Assignment Project

INFT4104 Selected topics in IT

By: Noor Alqusei 60098352 & Rawan Alqusei 60098355

Instructor Name: Dr. Abderrahmane Maaradji

Table of Contents

Chapter 1: Introduction

Overview of Air Quality Index (AQI)

Computation of the AQI requires an air pollutant concentration over a specified averaging period, obtained from an air monitor or model. Taken together, concentration and time represent the dose of the air pollutant. Health effects corresponding to a given dose are established by epidemiological research. Air pollutants vary in potency, and the function used to convert from air pollutant concentration to AQI varies by pollutant. Its air quality index values are typically grouped into ranges. Each range is assigned a descriptor, a color code, and a standardized public health advisory.

The AQI can increase due to an increase of air emissions. For example, during rush hour traffic or when there is an upwind forest fire or from a lack of dilution of air pollutants. Stagnant air, often caused by an anticyclone, temperature inversion, or low wind speeds lets air pollution remain in a local area, leading to high concentrations of pollutants, chemical reactions between air contaminants and hazy conditions.

On a day when the AQI is predicted to be elevated due to fine particle pollution, an agency or public health organization might:

- advise sensitive groups, such as the elderly, children and those with respiratory or cardiovascular problems or suffering from diseases, to avoid outdoor exertion.
- declare an "action day" to encourage voluntary measures to curtail air emissions, such as using public transportation.
- recommend the use of masks outdoors and air purifiers indoors to prevent fine particles from entering the lungs.

During a period of very poor air quality, such as an air pollution episode, when the AQI indicates that acute exposure may cause significant harm to the public health, agencies may invoke emergency plans that allow them to order major emitters (such as coal burning industries) to curtail emissions until the hazardous conditions abate.

Most air contaminants do not have an associated AQI. Many countries monitor ground-level ozone, particulates, sulfur dioxide, carbon monoxide and nitrogen dioxide, and calculate air quality indices for these pollutants.

The definition of the AQI in a particular nation reflects the discourse surrounding the development of national air quality standards in that nation. A website allowing government agencies anywhere in the world to submit their real-time air monitoring data for display using a common definition of the air quality index has recently become available. *(Wikipedia contributors, 2025)*

Machine Learning for AQI Forecasting

To maintain ambient air quality, regular monitoring and forecasting of air pollution is necessary. For that purpose, machine learning has emerged as a promising technique for predicting the Air Quality Index (AQI) compared to conventional methods. They apply the AQI to the city of Visakhapatnam, Andhra Pradesh, India, focusing on 12 contaminants and 10 meteorological parameters from July 2017 to September 2022. For this purpose, they employed several machine learning models, including

LightGBM, Random Forest, Catboost, Adaboost, and XGBoost. The results show that the Catboost model outperformed other models with an R2 correlation coefficient of 0.9998, a mean absolute error (MAE) of 0.60, a mean square error (MSE) of 0.58, and a root mean square error (RMSE) of 0.76. The Adaboost model had the least effective prediction with an R2 correlation coefficient of 0.9753. In summary, machine learning is a promising technique for predicting AQI with Catboost being the best-performing model for AQI prediction. Moreover, by leveraging historical data and machine learning algorithms enables accurate predictions of future urban air quality levels on a global scale. *(Ravindiran et al., 2023)*

Anomaly Detection in AQI Data

The anomaly detection system is successfully able to detect anomalies in the readings of gas sensors, temperature and humidity sensors in real-time. The system plots a real-time graph of all the readings which changes every twenty seconds. The working of the entire system is tested by intentionally creating the conditions in which anomalies are likely to occur. Anomalies are successfully detected and can be removed from analysis in real-time. This would prevent the triggering of false alarms in a factory or any other air pollution monitoring system. Filtering of data is essential before the analysis of any sensor data, but the data generated by the anomaly detection system can be directly used for analysis without filtering. The concept of detecting anomalies in real- time can be further extended to other real-time monitoring systems as well. *(Ijraset, n.d.)*

Chapter 2: System Design and Implementation

Hardware Setup

The hardware configuration for this project includes an Arduino Uno microcontroller, an MQ-135 gas sensor, and a 16x2 I2C LCD display. The MQ-135 sensor is designed to measure indoor air quality by detecting pollutants such as ammonia (NH3), nitrogen oxides (NOx), benzene, smoke, and carbon dioxide (CO2). These values give an estimate of the Air Quality Index (AQI).

To provide real-time visibility, a 16x2 LCD display was connected to the Arduino using I2C communication. The Arduino collects air quality readings from the MQ-135 sensor and displays them on the LCD screen. At the same time, the sensor values are printed to the Serial Monitor via Serial.println() for further processing and cloud transmission. The delay(5000) function ensures that data is updated every 5 seconds.

This setup enables both local visualization on the LCD and digital output for integration with Azure IoT Central.

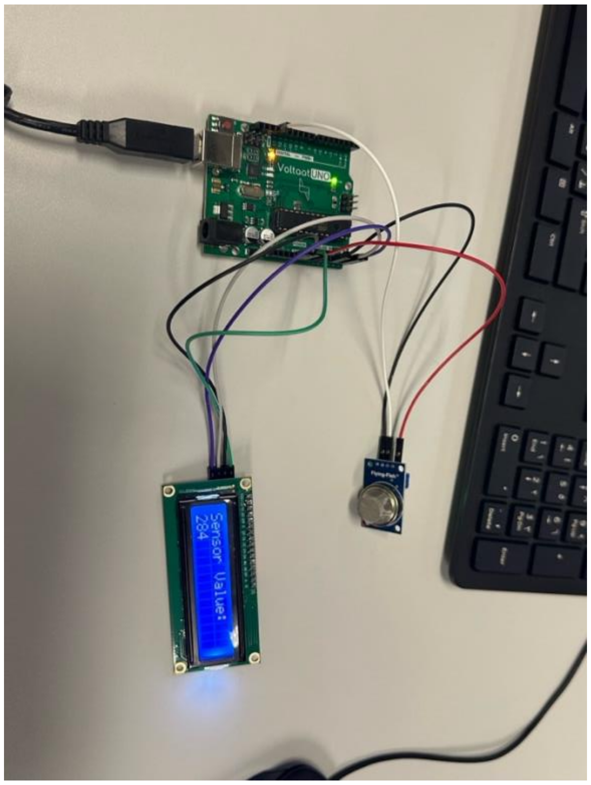


Figure 2.1 – Fully Assembled Hardware Setup

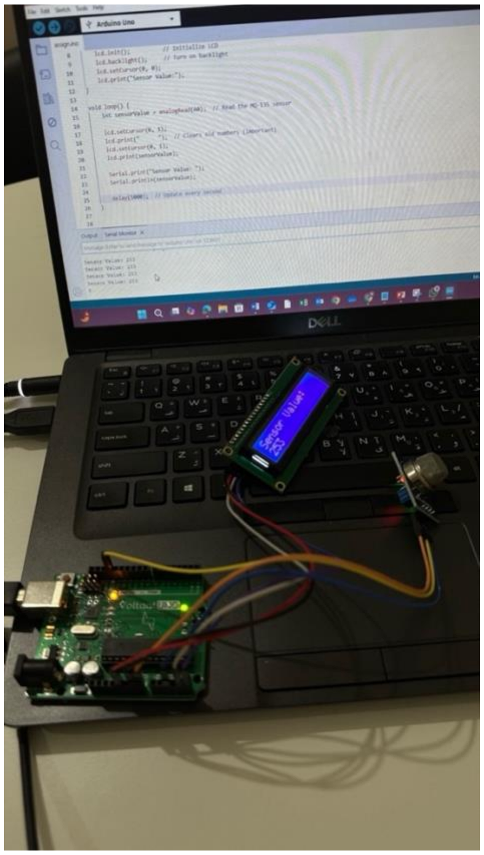
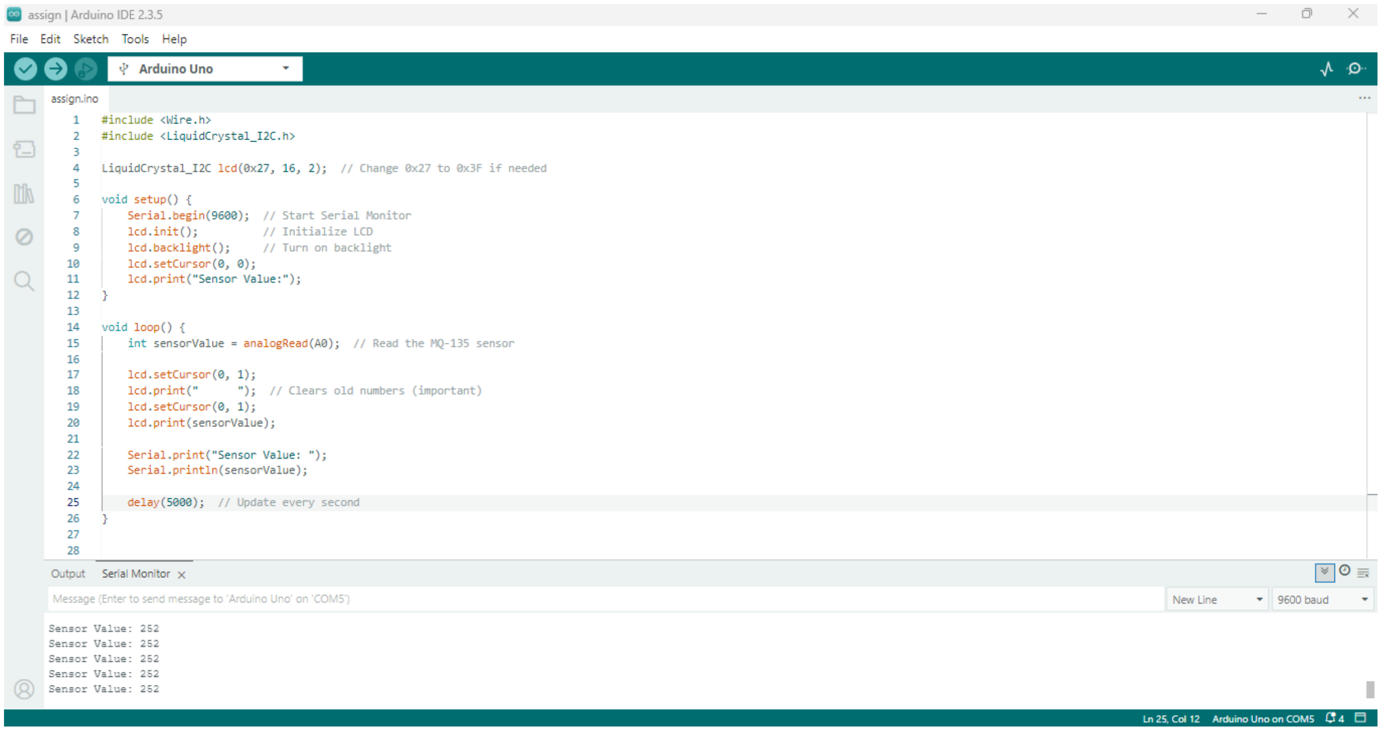


Figure 2.2 – Hardware Running with Code on Laptop


assign | Arduino IDE 2.3.5
File Edit Sketch Tools Help
Arduino Uno
assign.ino
Output
Serial Monitor
Message (Enter to send message to 'Arduino Uno' on 'COM5')
New Line
9600 baud
Ln 25, Col 12
Arduino Uno on COM5

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);  // Change 0x27 to 0x3F if needed

void setup() {
    Serial.begin(9600);  // Start Serial Monitor
    lcd.init();          // Initialize LCD
    lcd.backlight();     // Turn on backlight
    lcd.setCursor(0, 0);
    lcd.print("Sensor Value:");
}

void loop() {
    int sensorValue = analogRead(A0);  // Read the MQ-135 sensor

    lcd.setCursor(0, 1);
    lcd.print("      ");  // Clears old numbers (important)
    lcd.setCursor(0, 1);
    lcd.print(sensorValue);

