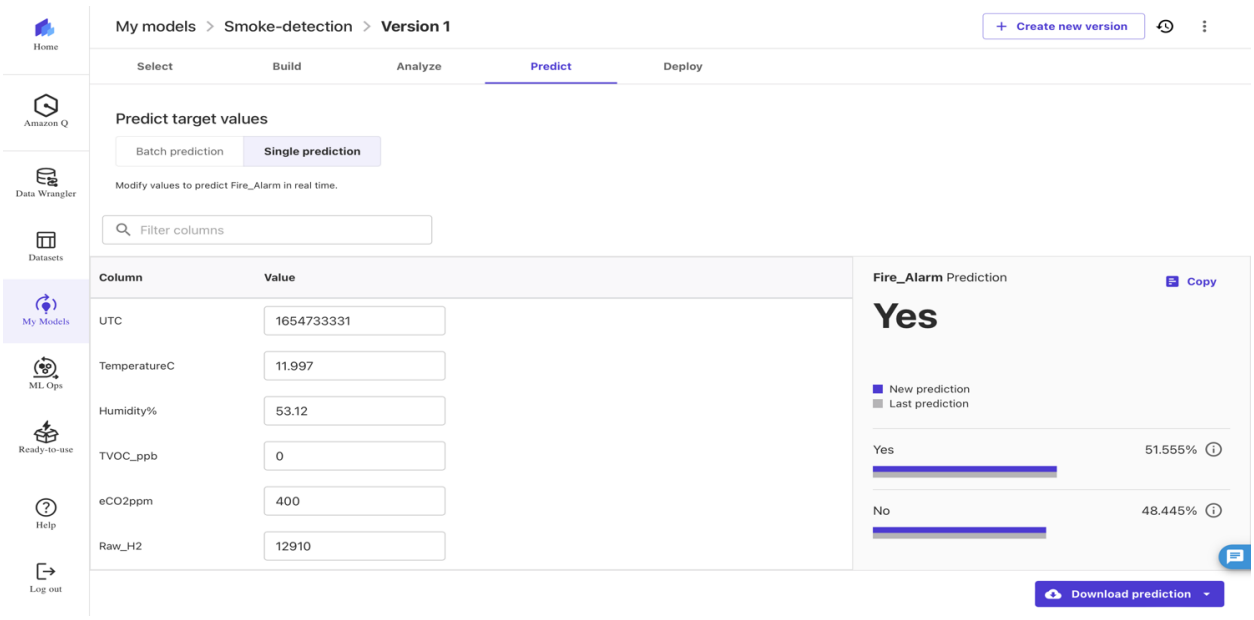
- Temporal Dependencies: Timestamp (18.867%) indicates potential over-reliance on temporal patterns, which could impact generalization to new conditions.
- Class Imbalance: Near-perfect accuracy raises concerns about potential overfitting and insufficient real-world scenario validation.

Model Performance

The model demonstrates exceptional performance with 100% precision, ensuring minimal false positives an essential factor for safety-critical applications like fire alarms. Additionally, its high recall underscores the model's reliability in accurately detecting true positive cases, thereby providing robust support for public safety.

Prediction

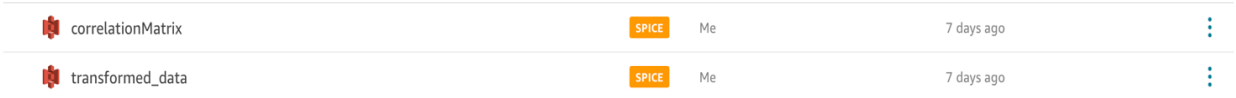
I used manual single prediction for predicting the target value i.e. fire alarm and got the below prediction.



4. Visualization

Connected QuickSight to the processed data in S3.

Added two datasets for creating visualization:

