Mini Project

Introduction

Goal of Project:

This project's objective is to efficiently manage large-scale datasets by designing and implementing a big data pipeline with AWS. Data ingestion, processing, analysis and development of trained machine learning models, and visualization are among the tasks that are part of the pipeline. It seeks to address important ethical issues including bias in training data and privacy concerns while fostering the development of crucial skills in big data engineering, cloud-based distributed systems, and analytics.

In order to improve decision-making and obtain real-time insights, this project also places a strong emphasis on automation. Using the power of AWS integrated tools like S3, EC2, Sage Maker, Quick sight, lambda functions we try to achieve above goals. This will help in understanding and learning of all AWS tools and technologies in processing and analyzing big data projects.

The deployment of this big data pipeline demonstrates the revolutionary possibilities of integrating cloud-based services and distributed computing to address challenging data issues. All things considered, this project acts as a guide for creating reliable, moral, and scalable data solutions, giving people the abilities and resources they need to succeed in the quickly developing big data and cloud computing industries.

Dataset Choice

The dataset I have chosen for this project is named HomeC.csv which is basically a IoT sensor data. The dataset consist of 29 columns of wide range including time, use [kW], gen [kW], House overall [kW], Dishwasher [kW], Furnace 1 [kW], Furnace 2 [kW], Home office [kW], Fridge [kW], Wine cellar [kW], Garage door [kW], Kitchen 12 [kW], Kitchen 14 [kW], Kitchen 38 [kW], Barn [kW], Well [kW], Microwave [kW], Living room [kW], Solar [kW], temperature, humidity, visibility, apparentTemperature, pressure, windSpeed, windBearing, precipIntensity, dewPoint, precipProbability.This dataset provides analysis of house parameters and its corresponding effects on temperature. For example , they have measured a dishwasher power based on its usage in kilo watts and temperature generated because of operation. The dataset consist of only numerical values and no few special numeric like exponentials have been calculated.

With millions of rows, the given dataset is a massive IoT sensor data file that is perfect for a big data pipeline project. With the use of this dataset, which records time-series data from Internet of Things sensors, I have investigated the practical difficulties of large-scale data engineering and analytics. Because dataset includes a variety of columns, this facilitate different analytics, aggregation, and transformation operations. Because of its variety in columns(29) and coverage of almost all power aspects such as pressure, temperature, windspeed , humidity etc. this helps in deep understanding and is a complete dataset.

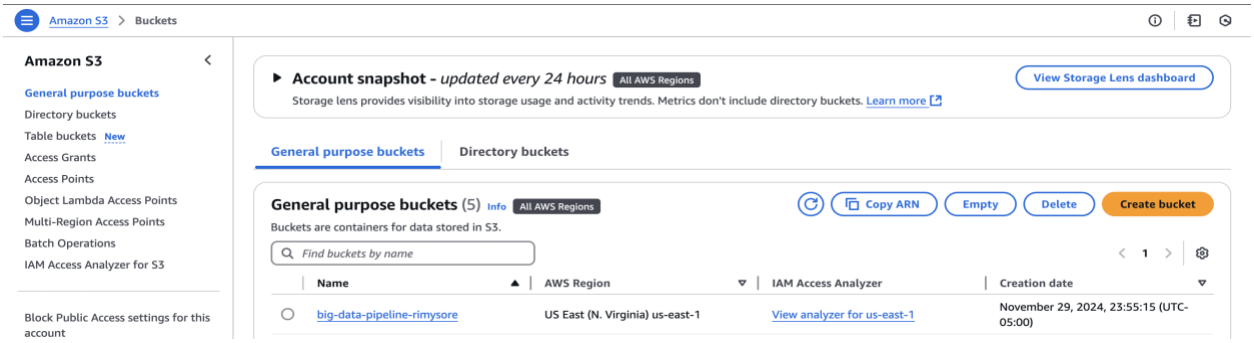
Methodology

2. Environment Setup

2.1 AWS S3 for Data Storage

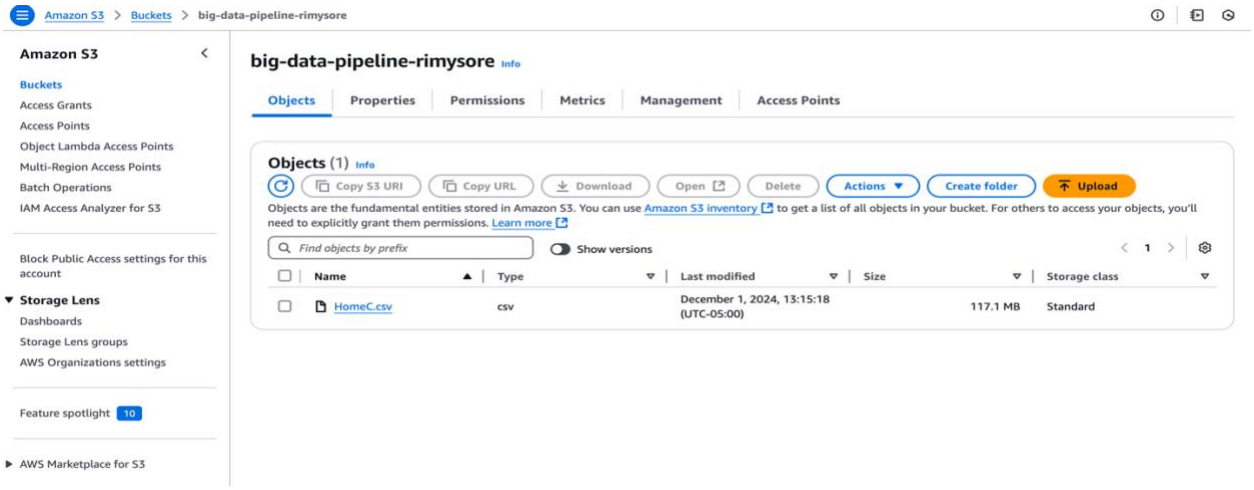
2.1.a task1 Creation of AWS S3 bucket to store data:

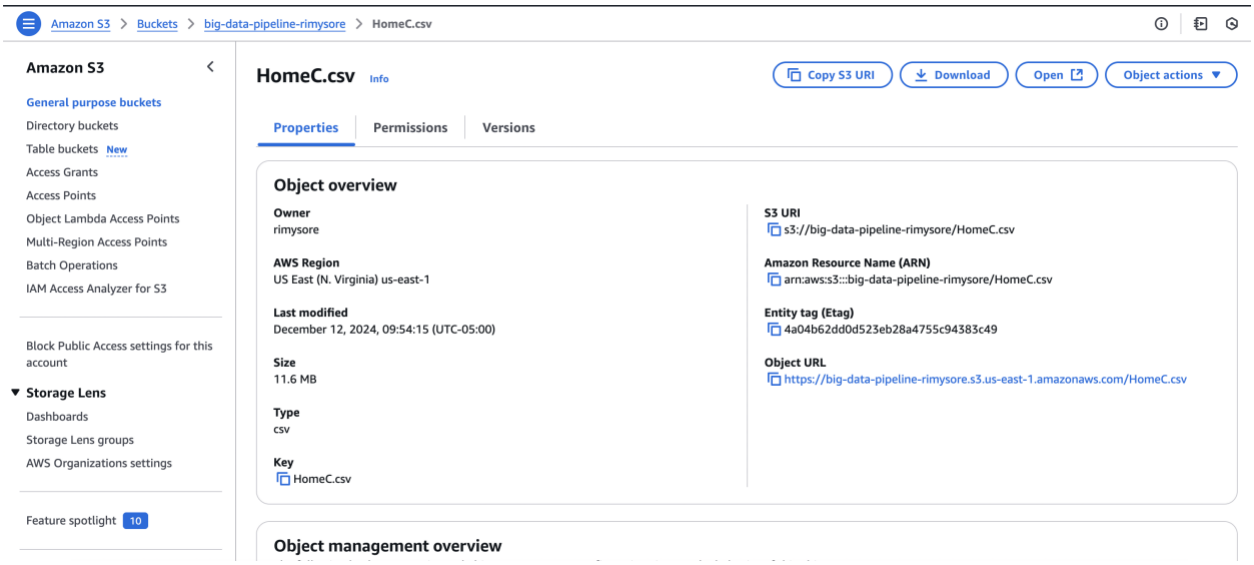
I have created a S3 bucket in aws to store both processed and raw dataset.



2.1.b task2 Upload the raw dataset to the S3 bucket

Uploaded the HomeC.csv file as a raw dataset.

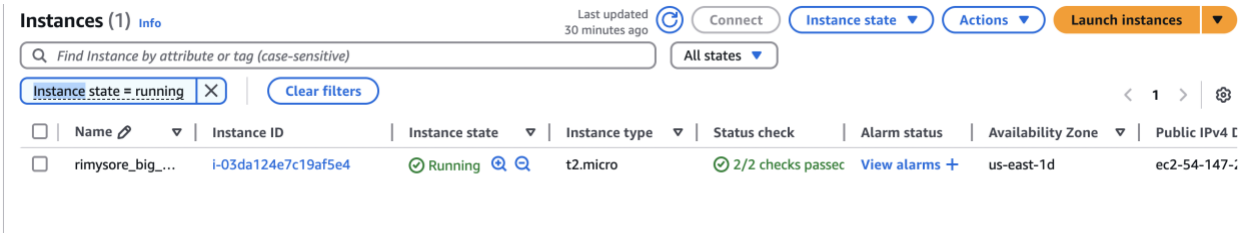




2.2 Linux Environment with PySpark

2.2.a task1 Set up a Linux-based environment, either locally or using an AWS EC2 Instance

Creation of EC2 instance in aws:



Connect to Linux-based EC2 instance using ssh command :

```
ssh: connect to host 54.221.22.215 port 22: Operation timed out
(base) rithvikms@Rithviks-MacBook-Pro-2 Mini Project % ssh -i "rimysore.pem" ec2-user@54.147.209.146
The authenticity of host '54.147.209.146 (54.147.209.146)' can't be established.
ED25519 key fingerprint is SHA256:4+vaycrvtme0swnIfIfSJOLwhTsfPQXLfeMJ2VuZQsU.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:7: 54.221.22.215
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '54.147.209.146' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Thu Dec  5 20:53:33 2024 from 73.146.57.63
```

2.2.b task2 Install PySpark for distributed data processing.

Used command - pip install pyspark to install pyspark.

```
[ec2-user@ip-172-31-26-144 ~]$ df -h /dev/xvda1
Filesystem      Size  Used Avail Use% Mounted on
/dev/xvda1       20G  3.7G   17G  19% /
[ec2-user@ip-172-31-26-144 ~]$ sudo yum clean all
sudo rm -rf /var/cache/yum
17 files removed
[ec2-user@ip-172-31-26-144 ~]$ df -h /dev/xvda1
Filesystem      Size  Used Avail Use% Mounted on
/dev/xvda1       20G  3.6G   17G  18% /
[ec2-user@ip-172-31-26-144 ~]$ sudo rm -rf /tmp/*
[ec2-user@ip-172-31-26-144 ~]$ df -h /tmp
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           475M     0  475M   0% /tmp
[ec2-user@ip-172-31-26-144 ~]$ mkdir -p /home/ec2-user/pip_temp
[ec2-user@ip-172-31-26-144 ~]$ export TMPDIR=/home/ec2-user/pip_temp/
[ec2-user@ip-172-31-26-144 ~]$ echo $TMPDIR
/home/ec2-user/pip_temp/
[ec2-user@ip-172-31-26-144 ~]$ 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 8.9 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 29.1 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
[ec2-user@ip-172-31-26-144 ~]$
```

2.2.c task3 Configure AWS CLI to interact with S3 buckets.

