

## **Assignment Project**

INFT4104 Selected topics in IT

By: Noor Alqusei 60098352 & Rawan Alqusei 60098355

Instructor Name: Dr. Abderrahmane Maaradji

## Table of Contents

<b>Chapter 1: Introduction.....</b>	<b>3</b>
Overview of Air Quality Index (AQI) .....	3
Machine Learning for AQI Forecasting.....	3
Anomaly Detection in AQI Data .....	4
<b>Chapter 2: System Design and Implementation .....</b>	<b>5</b>
Hardware Setup .....	5
Software Setup .....	6
Azure Blob Storage (Optional Feature) .....	9
<b>Chapter 3: Dataset and Preprocessing.....</b>	<b>11</b>
Dataset Overview.....	11
Preprocessing Steps: .....	11
<b>Chapter 4: Machine Learning for Prediction and Anomaly Detection.....</b>	<b>13</b>
AQI Prediction Using AutoARIMA .....	13
Anomaly Detection.....	19
Part 1 – AutoML Model Comparison .....	19
Part 2 – PCA-Based Unsupervised Detection.....	20
<b>Chapter 5: Results and Conclusions .....</b>	<b>22</b>
Results .....	22
Conclusion .....	22
<b>References.....</b>	<b>23</b>
<b>GitHub Repository: .....</b>	<b>23</b>

# 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)

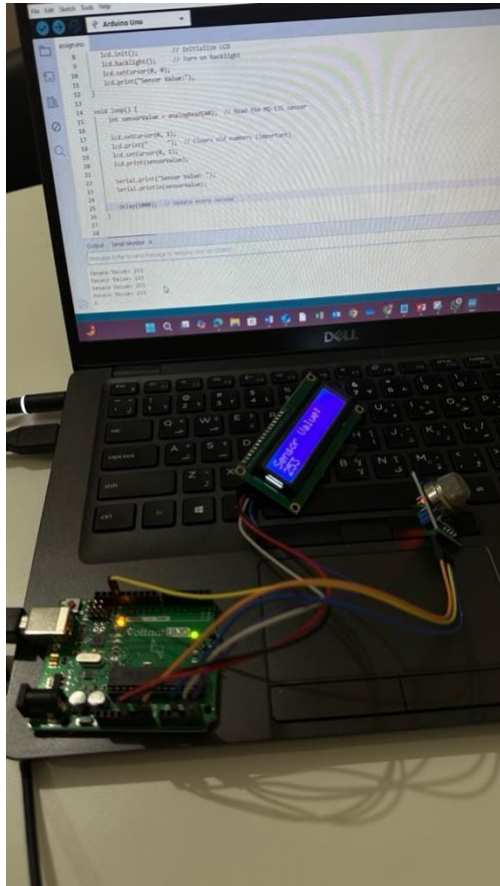
### Anomaly Detection in AQI Data

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 the project includes an Arduino Uno microcontroller and an MQ-135 air quality sensor, which is capable of detecting gases such as NH<sub>3</sub>, NO<sub>x</sub>, alcohol, benzene, smoke, and CO<sub>2</sub>. The sensor is used to collect real-time Air Quality Index (AQI) values from the environment.