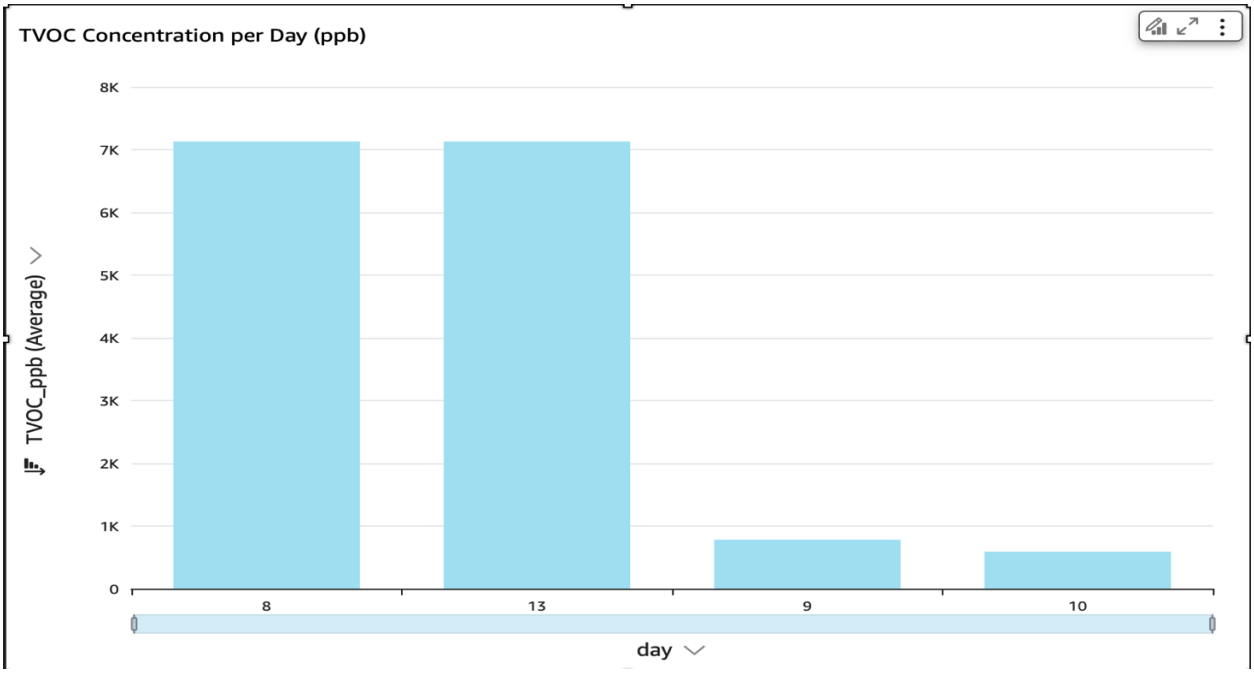


1. TVOC Concentration per Day (Bar Chart)

This bar chart provides a clear daily trend of Total Volatile Organic Compounds (TVOC) concentration in parts per billion (ppb) against days.

Insights:

- Peaks in TVOC levels may indicate specific environmental events, sensor malfunctions, or increased industrial or residential emissions.
- Analyzing these patterns helps identify days with unusually high pollution, enabling targeted environmental monitoring or intervention.

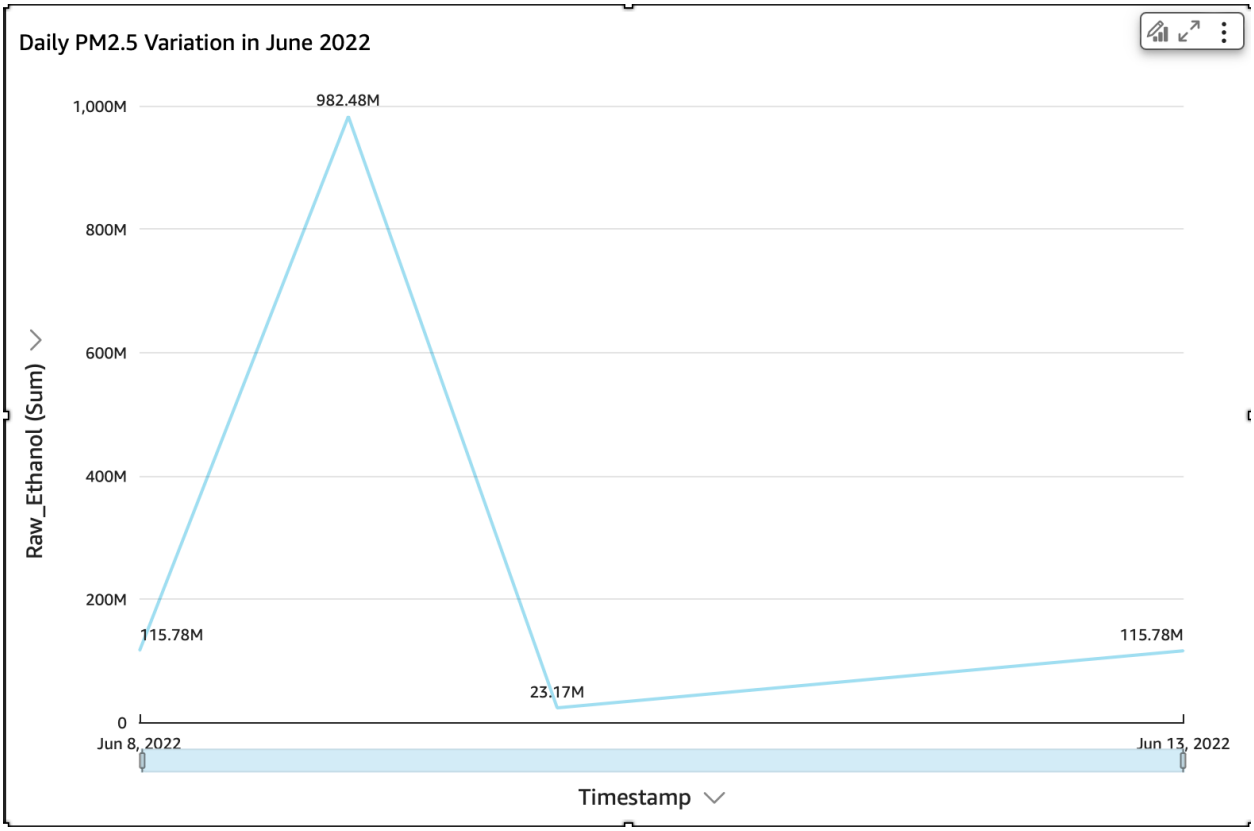


2. Daily PM2.5 Variation in June 2022 (Line Chart)

The line chart visualizes the day-to-day fluctuation in PM2.5 levels, showcasing pollution variations across June 2022.