Configured aws cli to interact with s3 bucket to fetch raw dataset from bucket and upload processed dataset back to bucket.

```
[ec2-user@ip-172-31-26-144 ~]$ aws configure
AWS Access Key ID [****************CKNA]: AKIAYLZZKBOQBATJCKNA
AWS Secret Access Key [****************lQUZ]: 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
Default region name [us-east-1]:
Default output format [json]:
[ec2-user@ip-172-31-26-144 ~]$ aws s3 ls
2024-11-30 04:55:15 big-data-pipeline-rimysore
```

3. Data Pipeline Tasks

Task 1: Data Ingestion from S3

- In this ingestion step I have used AWS CLI to interact with bucket.
- Connected to jupyter notebook via cli and developed pyspark code to pull raw data in s3 bucket (HomeC.csv) to pyspark environment.
- These below images show connection establishment.

```
[ec2-user@ip-172-31-26-144 ~]$ jupyter notebook --ip=0.0.0.0 --port=8888 --no-browser
[I 2024-12-12 16:05:04.975 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-12-12 16:05:04.979 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-12-12 16:05:04.984 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-12-12 16:05:04.988 ServerApp] notebook | extension was successfully linked.
[I 2024-12-12 16:05:05.662 ServerApp] notebook_shim | extension was successfully linked.
[I 2024-12-12 16:05:05.702 ServerApp] notebook_shim | extension was successfully loaded.
[I 2024-12-12 16:05:05.704 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2024-12-12 16:05:05.705 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2024-12-12 16:05:05.710 LabApp] JupyterLab extension loaded from /home/ec2-user/.local/lib/python3.9/site-packages/jupyterlab
[I 2024-12-12 16:05:05.711 LabApp] JupyterLab application directory is /home/ec2-user/.local/share/jupyter/lab
[I 2024-12-12 16:05:05.711 LabApp] Extension Manager is 'pypi'.
[W 2024-12-12 16:05:05.711 LabApp] Failed to instantiate the extension manager pypi. Falling back to read-only manager.
    Traceback (most recent call last):
      File "/home/ec2-user/.local/lib/python3.9/site-packages/jupyterlab/labapp.py", line 837, in initialize_handlers
        ext_manager = manager_factory(app_options, listings_config, self)
      File "/home/ec2-user/.local/lib/python3.9/site-packages/jupyterlab/extensions/__init__.py", line 46, in get_pypi_manager
        return PyPIExtensionManager(app_options, ext_options, parent)
      File "/home/ec2-user/.local/lib/python3.9/site-packages/jupyterlab/extensions/pypi.py", line 134, in __init__
        self._httpx_client = httpx.AsyncClient(proxies=proxies)
    TypeError: __init__() got an unexpected keyword argument 'proxies'
[I 2024-12-12 16:05:05.722 ServerApp] jupyterlab | extension was successfully loaded.
[I 2024-12-12 16:05:05.725 ServerApp] notebook | extension was successfully loaded.
[I 2024-12-12 16:05:05.726 ServerApp] Serving notebooks from local directory: /home/ec2-user
[I 2024-12-12 16:05:05.726 ServerApp] Jupyter Server 2.14.2 is running at:
[I 2024-12-12 16:05:05.726 ServerApp] http://ip-172-31-26-144.ec2.internal:8888/tree?token=f89eb3b724eec1de85ebb5935be6b44aedfbd9b0b49dbf56
[I 2024-12-12 16:05:05.726 ServerApp]     http://127.0.0.1:8888/tree?token=f89eb3b724eec1de85ebb5935be6b44aedfbd9b0b49dbf56
[I 2024-12-12 16:05:05.726 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 2024-12-12 16:05:05.730 ServerApp]

    To access the server, open this file in a browser:
        file:///home/ec2-user/.local/share/jupyter/runtime/jpserver-5150-open.html
    Or copy and paste one of these URLs:
```

```
(base) rithvikms@Rithviks-MacBook-Pro-2 Mini Project % ssh -i "rimysore.pem" -L 9999:localhost:8888 ec2-user@54.147.209.146
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
```

- Ingestion pyspark code and its output

```python
#task3 - data pipeline tasks
#Use AWS CLI or PySpark's built-in S3 support to load the dataset directly.
from pyspark.sql import SparkSession

# Initialize Spark session
spark = SparkSession.builder \
    .appName("Verify HomeC Dataset") \
    .getOrCreate()

# Path to the dataset on EC2
file_path = "/home/ec2-user/HomeC.csv"

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

# Show the schema of the dataset
print("Schema of the dataset:")
dataset.printSchema()

# Display the first 10 rows of the dataset
print("First 10 rows of the dataset:")
dataset.show(10, truncate=False)
```

```
Schema of the dataset:
root
 |-- time: string (nullable = true)
 |-- use [kW]: double (nullable = true)
 |-- gen [kW]: double (nullable = true)
 |-- House overall [kW]: double (nullable = true)
 |-- Dishwasher [kW]: double (nullable = true)
 |-- Furnace 1 [kW]: double (nullable = true)
 |-- Furnace 2 [kW]: double (nullable = true)
 |-- Home office [kW]: double (nullable = true)
 |-- Fridge [kW]: double (nullable = true)
 |-- Wine cellar [kW]: double (nullable = true)
 |-- Garage door [kW]: double (nullable = true)
 |-- Kitchen 12 [kW]: double (nullable = true)
 |-- Kitchen 14 [kW]: double (nullable = true)
 |-- Kitchen 38 [kW]: double (nullable = true)
 |-- Barn [kW]: double (nullable = true)
 |-- Well [kW]: double (nullable = true)
 |-- Microwave [kW]: double (nullable = true)
 |-- Living room [kW]: double (nullable = true)
 |-- Solar [kW]: double (nullable = true)
 |-- temperature: double (nullable = true)
 |-- humidity: double (nullable = true)
 |-- visibility: double (nullable = true)
 |-- apparentTemperature: double (nullable = true)
 |-- pressure: double (nullable = true)
 |-- windSpeed: double (nullable = true)
 |-- windBearing: double (nullable = true)
 |-- precipIntensity: double (nullable = true)
 |-- dewPoint: double (nullable = true)
 |-- precipProbability: double (nullable = true)
```

- Output – first 10 rows

```
+----------+-----------+-----------+--------------------+-----------------+----------------+----------------+-----------------+------------+----------
-----+-----------------+------------------+------------------+------------------+------------+-------------+-----------------+------------------+-----------+--
---------+--------+----------+--------------------+--------+---------+-----------+----------------+--------+-----------------+
|time      |use [kW]    |gen [kW]    |House overall [kW]|Dishwasher [kW]|Furnace 1 [kW]|Furnace 2 [kW]|Home office [kW]|Fridge [kW]|Wine cellar
[kW]|Garage door [kW]|Kitchen 12 [kW]|Kitchen 14 [kW]|Kitchen 38 [kW]|Barn [kW]   |Well [kW]   |Microwave [kW]|Living room [kW]|Solar [kW] |tem
perature|humidity|visibility|apparentTemperature|pressure|windSpeed|windBearing|precipIntensity|dewPoint|precipProbability|
+----------+-----------+-----------+--------------------+-----------------+----------------+----------------+-----------------+------------+----------
-----+-----------------+------------------+------------------+------------------+------------+-------------+-----------------+------------------+-----------+--
---------+--------+----------+--------------------+--------+---------+-----------+----------------+--------+-----------------+
|1451624400|0.932833333|0.003483333|0.932833333        |3.33E-5        |0.0207        |0.061916667   |0.442633333     |0.12415    |0.006983333
|0.013083333     |4.16667E-4     |1.5E-4         |0.0            |0.03135     |0.001016667|0.004066667   |0.001516667     |0.003483333|36.14
|0.62    |10.0      |29.26              |1016.91 |9.18     |282.0      |0.0             |24.4    |0.0              |
|1451624401|0.934333333|0.003466667|0.934333333        |0.0            |0.020716667   |0.063816667   |0.444066667     |0.124      |0.006983333
|0.013116667     |4.16667E-4     |1.5E-4         |0.0            |0.0315      |0.001016667|0.004066667   |0.00165         |0.003466667|36.14
|0.62    |10.0      |29.26              |1016.91 |9.18     |282.0      |0.0             |24.4    |0.0              |
|1451624402|0.931816667|0.003466667|0.931816667        |1.67E-5        |0.0207        |0.062316667   |0.446066667     |0.123533333|0.006983333
|0.013083333     |4.33333E-4     |1.66667E-4     |1.67E-5        |0.031516667|0.001      |0.004066667   |0.00165         |0.003466667|36.14
|0.62    |10.0      |29.26              |1016.91 |9.18     |282.0      |0.0             |24.4    |0.0              |
|1451624403|1.02205    |0.003483333|1.02205            |1.67E-5        |0.1069        |0.068516667   |0.446583333     |0.123133333|0.006983333
|0.013           |4.33333E-4     |2.16667E-4     |0.0            |0.0315      |0.001016667|0.004066667   |0.001616667     |0.003483333|36.14
|0.62    |10.0      |29.26              |1016.91 |9.18     |282.0      |0.0             |24.4    |0.0              |
|1451624404|1.1394     |0.003466667|1.1394             |1.33333E-4     |0.236933333   |0.063983333   |0.446533333     |0.12285    |0.00685
|0.012783333     |4.5E-4         |3.33333E-4     |0.0            |0.0315      |0.001016667|0.004066667   |0.001583333     |0.003466667|36.14
|0.62    |10.0      |29.26              |1016.91 |9.18     |282.0      |0.0             |24.4    |0.0              |
|1451624405|1.391866667|0.003433333|1.391866667        |2.83333E-4     |0.50325       |0.063666667   |0.447033333     |0.1223     |0.006716667
|0.012433333     |4.83333E-4     |5.66667E-4     |0.0            |0.03145     |0.001016667|0.004066667   |0.001583333     |0.003433333|36.14
|0.62    |10.0      |29.26              |1016.91 |9.18     |282.0      |0.0             |24.4    |0.0              |
|1451624406|1.366216667|0.00345    |1.366216667        |2.83333E-4     |0.4994        |0.063716667   |0.443266667     |0.12205    |0.006733333
|0.012416667     |5.16667E-4     |5.5E-4         |0.0            |0.03155     |0.001033333|0.004116667   |0.001533333     |0.00345    |36.14
|0.62    |10.0      |29.26              |1016.91 |9.18     |282.0      |0.0             |24.4    |0.0              |
|1451624407|1.4319     |0.003416667|1.4319             |2.5E-4         |0.477866667   |0.178633333   |0.444283333     |0.1218     |0.006783333
|0.01255         |4.83333E-4     |4.5E-4         |0.0            |0.031733333|0.001033333|0.0042        |0.00155         |0.003416667|36.14
|0.62    |10.0      |29.26              |1016.91 |9.18     |282.0      |0.0             |24.4    |0.0              |
|1451624408|1.6273     |0.003416667|1.6273             |1.83333E-4     |0.44765       |0.3657        |0.441466667     |0.121616667|0.00695
|0.012716667     |4.66667E-4     |3.0E-4         |1.67E-5        |0.031766667|0.001016667|0.0042        |0.001566667     |0.003416667|36.14
|0.62    |10.0      |29.26              |1016.91 |9.18     |282.0      |0.0             |24.4    |0.0              |
|1451624409|1.735383333|0.003416667|1.735383333        |1.67E-5        |0.17155       |0.6825        |0.438733333     |0.121633333|0.007233333
|0.01335         |3.66667E-4     |5.0E-5         |0.0            |0.031666667|0.001016667|0.0042        |0.001616667     |0.003416667|36.14
|0.62    |10.0      |29.26              |1016.91 |9.18     |282.0      |0.0             |24.4    |0.0              |
+----------+-----------+-----------+--------------------+-----------------+----------------+----------------+-----------------+------------+----------
-----+-----------------+------------------+------------------+------------------+------------+-------------+-----------------+------------------+-----------+--
---------+--------+----------+--------------------+--------+---------+-----------+----------------+--------+-----------------+
only showing top 10 rows
```

Task 2: Data Processing with PySpark

In this task we process raw data in pyspark environment and is divided in 2 steps.