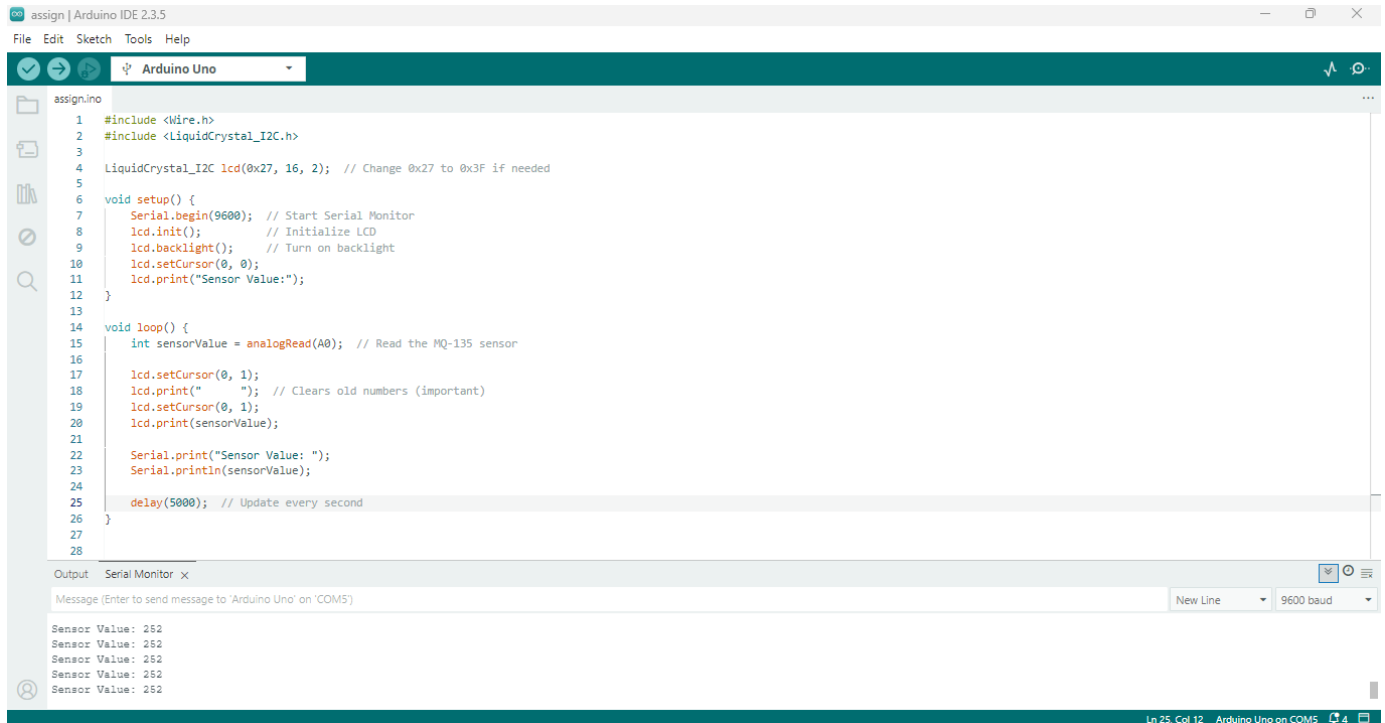


Figure 2.1 – Arduino AQI Sensing Setup

To enhance the user interface and data readability, an LCD display was integrated with the Arduino to show live AQI readings. The sensor readings were programmed to be collected and displayed every 5 seconds using the `delay(5000)` function. The data was also output to the Serial Monitor for further processing and transmission.

## 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:
  - o Establish a connection using the device's connection string.
  - o Read data from the Arduino serial port.
  - o Extract the numeric value from the incoming data (e.g., "Sensor Value: 255" → 255).
  - o 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\Deil\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=iotcc-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 2.3 – Python script transmitting data to Azure IoT Central

The top screenshot displays the 'MQ-135' device template configuration page in Azure IoT Central. The left sidebar shows the navigation menu with categories: Connect (Devices, Device groups, Device templat...), Analyze (Data explorer, Dashboards), Manage (Jobs), Extend (Rules, Data export), and Security. The main content area shows the 'MQ-135' template details, including a breadcrumb trail 'Device templates > MQ-135 > Model > MQ-135'. A dropdown menu is open, showing 'Model', 'MQ-135', 'Raw data', and 'Views'. The 'MQ-135' model is selected, showing its configuration form. The form includes fields for 'Display name' (airquality), 'Name' (airquality), 'Capability type' (Telemetry), and 'Semantic type' (None). There are also buttons for 'Save', 'Add capability', 'Edit identity', 'Export', 'Delete', and 'Edit DTDL'.