Insights:

- Steady trends suggest consistent air quality, while spikes can highlight potential sources of pollution, such as wildfires or construction activities.
- Observing these trends over time helps in correlating pollution events with external factors, aiding proactive environmental measures.

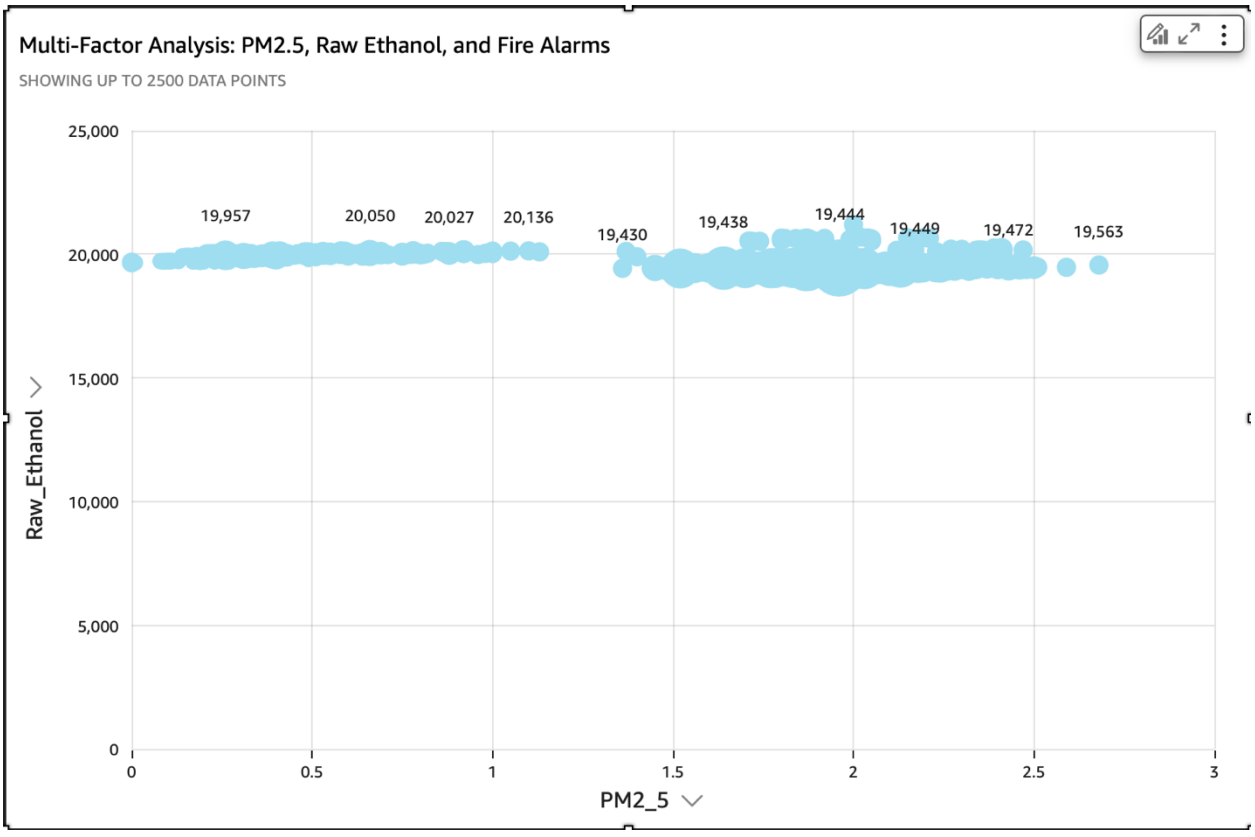


3. Multi-Factor Analysis: PM2.5, Raw Ethanol, and Fire Alarms (Scatter Plot)

This scatter plot highlights the interaction between PM2.5 levels, Raw Ethanol concentration, and the occurrence of fire alarms.

Insights:

- High concentrations of PM2.5 and Raw Ethanol often align with fire alarm triggers, showcasing their role in fire detection.
- Clusters of points reveal thresholds beyond which fire alarms are consistently activated, helping refine alarm sensitivity.