    Serial.print("Sensor Value: ");
    Serial.println(sensorValue);

    delay(5000);  // Update every second
}
```

```
Sensor Value: 252
Sensor Value: 252
Sensor Value: 252
Sensor Value: 252
Sensor Value: 252
```

Figure 2.3 – Arduino IDE Showing Sensor Code and Output

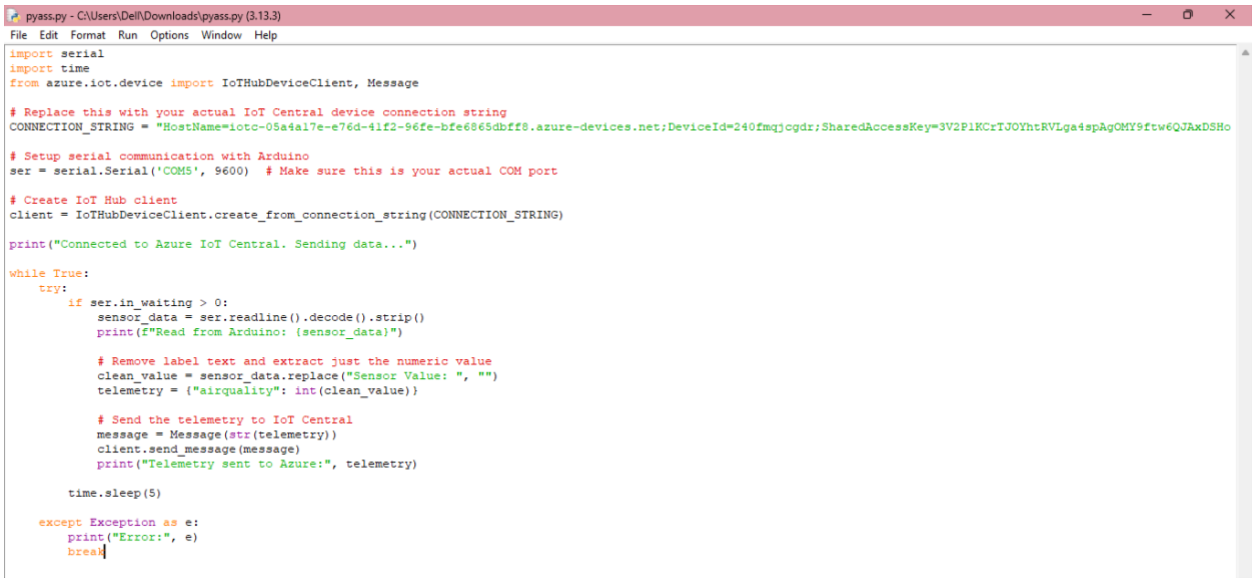
Software Setup

In this step, the Arduino was connected to Azure IoT Central and successfully transmitted real-time sensor data using Python. The MQ-135 air quality sensor was connected to the Arduino, and the sensor readings were sent via serial communication to a Python script. The script extracted the numeric air quality values, then packaged and sent them to the registered IoT Central device.

To achieve this, the following key tasks were performed:

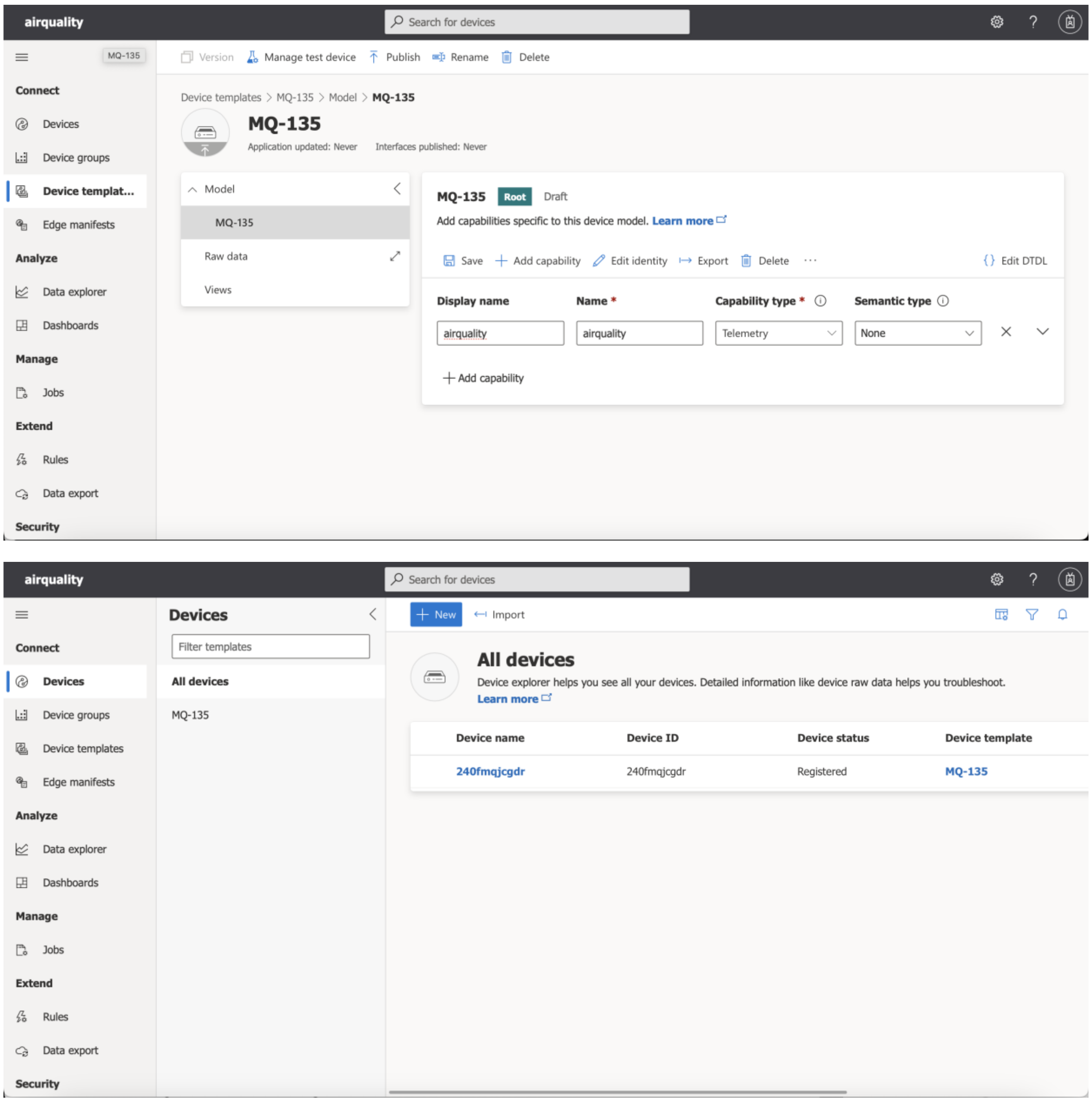
- A device template (MQ-135) was created with a telemetry capability named airquality.
- A real device was registered in Azure IoT Central using this template.
- A Python script was written using the azure.iot.device library to:
    - Establish a connection using the device's connection string.
    - Read data from the Arduino serial port.
    - Extract the numeric value from the incoming data (e.g., "Sensor Value: 255" → 255).
    - Send it as telemetry in JSON format to the connected device.

Once the Python script was executed, the data began flowing into Azure IoT Central, and the live telemetry appeared in the Raw data section of the registered device.


pyass.py - C:\Users\Dell\Downloads\pyass.py (3.13.3)
File Edit Format Run Options Window Help

```
import serial
import time
from azure.iot.device import IoTHubDeviceClient, Message

# Replace this with your actual IoT Central device connection string
CONNECTION_STRING = "HostName=iotc-05a4a17e-e76d-41f2-96fe-bfe6865dbff8.azure-devices.net;DeviceId=240fmqjcgdr;SharedAccessKey=3V2P1KCrTJOYhtRVLga4spAgOMY9ftw6QJAxDSHo

# Setup serial communication with Arduino
ser = serial.Serial('COM5', 9600)  # Make sure this is your actual COM port

# Create IoT Hub client
client = IoTHubDeviceClient.create_from_connection_string(CONNECTION_STRING)

print("Connected to Azure IoT Central. Sending data...")

while True:
    try:
        if ser.in_waiting > 0:
            sensor_data = ser.readline().decode().strip()
            print(f"Read from Arduino: {sensor_data}")

            # Remove label text and extract just the numeric value
            clean_value = sensor_data.replace("Sensor Value: ", "")
            telemetry = {"airquality": int(clean_value)}

            # Send the telemetry to IoT Central
            message = Message(str(telemetry))
            client.send_message(message)
            print("Telemetry sent to Azure:", telemetry)

        time.sleep(5)

    except Exception as e:
        print("Error:", e)
        break
```

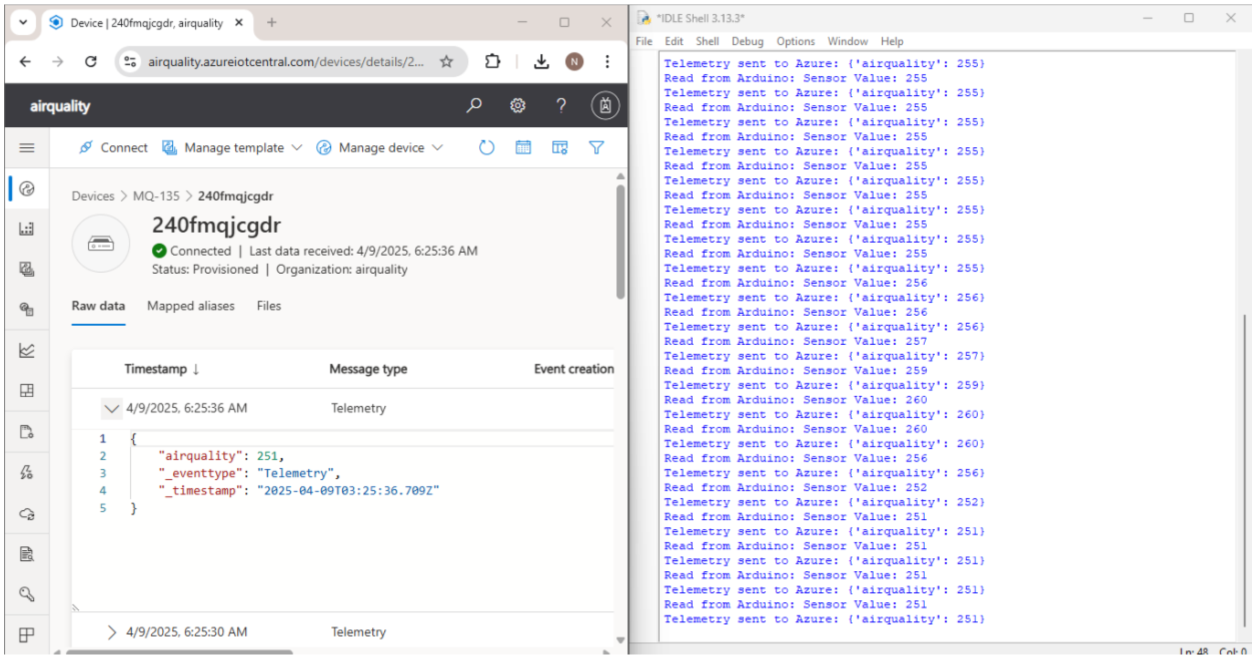
Figure 2.4 – Python script transmitting data to Azure IoT Central



Display name	Name *	Capability type *	Semantic type
airquality	airquality	Telemetry	None

Device name	Device ID	Device status	Device template
240fmqjcgdr	240fmqjcgdr	Registered	MQ-135

Figure 2.5 – Azure IoT Central Device Template and Registration



Devices > MQ-135 > 240fmqjcgdr

240fmqjcgdr

Connected | Last data received: 4/9/2025, 6:25:36 AM

Status: Provisioned | Organization: airquality

Timestamp ↓	Message type	Event creation
4/9/2025, 6:25:36 AM	Telemetry	
4/9/2025, 6:25:30 AM	Telemetry	