1. **Transformation** – Two new columns Year and Month has been added.

```
#task3 – Data Transformation: Create at least 2 new columns (e.g., Year, Month).
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

# Initialize Spark session
spark = SparkSession.builder \
    .appName("HomeC Data Transformation") \
    .getOrCreate()

# Path to the dataset on EC2
file_path = "/home/ec2-user/HomeC.csv"

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

# Transform 'time' column from Unix timestamp to readable timestamp
df_transformed = dataset.withColumn(
    "time",
    F.to_timestamp(F.from_unixtime(F.col("time")))
)

# Create new columns 'Year' and 'Month'
df_transformed = df_transformed.withColumn("Year", F.year("time")) \
                               .withColumn("Month", F.month("time"))

# Display the transformed dataset with new columns
print("Transformed dataset:")
df_transformed.show(10, truncate=False)
```

Output – Year and Month has been added at end.

```
Transformed dataset:
+--------------------+-----------+-----------+-----------------+--------------+-------------+-------------+---------------+-----------+--
-------------+---------------+--------------+--------------+--------------+-----------+-----------+-------------+---------------+-----
------+-----------+--------+----------+-----------------+--------+---------+-----------+--------------+--------+-----------------+----+---
--+
|time                |use [kW]   |gen [kW]   |House overall [kW]|Dishwasher [kW]|Furnace 1 [kW]|Furnace 2 [kW]|Home office [kW]|Fridge [kW]|Wi
ne cellar [kW]|Garage door [kW]|Kitchen 12 [kW]|Kitchen 14 [kW]|Kitchen 38 [kW]|Barn [kW]  |Well [kW]  |Microwave [kW]|Living room [kW]|Solar
[kW] |temperature|humidity|visibility|apparentTemperature|pressure|windSpeed|windBearing|precipIntensity|dewPoint|precipProbability|Year|Mont
h|
+--------------------+-----------+-----------+-----------------+--------------+-------------+-------------+---------------+-----------+--
-------------+---------------+--------------+--------------+--------------+-----------+-----------+-------------+---------------+-----
------+-----------+--------+----------+-----------------+--------+---------+-----------+--------------+--------+-----------------+----+---
--+
|2016-01-01 05:00:00|0.932833333|0.003483333|0.932833333       |3.33E-5       |0.0207       |0.061916667   |0.442633333     |0.12415    |0.
006983333     |0.013083333     |4.16667E-4     |1.5E-4        |0.0           |0.03135    |0.001016667|0.004066667   |0.001516667     |0.003
483333|36.14      |0.62    |10.0      |29.26             |1016.91 |9.18     |282.0      |0.0            |24.4    |0.0              |2016|1
|
```

2. Aggregation

- In aggregation step I have computed 5 key metrics
    i. Total Energy Usage by Month
    ii. Average Temperature by Month
    iii. Top 10 Months by Solar Energy Generation
    iv. Total Energy Generated vs. Consumed Per Year
    v. Monthly Average Usage for Top Appliances

```
#Task 3 - Data Aggregation: Compute at least 5 key metrics
#total_usage, total_solar, Top 10 Months by Solar Energy Generation, Energy Generated vs. Consumed Per Year
from pyspark.sql import functions as F

# 1. Total Energy Usage by Year and Month
total_usage_by_month = df_transformed.groupBy("Year", "Month").agg(
    F.sum("use [kW]").alias("Total_Usage_kW")
)
print("Total Energy Usage by Month:")
total_usage_by_month.show(10, truncate=False)

# 2. Average Monthly Temperature
avg_temp_by_month = df_transformed.groupBy("Year", "Month").agg(
    F.avg("temperature").alias("Avg_Temperature")
)
print("Average Temperature by Month:")
avg_temp_by_month.show(10, truncate=False)

# 3. Top 10 Months by Solar Energy Generation
top_solar_months = df_transformed.groupBy("Year", "Month").agg(
    F.sum("Solar [kW]").alias("Total_Solar_kW")
).orderBy(F.desc("Total_Solar_kW")).limit(10)
print("Top 10 Months by Solar Energy Generation:")
top_solar_months.show(truncate=False)

# 4. Total Energy Generated vs. Consumed Per Year
energy_comparison = df_transformed.groupBy("Year").agg(
    F.sum("use [kW]").alias("Total_Consumed_kW"),
    F.sum("gen [kW]").alias("Total_Generated_kW")
)
print("Energy Generated vs. Consumed Per Year:")
energy_comparison.show(truncate=False)

# 5. Monthly Average Usage for Top Consumer Appliances
# Selecting some key appliances for demonstration: 'Dishwasher [kW]', 'Fridge [kW]', 'Microwave [kW]'
avg_appliance_usage = df_transformed.groupBy("Year", "Month").agg(
    F.avg("Dishwasher [kW]").alias("Avg_Dishwasher_kW"),
    F.avg("Fridge [kW]").alias("Avg_Fridge_kW"),
    F.avg("Microwave [kW]").alias("Avg_Microwave_kW")
)
print("Monthly Average Usage for Top Appliances:")
avg_appliance_usage.show(10, truncate=False)
```

Total Energy Usage by Month:

```
+----+-----+----------------+
|Year|Month|Total_Usage_kW  |
+----+-----+----------------+
|NULL|NULL |NULL            |
|2016|1    |432839.7404670082|
+----+-----+----------------+
```

Average Temperature by Month:

```
[Stage 9:>                                                          (0 + 1) / 1]
+----+-----+----------------+
|Year|Month|Avg_Temperature |
+----+-----+----------------+
|NULL|NULL |NULL            |
|2016|1    |50.74193461135845|
+----+-----+----------------+
```

Top 10 Months by Solar Energy Generation:

```
+----+-----+----------------+
|Year|Month|Total_Solar_kW  |
+----+-----+----------------+
|2016|1    |38412.76483438449|
|NULL|NULL |NULL            |
+----+-----+----------------+
```

Energy Generated vs. Consumed Per Year:

```
+----+----------------+-----------------+
|Year|Total_Consumed_kW|Total_Generated_kW|
+----+----------------+-----------------+
|NULL|NULL            |NULL             |
|2016|432839.7404670082|38412.76483438449|
+----+----------------+-----------------+
```

Monthly Average Usage for Top Appliances:

```
[Stage 18:>                                                         (0 + 1) / 1]
+----+-----+-------------------+-------------------+-------------------+
|Year|Month|Avg_Dishwasher_kW  |Avg_Fridge_kW      |Avg_Microwave_kW   |
+----+-----+-------------------+-------------------+-------------------+
|NULL|NULL |NULL               |NULL               |NULL               |
|2016|1    |0.03136752497494595|0.06355641007374108|0.010982993427659423|
+----+-----+-------------------+-------------------+-------------------+
```

Task 3: Store Processed Data Back to S3

- To achieve this task, I have divided into 2 steps.
- First, I convert file into csv and store it in local storage.

```
output_path = "/home/ec2-user/procesed_data/"

# Save each DataFrame to local storage
# Save the transformed DataFrame to local storage
df_transformed.write.csv(output_path + "df_transformed.csv", header=True, mode="overwrite")
total_usage_by_month.write.csv(output_path + "total_usage_by_month.csv", header=True, mode="overwrite")
avg_temp_by_month.write.csv(output_path + "avg_temp_by_month.csv", header=True, mode="overwrite")
top_solar_months.write.csv(output_path + "top_solar_months.csv", header=True, mode="overwrite")
energy_comparison.write.csv(output_path + "energy_comparison.csv", header=True, mode="overwrite")
avg_appliance_usage.write.csv(output_path + "avg_appliance_usage.csv", header=True, mode="overwrite")
```

- Then Upload this from local storage to s3 bucket using boto3 package.

```
import boto3
import os

# Initialize S3 client
s3 = boto3.client("s3")

# S3 bucket name and folder
s3_bucket_name = "big-data-pipeline-rimysore"
s3_folder = "proccesed_data/"

# Function to upload files to S3
def upload_to_s3(local_folder, s3_bucket, s3_prefix):
    for root, dirs, files in os.walk(local_folder):
        for file in files:
            local_file_path = os.path.join(root, file)
            s3_file_path = s3_prefix + file
            print(f"Uploading {local_file_path} to s3://{s3_bucket}/{s3_file_path}")
            s3.upload_file(local_file_path, s3_bucket, s3_file_path)

# Upload the processed data to S3
upload_to_s3(output_path, s3_bucket_name, s3_folder)
```