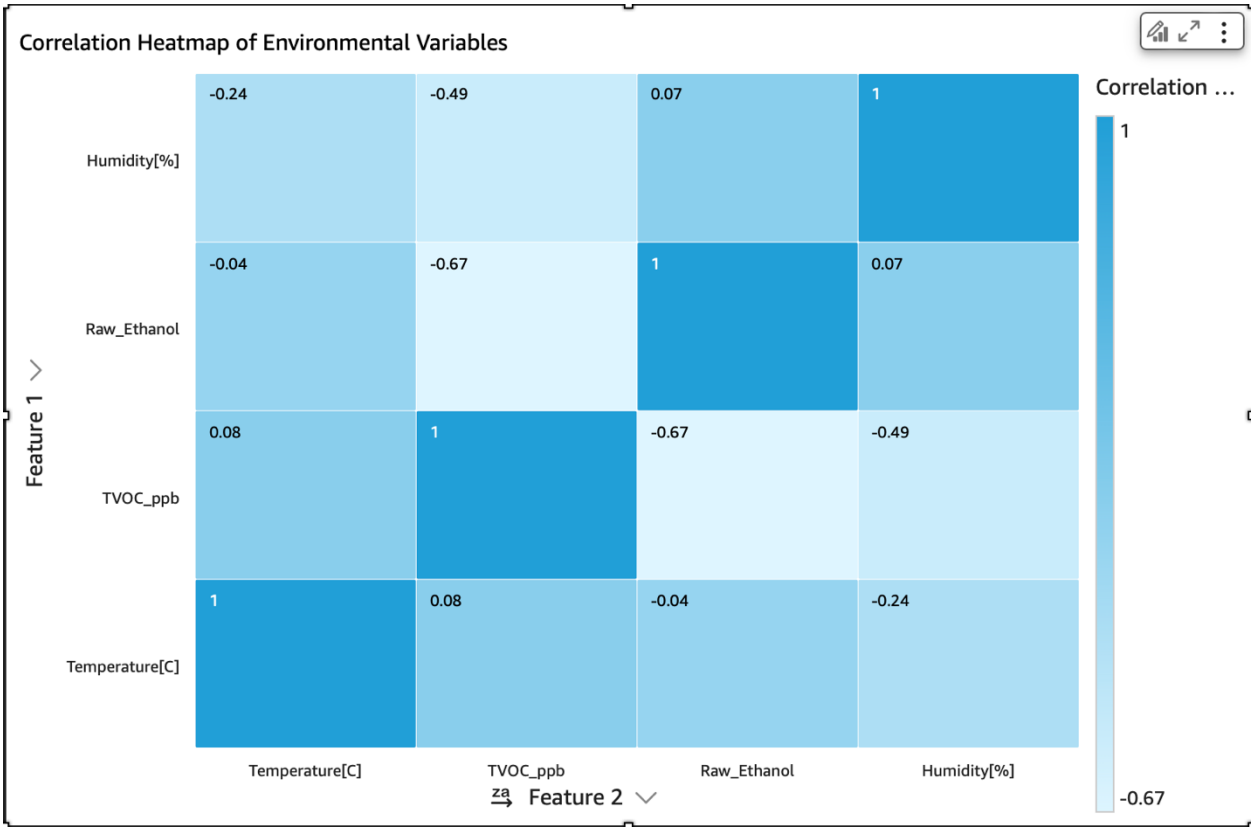


4. Correlation Heatmap of Environmental Variables (Heatmap)

This heatmap illustrates the strength of relationships between key environmental variables like Temperature, Humidity, TVOC, and Raw Ethanol.

Insights:

- Strong positive or negative correlations indicate how changes in one variable influence others.
- Identifying these relationships aids in predictive modeling and understanding environmental dynamics.



Bonus Task: Automation of the Pipeline

Objective:

Automate the entire data pipeline from ingestion to visualization, ensuring a seamless and efficient process for handling big data. This includes automating data retrieval, processing, storage, triggering, logging, notifications, and visualization updates.

Steps used:

1. Automation Script : Automate the retrieval, processing, and storage of datasets.

Implementation:

The script uses AWS Lambda to respond to S3 events triggered by new file uploads.

Steps performed by the script:

- Dynamic retrieval: Handles various files dynamically using bucket name and file key from S3 event triggers.
- Processing: Converts ‘UTC’ timestamps to human-readable ‘Timestamp’ while extracting ‘Year’ and ‘Month’.
- Storage: Processed files are uploaded back to the ‘processed/’ folder in S3 for downstream usage.

The below is a function written for automation process where in I have created a lambda function called SmokeDetectionPipelineTrigger.