```
{
    "airquality": 251,
    "_eventtype": "Telemetry",
    "_timestamp": "2025-04-09T03:25:36.709Z"
}
```

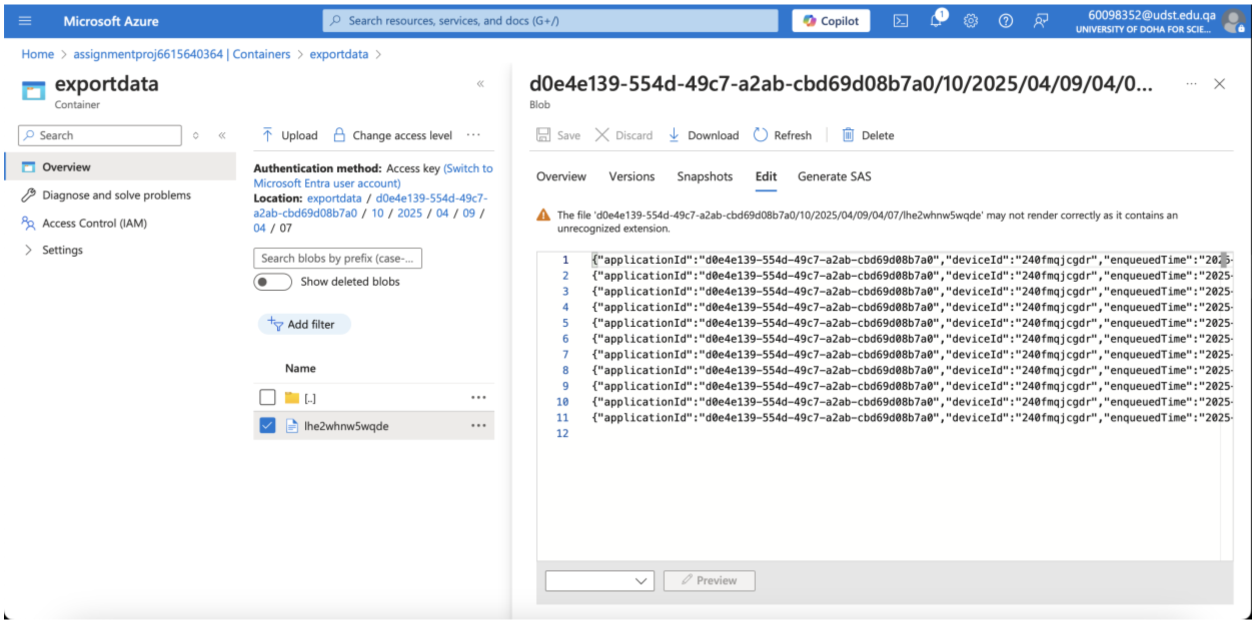
```
Telemetry sent to Azure: {'airquality': 255}
Read from Arduino: Sensor Value: 255
Telemetry sent to Azure: {'airquality': 255}
Read from Arduino: Sensor Value: 255
Telemetry sent to Azure: {'airquality': 255}
Read from Arduino: Sensor Value: 255
Telemetry sent to Azure: {'airquality': 255}
Read from Arduino: Sensor Value: 255
Telemetry sent to Azure: {'airquality': 255}
Read from Arduino: Sensor Value: 255
Telemetry sent to Azure: {'airquality': 255}
Read from Arduino: Sensor Value: 255
Telemetry sent to Azure: {'airquality': 255}
Read from Arduino: Sensor Value: 255
Telemetry sent to Azure: {'airquality': 255}
Read from Arduino: Sensor Value: 256
Telemetry sent to Azure: {'airquality': 256}
Read from Arduino: Sensor Value: 256
Telemetry sent to Azure: {'airquality': 256}
Read from Arduino: Sensor Value: 257
Telemetry sent to Azure: {'airquality': 257}
Read from Arduino: Sensor Value: 259
Telemetry sent to Azure: {'airquality': 259}
Read from Arduino: Sensor Value: 260
Telemetry sent to Azure: {'airquality': 260}
Read from Arduino: Sensor Value: 260
Telemetry sent to Azure: {'airquality': 260}
Read from Arduino: Sensor Value: 256
Telemetry sent to Azure: {'airquality': 256}
Read from Arduino: Sensor Value: 252
Telemetry sent to Azure: {'airquality': 252}
Read from Arduino: Sensor Value: 251
Telemetry sent to Azure: {'airquality': 251}
Read from Arduino: Sensor Value: 251
Telemetry sent to Azure: {'airquality': 251}
Read from Arduino: Sensor Value: 251
Telemetry sent to Azure: {'airquality': 251}
Read from Arduino: Sensor Value: 251
Telemetry sent to Azure: {'airquality': 251}
```

Figure 2.6 – Real-time AQI telemetry received in Azure IoT Central

Azure Blob Storage (Optional Feature)

As a validation and demonstration step, telemetry data from IoT Central was exported to Azure Blob Storage:

- Destination: Azure Blob Container named *exportdata*
- Data transformation was applied to export messages in structured JSON format
- Logs confirmed successful message storage with timestamps and AQI values


Home > assignmentproj6615640364 | Containers > exportdata >

exportdata
Container

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: exportdata / d0e4e139-554d-49c7-a2ab-cbd69d08b7a0 / 10 / 2025 / 04 / 09 / 04 / 07

Name
[..]
lhe2whnw5wqde

d0e4e139-554d-49c7-a2ab-cbd69d08b7a0/10/2025/04/09/04/0...
Blob

The file 'd0e4e139-554d-49c7-a2ab-cbd69d08b7a0/10/2025/04/09/04/07/lhe2whnw5wqde' may not render correctly as it contains an unrecognized extension.

```
{"applicationId":"d0e4e139-554d-49c7-a2ab-cbd69d08b7a0","deviceId":"240fmqjcgdr","enqueuedTime":"2025-
{"applicationId":"d0e4e139-554d-49c7-a2ab-cbd69d08b7a0","deviceId":"240fmqjcgdr","enqueuedTime":"2025-
{"applicationId":"d0e4e139-554d-49c7-a2ab-cbd69d08b7a0","deviceId":"240fmqjcgdr","enqueuedTime":"2025-
{"applicationId":"d0e4e139-554d-49c7-a2ab-cbd69d08b7a0","deviceId":"240fmqjcgdr","enqueuedTime":"2025-
{"applicationId":"d0e4e139-554d-49c7-a2ab-cbd69d08b7a0","deviceId":"240fmqjcgdr","enqueuedTime":"2025-
{"applicationId":"d0e4e139-554d-49c7-a2ab-cbd69d08b7a0","deviceId":"240fmqjcgdr","enqueuedTime":"2025-
{"applicationId":"d0e4e139-554d-49c7-a2ab-cbd69d08b7a0","deviceId":"240fmqjcgdr","enqueuedTime":"2025-
{"applicationId":"d0e4e139-554d-49c7-a2ab-cbd69d08b7a0","deviceId":"240fmqjcgdr","enqueuedTime":"2025-
{"applicationId":"d0e4e139-554d-49c7-a2ab-cbd69d08b7a0","deviceId":"240fmqjcgdr","enqueuedTime":"2025-
{"applicationId":"d0e4e139-554d-49c7-a2ab-cbd69d08b7a0","deviceId":"240fmqjcgdr","enqueuedTime":"2025-
{"applicationId":"d0e4e139-554d-49c7-a2ab-cbd69d08b7a0","deviceId":"240fmqjcgdr","enqueuedTime":"2025-
```

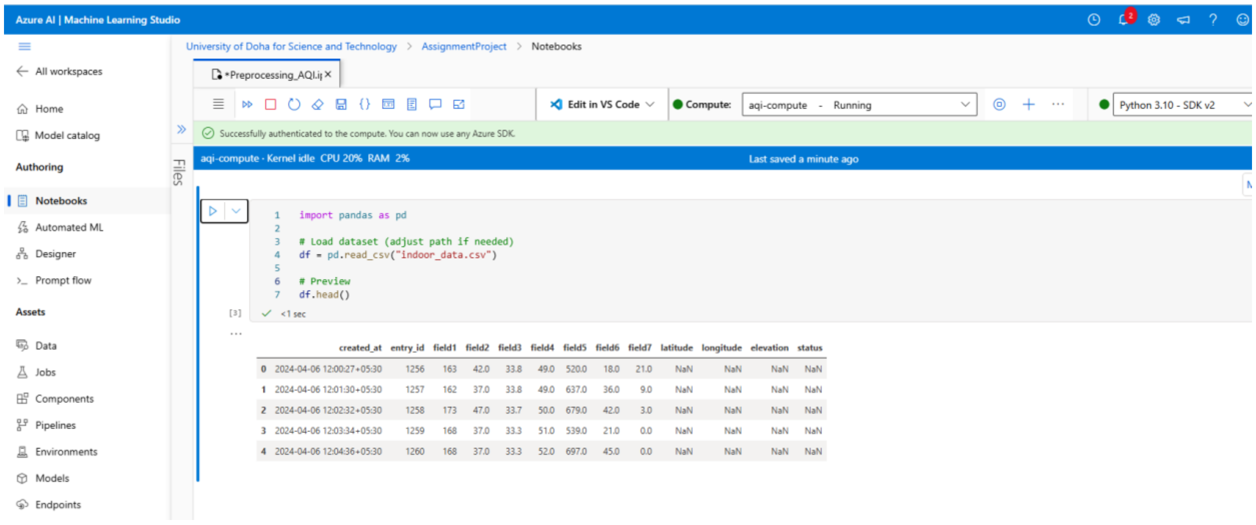
Figure 2.7 – Telemetry data exported to Azure Blob Storage

This workflow enabled a complete and functional IoT system for continuous air quality monitoring, data transmission, and cloud-based analysis.

Chapter 3: Dataset and Preprocessing

<u>Dataset Overview</u>

The dataset used for model training and testing was the Indoor Air Quality Dataset available on Kaggle. It contains 1333 rows and 13 columns of environmental sensor data.



```
Successfully authenticated to the compute. You can now use any Azure SDK.
aqi-compute · Kernel idle  CPU 20%  RAM  2%                Last saved a minute ago
```

```python
import pandas as pd

# Load dataset (adjust path if needed)
df = pd.read_csv("indoor_data.csv")

# Preview
df.head()
```

```
[3]  ✓ <1 sec

                  created_at  entry_id  field1  field2  field3  field4  field5  field6  field7  latitude  longitude  elevation  status
0  2024-04-06 12:00:27+05:30      1256     163    42.0    33.8    49.0   520.0    18.0    21.0       NaN        NaN        NaN     NaN
1  2024-04-06 12:01:30+05:30      1257     162    37.0    33.8    49.0   637.0    36.0     9.0       NaN        NaN        NaN     NaN
2  2024-04-06 12:02:32+05:30      1258     173    47.0    33.7    50.0   679.0    42.0     3.0       NaN        NaN        NaN     NaN
3  2024-04-06 12:03:34+05:30      1259     168    37.0    33.3    51.0   539.0    21.0     0.0       NaN        NaN        NaN     NaN
4  2024-04-06 12:04:36+05:30      1260     168    37.0    33.3    52.0   697.0    45.0     0.0       NaN        NaN        NaN     NaN
```

Figure 3.1 – Raw dataset preview in Azure ML Studio

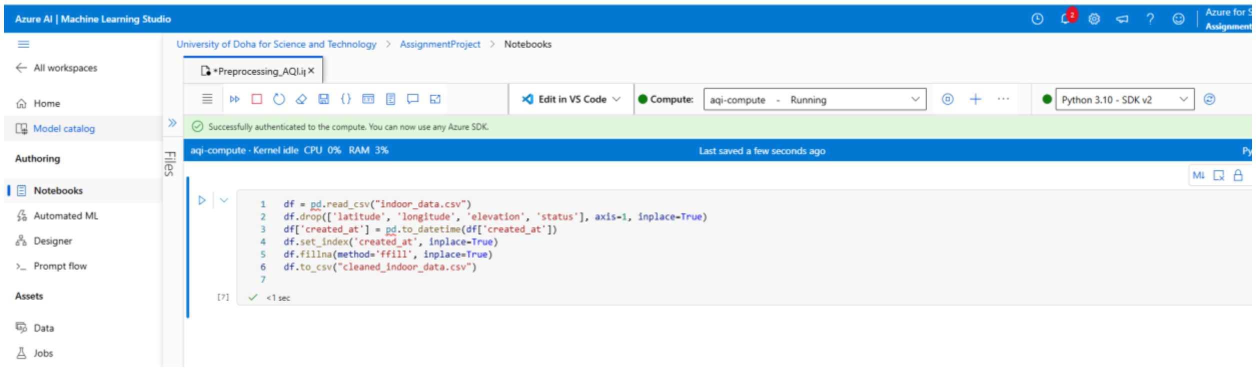
Key Features:

- **created_at:** Timestamp of data collection
- **entry_id:** Unique identifier for each record
- **field1 to field7:** Sensor readings including temperature, humidity, CO2, TVOC, PM2.5, PM10, and AQI
- **latitude, longitude, elevation, status:** Metadata columns that were found to be completely empty

The data types were primarily float64, making the dataset suitable for machine learning applications.

Preprocessing Steps:

1. **Importing Dataset in Azure ML Studio:** The dataset was uploaded and loaded into a Python notebook.
2. **Column Removal:** Non-informative columns (latitude, longitude, elevation, status) were dropped.
3. **Datetime Conversion:** The created_at column was converted into datetime format.
4. **Index Setting:** The created_at column was set as the index to enable time series forecasting.
5. **Missing Value Handling:** Applied forward-fill using fillna(method='ffill') to maintain temporal continuity.
6. **Output File:** Saved the cleaned dataset as cleaned_indoor_data.csv for use in Azure ML pipelines.



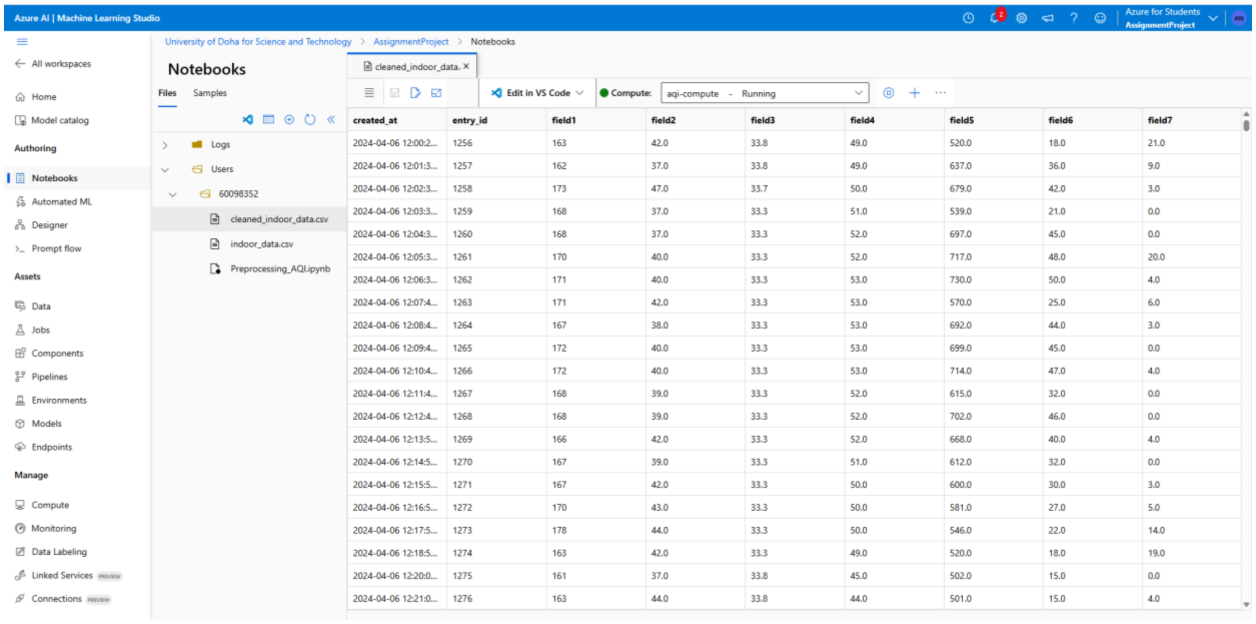
Successfully authenticated to the compute. You can now use any Azure SDK.

aqi-compute · Kernel idle CPU 0% RAM 3% Last saved a few seconds ago