Task 3 Output – Processed data stored to s3 bucket under processed_data/ folder.

```
Uploading /home/ec2-user/procesed_data/df_transformed.csv/part-00000-377f067d-495f-4880-8e5d-49b979dcdb6a-c000.csv to s3://big-data-pipeline-
rimysore/proccesed_data/part-00000-377f067d-495f-4880-8e5d-49b979dcdb6a-c000.csv
Uploading /home/ec2-user/procesed_data/df_transformed.csv/.part-00000-377f067d-495f-4880-8e5d-49b979dcdb6a-c000.csv.crc to s3://big-data-pipe
line-rimysore/proccesed_data/.part-00000-377f067d-495f-4880-8e5d-49b979dcdb6a-c000.csv.crc
Uploading /home/ec2-user/procesed_data/df_transformed.csv/_SUCCESS to s3://big-data-pipeline-rimysore/proccesed_data/_SUCCESS
Uploading /home/ec2-user/procesed_data/df_transformed.csv/._SUCCESS.crc to s3://big-data-pipeline-rimysore/proccesed_data/._SUCCESS.crc
Uploading /home/ec2-user/procesed_data/total_usage_by_month.csv/part-00000-f737bda6-3fe8-432d-959c-2c0834a141c6-c000.csv to s3://big-data-pip
eline-rimysore/proccesed_data/part-00000-f737bda6-3fe8-432d-959c-2c0834a141c6-c000.csv
Uploading /home/ec2-user/procesed_data/total_usage_by_month.csv/.part-00000-f737bda6-3fe8-432d-959c-2c0834a141c6-c000.csv.crc to s3://big-dat
a-pipeline-rimysore/proccesed_data/.part-00000-f737bda6-3fe8-432d-959c-2c0834a141c6-c000.csv.crc
Uploading /home/ec2-user/procesed_data/total_usage_by_month.csv/_SUCCESS to s3://big-data-pipeline-rimysore/proccesed_data/_SUCCESS
Uploading /home/ec2-user/procesed_data/total_usage_by_month.csv/._SUCCESS.crc to s3://big-data-pipeline-rimysore/proccesed_data/._SUCCESS.crc
Uploading /home/ec2-user/procesed_data/avg_temp_by_month.csv/part-00000-8149693c-0080-4915-ab12-76ea91494f55-c000.csv to s3://big-data-pipeli
ne-rimysore/proccesed_data/part-00000-8149693c-0080-4915-ab12-76ea91494f55-c000.csv
Uploading /home/ec2-user/procesed_data/avg_temp_by_month.csv/.part-00000-8149693c-0080-4915-ab12-76ea91494f55-c000.csv.crc to s3://big-data-p
ipeline-rimysore/proccesed_data/.part-00000-8149693c-0080-4915-ab12-76ea91494f55-c000.csv.crc
Uploading /home/ec2-user/procesed_data/avg_temp_by_month.csv/_SUCCESS to s3://big-data-pipeline-rimysore/proccesed_data/_SUCCESS
Uploading /home/ec2-user/procesed_data/avg_temp_by_month.csv/._SUCCESS.crc to s3://big-data-pipeline-rimysore/proccesed_data/._SUCCESS.crc
Uploading /home/ec2-user/procesed_data/top_solar_months.csv/part-00000-03df3206-7937-49fe-a81a-eb0704f431e6-c000.csv to s3://big-data-pipelin
e-rimysore/proccesed_data/part-00000-03df3206-7937-49fe-a81a-eb0704f431e6-c000.csv
Uploading /home/ec2-user/procesed_data/top_solar_months.csv/.part-00000-03df3206-7937-49fe-a81a-eb0704f431e6-c000.csv.crc to s3://big-data-pi
peline-rimysore/proccesed_data/.part-00000-03df3206-7937-49fe-a81a-eb0704f431e6-c000.csv.crc
Uploading /home/ec2-user/procesed_data/top_solar_months.csv/_SUCCESS to s3://big-data-pipeline-rimysore/proccesed_data/_SUCCESS
Uploading /home/ec2-user/procesed_data/top_solar_months.csv/._SUCCESS.crc to s3://big-data-pipeline-rimysore/proccesed_data/._SUCCESS.crc
Uploading /home/ec2-user/procesed_data/energy_comparison.csv/part-00000-397133a2-cbd9-42a7-8d85-9022ba2a0476-c000.csv to s3://big-data-pipeli
ne-rimysore/proccesed_data/part-00000-397133a2-cbd9-42a7-8d85-9022ba2a0476-c000.csv
Uploading /home/ec2-user/procesed_data/energy_comparison.csv/.part-00000-397133a2-cbd9-42a7-8d85-9022ba2a0476-c000.csv.crc to s3://big-data-p
ipeline-rimysore/proccesed_data/.part-00000-397133a2-cbd9-42a7-8d85-9022ba2a0476-c000.csv.crc
Uploading /home/ec2-user/procesed_data/energy_comparison.csv/_SUCCESS to s3://big-data-pipeline-rimysore/proccesed_data/_SUCCESS
Uploading /home/ec2-user/procesed_data/energy_comparison.csv/._SUCCESS.crc to s3://big-data-pipeline-rimysore/proccesed_data/._SUCCESS.crc
Uploading /home/ec2-user/procesed_data/avg_appliance_usage.csv/part-00000-6c3e0873-d40b-4d66-aca4-aea7f470def7-c000.csv to s3://big-data-pipe
line-rimysore/proccesed_data/part-00000-6c3e0873-d40b-4d66-aca4-aea7f470def7-c000.csv
Uploading /home/ec2-user/procesed_data/avg_appliance_usage.csv/.part-00000-6c3e0873-d40b-4d66-aca4-aea7f470def7-c000.csv.crc to s3://big-data
-pipeline-rimysore/proccesed_data/.part-00000-6c3e0873-d40b-4d66-aca4-aea7f470def7-c000.csv.crc
Uploading /home/ec2-user/procesed_data/avg_appliance_usage.csv/_SUCCESS to s3://big-data-pipeline-rimysore/proccesed_data/_SUCCESS
Uploading /home/ec2-user/procesed_data/avg_appliance_usage.csv/._SUCCESS.crc to s3://big-data-pipeline-rimysore/proccesed_data/._SUCCESS.crc
```

Name	Type	Last modified	Size	Storage class
HomeC.csv	csv	December 12, 2024, 09:54:15 (UTC-05:00)	11.6 MB	Standard
proccesed_data/	Folder	-	-	-

Task 4: Data Analysis Using Spark SQL

- I have executed vital 5 queries that derives key insights using SQL in pyspark environment.
- Avg temp month over month analysis, Identify average seasonal trends, Identify average seasonal trends for solar power, Identify average seasonal on generation, Identify average seasonal trends for fridge are the 5 insights derived.
- Read the csv file back to local pyspark and register dataframe as a temporary view to execute SQL queries.

```python
# Read the CSV file back into a DataFrame
total_usage_by_month_df = spark.read.csv(output_path + "total_usage_by_month.csv", header=True, inferSchema=True)
avg_temp_by_month = spark.read.csv(output_path + "avg_temp_by_month.csv", header=True, inferSchema=True)
top_solar_months = spark.read.csv(output_path + "top_solar_months.csv", header=True, inferSchema=True)
energy_comparison = spark.read.csv(output_path + "energy_comparison.csv", header=True, inferSchema=True)
avg_appliance_usage = spark.read.csv(output_path + "avg_appliance_usage.csv", header=True, inferSchema=True)

# Register the DataFrame as a temporary view
total_usage_by_month_df.createOrReplaceTempView("total_usage_by_month")
avg_temp_by_month.createOrReplaceTempView("avg_temp_by_month")
top_solar_months.createOrReplaceTempView("top_solar_months")
energy_comparison.createOrReplaceTempView("energy_comparison")
avg_appliance_usage.createOrReplaceTempView("avg_appliance_usage")

# Now, run your SQL query
# Query 1 - Avg temp month over month analysis
query_1 = """
SELECT Year, Month, SUM(Avg_Temperature) AS TotalUsageTemp
FROM avg_temp_by_month
GROUP BY Year, Month
ORDER BY TotalUsageTemp DESC
LIMIT 10
"""

result_1 = spark.sql(query_1)
result_1.show()

#query2 - Identify average seasonal trends
query_2 = """
SELECT Month, SUM(Total_Usage_kW) AS TotalUsage
FROM total_usage_by_month
GROUP BY Month
ORDER BY Month
"""
result_2 = spark.sql(query_2)
result_2.show()

#query3 - Identify average seasonal trends for solar power
query_3 = """
SELECT Month, SUM(Total_Solar_kW) AS TotalUsageSolar
FROM top_solar_months
GROUP BY Month
ORDER BY Month
"""
result_3 = spark.sql(query_3)
result_3.show()
```

```python
#query4 - Identify average seasonal on generation
query_4 = """
SELECT Year, SUM(Total_Generated_kW) AS TotalGenerated
FROM energy_comparison
GROUP BY Year
ORDER BY Year
"""
result_4 = spark.sql(query_4)
result_4.show()

#query5 - Identify average seasonal trends for fridge
query_5 = """
SELECT Month, SUM(Avg_Fridge_kW) AS TotalFridgeUsage
FROM avg_appliance_usage
GROUP BY Month
ORDER BY Month
"""
result_5 = spark.sql(query_5)
result_5.show()
```

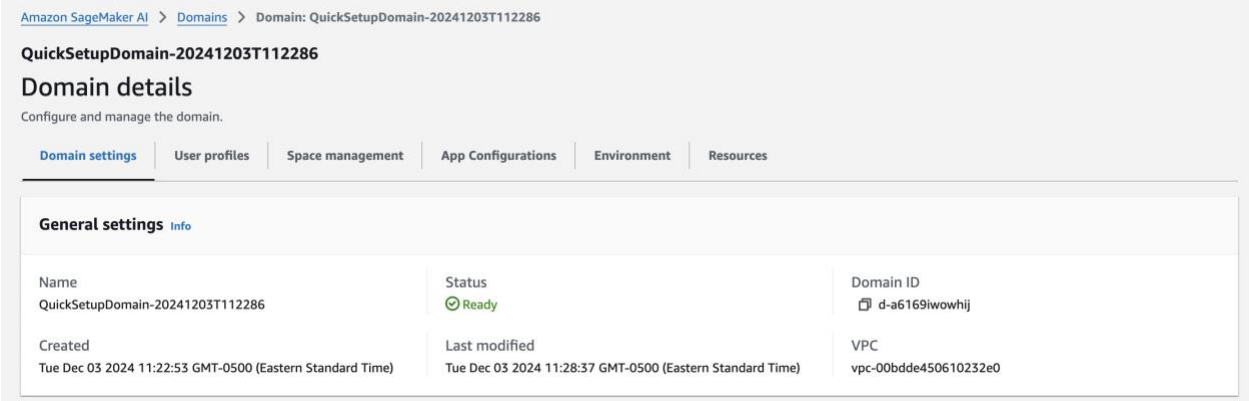
Task4 Ouput :

```
+----+-----+-----------------+
|Year|Month|   TotalUsageTemp|
+----+-----+-----------------+
|2016|    1|50.74193461135845|
|NULL| NULL|             NULL|
+----+-----+-----------------+

+-----+-----------------+
|Month|       TotalUsage|
+-----+-----------------+
| NULL|             NULL|
|    1|432839.7404670082|
+-----+-----------------+

+-----+-----------------+
|Month|  TotalUsageSolar|
+-----+-----------------+
| NULL|             NULL|
|    1|38412.76483438449|
+-----+-----------------+

+----+-----------------+
|Year|   TotalGenerated|
+----+-----------------+
|NULL|             NULL|
|2016|38412.76483438449|
+----+-----------------+

+-----+-------------------+
|Month|   TotalFridgeUsage|
+-----+-------------------+
| NULL|               NULL|
|    1|0.06355641007374108|
+-----+-------------------+
```

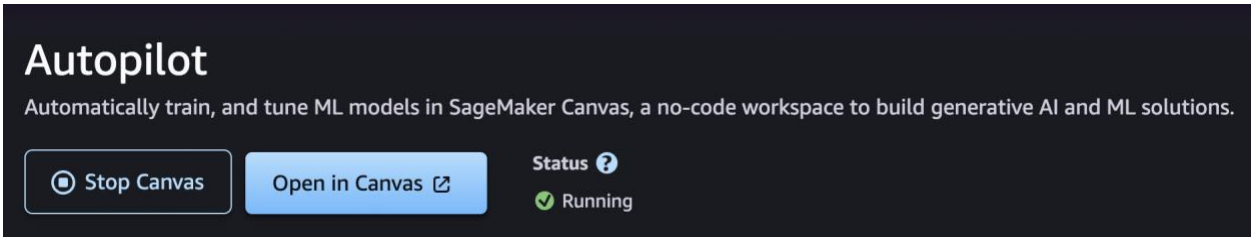
Task 5: Machine Learning with AWS SageMaker Autopilot

AWS Sagemaker is an integrated tool to build and deploy a Machine learning model with ease. Basic concept is to store some dataset into sagemaker polatform (Studio) and choose a target variable on which we build our model around.

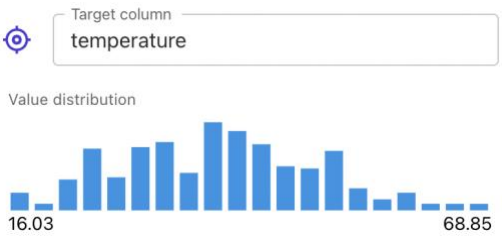
1. I have downloaded my processed data from S3 bucket and created a domain in sagemaker to start the studio.



