INDUSTRIAL EQUIPMENT MONITORING AND PREDICTIVE MAINTENANCE

1. ABSTRACT

This project demonstrates an end-to-end industrial equipment monitoring and predictive maintenance system using IoT and cloud technologies. The industrial sensor data was generated using Python telemetry and sent to AWS IoT Core, then stored in AWS Timestream. Data was transferred to Azure Blob Storage via AWS Lambda for further processing. Power BI Desktop was connected to Azure Blob to visualize raw and processed data. Predictive maintenance was implemented using Azure Machine Learning, with results saved as CSV files. Finally, Azure Blob data was linked to Azure Log Analytics and monitored using Grafana dashboards. The system provides real-time insights into equipment health, enabling proactive maintenance and reducing downtime.

2. INTRODUCTION

In industrial settings, equipment such as motors, pumps, compressors, and turbines are critical to production and operational efficiency. Any unplanned downtime or failure of such machinery can lead to significant operational losses, reduced productivity, and increased maintenance costs.

With the rise of Industry 4.0 and the adoption of IoT (Internet of Things), a paradigm shift has occurred from reactive to predictive maintenance. By equipping machinery with sensors that monitor temperature, vibration, pressure, and other performance parameters in real time, it becomes possible to collect valuable operational data. This data, when analysed using machine learning techniques, can help predict equipment failures before they occur. Predictive maintenance not only reduces downtime but also optimizes maintenance scheduling, enhances asset life, and improves safety.

This project aims to improve the industrial maintenance by replacing manual checks with real-time monitoring and predictive insights. It uses IoT, cloud platforms, and machine learning to reduce downtime and costs. The end-to-end flow includes sensor data simulation, cloud routing, predictive analysis, and visualization using Power BI and Grafana.

The primary goal of this project is to design and implement a predictive maintenance system that leverages IoT and cloud-based platforms to monitor industrial equipment in real time and provide actionable insights through intelligent visualizations. The specific objectives of the project include:

- Routing simulated sensor data to AWS IoT Core for initial ingestion and device management.
- Transferring the data from AWS to Azure Blob Storage using AWS Lambda, allowing interoperability between cloud platforms.
- Connecting Power BI to Azure Blob for visualizing both raw and processed sensor data in a user-friendly dashboard format.
- Utilizing Azure Machine Learning to build a predictive maintenance model that forecasts the probability of equipment failure.
- Sending data from Azure Blob to Azure Log Analytics to monitor key metrics, and visualizing them in real-time using Grafana dashboards.
- Generating alerts and visual indicators in Grafana to notify users about high-risk equipment based on failure probability.

3. TECHNOLOGIES USED

- **IoT Devices and Sensors**: The project simulates data from various industrial-grade sensors including temperature, vibration, pressure, and distance sensors. These virtual devices replicate real-world equipment behaviour and environmental conditions typically found in manufacturing or heavy machinery settings. The data generated by these sensors serves as the foundational input for downstream analytics, allowing the system to perform performance monitoring, anomaly detection, and predictive maintenance analysis. By simulating different operational states and potential failure modes, the system provides a realistic framework for testing and validating the analytics pipeline.
- **Middleware:** For efficient and reliable message delivery, the system employs the MQTT (Message Queuing Telemetry Transport) protocol—a lightweight, publish-

subscribe network protocol designed for constrained devices and low-bandwidth networks. MQTT ensures minimal overhead and near real-time transmission of telemetry data between the simulated IoT sensors and the cloud services. This layer acts as the communication backbone of the system, ensuring seamless integration and data flow across components.

- **Cloud Infrastructure**: The cloud infrastructure supporting this project is designed using a hybrid architecture that leverages the strengths of both AWS and Azure platforms. Initially, sensor data transmitted via the MQTT protocol is ingested through AWS IoT Core, which acts as the entry point for managing and processing IoT device data. This data is stored in AWS Timestream, a purpose-built time-series database optimized for the efficient storage and querying of chronological sensor data. To enable further processing and advanced analytics on a different cloud platform, a custom AWS Lambda function is configured to automatically transfer the time-series data from AWS Timestream into Azure Blob Storage. This cross-cloud integration facilitates a seamless flow of data between AWS and Azure services. Once the data resides in Azure, it is fed into Azure Machine Learning, where it is used to train and evaluate predictive models, perform anomaly detection, and generate maintenance forecasts. This integrated pipeline allows for scalable, real-time processing while maintaining flexibility and interoperability across cloud ecosystems.
- **Analytics and Visualization Tools:** To extract actionable insights from the collected sensor data, the system incorporates two powerful visualization platforms:

- **Power BI** is used to create interactive dashboards that present both raw and modeled sensor data. These dashboards provide intuitive visualizations, trend analysis, and summary statistics for stakeholders and operators.
- **Grafana**, integrated with Azure Monitor and Log Analytics, delivers real-time monitoring dashboards. These are particularly useful for tracking live equipment metrics, detecting threshold violations, and generating alerts for unusual behaviour.

4. SYSTEM ARCHITECTURE

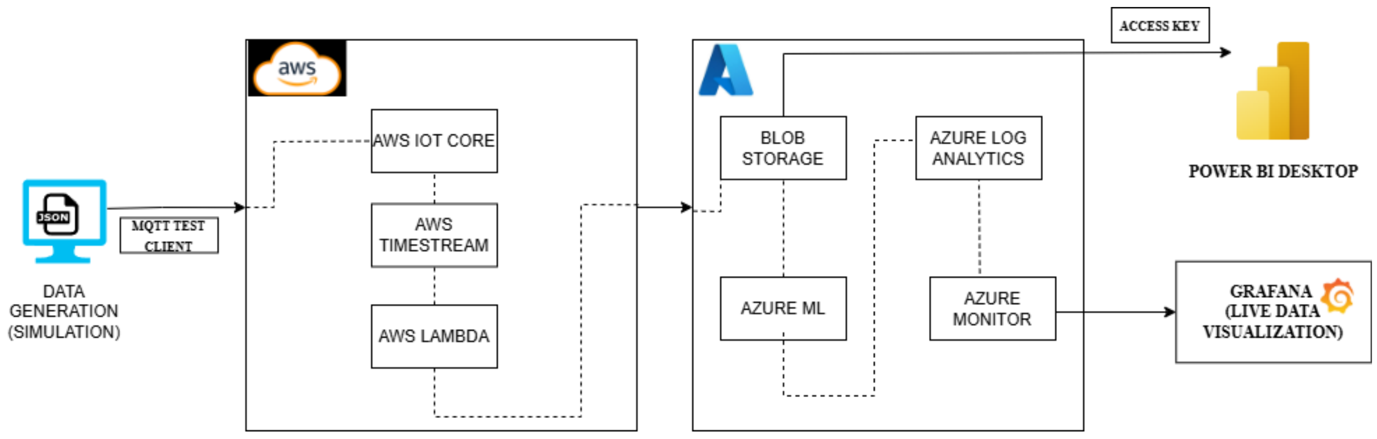


FIGURE 01- END TO END SYSTEM ARCHITECTURE

The system architecture for the Industrial Equipment Monitoring and Predictive Maintenance project is designed to efficiently collect, process, store, and visualize real-time data from industrial equipment. It includes the following key components:

➢ DATA GENERATION (SIMULATION)

- This component uses a python code-based data generator, running on a local machine to produce realistic sensor data (e.g., temperature, pressure, vibration).
- The data is formatted in JSON and sent to the cloud via the MQTT (Message Queuing Telemetry Transport) protocol, which is lightweight and ideal for IoT communications.

- **AWS IOT CORE**

- Acts as the primary gateway for securely connecting IoT devices to the cloud.
- It handles device authentication, message routing, and real-time data ingestion, allowing seamless integration with other AWS services.
- Supports low-latency, bi-directional communication between IoT devices and the cloud.

- **AWS TIMESTREAM**

- A fully managed, serverless, time-series database designed for IoT applications.
- It efficiently captures, stores, and queries high-volume, time-stamped data from connected devices.
- Features automated data tiering and built-in analytics, making it ideal for handling large-scale telemetry data.

- **AWS LAMBDA**

- A serverless compute service that triggers automatically when new data is available in Timestream.
- It processes incoming telemetry, applies data transformation, filters out anomalies, and forwards the cleaned data to Azure for deeper analysis.
- This component reduces infrastructure management and lowers costs, as you only pay for the actual compute time.

- **AZURE BLOB STORAGE**

- Provides scalable, cost-effective storage for raw and processed telemetry data.
- Acts as a central data lake, supporting both structured and unstructured data for further analytics and machine learning.
- The data is stored in a secure, high-availability environment, making it ideal for long-term archival and batch processing.

- **AZURE ML**

  - Used for developing, training, and deploying machine learning models for predictive maintenance.
  - Leverages historical data stored in Blob Storage to identify early warning signs of equipment failure, reducing downtime and maintenance costs.