The bottom screenshot displays the 'All devices' registration page. The left sidebar is the same as the top screenshot. The main content area shows the 'All devices' page with a 'Filter templates' input field. Below the input field, there is a table listing the registered devices. The table has four columns: 'Device name', 'Device ID', 'Device status', and 'Device template'. The table contains one row with the following data:

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

Figure 2.4 – Azure IoT Central Device Template and Registration



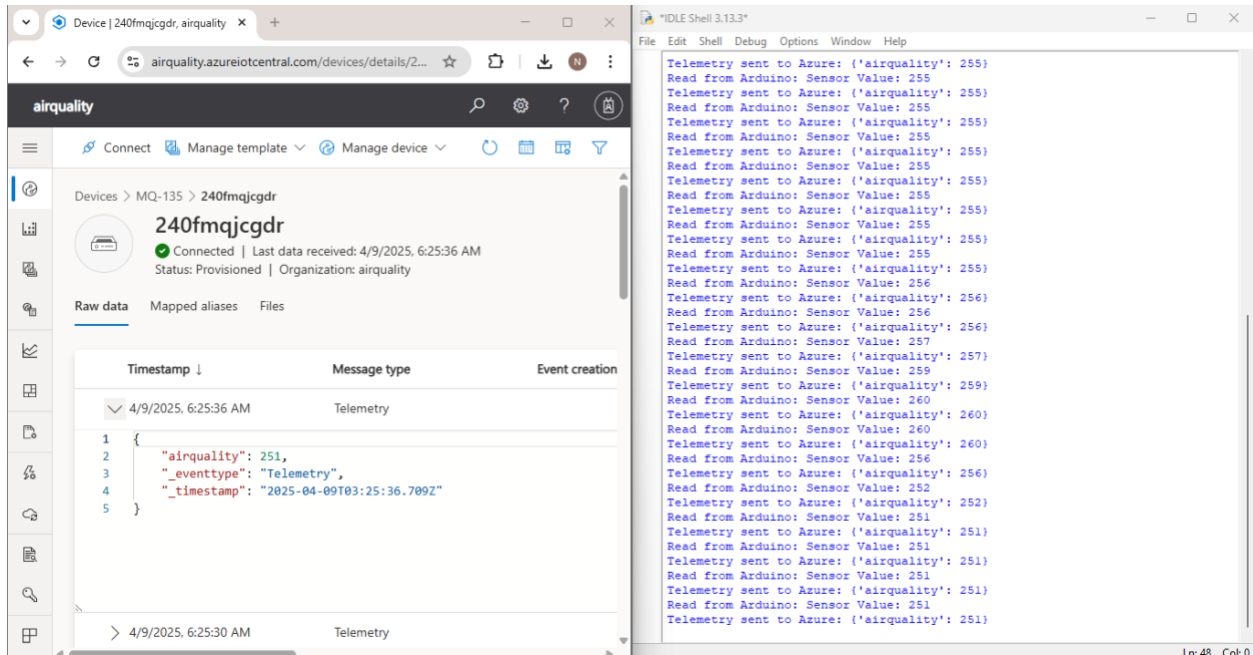


Figure 2.5 – Real-time AQI telemetry received in Azure IoT Central

## 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

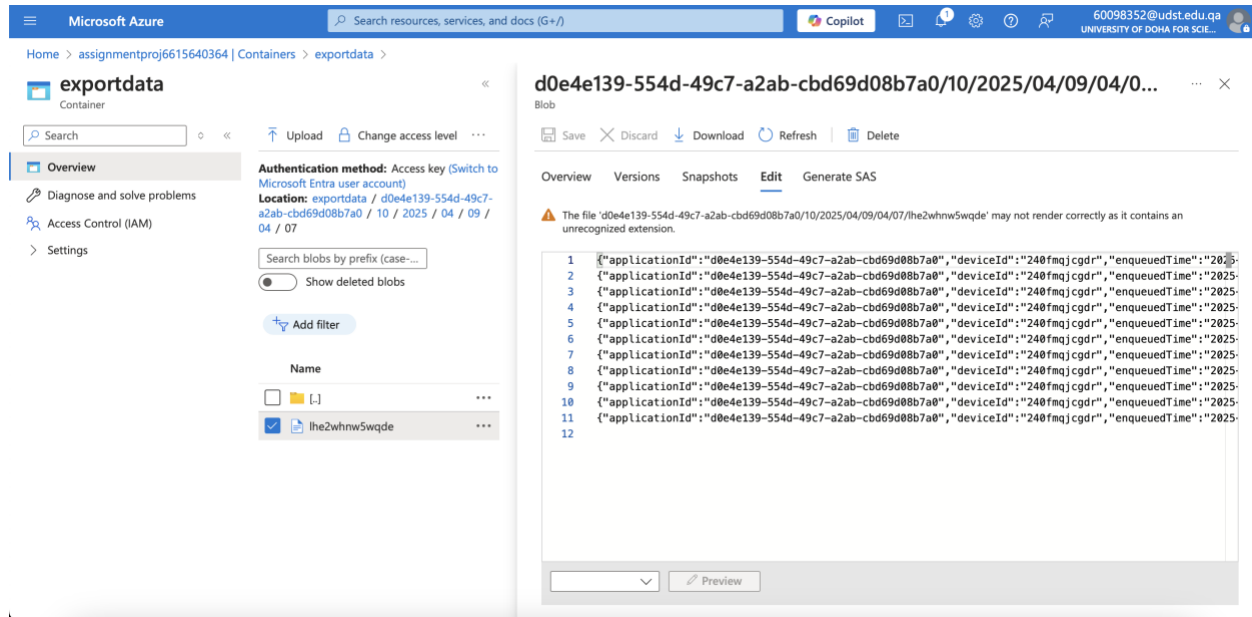


Figure 2.6 – Telemetry data exported to Azure Blob Storage

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

### Dataset Overview

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.