lambda_function.py

```python
import boto3
import os
import pandas as pd
from datetime import datetime

s3_client = boto3.client('s3')
sns_client = boto3.client('sns')

def lambda_handler(event, context):
    try:
        for record in event['Records']:
            # This block ynamically retrieve bucket name and file key
            bucket_name = record['s3']['bucket']['name']
            file_key = record['s3']['object']['key']
            print(f"Processing file: {file_key} from bucket: {bucket_name}")

            # Downloads the raw file from S3 to the Lambda temporary directory for processing
            local_file_path = f"/tmp/{os.path.basename(file_key)}"
            s3_client.download_file(bucket_name, file_key, local_file_path)

            # Process the file: Add 'Timestamp', 'Year', and 'Month' columns
            processed_file_path = f"/tmp/processed-{os.path.basename(file_key)}"
            df = pd.read_csv(local_file_path)

            # Add "Timestamp" column based on 'UTC'
            df['Timestamp'] = pd.to_datetime(df['UTC'], unit='s')
            # Add "Year" and "Month" columns from 'Timestamp'
            df['Year'] = df['Timestamp'].dt.year
            df['Month'] = df['Timestamp'].dt.month


            df.to_csv(processed_file_path, index=False)

            # Upload the processed file back to the S3 bucket
            processed_key = f"processed/{os.path.basename(file_key)}"
            s3_client.upload_file(processed_file_path, bucket_name, processed_key)
            print(f"Processed file uploaded to: {processed_key}")

        # Notify success using AWS SNS
        sns_client.publish(
            TopicArn="arn:aws:sns:us-east-1:727646487685:PipelineNotifications",
            Message=f"File {file_key} successfully processed and uploaded to {processed_key}.",
            Subject="Pipeline Success"
        )
        return {"statusCode": 200, "body": "File processed successfully!"}

    except Exception as e:
        # Notify failure using AWS SNS
        sns_client.publish(
            TopicArn="arn:aws:sns:us-east-1:727646487685:PipelineNotifications",
            Message=f"Error processing file {file_key}: {str(e)}",
            Subject="Pipeline Failure"
        )
        raise e
```

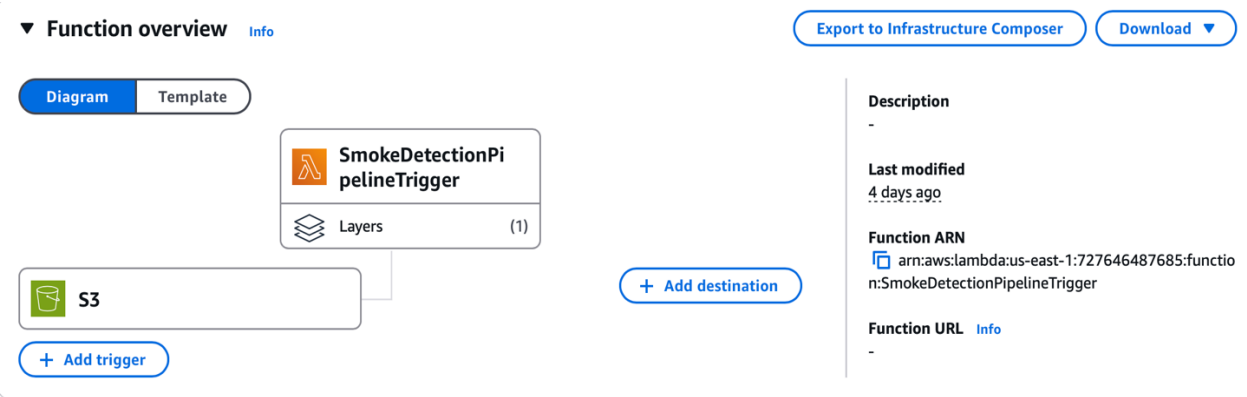
2. Set Up Scheduling Tools

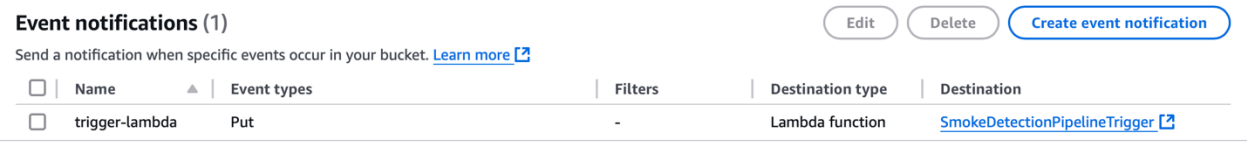
Scheduled the pipeline to trigger automatically.

AWS Lambda: Configured to trigger upon new uploads to the S3 bucket using S3 event notifications.

Layers: Added a pandas layer to support pandas for python 3.9.

In s3 bucket, I have added event notification where in the destination is given as the lambda function. (trigger-lambda)