- **AZURE LOG ANALYTICS**

  - Collects, organizes, and analyzes telemetry and log data for operational insights.
  - Integrates seamlessly with Azure Monitor to provide real-time alerts and detailed analytics for predictive maintenance.

- **AZURE MONITOR**

  - Monitors real-time data streams for critical metrics like temperature, vibration, and operating hours.
  - Generates automated alerts based on predefined thresholds, allowing for proactive maintenance and faster response to equipment anomalies.

- **GRAFANA (LIVE DATA VISUALIZATION)**

  - Offers customizable, real-time dashboards for visualizing key performance indicators (KPIs) and machine health metrics.
  - Directly connects to Azure Monitor for live data insights, supporting powerful data visualizations and alerting features.

- **POWER BI DESKTOP**

  - Provides business intelligence capabilities, enabling the creation of interactive reports and data visualizations.
  - Connects to Azure Blob Storage to generate insights from historical data, supporting strategic decision-making and performance optimization.

5. METHODOLOGY

This section outlines the approach taken to implement the Industrial Equipment Monitoring and Predictive Maintenance system, including data collection, processing, security measures, and visualization techniques.

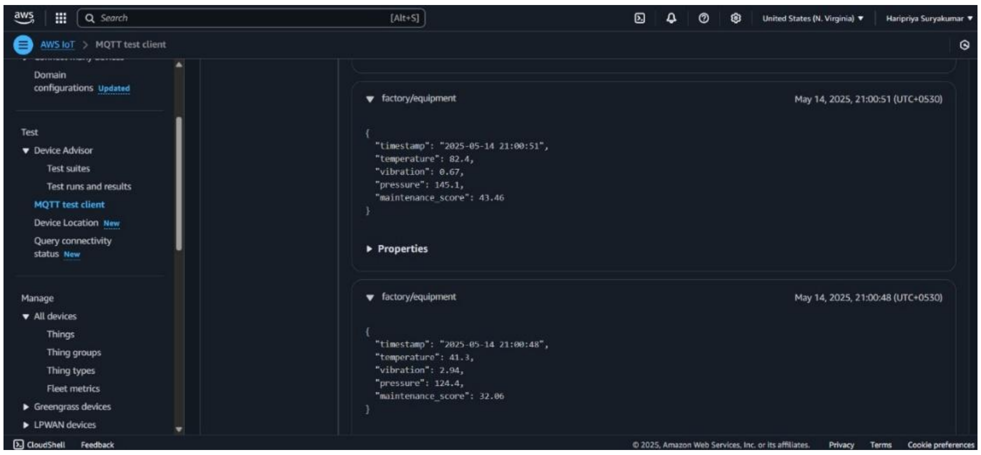
5.1 OVERVIEW OF PROJECT IMPLEMENTATION

The project was implemented using a combination of AWS and Azure cloud platforms to ensure efficient data collection, processing, and analysis. The overall workflow involves real-time data ingestion from IoT devices, serverless data processing, and predictive analytics to identify equipment anomalies and optimize maintenance schedules.

5.2 DATA COLLECTION AND PROCESSING METHODS

Data was generated using a simulated MQTT client to mimic real-world industrial sensor outputs. This data was sent to AWS IoT Core, where it was securely ingested and routed to AWS Timestream for efficient time-series storage. The data flow included the following key steps:

- **Data Generation:** Simulated equipment data (e.g., temperature, pressure, vibration) was produced using MQTT test clients, formatted as JSON.
- **Data Ingestion:** AWS IoT Core received and authenticated the incoming messages, ensuring secure device communication.



factory/equipment May 14, 2025, 21:00:51 (UTC+0530)

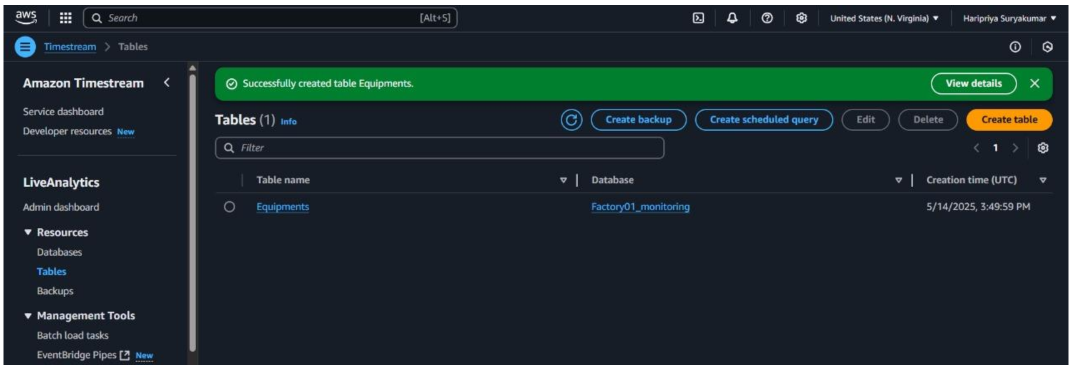
```
{
  "timestamp": "2025-05-14 21:00:51",
  "temperature": 82.4,
  "vibration": 0.67,
  "pressure": 145.1,
  "maintenance_score": 43.46
}
```

factory/equipment May 14, 2025, 21:00:48 (UTC+0530)

```
{
  "timestamp": "2025-05-14 21:00:48",
  "temperature": 41.3,
  "vibration": 2.94,
  "pressure": 124.4,
  "maintenance_score": 32.06
}
```

FIGURE 02- MQTT TEST CLIENT RECEIVING SIMULATED DATA

- **Data Processing:** AWS Lambda functions were triggered to process and transform the data, filtering out noise and preparing it for long-term storage.



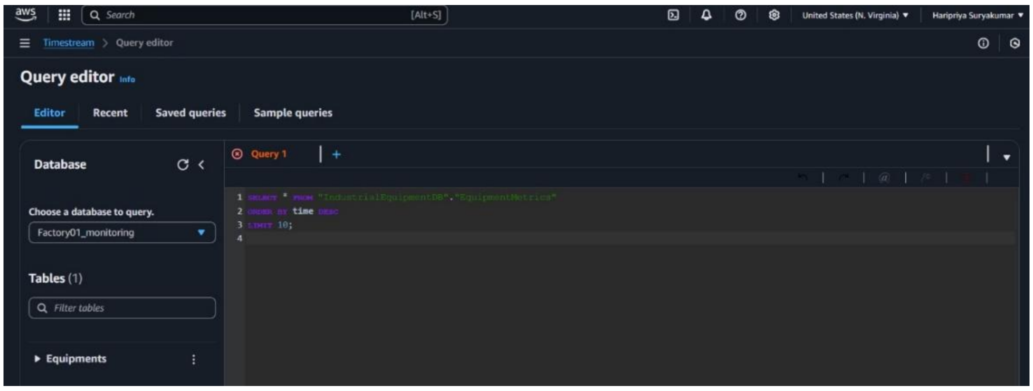
Successfully created table Equipments.

Tables (1)

Table name	Database	Creation time (UTC)
Equipments	Factory01_monitoring	5/14/2025, 3:49:59 PM

FIGURE 03- AWS TIMESTREAM CONFIGURATION

- **Data Storage:** The processed data was then sent to Azure Blob Storage, creating a centralized data lake for historical analysis and machine learning.



Choose a database to query: Factory01_monitoring

```sql
SELECT * FROM "IndustrialEquipmentDB"."EquipmentMetrics"
ORDER BY time DESC
LIMIT 10;
```

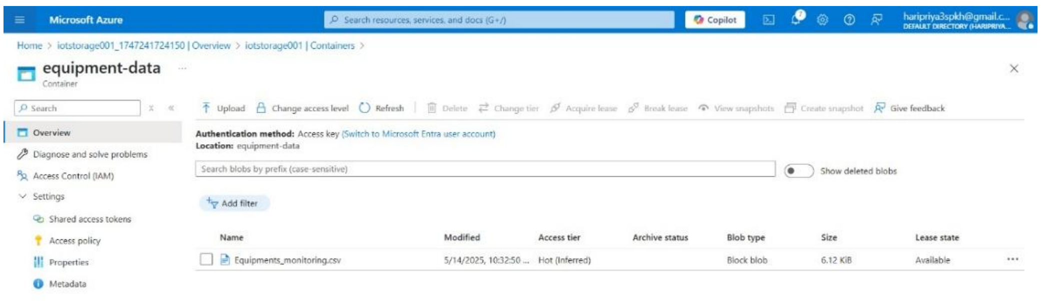
FIGURE 04- TIMESTREAM QUERY EDITOR

5.3 SECURITY AND MIDDLEWARE INTEGRATION

Security was a critical aspect of this project, ensuring data integrity and confidentiality at every stage:

- **Authentication and Authorization:** AWS IoT Core used device certificates and policies to securely authenticate incoming data streams.
- **Data Encryption**: All data in transit and at rest was encrypted using TLS and AES-256, respectively.
- **Access Management**: IAM roles and policies were configured to limit data access, ensuring only authorized services could interact with the IoT data.