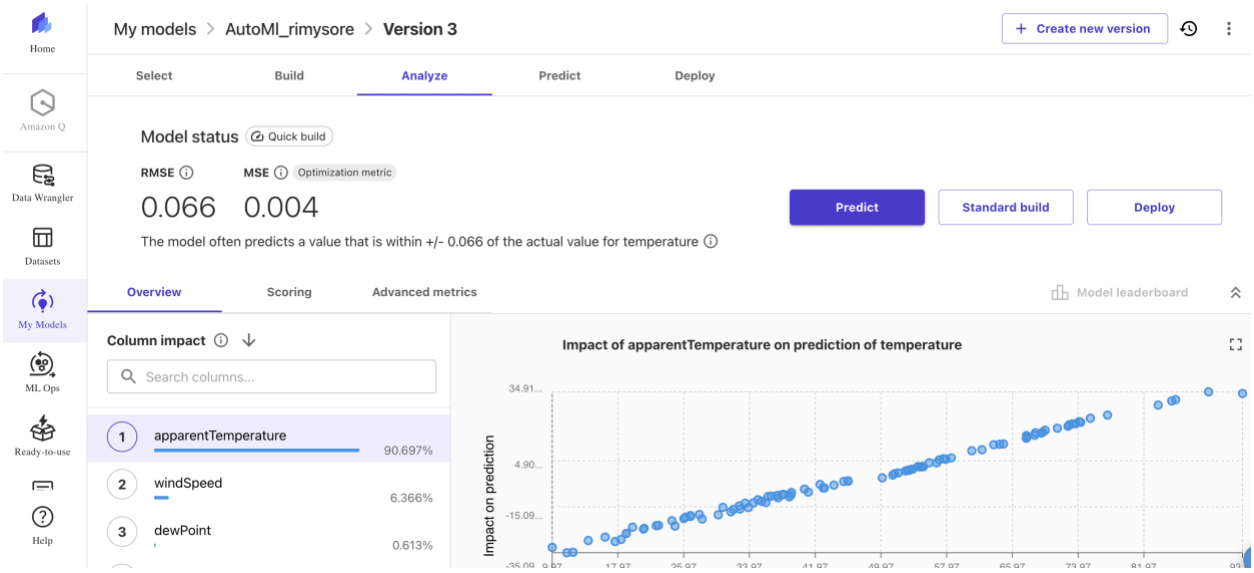
2. In Studio mode , we start canvas server to start running sagemaker.



3. I have selected Temperature parameter as **Target value.**



4. Since my dataset consist only of numeric values I have used numeric prediction for configuration of model. So my output values will be all numerics.
5. Sagemaker Analyze dashboard



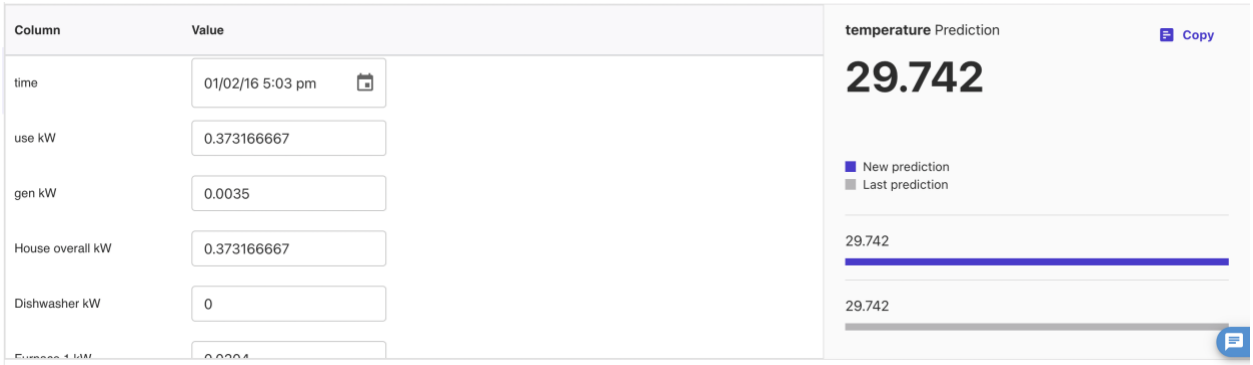
Above figure depicts the summary of machine learning model built using SageMaker's AutoML platform where target variable is Temperature for my processed dataset HomeC.csv. So this dashboard gives all insights like :

- Root Mean Square Error(RMSE) = 0.066 (Lower RMSE values suggest better predictive accuracy)
- Mean Squared Error (MSE) = 0.004 (low value reflects a well-fitted model)
- Model Status = Quick Build

Under Column Impact Analysis we can see that **apparentTemperature** is the key driver for predicting actual temperature. The upward trend in the plot

indicates a positive correlation: as apparentTemperature increases, the predicted temperature also increases.

Prediction:



Above figure represents the prediction part in aws sagemaker. Where it predicts our target value **temperature.** It illustrates the application of sophisticated predictive analytics for temperature forecasting and energy management through a machine learning-based energy prediction interface. Time, use kW, gen kW, house overall kW, and other appliance-specific power usage metrics are among the energy-related parameters that the model accepts as inputs. The model forecasts the ambient temperature (29.742) based on these parameters, which is crucial for resource management and energy optimization.

AWS SageMaker, a cloud-based machine learning service, might be used to create the system's expected real-time inference endpoint. For applications that demand quick decisions, SageMaker offers a smooth framework for training, deploying, and testing models, enabling low-latency predictions. To ensure model accuracy and consistency over time, users can test and compare

forecasts using the dashboard interface (e.g., "New prediction" vs. "Last prediction").

Ethical Issues in Task5 :

1. **Bias in Training data**:

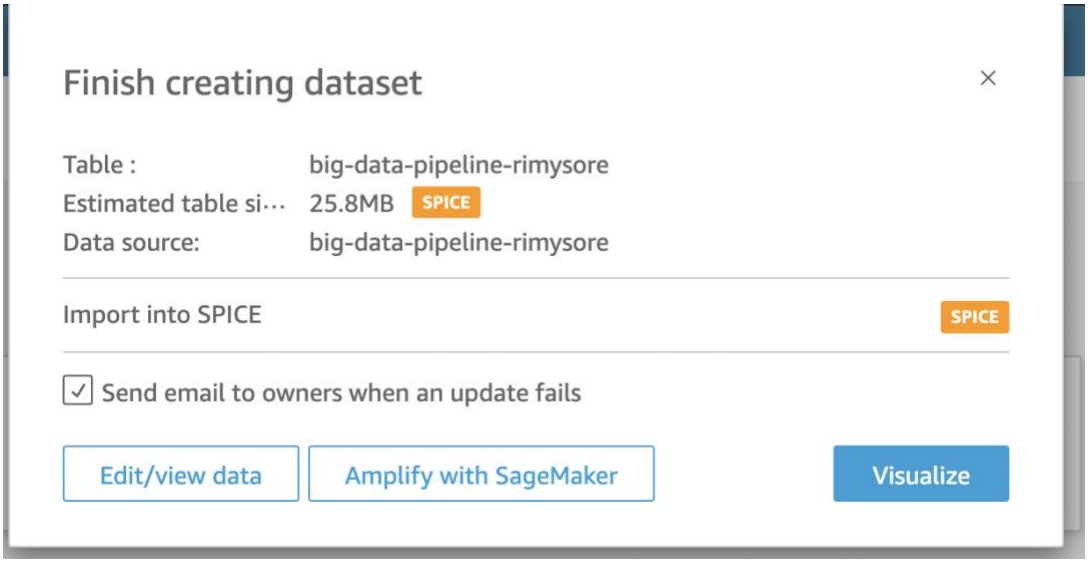
   As evidenced by the model's strong dependence on apparentTemperature (90.697%), predictions are largely dependent on this one characteristic. The low influence of secondary features such as windSpeed (6.366%) and dewPoint (0.613%) suggests that the feature space is not very diverse. If one prominent feature (apparentTemperature) is overused and does not generalize well across situations (e.g., geographic locations or seasons), it may result in skewed forecasts. The model might not function fairly in these circumstances if the training data disproportionately depicts particular weather patterns, geographical areas, or seasons.

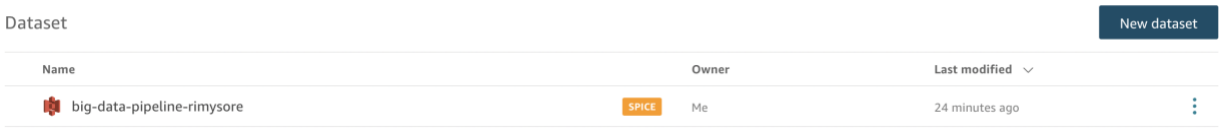
2. **Privacy Concerns:**

   The model's use of environmental data, as shown by the feature list (apparentTemperature, windSpeed, and dewPoint), may indirectly link to sensitive information if the dataset contains geographical or Internet of Things data. Re-identifying individuals or locations could be one of the privacy dangers if the data include geolocation or information collected from the Internet of Things. Forecasts may inadvertently disclose sensitive or private environmental patterns associated with certain regions.

4. Visualization

- Utilized help of quicksight tool powered by AWS.
- Connected S3 bucket pipeline processed folder , where we have processed file to visualize .





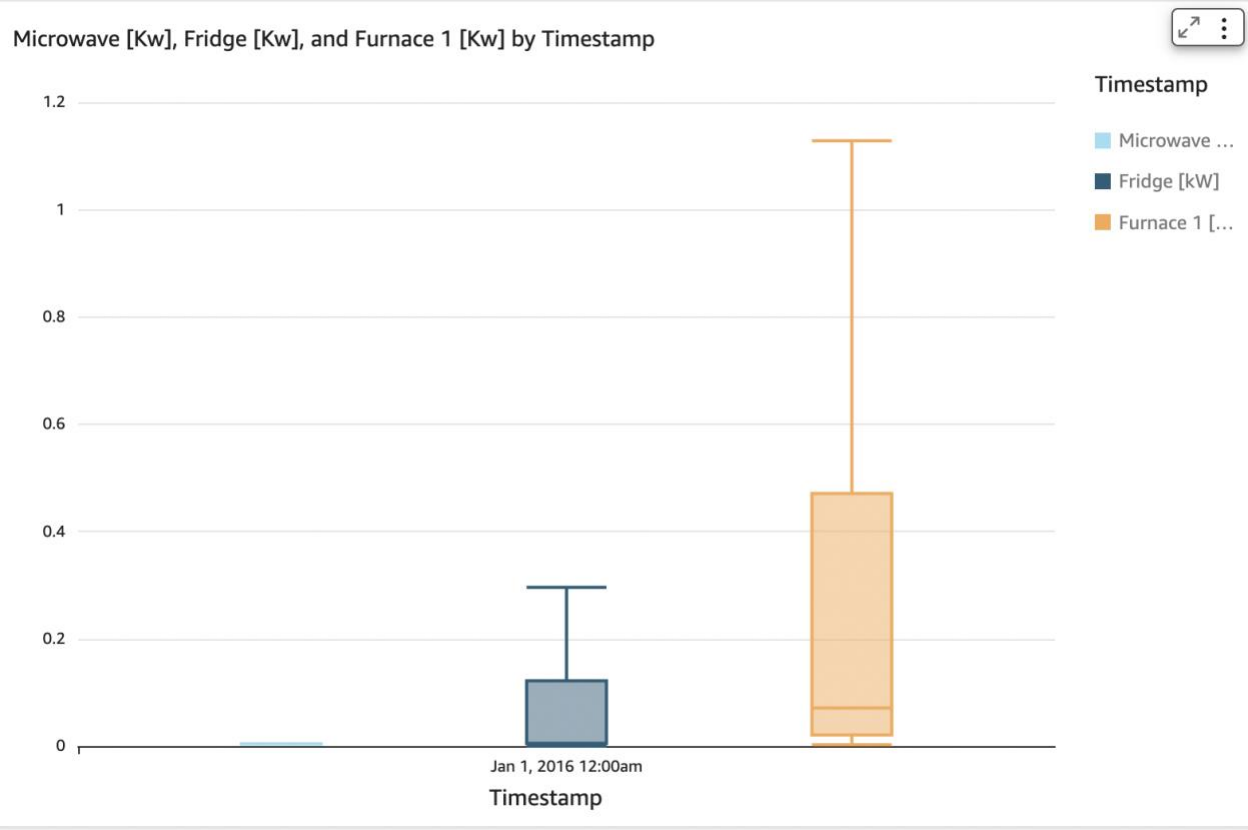
- For my dataset I have considered providing a comprehensive view of the relationship between environmental conditions (temperature, pressure) and appliance power consumption in the monitored system.