```python
df = pd.read_csv("indoor_data.csv")
df.drop(['latitude', 'longitude', 'elevation', 'status'], axis=1, inplace=True)
df['created_at'] = pd.to_datetime(df['created_at'])
df.set_index('created_at', inplace=True)
df.fillna(method='ffill', inplace=True)
df.to_csv("cleaned_indoor_data.csv")
```

[7] ✓ <1 sec

Figure 3.2 – Python notebook for preprocessing: dropping columns and handling missing values



created_at	entry_id	field1	field2	field3	field4	field5	field6	field7
2024-04-06 12:00:2...	1256	163	42.0	33.8	49.0	520.0	18.0	21.0
2024-04-06 12:01:3...	1257	162	37.0	33.8	49.0	637.0	36.0	9.0
2024-04-06 12:02:3...	1258	173	47.0	33.7	50.0	679.0	42.0	3.0
2024-04-06 12:03:3...	1259	168	37.0	33.3	51.0	539.0	21.0	0.0
2024-04-06 12:04:3...	1260	168	37.0	33.3	52.0	697.0	45.0	0.0
2024-04-06 12:05:3...	1261	170	40.0	33.3	52.0	717.0	48.0	20.0
2024-04-06 12:06:3...	1262	171	40.0	33.3	53.0	730.0	50.0	4.0
2024-04-06 12:07:4...	1263	171	42.0	33.3	53.0	570.0	25.0	6.0
2024-04-06 12:08:4...	1264	167	38.0	33.3	53.0	692.0	44.0	3.0
2024-04-06 12:09:4...	1265	172	40.0	33.3	53.0	699.0	45.0	0.0
2024-04-06 12:10:4...	1266	172	40.0	33.3	53.0	714.0	47.0	4.0
2024-04-06 12:11:4...	1267	168	39.0	33.3	52.0	615.0	32.0	0.0
2024-04-06 12:12:4...	1268	168	39.0	33.3	52.0	702.0	46.0	0.0
2024-04-06 12:13:5...	1269	166	42.0	33.3	52.0	668.0	40.0	4.0
2024-04-06 12:14:5...	1270	167	39.0	33.3	51.0	612.0	32.0	0.0
2024-04-06 12:15:5...	1271	167	42.0	33.3	50.0	600.0	30.0	3.0
2024-04-06 12:16:5...	1272	170	43.0	33.3	50.0	581.0	27.0	5.0
2024-04-06 12:17:5...	1273	178	44.0	33.3	50.0	546.0	22.0	14.0
2024-04-06 12:18:5...	1274	163	42.0	33.3	49.0	520.0	18.0	19.0
2024-04-06 12:20:0...	1275	161	37.0	33.8	45.0	502.0	15.0	0.0
2024-04-06 12:21:0...	1276	163	44.0	33.8	44.0	501.0	15.0	4.0

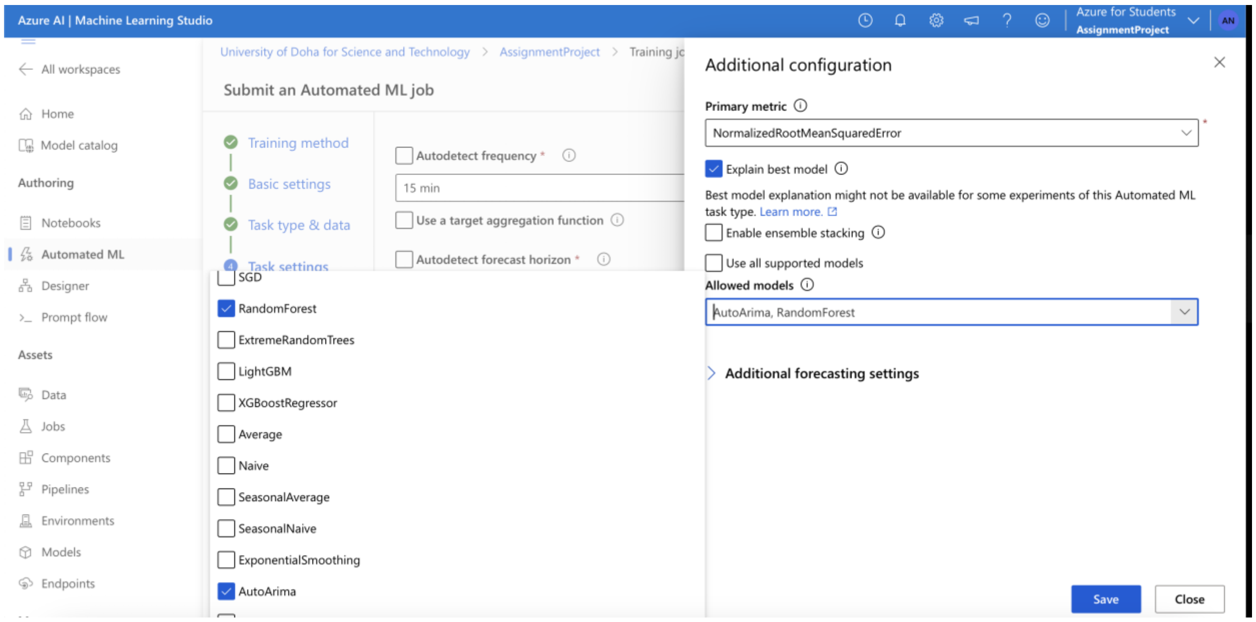
Figure 3.3 – Cleaned dataset ready for model training

Chapter 4: Machine Learning for Prediction and Anomaly Detection

AQI Prediction Using AutoARIMA

Azure AutoML was used to train a Time Series Forecasting model on the cleaned dataset.

- **Task Type:** Time Series Forecasting
- **Target Column:** field2 (assumed AQI)
- **Time Column:** created_at
- **Model Chosen**: AutoARIMA (best NRMSE)



☐ Autodetect frequency *

15 min

☐ Use a target aggregation function

☐ Autodetect forecast horizon *

☐ SGD
☑ RandomForest
☐ ExtremeRandomTrees
☐ LightGBM
☐ XGBoostRegressor
☐ Average
☐ Naive
☐ SeasonalAverage
☐ SeasonalNaive
☐ ExponentialSmoothing
☑ AutoArima

Additional configuration

Primary metric: NormalizedRootMeanSquaredError *

☑ Explain best model

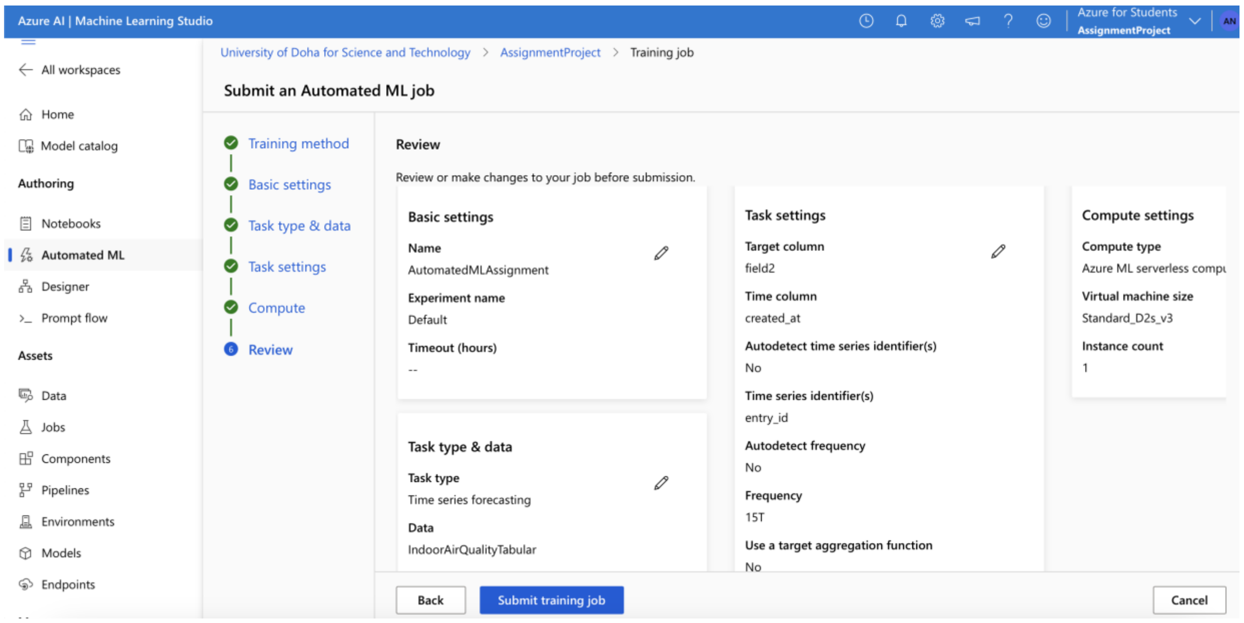
Best model explanation might not be available for some experiments of this Automated ML task type. Learn more.

☐ Enable ensemble stacking

☐ Use all supported models

Allowed models: AutoArima, RandomForest

> Additional forecasting settings



Submit an Automated ML job

Review

Review or make changes to your job before submission.