- **Middleware (AWS Lambda):** Acted as a secure processing layer, enforcing data validation before forwarding to Azure.



equipment-data

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: equipment-data

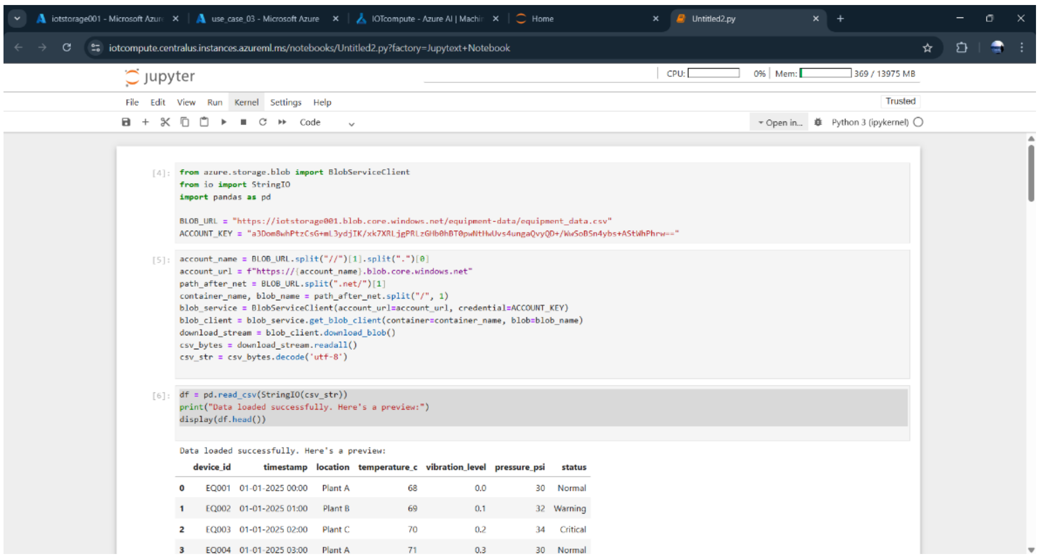
Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
Equipments_monitoring.csv	5/14/2025, 10:32:50 ...	Hot (Inferred)		Block blob	6.12 KiB	Available

FIGURE 05- AZURE BLOB STORAGE CONTAINER

5.4 DATA ANALYSIS APPROACH (USING AZURE AND POWER BI)

Once data was securely stored in Azure Blob Storage, it was analysed using a combination of Azure Machine Learning and Power BI:

- **Data Preparation:** Data was cleaned and structured for analysis using Azure ML pipelines.
- **Predictive Analytics:** Machine learning models were trained to identify early signs of equipment failure, leveraging historical telemetry data.


iotstorage001 - Microsoft Azure
use_case_03 - Microsoft Azure
IOTcompute - Azure AI | Machi
Home
Untitled2.py
iotcompute.centralus.instances.azureml.ms/notebooks/Untitled2.py?factory=Jupytext+Notebook
jupyter
CPU: 0% | Mem: 369 / 13975 MB
File
Edit
View
Run
Kernel
Settings
Help
Trusted
Code
Open in...
Python 3 (ipykernel)

```
[4]: from azure.storage.blob import BlobServiceClient
     from io import StringIO
     import pandas as pd

     BLOB_URL = "https://iotstorage001.blob.core.windows.net/equipment-data/equipment_data.csv"
     ACCOUNT_KEY = "a3DomBwhPtzCsG+mL3ydjIK/xk7XRLjgPRLzGHb0hBT0pwNtHwUvs4ungaQvyQD+/WwSoBSn4ybs+AStWhPhrw=="

[5]: account_name = BLOB_URL.split("//")[1].split(".")[0]
     account_url = f"https://{account_name}.blob.core.windows.net"
     path_after_net = BLOB_URL.split(".net/")[1]
     container_name, blob_name = path_after_net.split("/", 1)
     blob_service = BlobServiceClient(account_url=account_url, credential=ACCOUNT_KEY)
     blob_client = blob_service.get_blob_client(container=container_name, blob=blob_name)
     download_stream = blob_client.download_blob()
     csv_bytes = download_stream.readall()
     csv_str = csv_bytes.decode('utf-8')

[6]: df = pd.read_csv(StringIO(csv_str))
     print("Data loaded successfully. Here's a preview:")
     display(df.head())
```

```
Data loaded successfully. Here's a preview:
```

	device_id	timestamp	location	temperature_c	vibration_level	pressure_psi	status
0	EQ001	01-01-2025 00:00	Plant A	68	0.0	30	Normal
1	EQ002	01-01-2025 01:00	Plant B	69	0.1	32	Warning
2	EQ003	01-01-2025 02:00	Plant C	70	0.2	34	Critical
3	EQ004	01-01-2025 03:00	Plant A	71	0.3	30	Normal

FIGURE 06- AZURE ML VM - COMPUTING

- **Business Intelligence:** Power BI Desktop was used to generate insightful, interactive reports for operational decision-making.

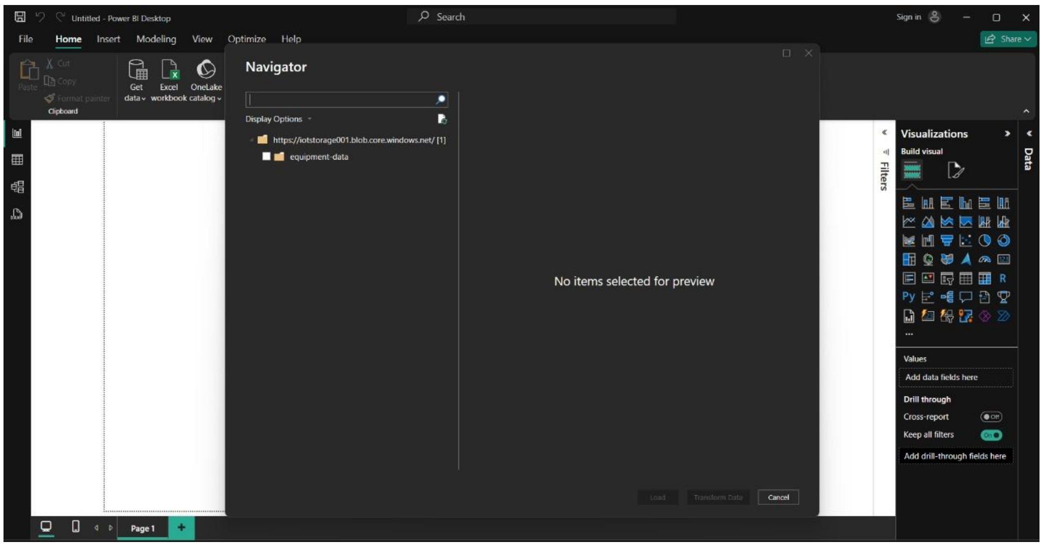


FIGURE 07- CONNECTING POWER BI DESKTOP TO AZURE BLOB STORAGE VIA ACESS KEY

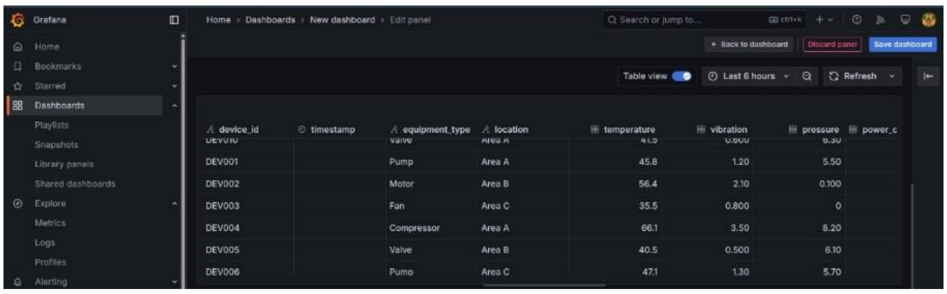
5.5 VISUALIZATION TECHNIQUES (USING GRAFANA)

Real-time data visualization was achieved using Grafana, which provides rich, interactive dashboards:

- **Live Metrics**: The system is designed to provide continuous monitoring of key performance metrics such as temperature, vibration, and pressure in near real-time. These live metrics are collected from simulated IoT sensors and visualized using dynamic dashboards, offering instant feedback on equipment behaviours and operational conditions. By streaming this data in real time, the system allows operators to maintain constant awareness of asset performance, identify anomalies as they occur, and make informed decisions without delay. This capability is especially valuable in industrial settings where rapid response can prevent equipment damage or production downtime.
- **Custom Dashboards**: Custom dashboards were developed to present a clear, role-specific view of system performance and asset health. These dashboards aggregate sensor data, analytics outputs, and alert statuses into visually intuitive

layouts tailored to different user groups—such as maintenance personnel, system administrators, or operations managers. Users can track historical trends, compare metrics across multiple assets, and monitor real-time performance within a single interface. This customization improves usability and ensures that stakeholders receive only the most relevant insights for their responsibilities, thus enhancing situational awareness and decision-making.