3. Implement Logging and Notifications

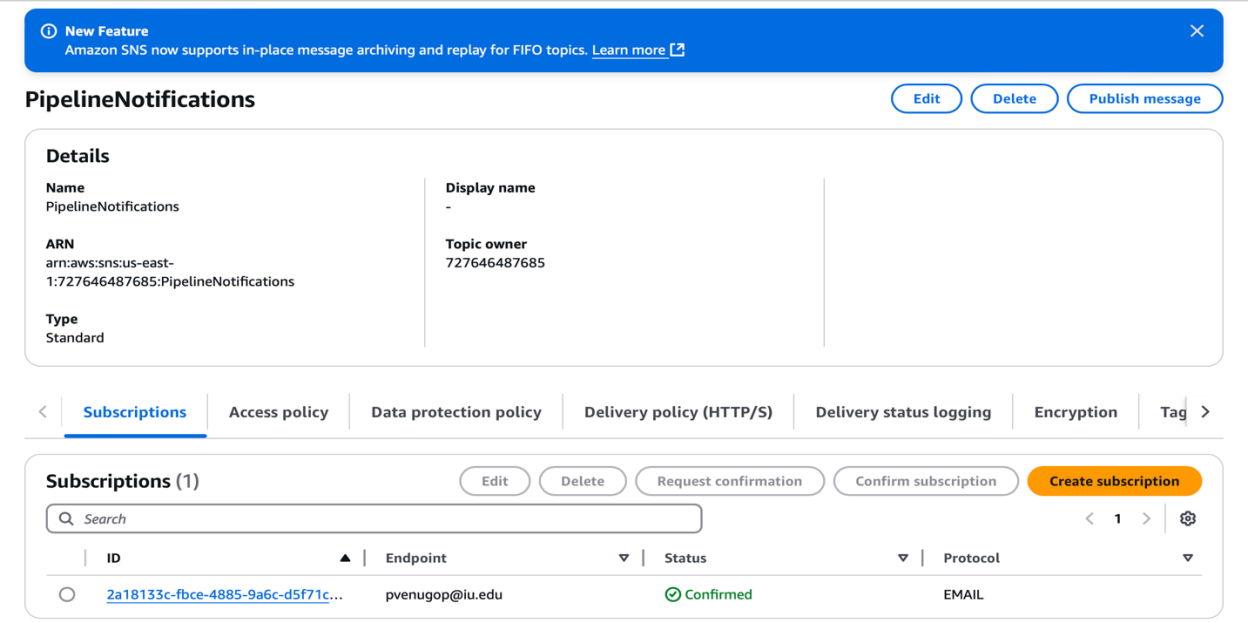
Provide real-time updates on the pipeline's status to ensure visibility and quick issue resolution.

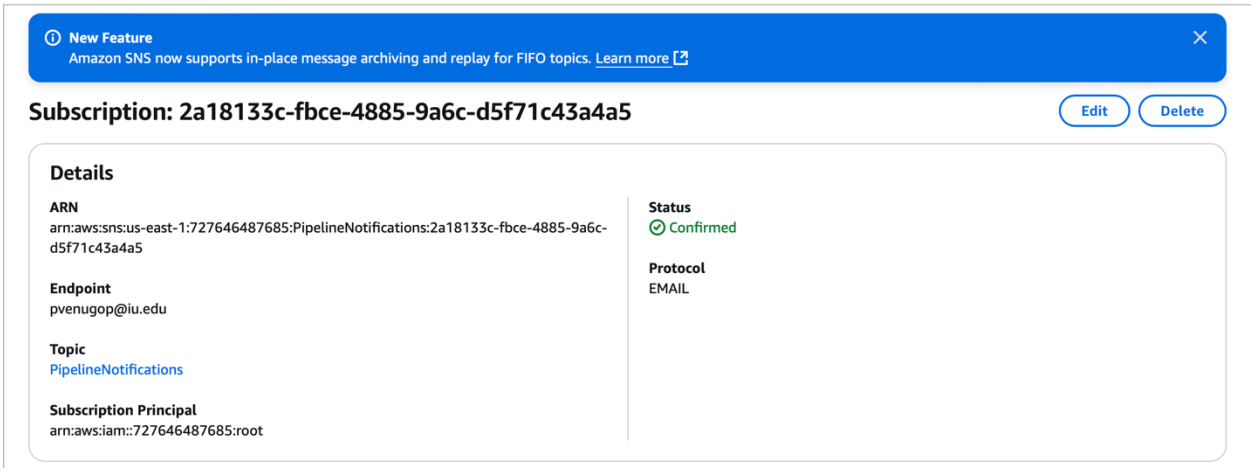
Logging: All events and errors are logged in AWS CloudWatch, allowing detailed monitoring of the pipeline’s progress.