Basic settings

Name
AutomatedMLAssignment

Experiment name
Default

Timeout (hours)
--

Task type & data

Task type
Time series forecasting

Data
IndoorAirQualityTabular

Task settings

Target column
field2

Time column
created_at

Autodetect time series identifier(s)
No

Time series identifier(s)
entry_id

Autodetect frequency
No

Frequency
15T

Use a target aggregation function
No

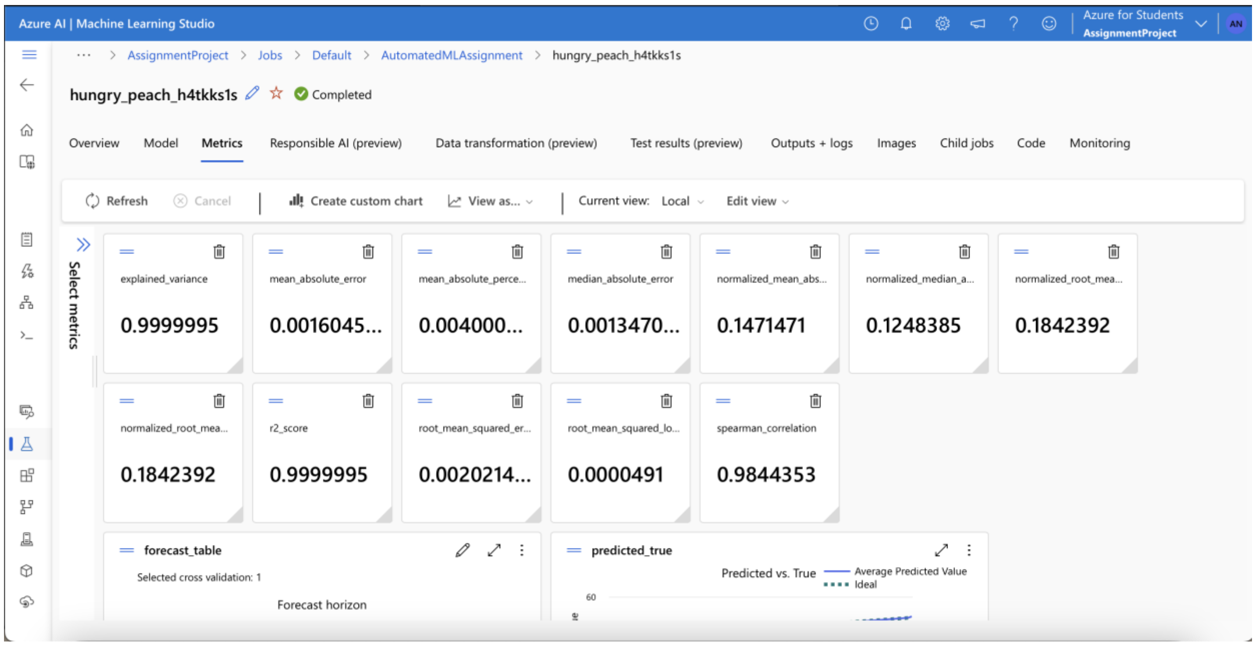
Compute settings

Compute type
Azure ML serverless compu

Virtual machine size
Standard_D2s_v3

Instance count
1

Figure 4.1 – Azure AutoML Time Series Forecasting job configuration

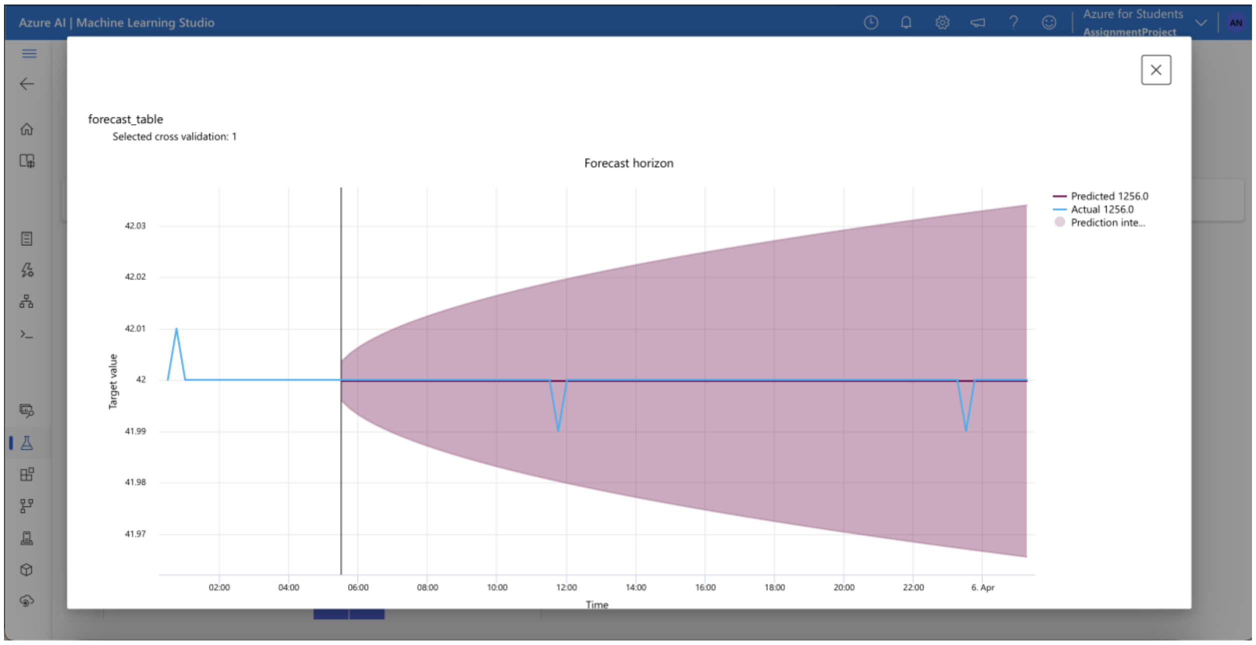


hungry_peach_h4tkks1s Completed

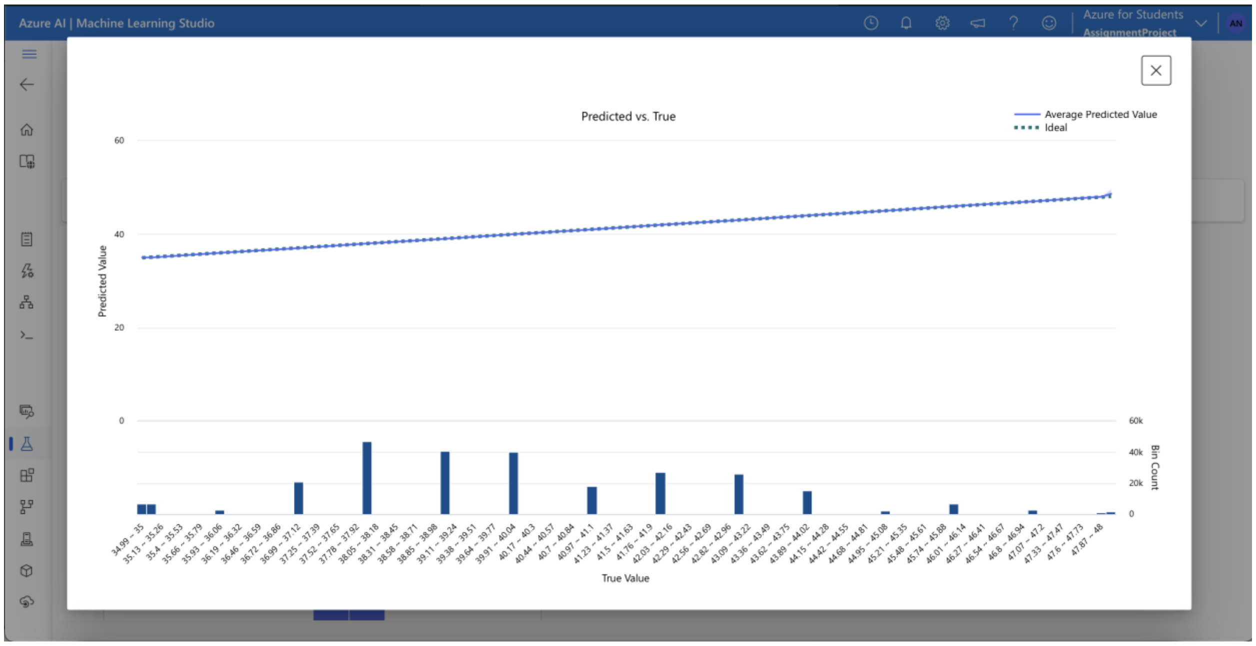
explained_variance	0.9999995
mean_absolute_error	0.0016045...
mean_absolute_perce...	0.004000...
median_absolute_error	0.0013470...
normalized_mean_abs...	0.1471471
normalized_median_a...	0.1248385
normalized_root_mea...	0.1842392
normalized_root_mea...	0.1842392
r2_score	0.9999995
root_mean_squared_er...	0.0020214...
root_mean_squared_lo...	0.0000491
spearman_correlation	0.9844353

forecast_table
Selected cross validation: 1
Forecast horizon

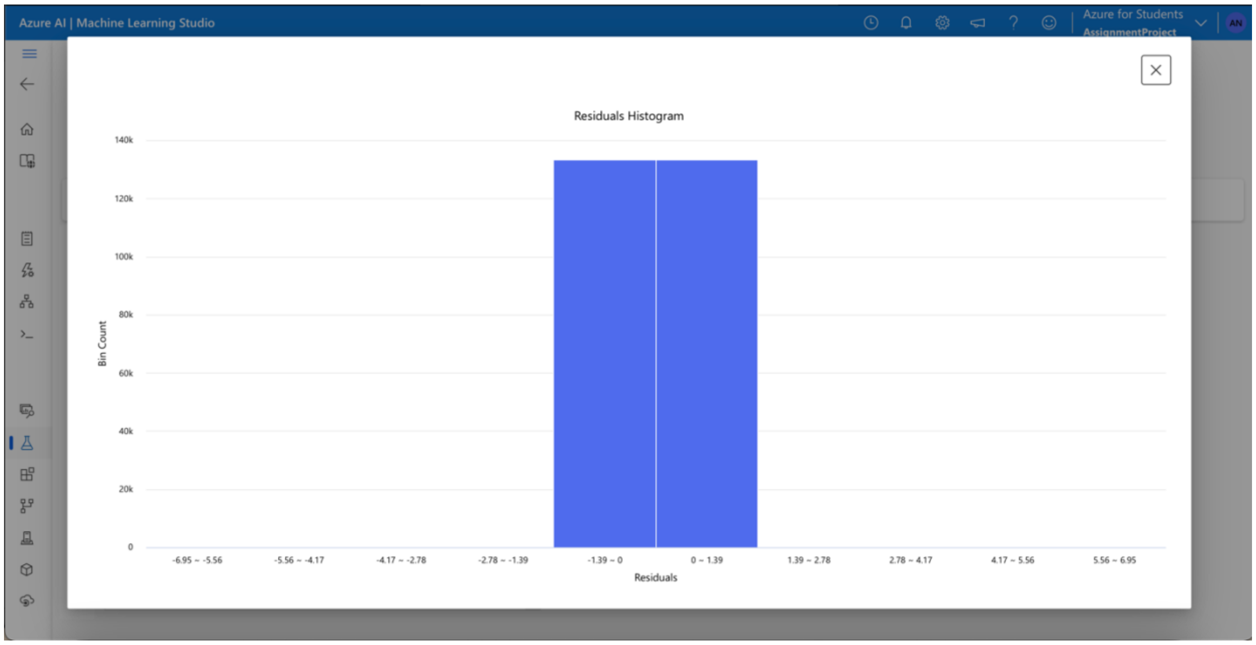
predicted_true
Predicted vs. True
Average Predicted Value
Ideal
60



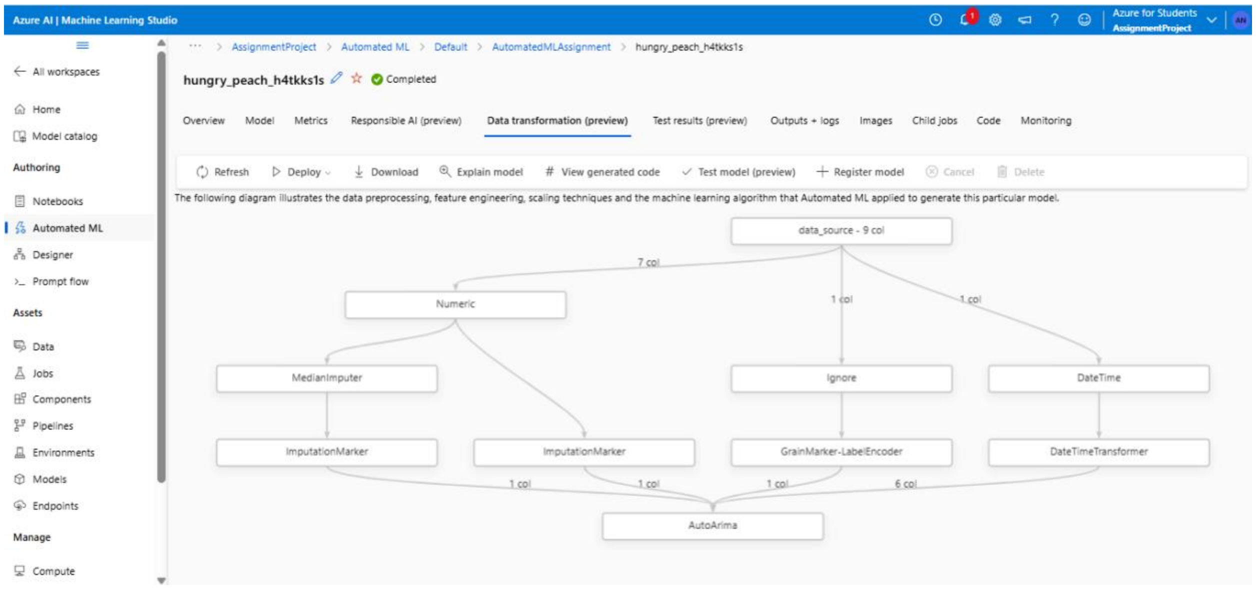
This chart shows the model's forecast of future air quality values (purple line), compared to the actual values (blue line). The shaded purple area shows the confidence range, meaning the model is quite sure the true values will fall inside it.



The predicted values (blue line) closely match the ideal line (dotted), showing that the model's predictions are very close to actual results.



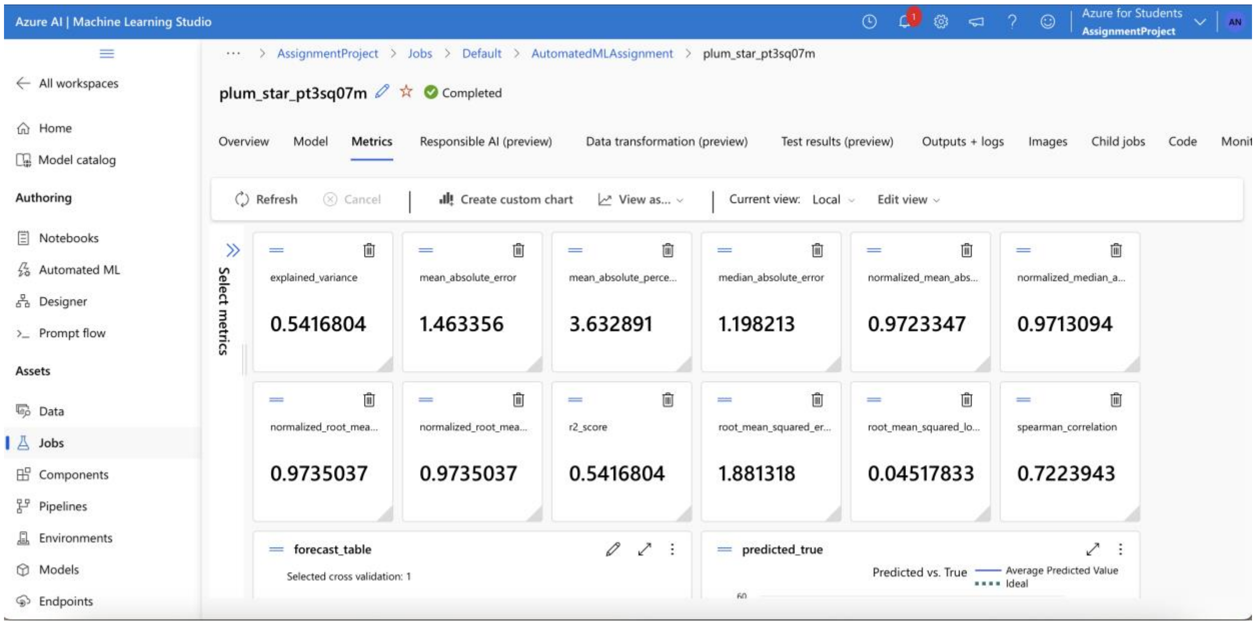
This histogram shows the distribution of prediction errors (residuals). Most errors are small and centered around zero, meaning the model made very few big mistakes.



The following diagram illustrates the data preprocessing, feature engineering, scaling techniques and the machine learning algorithm that Automated ML applied to generate this particular model.

Figure 4.2 – AutoARIMA Model Evaluation Summary

<u>AutoARIMA model performance including metrics (R², RMSE, MAE), forecast chart, residual histogram, predicted values comparison, and preprocessing steps.</u>

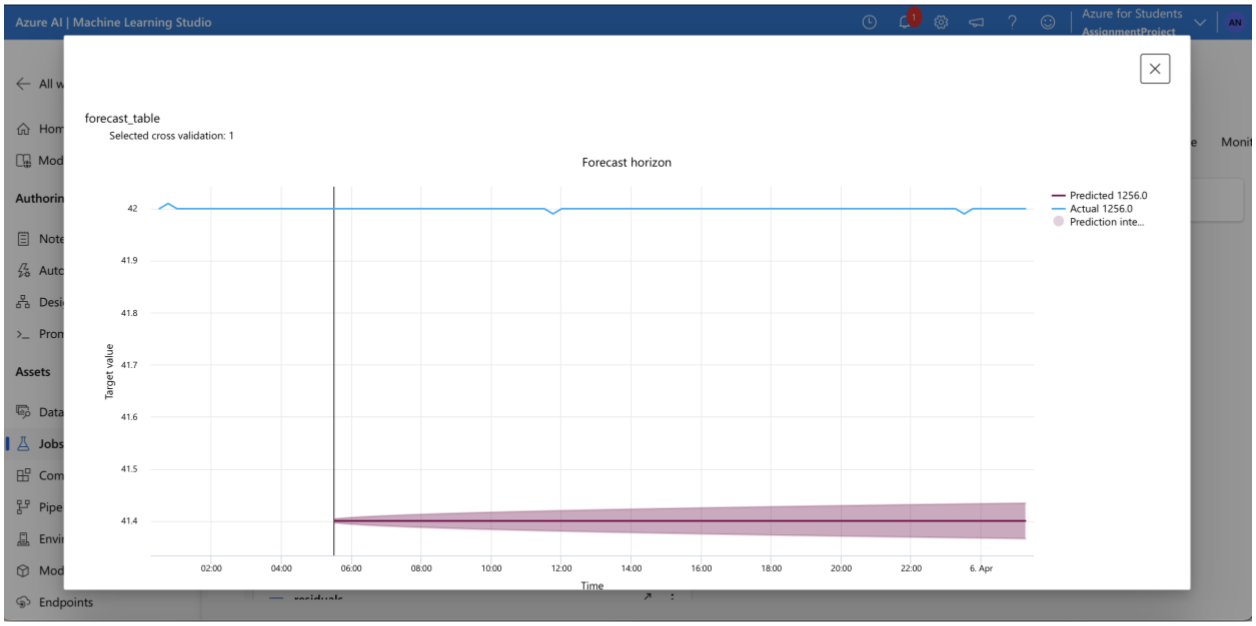


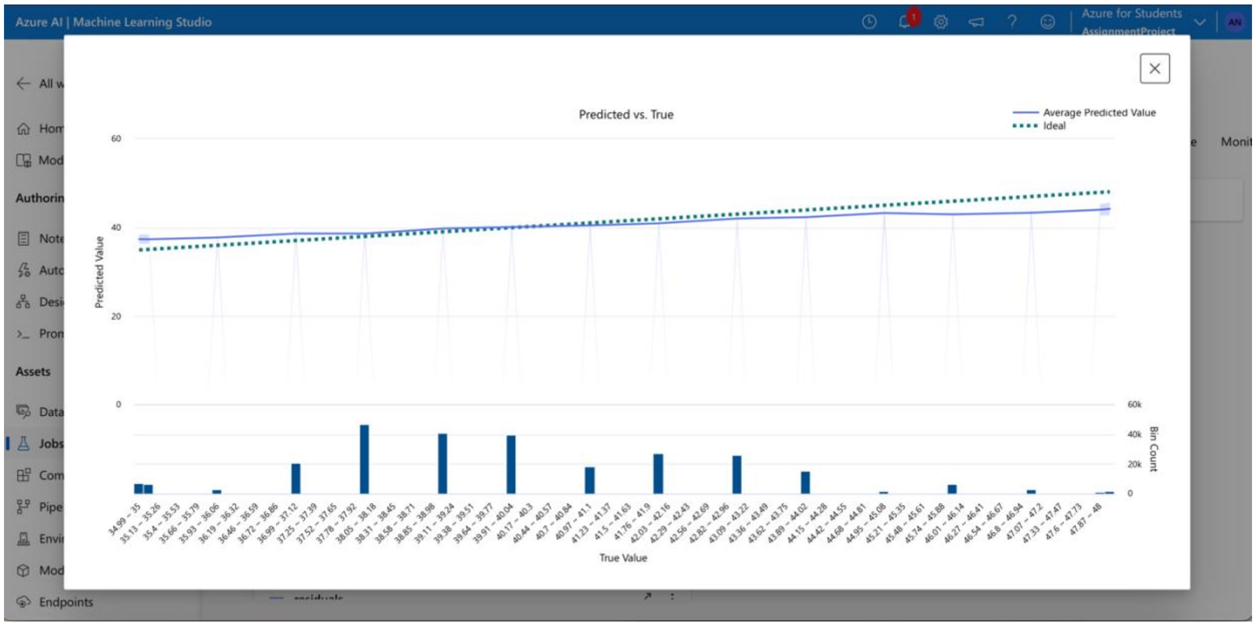
plum_star_pt3sq07m Completed

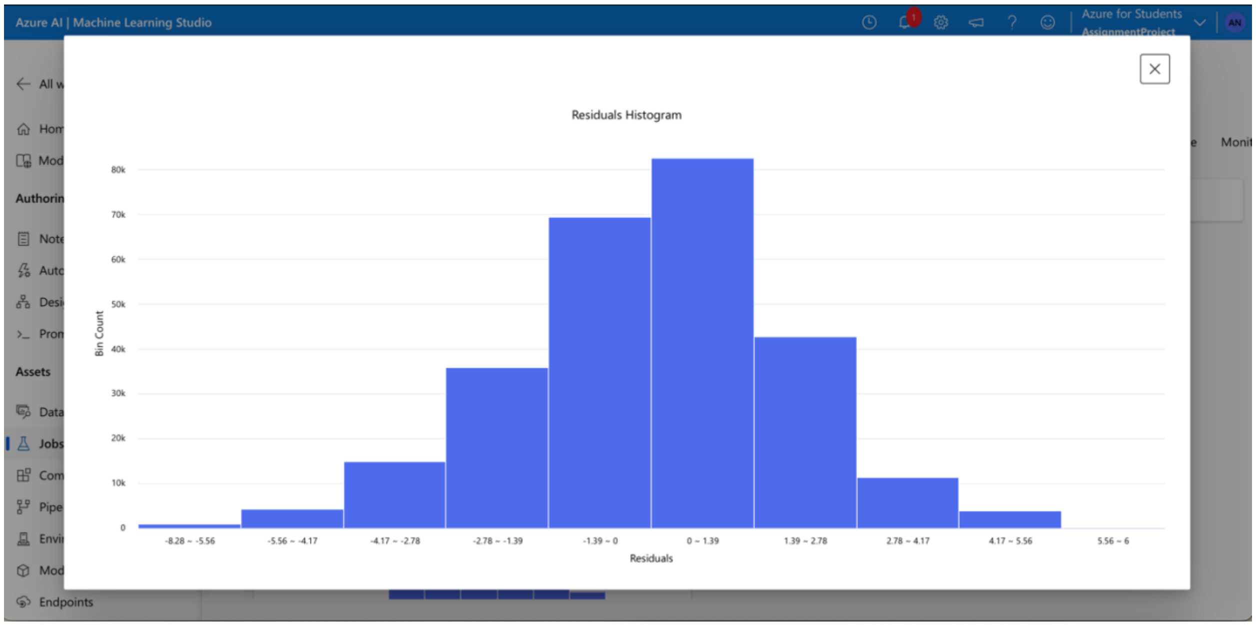
explained_variance	0.5416804
mean_absolute_error	1.463356
mean_absolute_perce...	3.632891
median_absolute_error	1.198213
normalized_mean_abs...	0.9723347
normalized_median_a...	0.9713094
normalized_root_mea...	0.9735037
normalized_root_mea...	0.9735037
r2_score	0.5416804
root_mean_squared_er...	1.881318
root_mean_squared_lo...	0.04517833
spearman_correlation	0.7223943

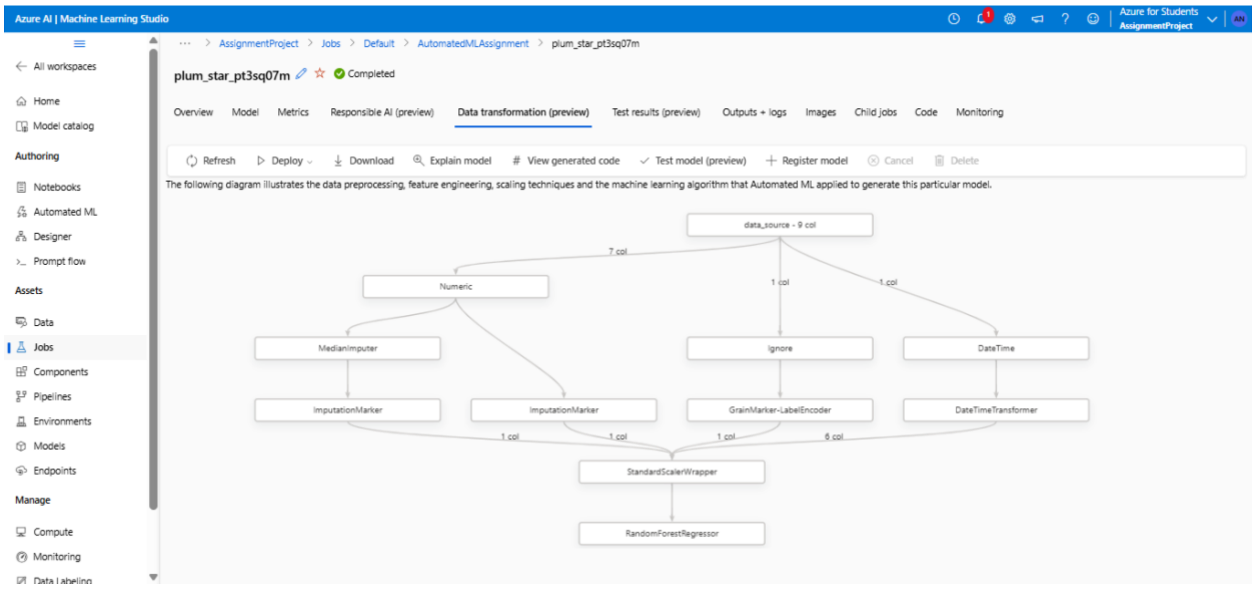
forecast_table