- **Alerting**: To support proactive maintenance and reduce unplanned downtime, the system incorporates an automated alerting mechanism. Alerts are configured to trigger when critical metrics, such as temperature or vibration levels, exceed predefined thresholds. When such a breach is detected, notifications are immediately sent to relevant personnel through preferred communication channels, including email or SMS. This allows maintenance teams to act swiftly before minor issues escalate into major failures. The alerting system ensures a timely response to equipment anomalies, reinforcing operational safety, improving reliability, and optimizing maintenance schedules.



device_id	timestamp	equipment_type	location	temperature	vibration	pressure	power_c
DEV010		Valve	Area A	41.5	0.600	6.30	
DEV001		Pump	Area A	45.8	1.20	5.50	
DEV002		Motor	Area B	56.4	2.10	0.100	
DEV003		Fan	Area C	35.5	0.800	0	
DEV004		Compressor	Area A	66.1	3.50	8.20	
DEV005		Valve	Area B	40.5	0.500	6.10	
DEV006		Pump	Area C	47.1	1.30	5.70	

FIGURE 08– DATA IN TABLE VIEW IN GRAFANA

6. RESULTS AND DISCUSSIONS

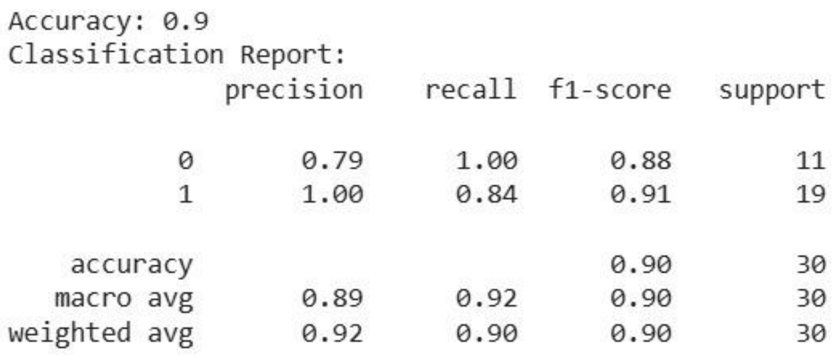
This section presents the outcomes of the data analysis and visualization efforts, highlighting the key insights derived from the project and evaluating how effectively the solution meets its objectives.

6.1 DATA ANALYSIS RESULTS

The predictive maintenance model developed as part of this project demonstrated strong performance in classifying equipment conditions into "Repair" and "No Repair" categories. The model achieved an overall accuracy of 90%, indicating that nine out of ten predictions were correct on the test dataset.

A detailed examination of the classification report reveals:

- For the "No Repair" class, the model showed a precision of 79% and a perfect recall of 100%. This means that when the model predicted "No Repair," it was correct 79% of the time, and it successfully identified all actual "No Repair" instances without missing any.
- For the "Repair" class, the model achieved a precision of 100% and a recall of 84%. In other words, every prediction of "Repair" was accurate, though 16% of actual "Repair" cases were missed by the model and incorrectly classified as "No Repair."
- The F1-scores, which balance precision and recall, were 0.88 for "No Repair" and 0.91 for "Repair," reflecting reliable model performance across both classes.



```
Accuracy: 0.9
Classification Report:
              precision    recall  f1-score   support

           0       0.79      1.00      0.88        11
           1       1.00      0.84      0.91        19

    accuracy                           0.90        30
   macro avg       0.89      0.92      0.90        30
weighted avg       0.92      0.90      0.90        30
```

FIGURE 09- CLASSIFICATION REPORT

The confusion matrix offers deeper insight into the classification performance of the predictive maintenance model, especially regarding its ability to correctly identify instances where no repair is required. In this case, the model demonstrated exceptional accuracy by correctly classifying all 11 instances that were actually labeled as "No Repair." This means that there were zero false positives for this particular class—none of the cases that required no intervention were mistakenly flagged for repair. This level of precision is highly valuable in industrial environments, as it ensures that maintenance teams are not directed toward equipment that is functioning normally. Avoiding unnecessary maintenance not only conserves resources but also reduces potential risks associated with over-servicing, such as introducing new faults or increasing downtime due to avoidable inspections.

Furthermore, the model's perfect classification of "No Repair" cases indicates that it has effectively learned to distinguish between normal operational patterns and those signaling degradation or failure. This distinction is critical in predictive maintenance systems, where the goal is not only to detect potential issues but also to minimize false alarms that could lead to workflow interruptions. The ability to accurately identify healthy equipment contributes significantly to operational efficiency, allowing maintenance efforts to be focused where they are truly needed. Over time, this precision supports a more cost-effective and streamlined maintenance strategy, increases the trust of end-users in AI-driven recommendations, and enhances the overall credibility of the predictive system.

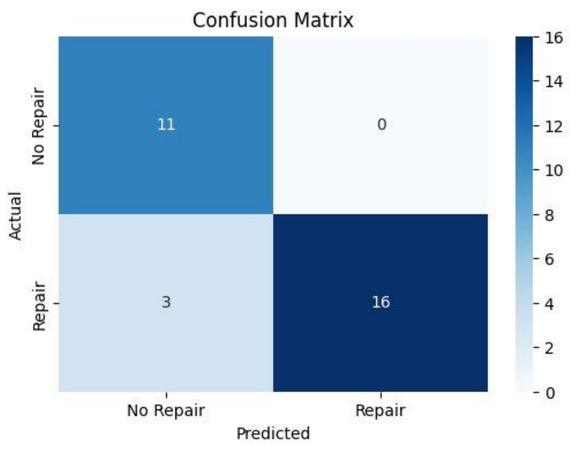


FIGURE 10- CONFUSION MATRIX

6.2 INSIGHTS DERIVED FROM THE POWER BI DASHBOARDS

Power BI dashboards provided comprehensive visual summaries of historical and real-time data. Users could easily track equipment performance trends, compare operational parameters across different assets, and monitor key indicators such as temperature fluctuations and vibration levels. The interactive reports facilitated detailed drill-down analysis, empowering stakeholders to make data-driven maintenance decisions.

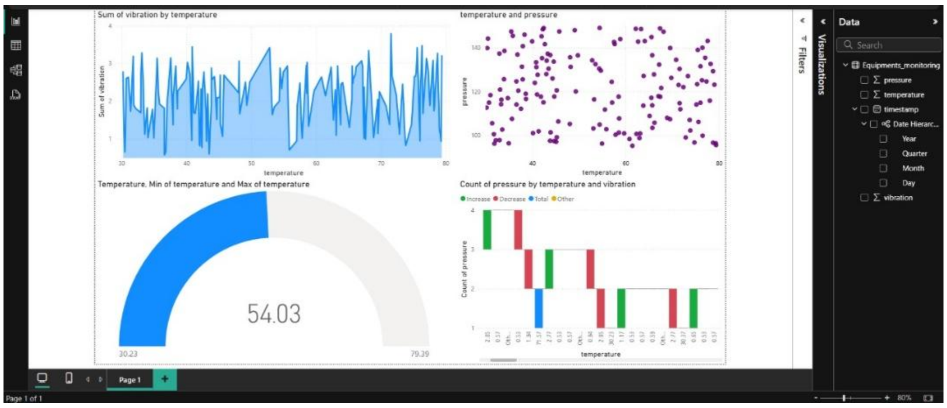

Sum of vibration by temperature
temperature and pressure
Temperature, Min of temperature and Max of temperature
54.03
30.23
79.39
Count of pressure by temperature and vibration
Data
Equipments_monitoring
pressure
temperature
timestamp
vibration
Page 1

FIGURE 11- POWER BI DASHBOARD (RAW DATA)