Visualizations:

Graph 1: Timestamp Analysis

A line graph showing the relationship between three variables:

- Microwave power consumption [kW]
- Fridge power consumption [kW]
- Furnace 1 power consumption [kW]
  The graph tracks these measurements over time, showing their relative power usage patterns.

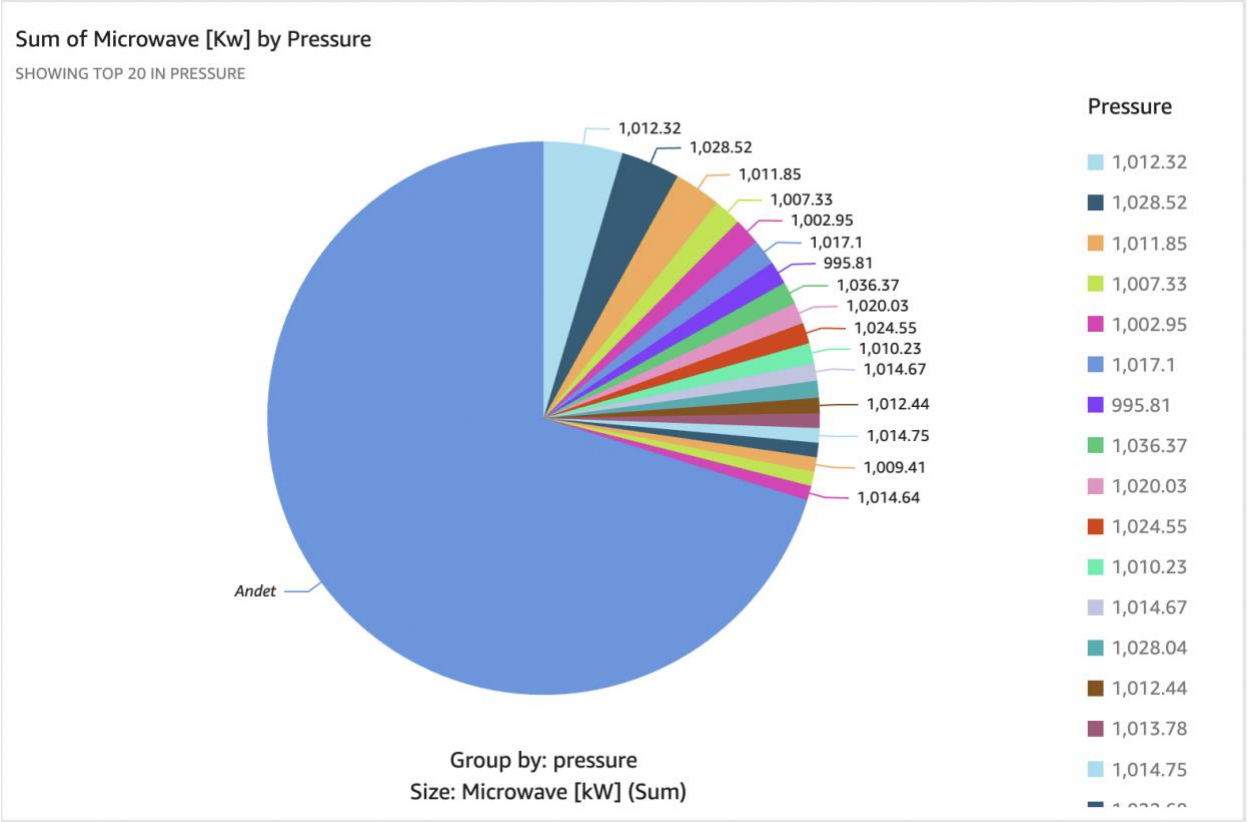


Graph 2: Pressure Distribution

A Pie chart displaying:

- Top 20 pressure measurements
- Data grouped by pressure values
- Size metric based on Microwave power consumption [kW]

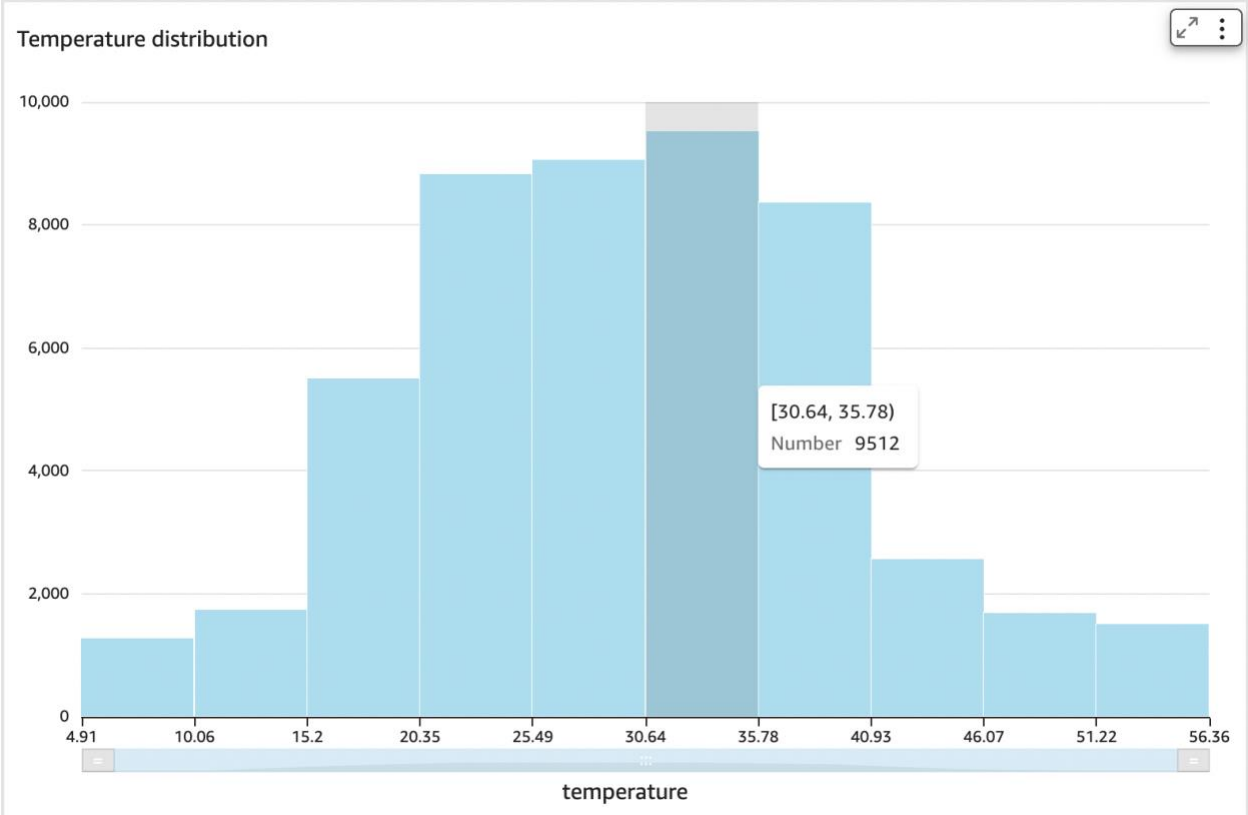
  The visualization helps identify the most common pressure ranges and their correlation with microwave usage.



Graph 3: Temperature Distribution

Shows the distribution of temperature readings with:

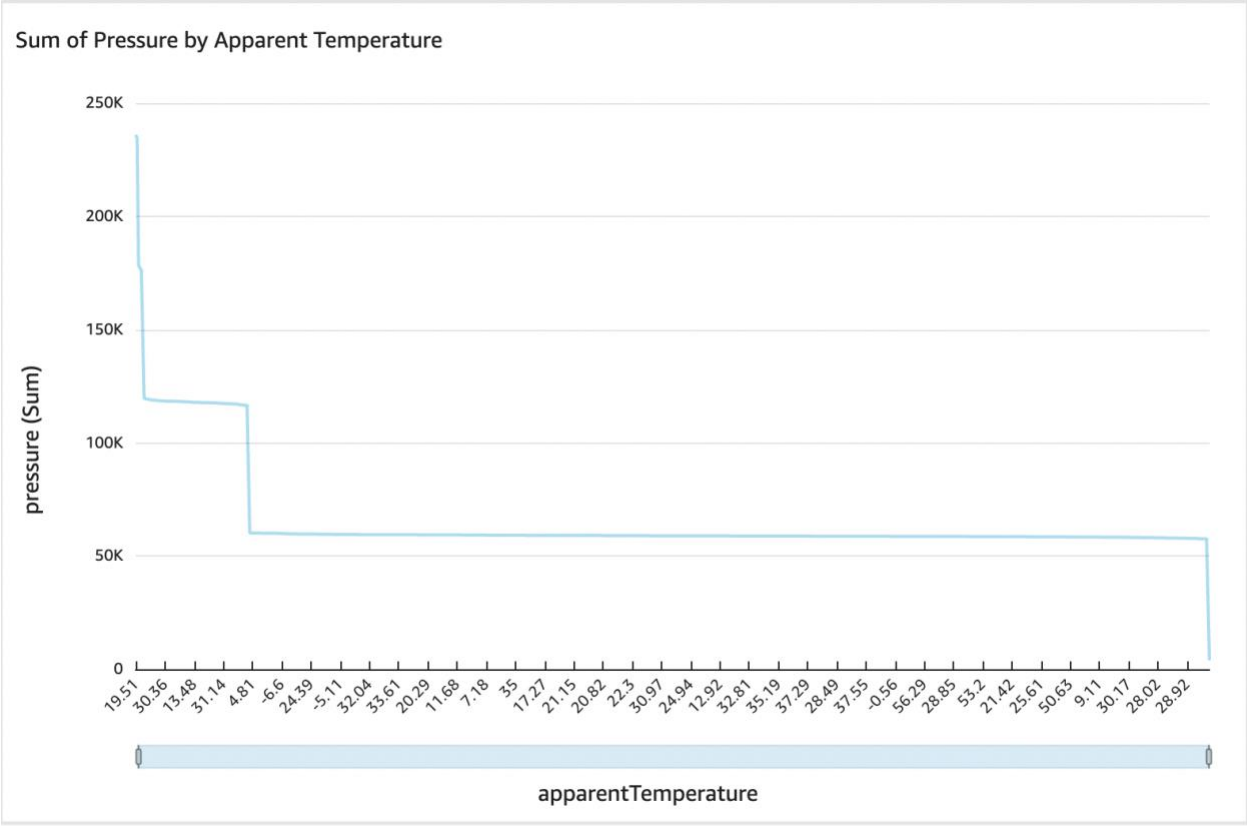
- Total sample size of 50K measurements
- Temperature range from -7.74°C to 33.88°C
- Highest recorded temperature at 33.88°C
- Two negative temperature readings (-7.74°C and -1.04°C)



Graph 4: Pressure vs Apparent Temperature

A gauge meter visualization showing:

- Relationship between pressure and apparent temperature
- Maximum scale value of 1,175,479.88
- Current reading at 11,147.34%
- Indicator position at 10.54



5.Bonus Task: Automation of the Pipeline

By executing code in response to particular events, AWS Lambda, a serverless computing service, helps developers automate workflows and processes without having to worry about maintaining underlying infrastructure. AWS Lambda's event-driven architecture allows it to be easily linked with other AWS services, including S3, DynamoDB, and CloudWatch, to initiate workflows in response to scheduled events, file uploads, and database updates. For example, by running code when fresh data is uploaded to an S3 bucket or at predetermined intervals using CloudWatch Events, Lambda can automate processes like data ingestion, transformation, and loading in a data pipeline. In

order to provide real-time alerts on pipeline success or failure, Lambda also facilitates interaction with email or AWS Simple Notification Service (SNS).