Selected cross validation: 1

predicted_true
Predicted vs. True — Average Predicted Value, Ideal


forecast_table
Selected cross validation: 1


Azure AI | Machine Learning Studio
Predicted vs. True
Average Predicted Value
Ideal
Predicted Value
Bin Count
True Value


Azure AI | Machine Learning Studio
Residuals Histogram
Bin Count
Residuals



The following diagram illustrates the data preprocessing, feature engineering, scaling techniques and the machine learning algorithm that Automated ML applied to generate this particular model.

Figure 4.3 – RandomForest Model Evaluation Summary

RandomForest model performance with evaluation metrics, forecast chart, and residual plot. Demonstrates weaker prediction compared to AutoARIMA.

Anomaly Detection

Part 1 – AutoML Model Comparison

In this step, we compared two machine learning models trained on the cleaned dataset to identify which one performs better in predicting AQI values and detecting anomalies.

Two models were compared:

1. AutoARIMA
2. RandomForest (with StandardScalerWrapper)

We used the evaluation metrics provided by Azure AutoML, including R² Score, Normalized Root Mean Squared Error (NRMSE), and Mean Absolute Error (MAE). The comparison is shown in the table below:

Metric	AutoARIMA	RandomForest
R^2 Score	0.9999995	0.5416804
Normalized RMSE	0.1842392	0.9735037
Mean Absolute Error	0.0016045	1.463356

From the results, we can see that AutoARIMA performed much better. It achieved a higher R² score and lower error values compared to the RandomForest model.

Even though both models were trained using the same cleaned dataset, RandomForest showed less accurate predictions and higher errors. This indicates that AutoARIMA is more suitable for AQI forecasting and detecting anomalies.

Part 2 – PCA-Based Unsupervised Detection

In addition to the AutoML model comparison, we also implemented unsupervised anomaly detection using PCA-Based Anomaly Detection in Azure Machine Learning Designer.

The cleaned AQI dataset (cleaned_indoor_data.csv) was passed through the following steps:

1. **Clean Missing Data**: Removed any null entries.
2. **Split Data**: Divided the dataset into training (70%) and testing (30%).
3. **PCA-Based Anomaly Detection**: Used to generate a PCA model that detects outliers without needing labeled anomaly data.
4. **Train Anomaly Detection Model**: Trained the PCA model using the training set.
5. **Score Model**: Tested the model using the test set and produced outputs including:
    a. **Scored Labels**: 1 indicates anomaly, 0 indicates normal
    b. **Scored Probabilities**: The confidence that a row is not an anomaly

The model detected a small number of anomalies, approximately 1.5% of the dataset, with most data labeled as normal. These results align with expectations for indoor AQI values, which typically remain stable except for brief fluctuations.