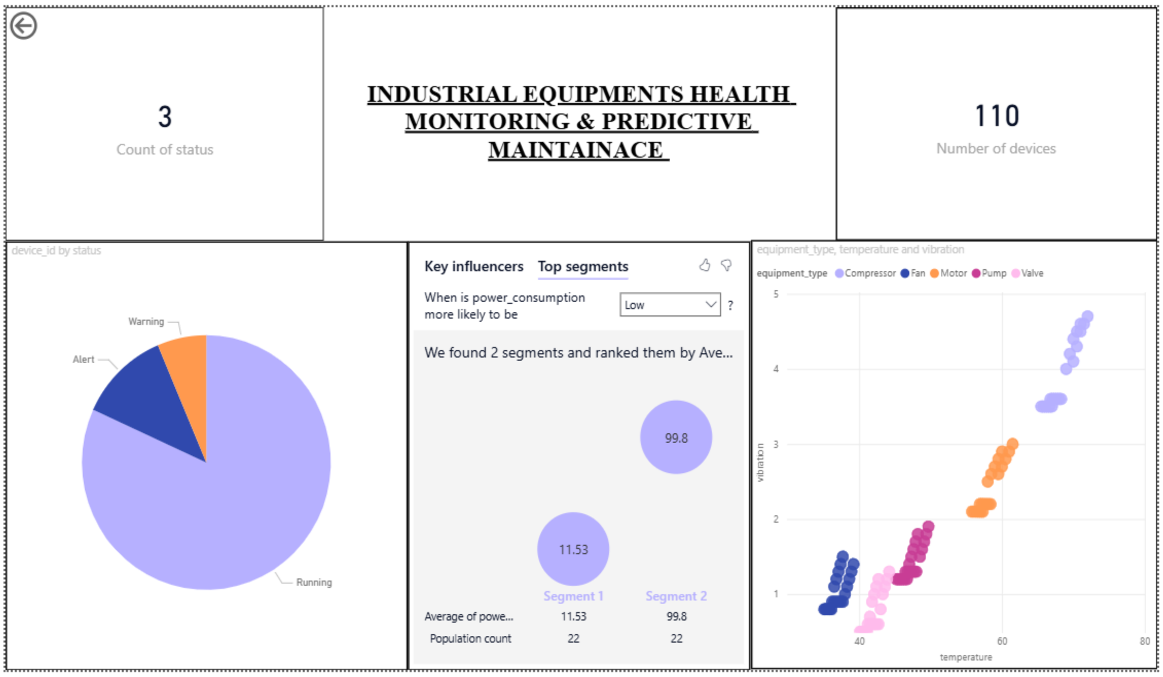

3
Count of status
INDUSTRIAL EQUIPMENTS HEALTH MONITORING & PREDICTIVE MAINTAINACE
110
Number of devices
device_id by status
Warning
Alert
Running
Key influencers
Top segments
When is power_consumption more likely to be
Low
We found 2 segments and ranked them by Ave...
99.8
11.53
Segment 1
Segment 2
Average of powe... 11.53 99.8
Population count 22 22
equipment_type, temperature and vibration
equipment_type Compressor Fan Motor Pump Valve
vibration
temperature

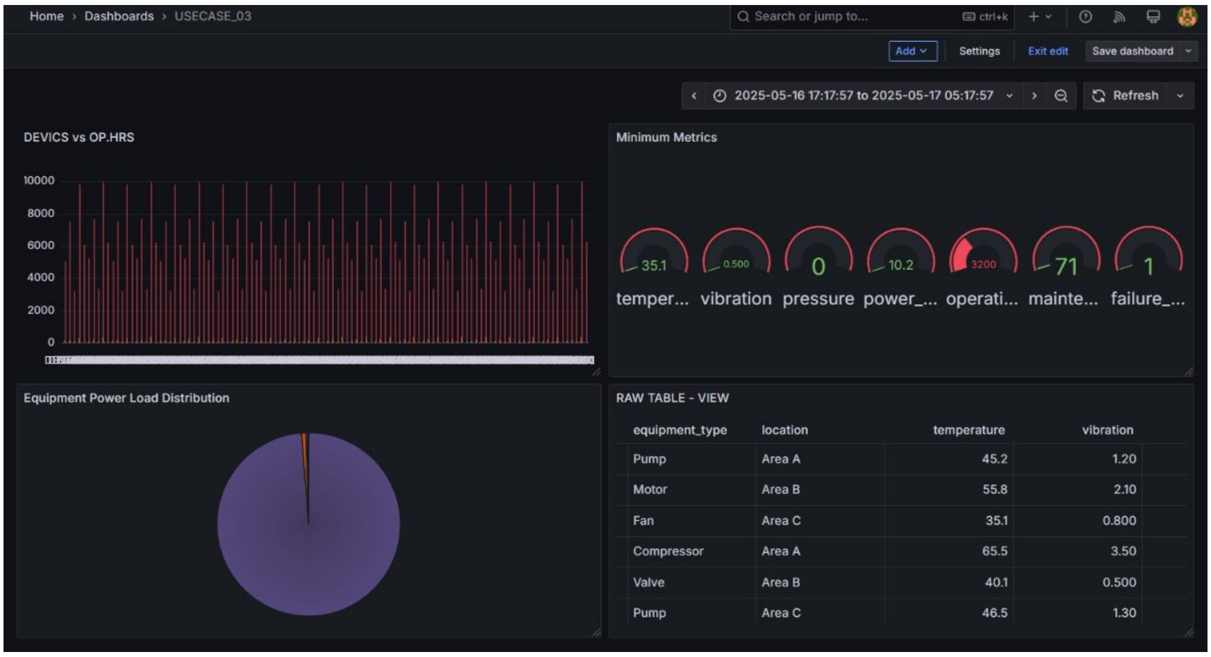
FIGURE 12- POWER BI DASHBOARD(PREDICTED DATA)

6.3 REAL-TIME MONITORING INSIGHTS FROM GRAFANA

Grafana dashboards provided real-time visibility into equipment health, transforming raw telemetry data into actionable insights for maintenance teams. These dashboards tracked key performance metrics such as temperature, vibration, and power consumption, presenting this data in intuitive, interactive formats.

Beyond just visualizing live metrics, the Grafana dashboards were configured to display critical status indicators, including the number of devices approaching failure and those needing immediate repair. For instance, live charts highlighted equipment nearing predefined failure thresholds, allowing operators to proactively address issues before they escalate. This included tracking rapid temperature spikes, abnormal vibration patterns, or pressure surges that often precede mechanical breakdowns. Additionally, dedicated panels summarized the number of devices already exhibiting critical fault conditions, signaling the need for immediate intervention to prevent costly downtime.

The dashboards also provided an overall view of system health, offering at-a-glance summaries of the percentage of devices operating normally, those approaching failure, and those requiring urgent repair. This holistic view enabled maintenance teams to prioritize resources effectively and reduce unplanned outages.



equipment_type	location	temperature	vibration
Pump	Area A	45.2	1.20
Motor	Area B	55.8	2.10
Fan	Area C	35.1	0.800
Compressor	Area A	65.5	3.50
Valve	Area B	40.1	0.500
Pump	Area C	46.5	1.30

FIGURE 13- GRAFANA REAL TIME DATA DASHBOARD

6.4 MEETING PROJECT OBJECTIVES

Based on the analysis of the visual outputs and performance metrics, the implemented solution successfully fulfils the project objectives of monitoring industrial equipment and enabling predictive maintenance. The integration of AWS and Azure platforms provided a robust framework for secure and efficient data ingestion, processing, and storage.

The predictive maintenance model's high accuracy and clear classification results demonstrate its capability to reliably identify equipment needing repair, supporting proactive decision-making. Power BI dashboards offer comprehensive, interactive visualizations that translate complex data into actionable insights for operational teams. Real-time Grafana monitoring enhances responsiveness by delivering live alerts and status updates, ensuring immediate attention to critical issues.

Together, these visualizations confirm that the system meets the goals of reducing unplanned downtime, lowering maintenance costs, and improving overall asset utilization. The combination of historical data analytics and real-time monitoring enables a comprehensive approach to equipment management, validating the effectiveness of the solution.

7. CHALLENGES AND SOLUTIONS

During the course of this project, several key challenges were encountered in the implementation of the industrial equipment monitoring and predictive maintenance system. These challenges, along with the solutions applied, are outlined below:

7.1 KEY CHALLENGES:

- **Data Communication and Integration:** Ensuring seamless and reliable communication between diverse IoT devices, AWS services, and Azure components posed integration complexities, especially with differing protocols and data formats.
- **Sensor Data Accuracy and Calibration:** Simulating realistic sensor data required careful calibration to reflect true operational conditions and avoid misleading analytics.
- **Latency and Real-Time Processing:** Managing real-time data flow with minimal latency was critical for effective monitoring and timely alerts, which required optimizing data pipelines and processing functions.

7.2 SOLUTIONS IMPLEMENTED:

- **Middleware and Standardized Data Formats:** AWS Lambda functions were utilized as middleware to preprocess and normalize incoming data, ensuring compatibility between IoT devices and cloud services.
- **Sensor Data Simulation Tools:** Custom MQTT clients with configurable parameters were used to generate calibrated and realistic sensor data streams.
- **Efficient Serverless Architecture:** Leveraging AWS IoT Core, Timestream, and Lambda enabled scalable, event-driven data processing, minimizing latency and improving responsiveness.

8. CONCLUSION

This project successfully developed an industrial equipment monitoring and predictive maintenance system, integrating real-time data ingestion, machine learning, and interactive data visualization. The solution effectively addressed the key objectives of reducing unplanned downtime, optimizing maintenance schedules, and improving asset utilization. The seamless integration of AWS and Azure platforms provided a robust and scalable infrastructure for real-time monitoring and long-term data analytics.

The high accuracy of the predictive models and the insightful Power BI dashboards demonstrated the practical value of data-driven maintenance strategies. Real-time alerts through Grafana further enhanced the system’s responsiveness, enabling rapid intervention when anomalies were detected. Overall, the project highlights the potential of IoT and cloud technologies in transforming industrial maintenance practices.