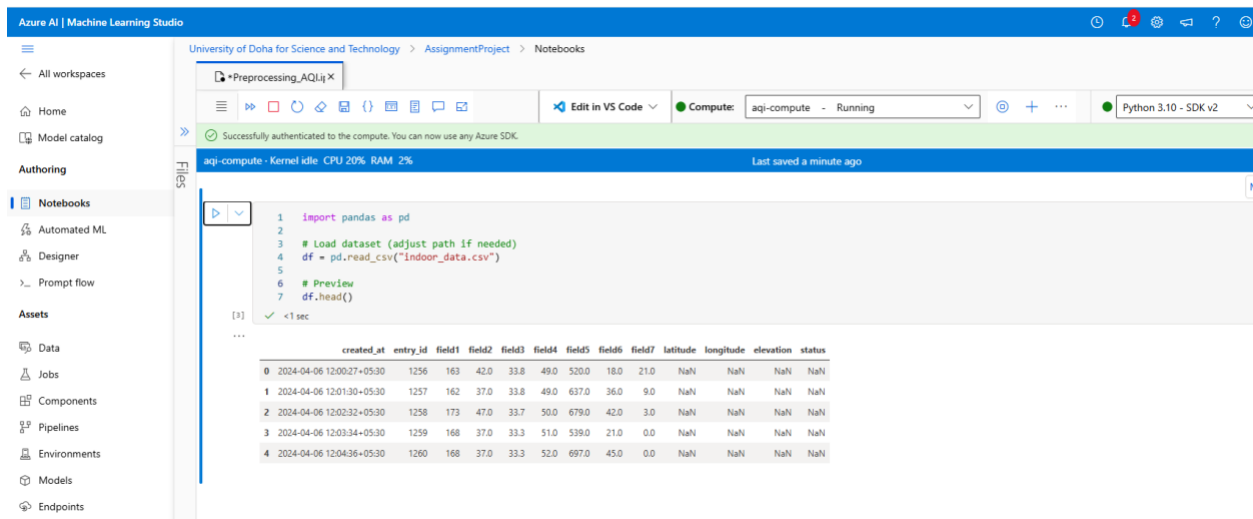


Figure 3.1 – Raw dataset preview in Azure ML Studio

### 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.

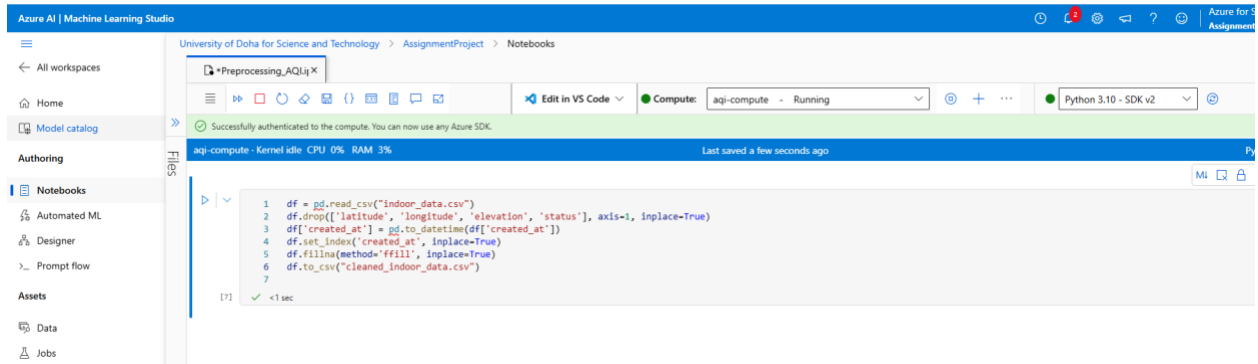