- Executed automation steps in AWS to achieve Bonus part.
- Tools used – AWS Lambda, SNS, S3 etc.

1.**Develop an Automation Script:**

lambda_function.py

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

# Initialize AWS services
s3_client = boto3.client('s3')
sns_client = boto3.client('sns')

def lambda_handler(event, context):
    try:
        # Loop through each record in the S3 event
        for record in event['Records']:
            # Dynamically 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}")

            # Download the raw file from S3 to the Lambda temporary directory
            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)

            df['Timestamp'] = pd.to_datetime(df['time'], unit='s')
            # Add "Year" and "Month" columns
```

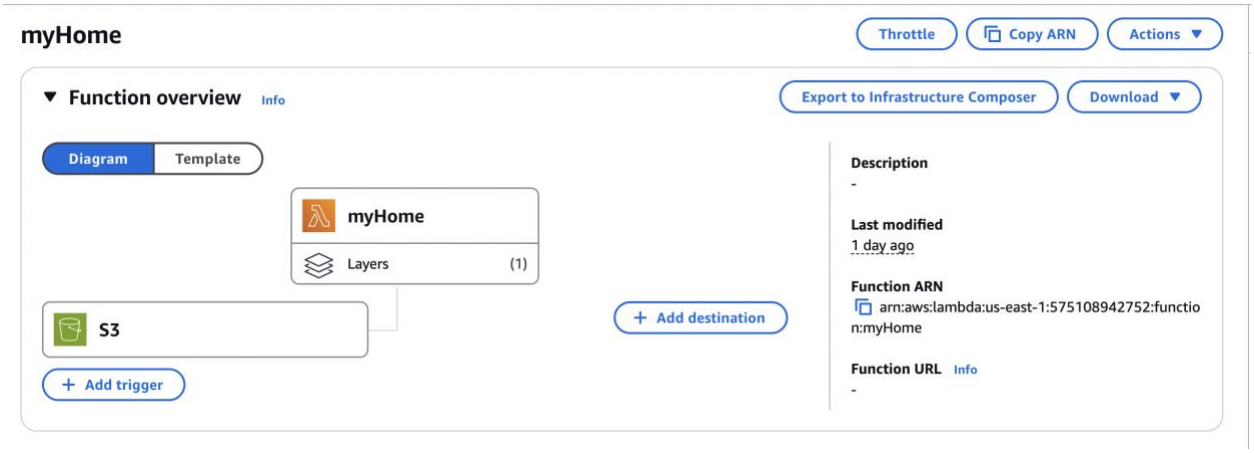
```
        df['Year'] = df['Timestamp'].dt.year
        df['Month'] = df['Timestamp'].dt.month

        # Save the processed file locally
        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-1:575108942752:homeCMessage",
        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-1:575108942752:homeCMessage",
        Message=f"Error processing file {file_key}: {str(e)}",
        Subject="Pipeline Failure"
    )
    raise e
```

Explanation:

Above is the automation script in python language for data retrieval, processing, and storage steps. Starting with boto3 which is aws sdk for python execution. Initialize the aws service to fetch data from s3 bucket dynamically while configured sns client listens to triggers.Then we download this raw file to local storage in lambda. Inside lambda_handler() function, we provide the bucket details and records to be recorded. This is passed as a parameter to function using ARN attribute, which is generated when we setup

simple notification service for our project. Timestamp is converted into year and month to fetch proper data inside function itself.

2. **Set Up Scheduling Tools**:



Utilized AWS Lambda assist to achieve scheduling tasks. As in above image, I have setup lambda triggers and named it as **myHome**. This is responsible for all scheduled jobs. The Lambda function is triggered by **an S3 bucket.** Whenever a new file is uploaded ito s3 bucket , it triggers lambda automation. This event-driven mechanism eliminates the need for manual intervention to initiate the pipeline. This setup takes ARN value to confirm parameter for trigger and to fetch data. Any logs generated will be stored in Cloudwatch logs.

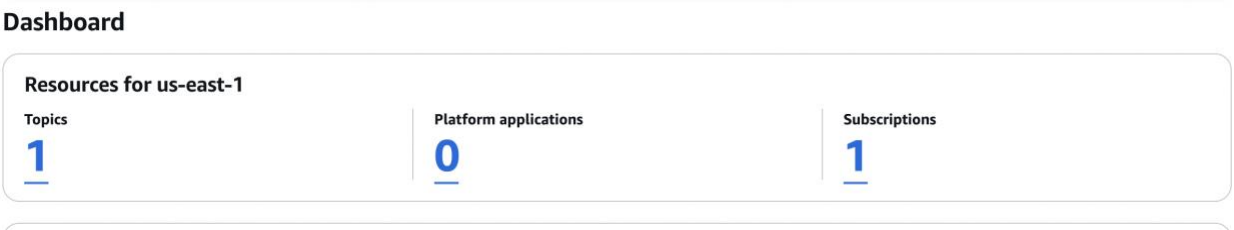
Template:

```
# This AWS SAM template has been generated from your function's configuration. If
# your function has one or more triggers, note that the AWS resources associated
# with these triggers aren't fully specified in this template and include
# placeholder values. Open this template in AWS Infrastructure Composer or your
# favorite IDE and modify it to specify a serverless application with other AWS
# resources.
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Application Model template describing your function.
Resources:
  myHome:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 303
      Handler: lambda_function.lambda_handler
      Runtime: python3.9
      Architectures:
        - x86_64
      EphemeralStorage:
        Size: 512
      EventInvokeConfig:
        MaximumEventAgeInSeconds: 21600
        MaximumRetryAttempts: 2
      Layers:
        - arn:aws:lambda:us-east-1:336392948345:layer:AWSSDKPandas-Python39:26
      PackageType: Zip
      Policies:
        - Statement:
            - Effect: Allow
              Action:
                - logs:CreateLogGroup
              Resource: arn:aws:logs:us-east-1:575108942752:*
            - Effect: Allow
              Action:
                - logs:CreateLogStream
                - logs:PutLogEvents
              Resource:
                - >-
                  arn:aws:logs:us-east-1:575108942752:log-group:/aws/lambda/myHome:*
            - Sid: SNSFullAccess
              Effect: Allow
              Action:
                - sns:*
              Resource: '*'
            - Sid: SMSAccessViaSNS
              Effect: Allow
              Action:
                - sms-voice:DescribeVerifiedDestinationNumbers
                - sms-voice:CreateVerifiedDestinationNumber
                - sms-voice:SendDestinationNumberVerificationCode
                - sms-voice:SendTextMessage
                - sms-voice:DeleteVerifiedDestinationNumber
                - sms-voice:VerifyDestinationNumber
                - sms-voice:DescribeAccountAttributes
                - sms-voice:DescribeSpendLimits
                - sms-voice:DescribePhoneNumbers
                - sms-voice:SetTextMessageSpendLimitOverride
                - sms-voice:DescribeOptedOutNumbers
                - sms-voice:DeleteOptedOutNumber
              Resource: '*'
              Condition:
                StringEquals:
                  aws:CalledViaLast: sns.amazonaws.com
            - Effect: Allow
              Action:
                - s3:*
                - s3-object-lambda:*
              Resource: '*'
      RecursiveLoop: Terminate
```

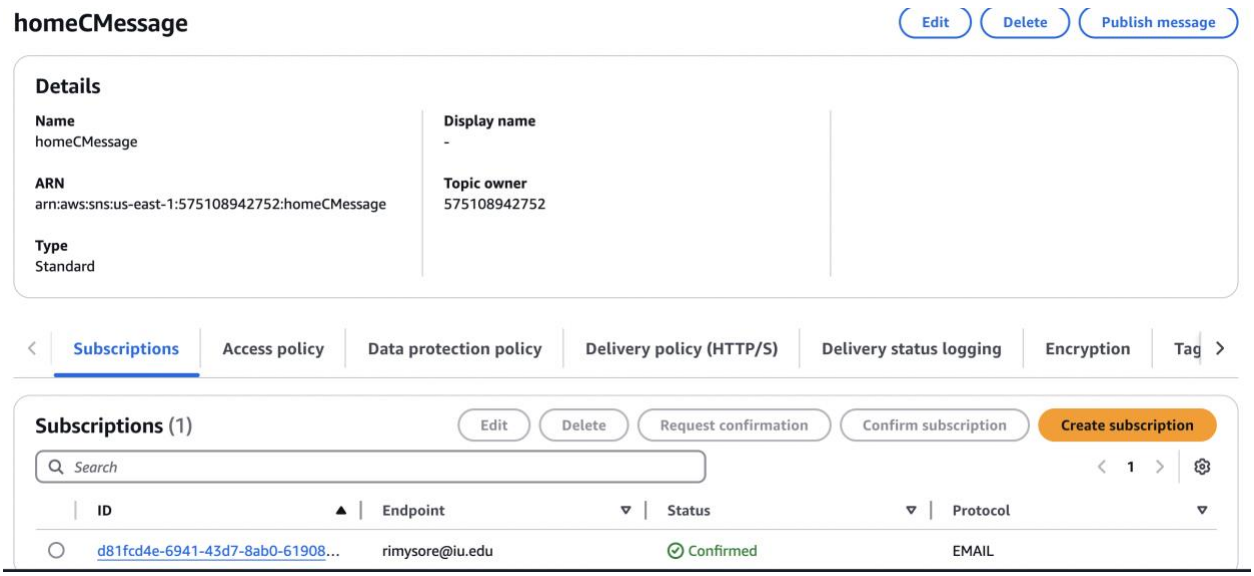
```
    SnapStart:
      ApplyOn: None
    Events:
      BucketEvent1:
        Type: S3
        Properties:
          Bucket:
            Ref: Bucket1
          Events:
            - s3:ObjectCreated:*
    RuntimeManagementConfig:
      UpdateRuntimeOn: Auto
Bucket1:
  Type: AWS::S3::Bucket
  Properties:
    VersioningConfiguration:
      Status: Enabled
    BucketEncryption:
      ServerSideEncryptionConfiguration:
        - ServerSideEncryptionByDefault:
            SSEAlgorithm: AES256
BucketPolicy1:
  Type: AWS::S3::BucketPolicy
  Properties:
    Bucket: Bucket1
    PolicyDocument:
      Statement:
        - Action: s3:*
          Effect: Deny
          Principal: '*'
          Resource:
            - arn:aws:s3:::Bucket1/*
            - arn:aws:s3:::Bucket1
          Condition:
            Bool:
              aws:SecureTransport: false
```

3.Implement Logging and Notifications:

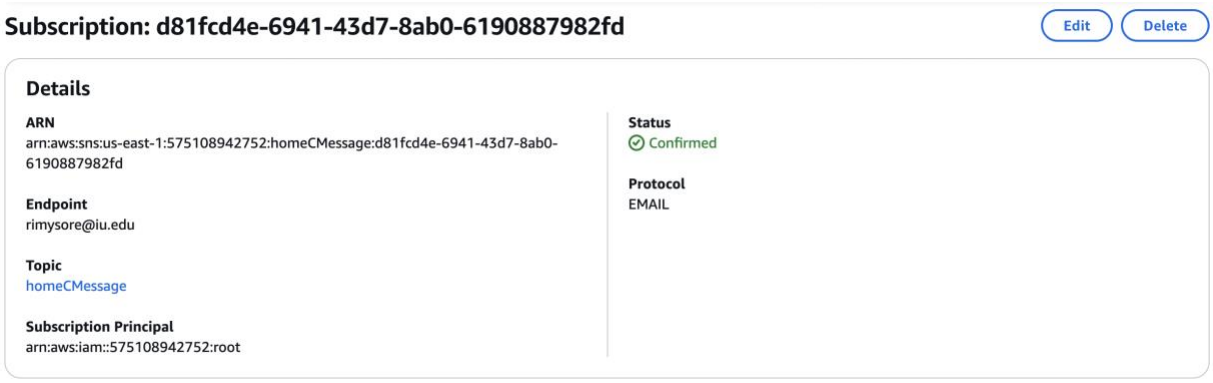
- Created a topic in sns aws to enable notifications for event trigger.



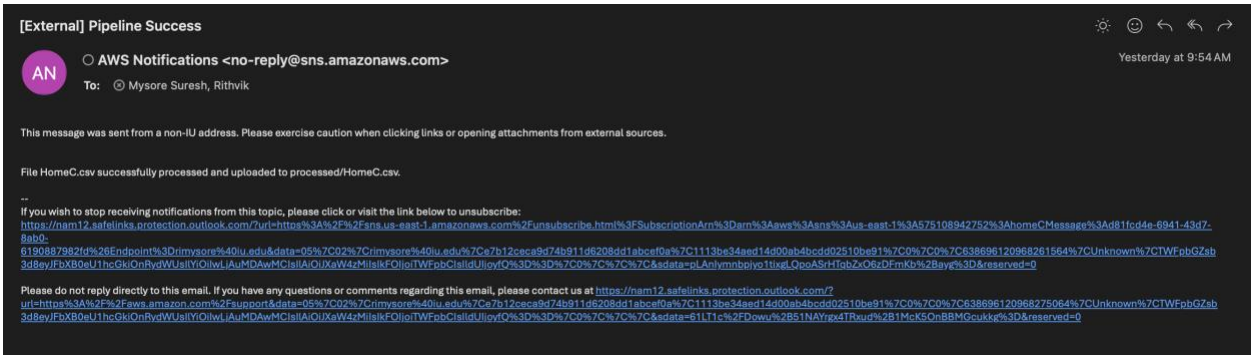
- Function name and subscription details



- Created a new subscription and added my IU mail id to get notified whenever a event is triggered in lambda.

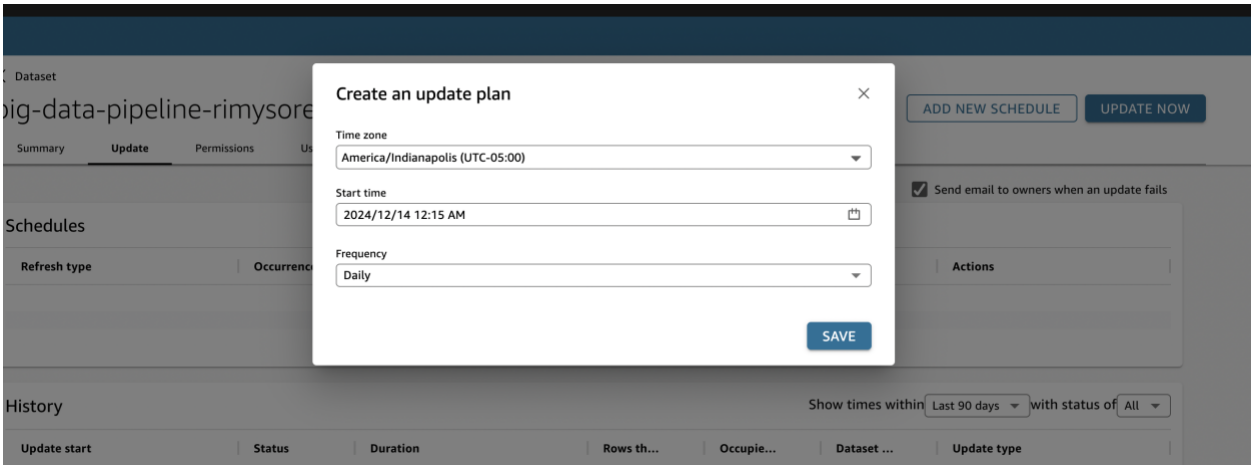


- Got a Pipeline success email when a file HomeC.csv was added successfully to the s3 bucket.

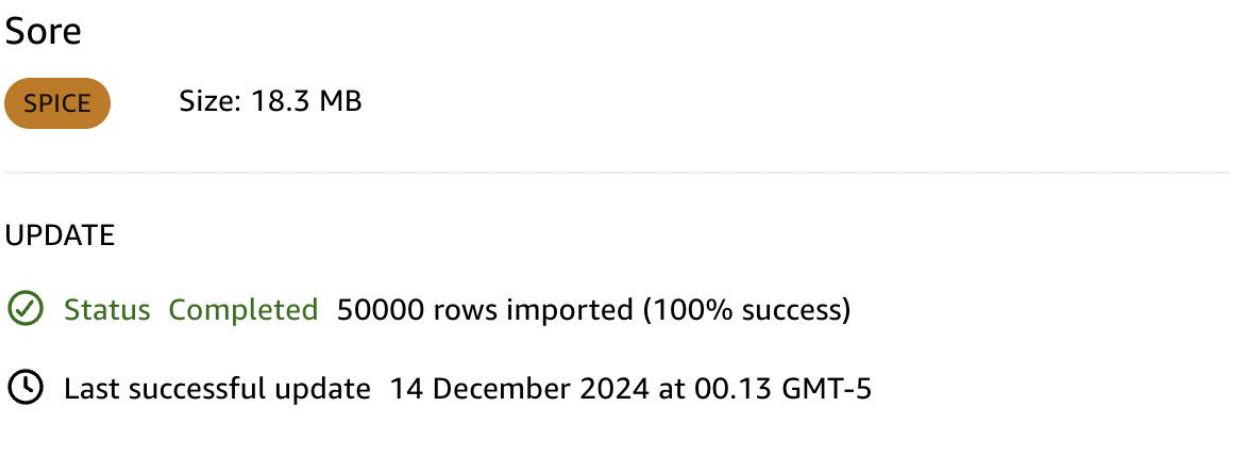


4.Automate Dashboard Updates

- Scheduling recurring data refreshes for datasets loaded into SPICE is possible using AWS QuickSight.
- Depending on the needs of the data flow and the objectives of real-time analysis, we can set up refresh intervals that range from hourly to daily.
- QuickSight is connected to processed data stored in AWS S3 in processed folder, allowing for dynamic dashboard changes upon the upload of fresh data.

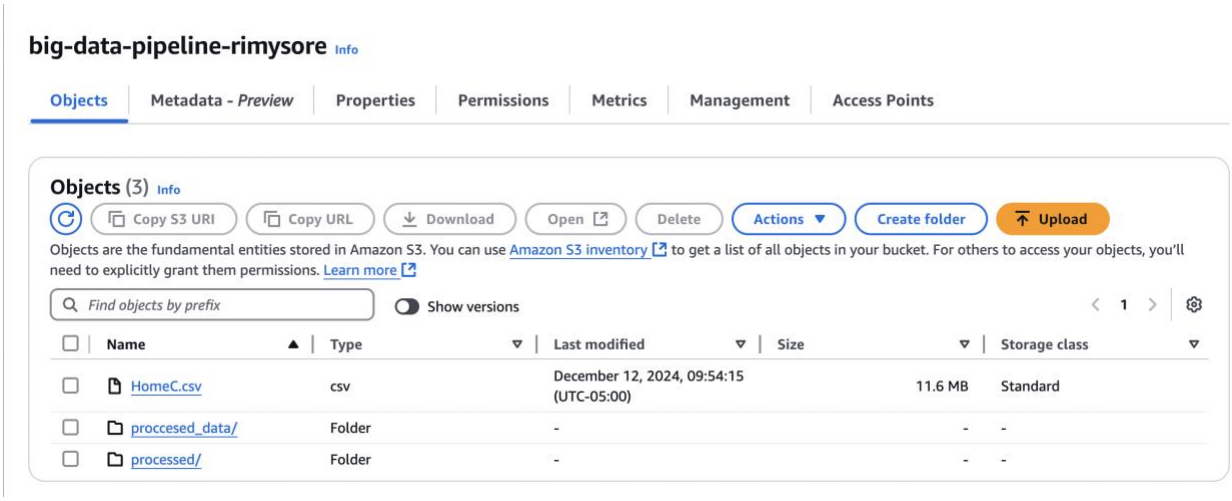


- Below screenshot depicts data being refreshed periodically.

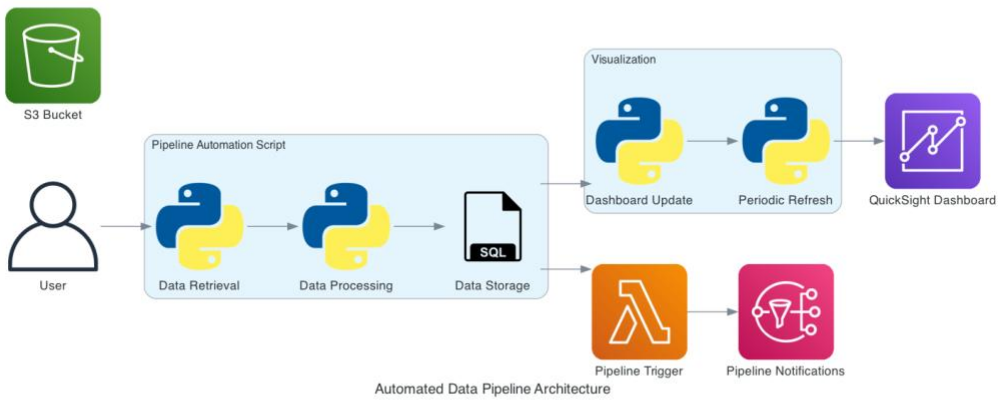


Output after automation:

So once I complete this automation setup(lambda function, job schedule, sns) and upload my HomeC.csv file to s3 bucket it will perform automation tasks to successfully upload data , retrieve data, process the data(add year and month) and then store it in a folder named processed.



Bonus: Architecture of entire automation



- Above diagram gives architecture of whole pipeline used in our project.
- S3 bucket is connected to instance and data is modified inside bucket.
- All operations like data retrieval, data processing, and storage happens in successive tasks.
- Lambda function triggers the main automation concepts. When a new file is uploaded to s3 bucket, it processes it and stores in new folder.
- Simple notification service triggers notification to responsible user about all these operations.
- Whether pipeline is success or failure.

- For Visualization part, quicksight is configured for aut refresh on timely fashion and it updates database whenever a new data is added to s3 bucket .

Conclusion

To sum up, this project shows how to successfully use the robust tools and services provided by AWS and PySpark to construct a whole big data pipeline, from data intake and processing to analysis, visualization, and automation. I have developed a thorough understanding of big data analytics and cloud-based distributed systems by completing all of the tasks listed, including deploying scalable storage in S3, using PySpark for distributed data processing, using AWS SageMaker for machine learning, and using AWS QuickSight to create dynamic dashboards. The workflow is smooth and reliable since the pipeline is automated with AWS Lambda and scheduling tools, which guarantees effectiveness and real-time data processing. In addition to showcasing my technological expertise, this project illustrates how crucial it is to address ethical issues like data privacy and bias in machine learning, encouraging the appropriate and useful use of data engineering techniques to address real-world problems.