However, the project also revealed opportunities for further enhancement, such as improving model precision for under-represented failure modes and optimizing data pipelines for even lower latency. These considerations will guide future iterations of the system.

9. FUTURE WORK

Several potential extensions can be pursued to enhance the capabilities and impact of this project:

- **Integration of Additional Sensors:** Expanding the range of monitored parameters (e.g., pressure, humidity, RPM) to capture more comprehensive equipment health data.
- **Advanced Analytics and AI Models:** Implementing deep learning models or anomaly detection algorithms for more accurate failure prediction and root cause analysis.
- **Edge Computing for Faster Response:** Shifting some data processing to edge devices to reduce latency and improve real-time decision-making.
- **Scalability for Large-Scale Deployment:** Optimizing the architecture for deployment across multiple sites or even global manufacturing networks.
- **Integration with Maintenance Management Systems:** Connecting the predictive maintenance system with enterprise asset management (EAM) platforms for automated maintenance scheduling.
- **Enhanced Security Measures:** Incorporating blockchain or zero-trust architecture for more robust data security.

10. REFERENCES

- https://aws.amazon.com/blogs/iot/using-aws-iot-for-predictive-maintenance/
- https://www.cloudjournee.com/blog/aws-iot-real-time-monitoring-predictive-maintenance-manufacturing/
- https://docs.aws.amazon.com/lambda/
- https://learn.microsoft.com/en-us/azure/machine-learning/?view=azureml-api-2
- https://grafana.com/docs/

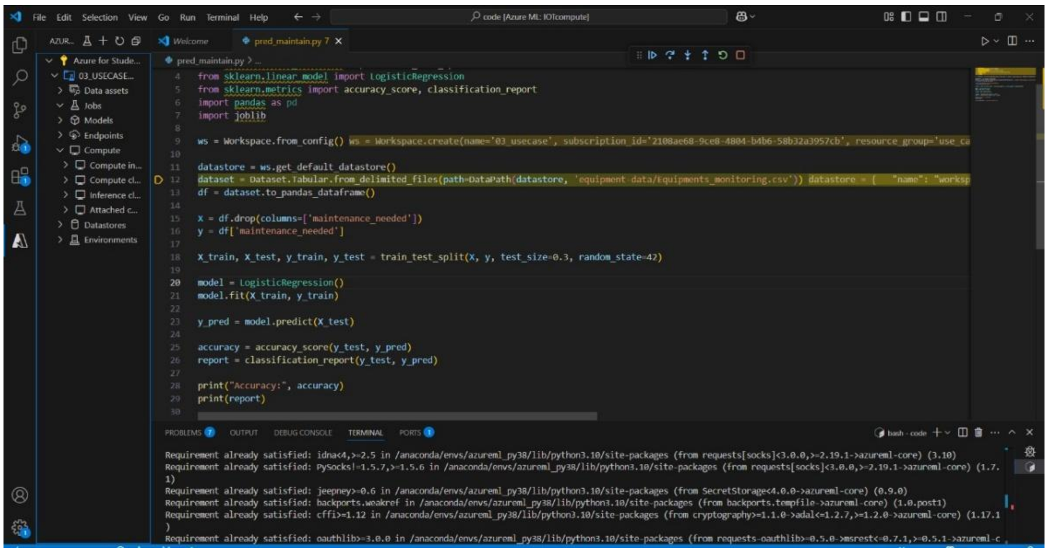
11. APPENDICES

APPENDIX A: CODE SNIPPETS

A.1 Python Data Simulator Code

```
import ssl

import time

import json

import random

from paho.mqtt import client as mqtt_client

AWS_ENDPOINT = "a3dmb30l73pi7o-ats.iot.us-east-1.amazonaws.com"

PORT = 8883

TOPIC = "factory/equipment"

CLIENT_ID = "equipment-sensor-1"

# Certificate paths

CA_PATH = "AmazonRootCA1.pem"

CERT_PATH = "certificate.pem.crt"

KEY_PATH = "private.pem.key"

def on_connect(client, userdata, flags, rc):

    if rc == 0:

        print("Connected to AWS IoT Core")

    else:
        print(f"Connection failed with code {rc}")
def generate_sensor_data():
    temperature = round(random.uniform(35, 85), 1)
    vibration = round(random.uniform(0.1, 3.5), 2)
    pressure = round(random.uniform(90, 150), 2)
    maintenance_score = round(
        max(0, 100 - (temperature * 0.4 + vibration * 15 + (pressure - 100) * 0.3)), 2
    )
    return {
        "timestamp": time.strftime('%Y-%m-%d %H:%M:%S'),
        "temperature": temperature,
        "vibration": vibration,
        "pressure": pressure,
        "maintenance_score": maintenance_score
    }
def publish_data(client):
    while True:
        data = generate_sensor_data()
        payload = json.dumps(data)
        result = client.publish(TOPIC, payload)
        status = result[0]
        if status == 0:
            print(f"Published: {payload}")
        else:
            print("Failed to send message")
        time.sleep(3)
def connect_mqtt():
    client = mqtt_client.Client(CLIENT_ID)
    client.on_connect = on_connect
    client.tls_set(
        ca_certs=CA_PATH,
        certfile=CERT_PATH,
        keyfile=KEY_PATH,
        tls_version=ssl.PROTOCOL_TLSv1_2
    )
    client.connect(AWS_ENDPOINT, PORT)
    return client
if __name__ == '__main__':
    client = connect_mqtt()
    client.loop_start()
```

publish_data(client)



```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd
import joblib

ws = Workspace.from_config() ws = Workspace.create(name='03_usecase', subscription_id='2108ae68-9ce8-4804-b4b6-58b32a3957cb', resource_group='use_ca

datastore = ws.get_default_datastore()
dataset = Dataset.Tabular.from_delimited_files(path=DataPath(datastore, 'equipment-data/Equipments_monitoring.csv')) datastore = {   "name": "worksp
df = dataset.to_pandas_dataframe()

X = df.drop(columns=['maintenance_needed'])
y = df['maintenance_needed']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

model = LogisticRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print(report)
```

```
Requirement already satisfied: idna<4,>=2.5 in /anaconda/envs/azureml_py38/lib/python3.10/site-packages (from requests[socks]<3.0.0,>=2.19.1->azureml-core) (3.10)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /anaconda/envs/azureml_py38/lib/python3.10/site-packages (from requests[socks]<3.0.0,>=2.19.1->azureml-core) (1.7.
1)
Requirement already satisfied: jeepney>=0.6 in /anaconda/envs/azureml_py38/lib/python3.10/site-packages (from SecretStorage<4.0.0->azureml-core) (0.9.0)
Requirement already satisfied: backports.weakref in /anaconda/envs/azureml_py38/lib/python3.10/site-packages (from backports.tempfile->azureml-core) (1.0.post1)
Requirement already satisfied: cffi>=1.12 in /anaconda/envs/azureml_py38/lib/python3.10/site-packages (from cryptography>=1.1.0->adal<=1.2.7,>=1.2.0->azureml-core) (1.17.1
)
Requirement already satisfied: oauthlib>=3.0.0 in /anaconda/envs/azureml_py38/lib/python3.10/site-packages (from requests-oauthlib>=0.5.0->msrest<=0.7.1,>=0.5.1->azureml-c
```

FIGURE 14- SIMULATED EQUIPMENTS' DATA

A.2 AWS Lambda Data Forwarding Code

import os

import boto3

import json

from azure.storage.blob import BlobServiceClient

from datetime import datetime

CONNECTION_STRING = "DefaultEndpointsProtocol=https;AccountName=iotstorage001;AccountKey=lVN4x+LFlk1lMCPL/wF2zj+h1WxmlTl1eQiSlSwRGCK//xLOimzKSv6MYlHAyFbFhmSQBWgw7J1O+AStEUJRxw==;EndpointSuffix=core.windows.net"

CONTAINER_NAME = "equipment-data"

BLOB_NAME = "Equipments_monitoring.csv"

DATABASE_NAME = "Factory01_monitoring"

```
TABLE_NAME = "equipments"

def lambda_handler(event, context):

    try:

        # Initialize AWS Timestream and Azure Blob clients

        timestream_client = boto3.client('timestream-query')

        blob_service_client                                                          =
BlobServiceClient.from_connection_string(CONNECTION_STRING)

        blob_client                                                                  =
blob_service_client.get_blob_client(container=CONTAINER_NAME,
blob=BLOB_NAME)

        # Query Timestream for recent data

        query = f"""

        SELECT

            time,

            device_id,

            temperature,

            vibration,

            status

        FROM "{DATABASE_NAME}"."{TABLE_NAME}"

        ORDER BY time DESC

        LIMIT 50
```

```
"""

response = timestream_client.query(QueryString=query)

# Prepare data as CSV

csv_data = ""

for row in response['Rows']:

    timestamp = row['Data'][0]['ScalarValue']

    device_id = row['Data'][1]['ScalarValue']

    temperature = row['Data'][2]['ScalarValue']

    vibration = row['Data'][3]['ScalarValue']

    status = row['Data'][4]['ScalarValue']

    csv_data += f"{timestamp},{device_id},{temperature},{vibration},{status}\n"

try:

    existing_data = blob_client.download_blob().content_as_text()

    new_data = existing_data + csv_data

except Exception as e:

    # If blob doesn't exist, create a new one

    new_data = "timestamp,device_id,temperature,vibration,status\n" + csv_data

blob_client.upload_blob(new_data, blob_type="BlockBlob", overwrite=True)

print("Data successfully forwarded to Azure Blob Storage.")

return {
```

```
            'statusCode': 200,

            'body': 'Data successfully forwarded to Azure Blob Storage.'

        }

    except Exception as e:

        print(f"Error: {e}")

        return {

            'statusCode': 500,

            'body': f"Error forwarding data: {str(e)}"

        }
```