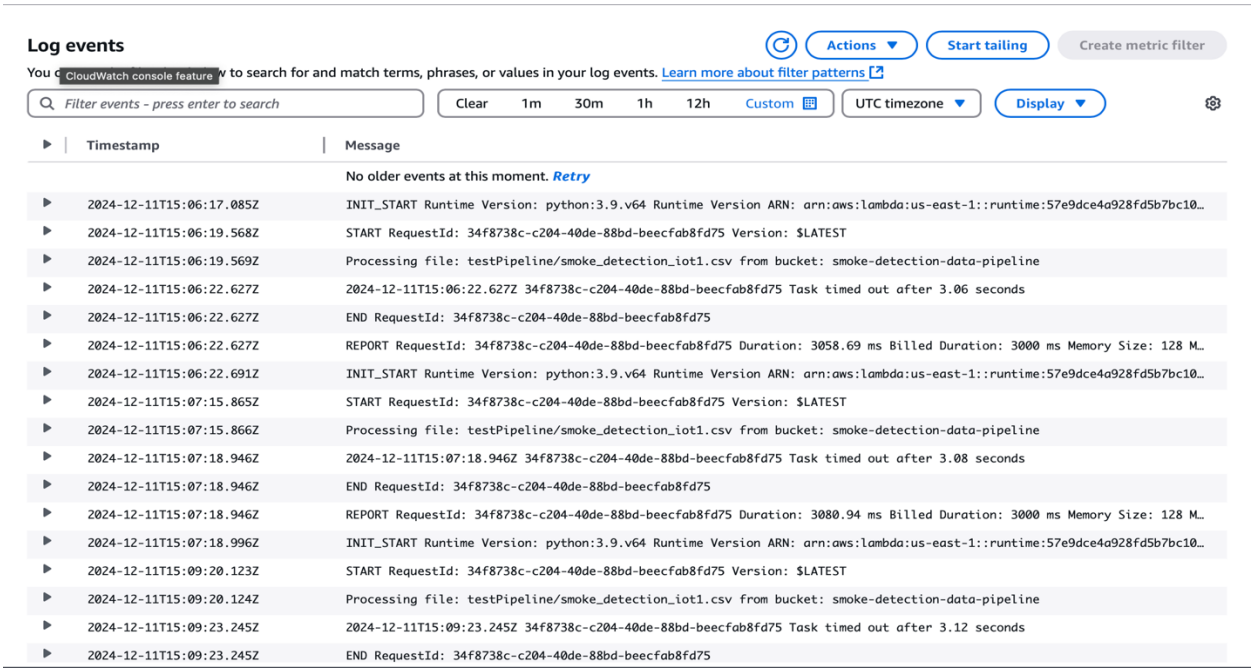
Notifications:

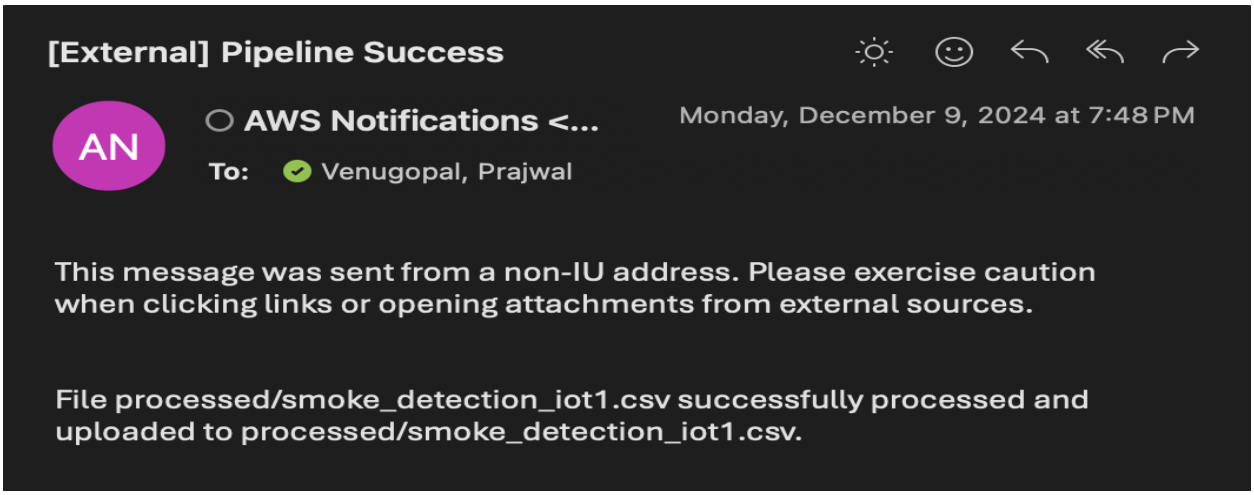
AWS SNS (Simple Notification Service): Sends notifications to subscribed users about the pipeline status (success or failure).

Notifications contain details about the processed file and any errors encountered. The sns name given is PipelineNotifications.