This method successfully complements the supervised AutoML approach by showing how unsupervised techniques like PCA can also identify irregular patterns in environmental sensor data.

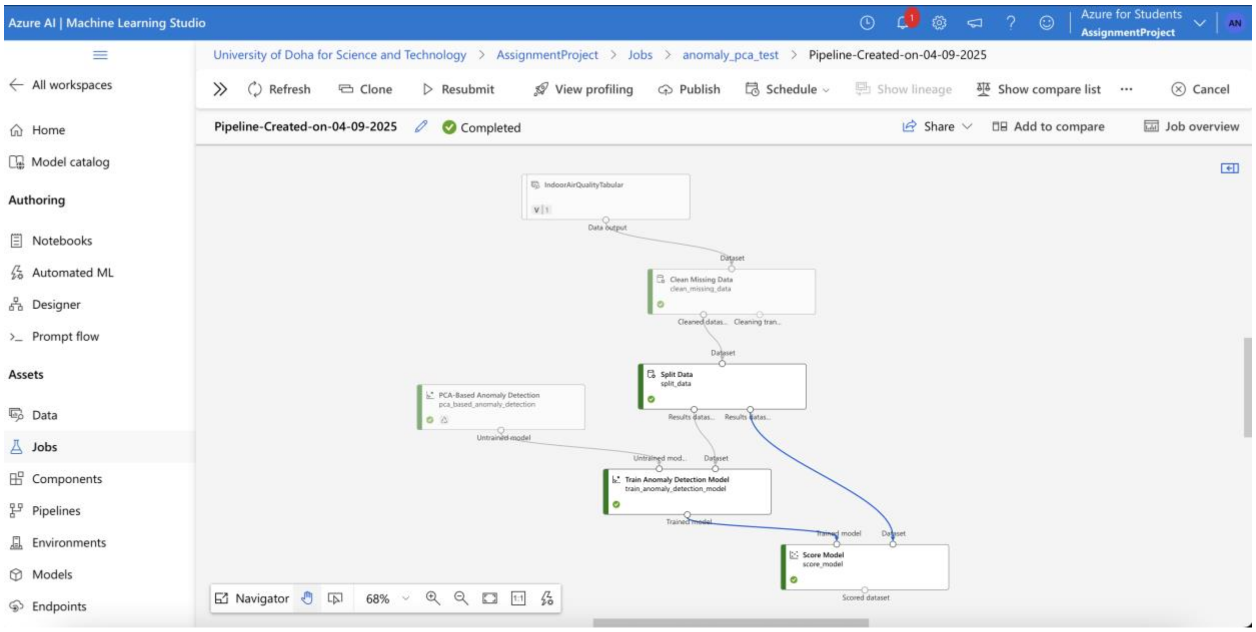
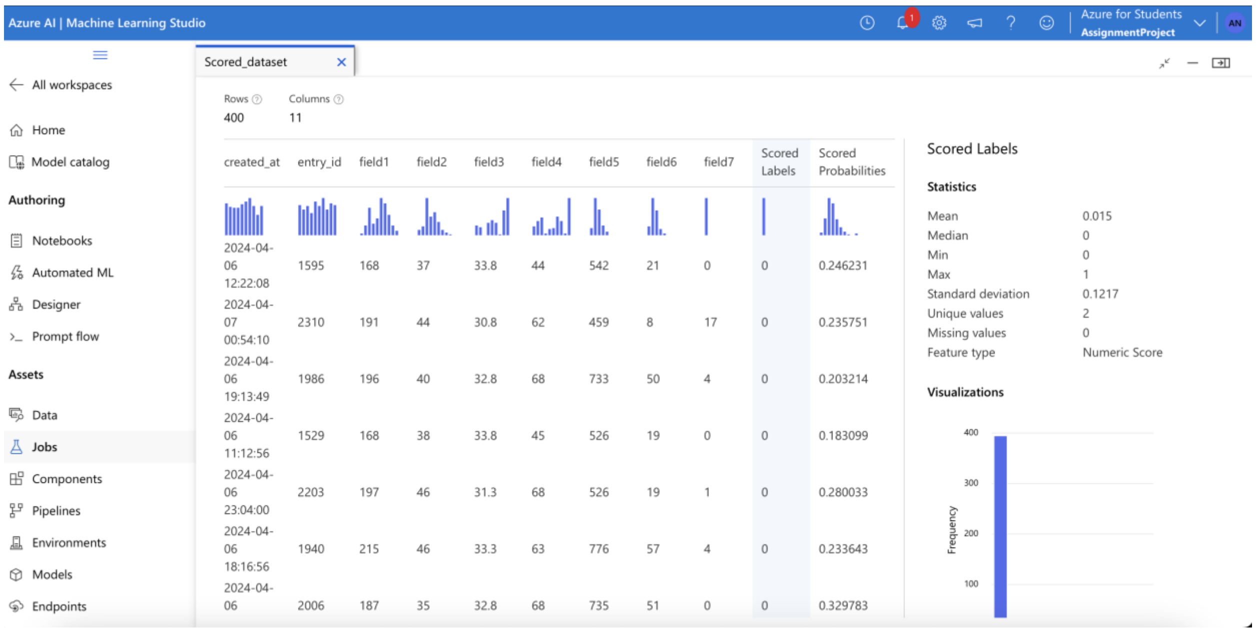


Figure 4.6 – PCA pipeline in Azure ML Designer



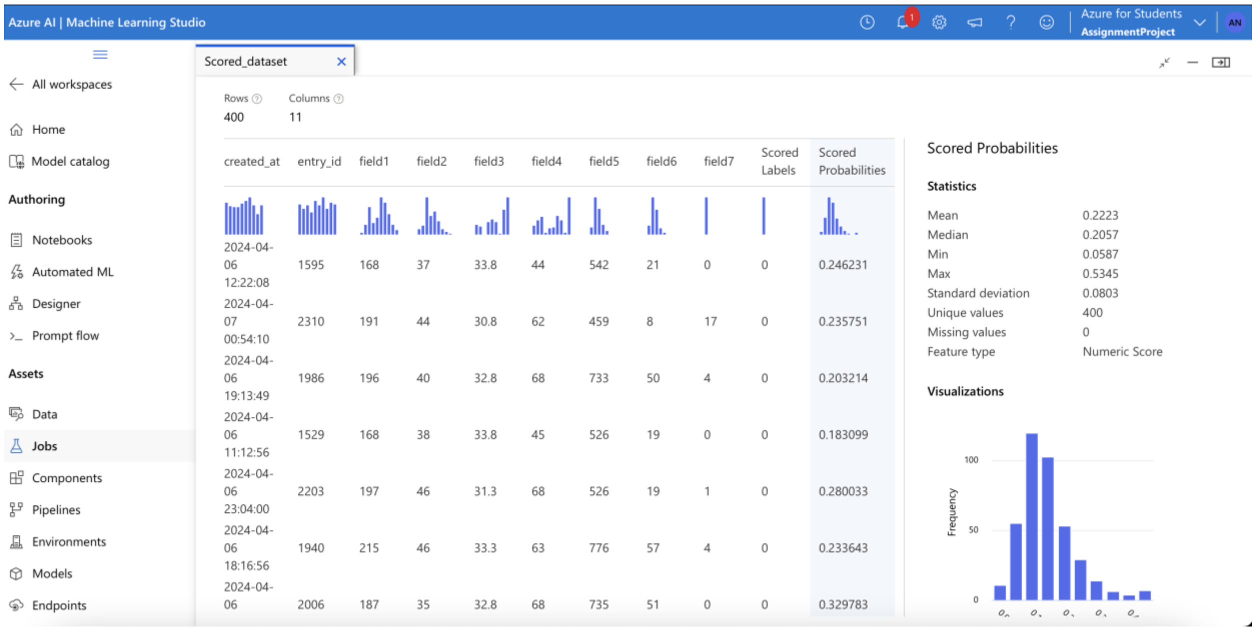
Rows 400 Columns 11

created_at	entry_id	field1	field2	field3	field4	field5	field6	field7	Scored Labels	Scored Probabilities
2024-04-06 12:22:08	1595	168	37	33.8	44	542	21	0	0	0.246231
2024-04-07 00:54:10	2310	191	44	30.8	62	459	8	17	0	0.235751
2024-04-06 19:13:49	1986	196	40	32.8	68	733	50	4	0	0.203214
2024-04-06 11:12:56	1529	168	38	33.8	45	526	19	0	0	0.183099
2024-04-06 23:04:00	2203	197	46	31.3	68	526	19	1	0	0.280033
2024-04-06 18:16:56	1940	215	46	33.3	63	776	57	4	0	0.233643
2024-04-06	2006	187	35	32.8	68	735	51	0	0	0.329783

Scored Labels

Statistics

Mean	0.015
Median	0
Min	0
Max	1
Standard deviation	0.1217
Unique values	2
Missing values	0
Feature type	Numeric Score



Rows 400 Columns 11

created_at	entry_id	field1	field2	field3	field4	field5	field6	field7	Scored Labels	Scored Probabilities
2024-04-06 12:22:08	1595	168	37	33.8	44	542	21	0	0	0.246231
2024-04-07 00:54:10	2310	191	44	30.8	62	459	8	17	0	0.235751
2024-04-06 19:13:49	1986	196	40	32.8	68	733	50	4	0	0.203214
2024-04-06 11:12:56	1529	168	38	33.8	45	526	19	0	0	0.183099
2024-04-06 23:04:00	2203	197	46	31.3	68	526	19	1	0	0.280033
2024-04-06 18:16:56	1940	215	46	33.3	63	776	57	4	0	0.233643
2024-04-06	2006	187	35	32.8	68	735	51	0	0	0.329783

Scored Probabilities

Statistics

Mean	0.2223
Median	0.2057
Min	0.0587
Max	0.5345
Standard deviation	0.0803
Unique values	400
Missing values	0
Feature type	Numeric Score

Figure 4.7 – Scored output from PCA anomaly detection

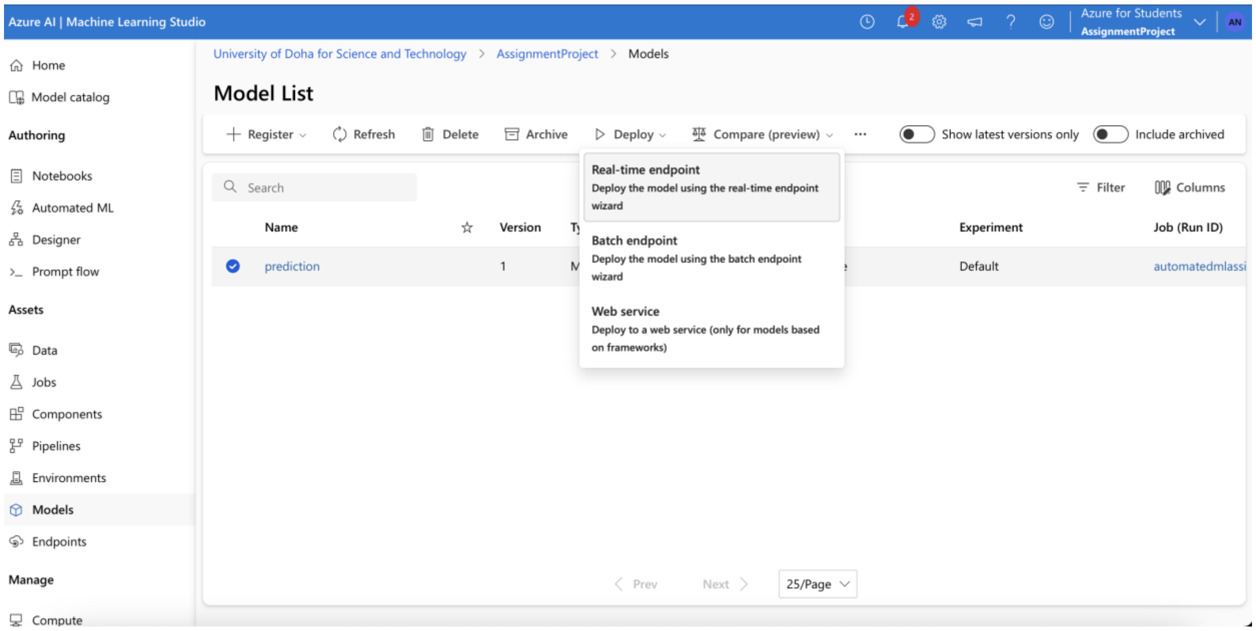
Model Deployment and Testing

After training and selecting the best model using Azure AutoML, we proceeded to deploy the model as a real-time endpoint in Azure Machine Learning Studio.

Deployment Process:

1. We navigated to the Models section and selected the trained model named *prediction*.

2. Under the Deploy dropdown, we selected Real-time endpoint.
3. We configured the deployment:
   a. **Virtual Machine size**: Standard_D2as_v4 (2 cores, 8 GB RAM)
   b. **Endpoint name**: assignmentproject-nmbuq
   c. **Deployment name**: prediction-1
   d. **Inferencing data collection**: Enabled
4. After deploying, the provisioning status changed to "*Succeeded*", confirming that the model was successfully deployed.



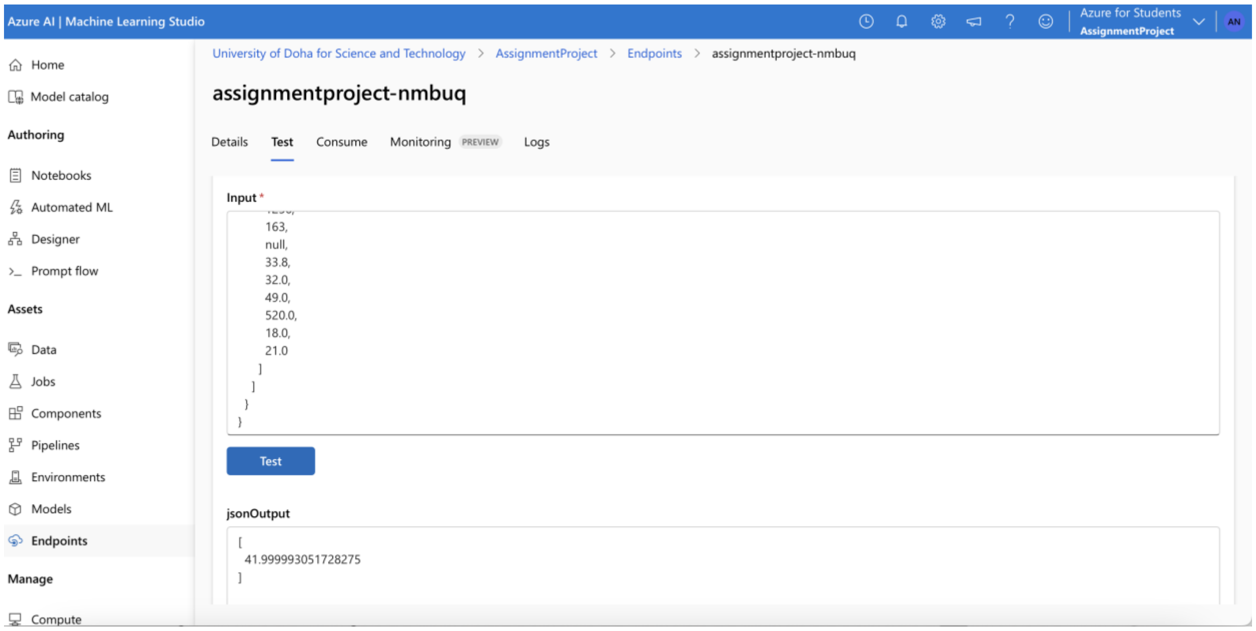
Name	Version	Ty	Experiment	Job (Run ID)
prediction	1	M	Default	automatedmlassi

Real-time endpoint
Deploy the model using the real-time endpoint wizard

Batch endpoint
Deploy the model using the batch endpoint wizard

Web service
Deploy to a web service (only for models based on frameworks)

Figure 4.8 – Azure AutoML Model Deployment Menu



assignmentproject-nmbuq

Input *