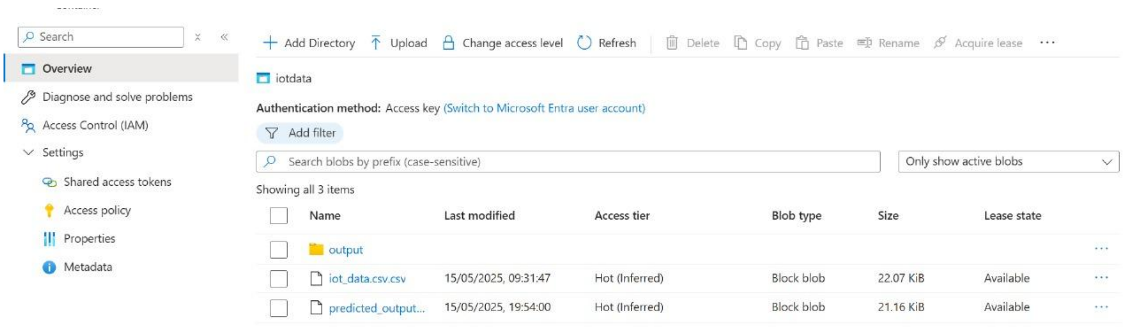
A.3 MODEL TRAINING AND PREDICTIVE ANALYSIS

```
!pip install azure-storage-blob --quiet

from azure.storage.blob import BlobServiceClient

import pandas as pd

from io import StringIO

from azure.storage.blob import BlobServiceClient

connection_string = (

    "DefaultEndpointsProtocol=https;"

    "AccountName=iotstorage001;"


"AccountKey=rvDUHzZGsL86iZKIx3Und6s69KVlbnsBehREy5TNERK5TGG8Q1r+b
PxZPrrTWCCbRV9C12jyRA4M+AStZ13p8w==;"
```

```
    "EndpointSuffix=core.windows.net"
)
blob_service_client = BlobServiceClient.from_connection_string(connection_string)
blob_client = blob_service_client.get_blob_client(container=container_name,
blob=blob_name)
downloaded_blob = blob_client.download_blob()
csv_content = downloaded_blob.content_as_text()
downloaded_blob = blob_client.download_blob()
csv_content = downloaded_blob.content_as_text()
df = pd.read_csv(StringIO(csv_content))
df.head()
df.info()
df.isnull().sum()
df.describe()
from azure.storage.blob import BlobServiceClient
import json
import pandas as pd
import io
connection_string = (
    "DefaultEndpointsProtocol=https;"
    "AccountName=iotstorage001;"
```

```
"AccountKey=rvDUHzZGsL86iZKIx3Und6s69KVlbnsBehREy5TNERK5TGG8Q1r+b
PxZPrrTWCCbRV9C12jyRA4M+AStZ13p8w==;"

    "EndpointSuffix=core.windows.net"

)

container_name = "iotdata"

blob_name = "output/2025/05/15/.json"

blob_service_client = BlobServiceClient.from_connection_string(connection_string)

blob_client      =      blob_service_client.get_blob_client(container=container_name,
blob=blob_name)

blob_list = container_client.list_blobs()

for blob in blob_list:

    print(blob.name)

df['timestamp'] = pd.to_datetime(df['timestamp'])

import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))

plt.plot(df['timestamp'], df['temperature'], label='Temperature (°C)', color='tomato')

plt.plot(df['timestamp'], df['humidity'], label='Humidity (%)', color='skyblue')

plt.xlabel('Timestamp')

plt.ylabel('Sensor Readings')

plt.title('Temperature and Humidity Over Time')

plt.legend()
plt.xticks(rotation=45)

plt.tight_layout()

plt.show()

cols_to_drop = ['timestamp', 'EventProcessedUtcTime', 'EventEnqueuedUtcTime',
'PartitionId',

            'MessageId', 'CorrelationId', 'ConnectionDeviceId',
'ConnectionDeviceGenerationId', 'EnqueuedTime']

df_clean = df.drop(columns=[col for col in cols_to_drop if col in df.columns])

X = df_clean.drop(columns=['temperature'])

y = df_clean['temperature']

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestRegressor()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))

df['predicted_temperature'] = model.predict(X)

df.to_csv("predicted_output.csv", index=False)

from azure.storage.blob import BlobServiceClient
```

```
connection_string = (
    "DefaultEndpointsProtocol=https;"
    "AccountName=iotstorage001;"
"AccountKey=rvDUHzZGsL86iZKIx3Und6s69KVlbnsBehREy5TNERK5TGG8Q1r+bPxZPrrTWCCbRV9C12jyRA4M+AStZ13p8w==;"
    "EndpointSuffix=core.windows.net"
)
container_name = "iotdata"
blob_name = "predicted_output.csv"
blob_service_client = BlobServiceClient.from_connection_string(connection_string)
blob_client = blob_service_client.get_blob_client(container=container_name, blob=blob_name)
with open("predicted_output.csv", "rb") as data:
    blob_client.upload_blob(data, overwrite=True)
```



iotdata

Authentication method: Access key (Switch to Microsoft Entra user account)

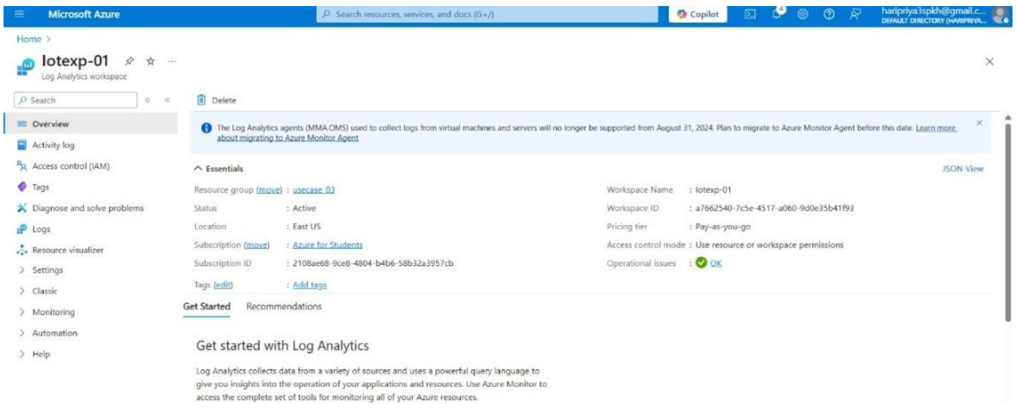
Only show active blobs

Showing all 3 items

Name	Last modified	Access tier	Blob type	Size	Lease state
output					
iot_data.csv.csv	15/05/2025, 09:31:47	Hot (Inferred)	Block blob	22.07 KiB	Available
predicted_output...	15/05/2025, 19:54:00	Hot (Inferred)	Block blob	21.16 KiB	Available

FIGURE 15- PREDICTIVE DATA STORED IN AZURE BLOB

APPENDIX B: SCREENSHOTS



The Log Analytics agents (MMA.OMS) used to collect logs from virtual machines and servers will no longer be supported from August 31, 2024. Plan to migrate to Azure Monitor Agent before this date. Learn more about migrating to Azure Monitor Agent