Figure 3.2 – Python notebook for preprocessing: dropping columns and handling missing values

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

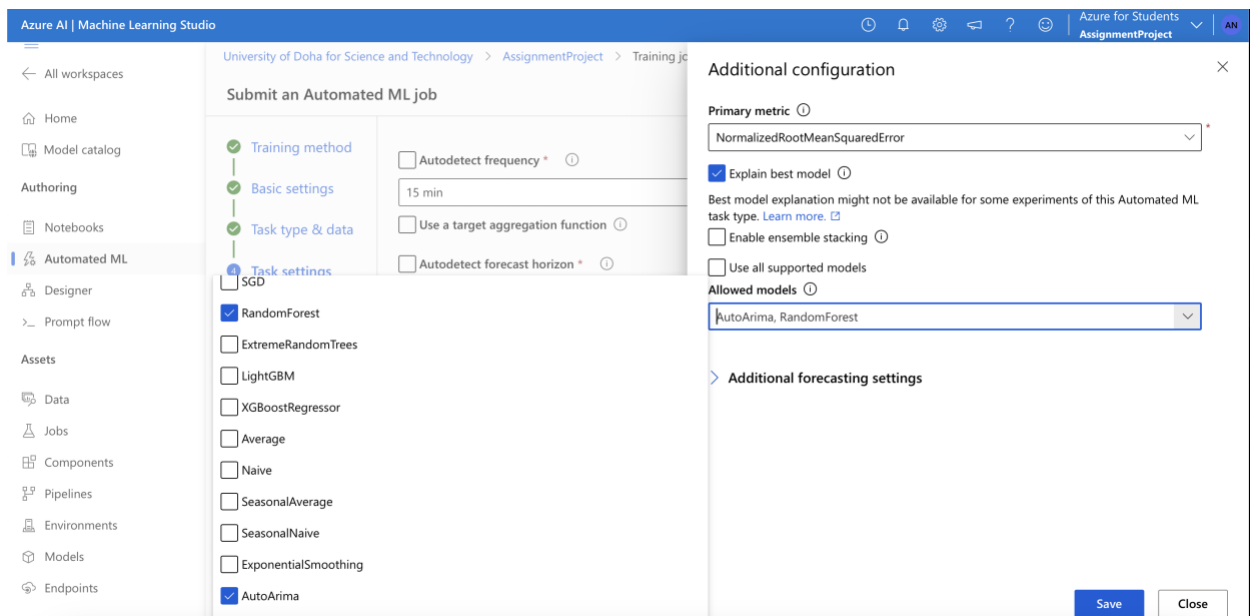
Figure 3.3 – Cleaned dataset ready for model training