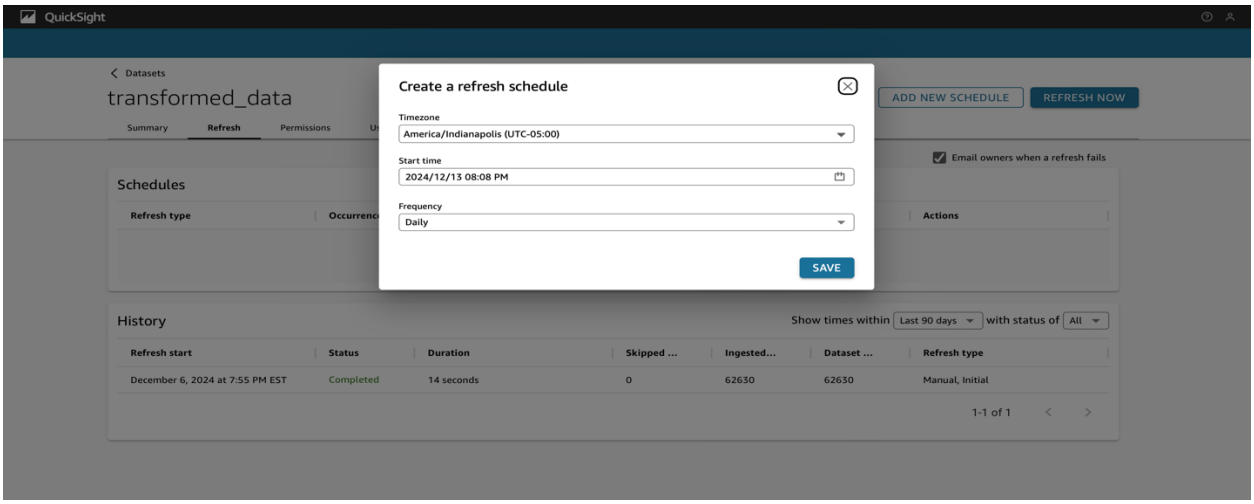


4. Automate Dashboard Updates

Enable real-time updates of visualizations for data insights.

AWS Quick Sight: Configured to automatically refresh datasets connected to the processed files in S3.

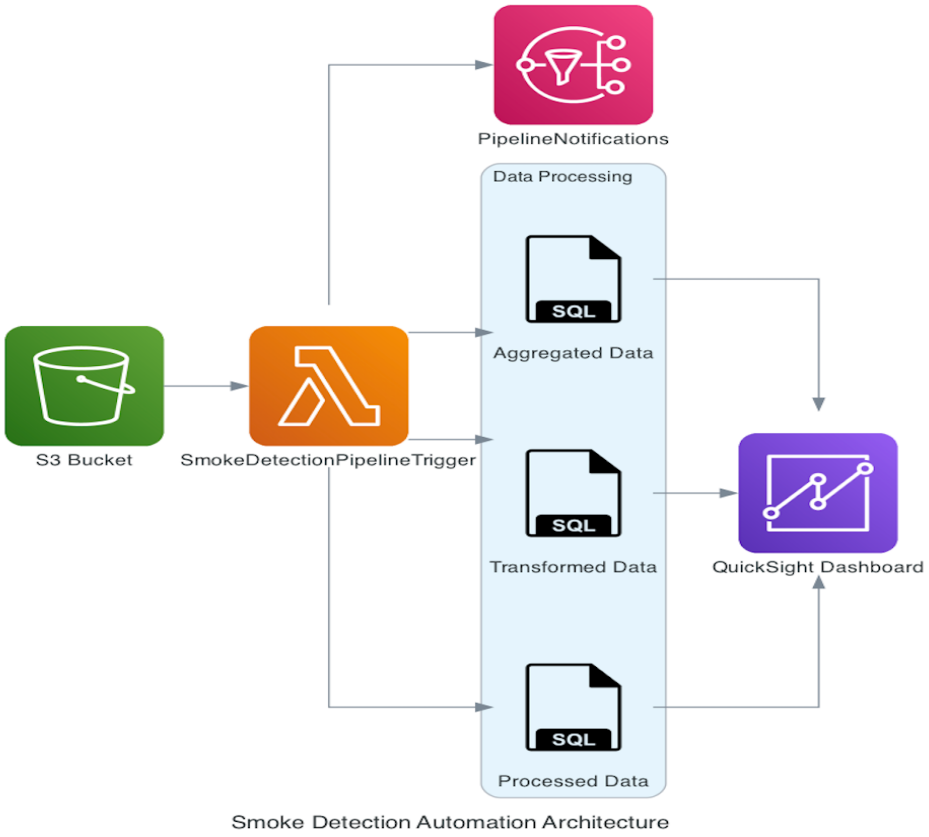
Key Features: Dynamic dashboard updates upon new data ingestion.



Benefits of Automation

- Efficiency: Eliminates manual intervention for data processing and visualization updates.
- Scalability: Handles large datasets and frequent uploads seamlessly.
- Monitoring: Provides transparency into the pipeline's health through real-time notifications and logs.
- Timeliness: Ensures dashboards are always up-to-date for immediate insights.

Architecture Diagram of automation process



Conclusion

This will show the effective execution of this project and give examples that combine AWS with PySpark in the creation and development of a scalable big data pipeline. From ingestion through automation, the project touches on key elements in the process of data engineering, big data analytics, and cloud computing to guarantee proficient processing and visualization of extensive datasets.

Key highlights of this project include:

- Scalable Data Pipeline: By using AWS S3 to store the data reliably and PySpark for distributed processing, this provides a scalable and high-performance IoT sensor data handling capability.
- Insightful Analysis: Use of data transformations, aggregations, and advanced Spark SQL queries gave much insight into environmental conditions and trends relating to fire alarms.
- Integration of Machine Learning: The AutoML application in the scenario is seen with AWS SageMaker Autopilot to train a classification model up to almost perfect accuracy for predictive analytics.
- Dynamic Visualization: The creation of the visually appealing dashboards with AWS QuickSight made the data more interpretable so that stakeholders could easily extract actionable insights.
- End-to-End Automation: The pipeline's automation using AWS Lambda and SNS streamlined operations, ensuring timely notifications and real-time dashboard updates for a fully integrated workflow.