Essentials JSON View

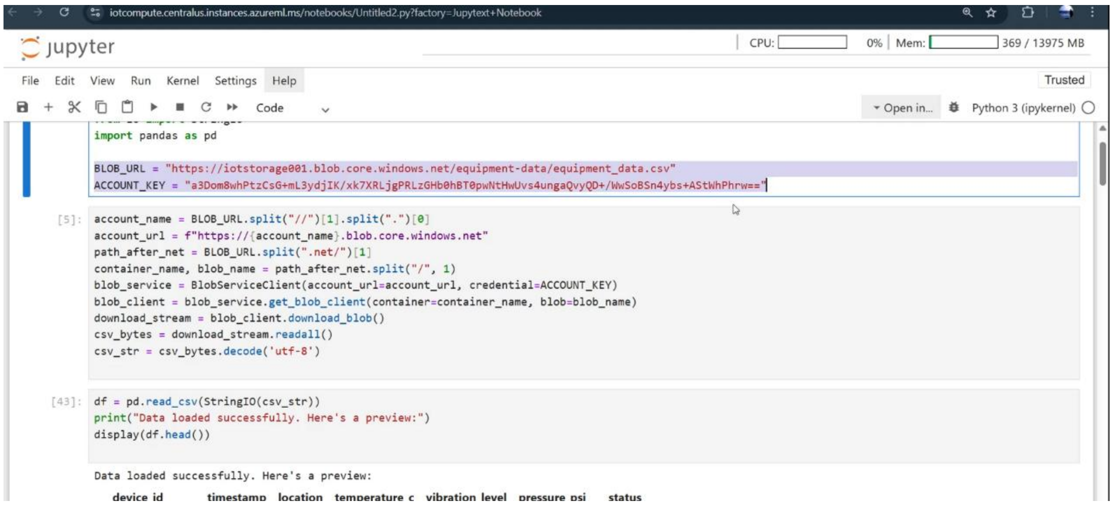
Resource group (move)	: usecase 03
Status	: Active
Location	: East US
Subscription (move)	: Azure for Students
Subscription ID	: 2108ae68-9ce8-4804-b4b6-58b32a3957cb
Tags (edit)	: Add tags
Workspace Name	: Iotexp-01
Workspace ID	: a7662540-7c5e-4517-a060-9d0e35b41f93
Pricing tier	: Pay-as-you-go
Access control mode	: Use resource or workspace permissions
Operational issues	: ✓ OK

Get Started Recommendations

Get started with Log Analytics

Log Analytics collects data from a variety of sources and uses a powerful query language to give you insights into the operation of your applications and resources. Use Azure Monitor to access the complete set of tools for monitoring all of your Azure resources.

FIGURE 16- AZURE LOG ANALYTICS

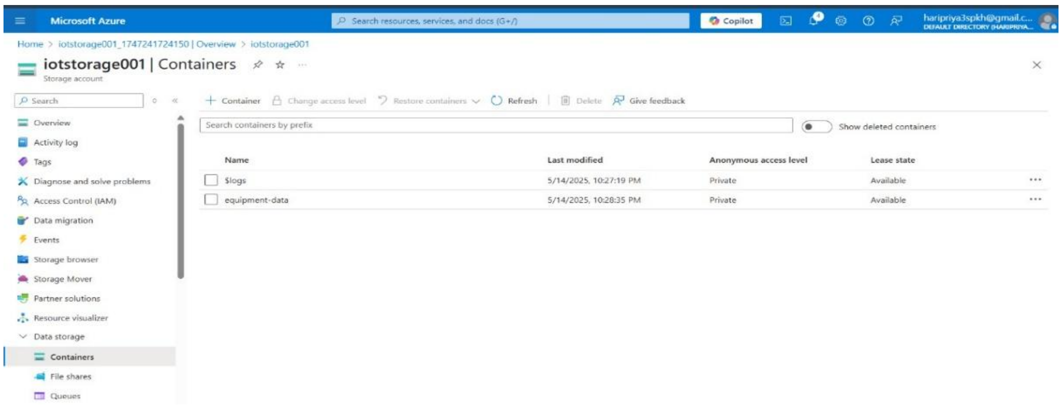


```
import pandas as pd

BLOB_URL = "https://iotstorage001.blob.core.windows.net/equipment-data/equipment_data.csv"
ACCOUNT_KEY = "a3Dom8whPtzCsG+mL3ydjIK/xk7XRLjgPRLzGHb0hBT0pwNtHwUvs4ungaQvyQD+/WwSoBSn4ybs+AStWhPhrw=="
```

```
[5]: account_name = BLOB_URL.split("//")[1].split(".")[0]
     account_url = f"https://{account_name}.blob.core.windows.net"
     path_after_net = BLOB_URL.split(".net/")[1]
     container_name, blob_name = path_after_net.split("/", 1)
     blob_service = BlobServiceClient(account_url=account_url, credential=ACCOUNT_KEY)
     blob_client = blob_service.get_blob_client(container=container_name, blob=blob_name)
     download_stream = blob_client.download_blob()
     csv_bytes = download_stream.readall()
     csv_str = csv_bytes.decode('utf-8')
```

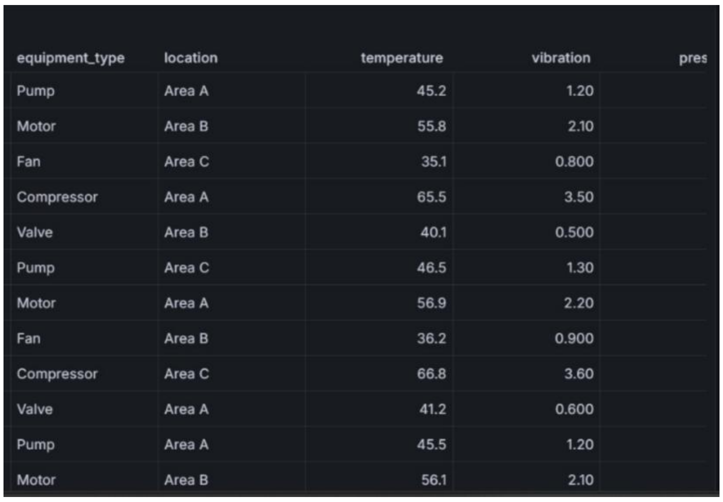
```
[43]: df = pd.read_csv(StringIO(csv_str))
      print("Data loaded successfully. Here's a preview:")
      display(df.head())

      Data loaded successfully. Here's a preview:
         device_id     timestamp  location  temperature_c  vibration_level  pressure_psi  status
```

FIGURE 17- DATASET LOADING IN AZURE ML – JUPYTER



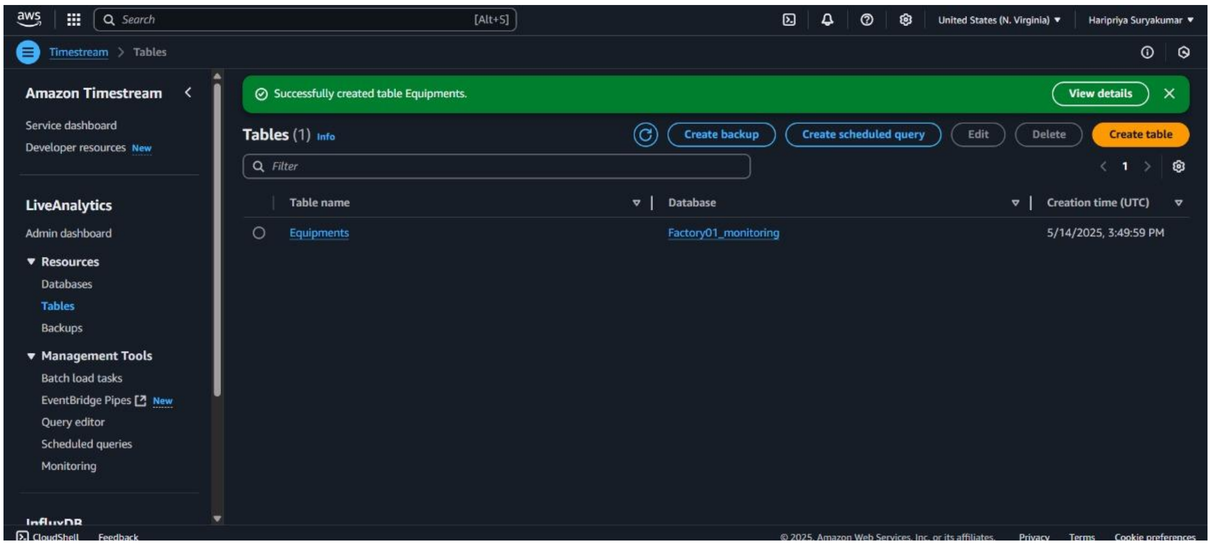
Name	Last modified	Anonymous access level	Lease state
$logs	5/14/2025, 10:27:19 PM	Private	Available
equipment-data	5/14/2025, 10:28:35 PM	Private	Available

FIGURE 18- CONTAINER – AZURE BLOB STORAGE



equipment_type	location	temperature	vibration	pres
Pump	Area A	45.2	1.20	
Motor	Area B	55.8	2.10	
Fan	Area C	35.1	0.800	
Compressor	Area A	65.5	3.50	
Valve	Area B	40.1	0.500	
Pump	Area C	46.5	1.30	
Motor	Area A	56.9	2.20	
Fan	Area B	36.2	0.900	
Compressor	Area C	66.8	3.60	
Valve	Area A	41.2	0.600	
Pump	Area A	45.5	1.20	
Motor	Area B	56.1	2.10	

FIGURE 19- TABLE VIEW – GRAFANA



Successfully created table Equipments.

Tables (1) Info

Table name	Database	Creation time (UTC)
Equipments	Factory01_monitoring	5/14/2025, 3:49:59 PM

FIGURE 20- AWS TIMESTREAM CONFIGURATION

12. URL

https://github.com/Haripriya-Suryakumar/Equipments_monitoring