## 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)



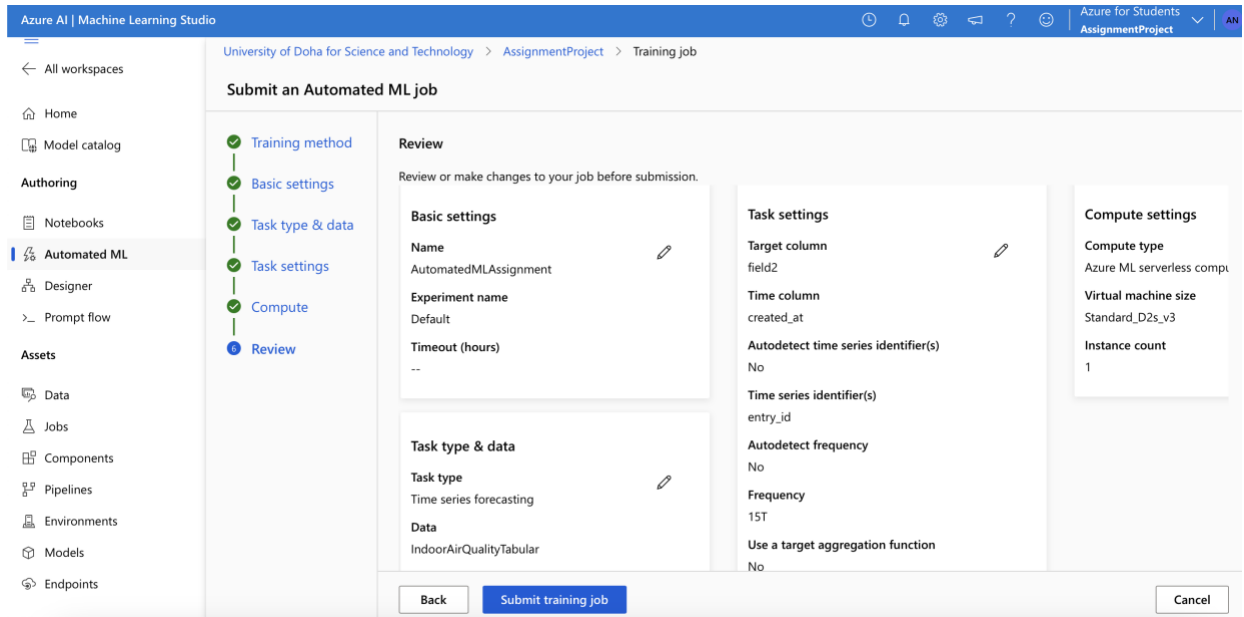