```
        163,
        null,
        33.8,
        32.0,
        49.0,
        520.0,
        18.0,
        21.0
      ]
    ]
  }
}
```

jsonOutput

```
[
  41.999993051728275
]
```

Figure 4.9 – Endpoint Test Output

Testing the Endpoint: We used sample AQI data in the Test tab of the deployed endpoint. The system returned a valid AQI prediction in JSON format, confirming that real-time inference was functioning correctly.

Chapter 5: Results and Conclusions

Results

By the end of our project, we were able to clearly see which machine learning model worked better for predicting the Air Quality Index (AQI). Using Azure AutoML, the system chose AutoARIMA as the best model. It gave us really high accuracy with an R^2 score of almost 1 (0.9999995), and very low error values (RMSE = 0.0020, MAE = 0.0016). This means it was very good at predicting the AQI based on past data, and the results were very close to the actual values.

We also tried RandomForest, but it didn't work as well. Its R^2 score was only 0.54, and the error values were much higher (RMSE ≈ 0.97, MAE ≈ 1.46). This showed us that RandomForest wasn't the right model for this kind of time-based data. When we compared the graphs of predicted and real values, AutoARIMA's results were much closer to the real data.

To make our system more reliable, we used both supervised and unsupervised methods for anomaly detection. The supervised method was already part of AutoML. Then we used a PCA-based method in Azure ML Designer to detect outliers without needing labeled data. This method found that around 1.5% of the dataset had unusual values, which makes sense since indoor air quality usually stays steady.

We did all the data cleaning, training, and testing in Azure ML Studio. The platform also gave us graphs like residual histograms and forecast charts, which made it easier to understand the model's accuracy. This made our project not only effective but also easy to explain and present.

What made our solution even more complete was deploying the model as a real-time service. We connected it with our Arduino and MQ-135 sensor using Azure IoT Central. This allowed us to send real-time AQI data to Azure and get predictions immediately. We also exported some of this data to Azure Blob Storage to prove that the system was working and that the data was being saved properly.

Conclusion

In conclusion, our project proved that we could successfully design and build a working IoT-based air quality monitoring system. We connected our Arduino with a gas sensor and used Microsoft Azure to collect, clean, and analyze the data. The AutoARIMA model gave very good predictions, and PCA helped us catch anomalies without needing any labels.

Our setup was simple, cloud-based, and could easily be expanded in the future. We also made sure it worked with real-time data and stored everything in the cloud. This means the system could be used in real situations like schools, offices, or even smart cities.

Overall, we were able to meet all the goals of the project. We built a working system, learned how to use machine learning in Azure, and explored how IoT and cloud computing can help in monitoring the environment. It was a great learning experience that also showed how technology can be used to support health and safety.

References

Ijraset. (n.d.). *Anomaly detection in ambient air quality*. IJRASET. https://www.ijraset.com/research-paper/anomaly-detection-in-ambient-air-quality

Ravindiran, G., Hayder, G., Kanagarathinam, K., Alagumalai, A., & Sonne, C. (2023). Air quality prediction by machine learning models: A predictive study on the indian coastal city of Visakhapatnam. *Chemosphere*, *338*, 139518. https://doi.org/10.1016/j.chemosphere.2023.139518

Wikipedia contributors. (2025, January 15). *Air quality index*. Wikipedia. https://en.wikipedia.org/wiki/Air_quality_index#Overview

GitHub Repository:

https://github.com/euphoraw/AQI-Prediction-Anomaly-Detection