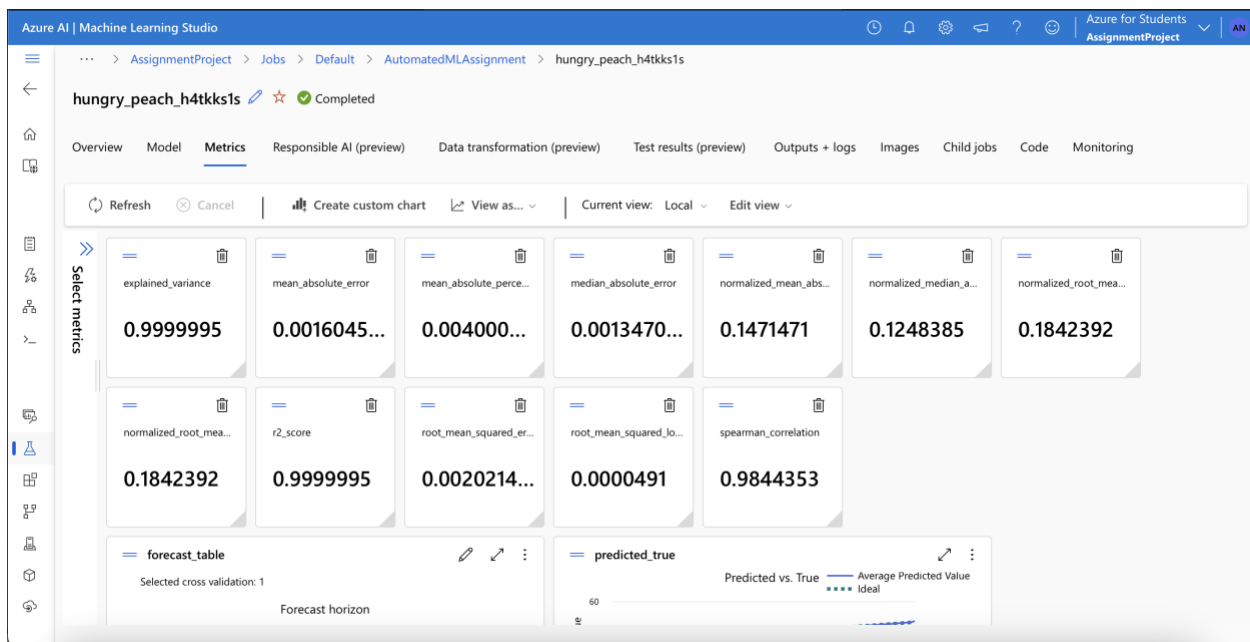
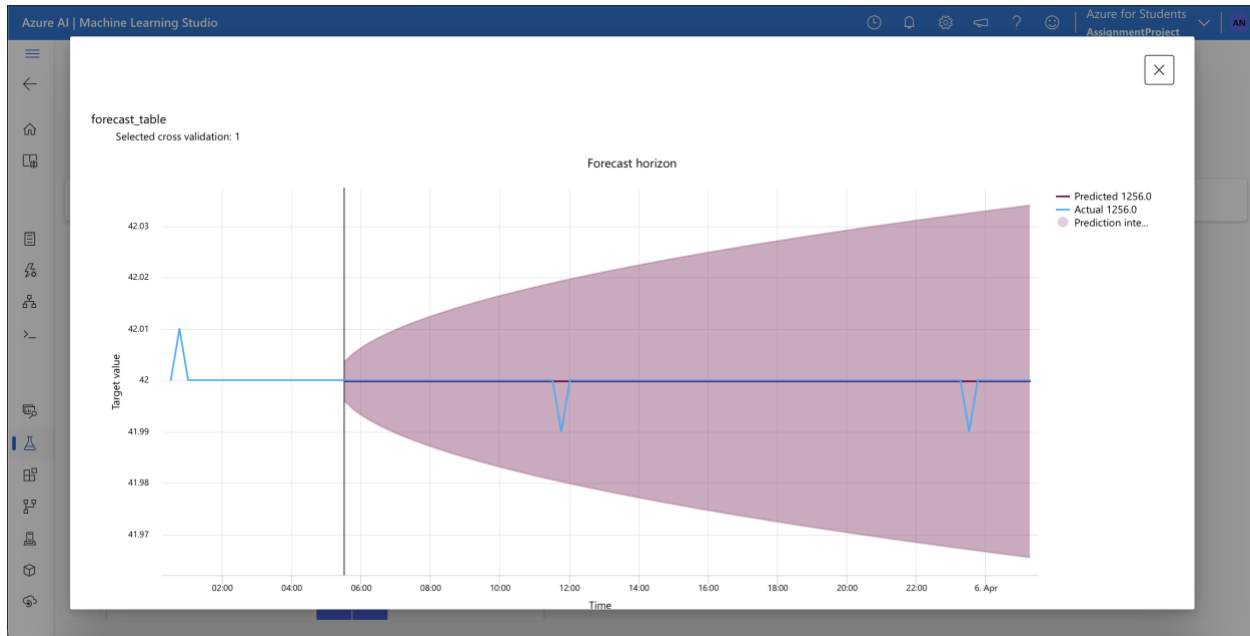
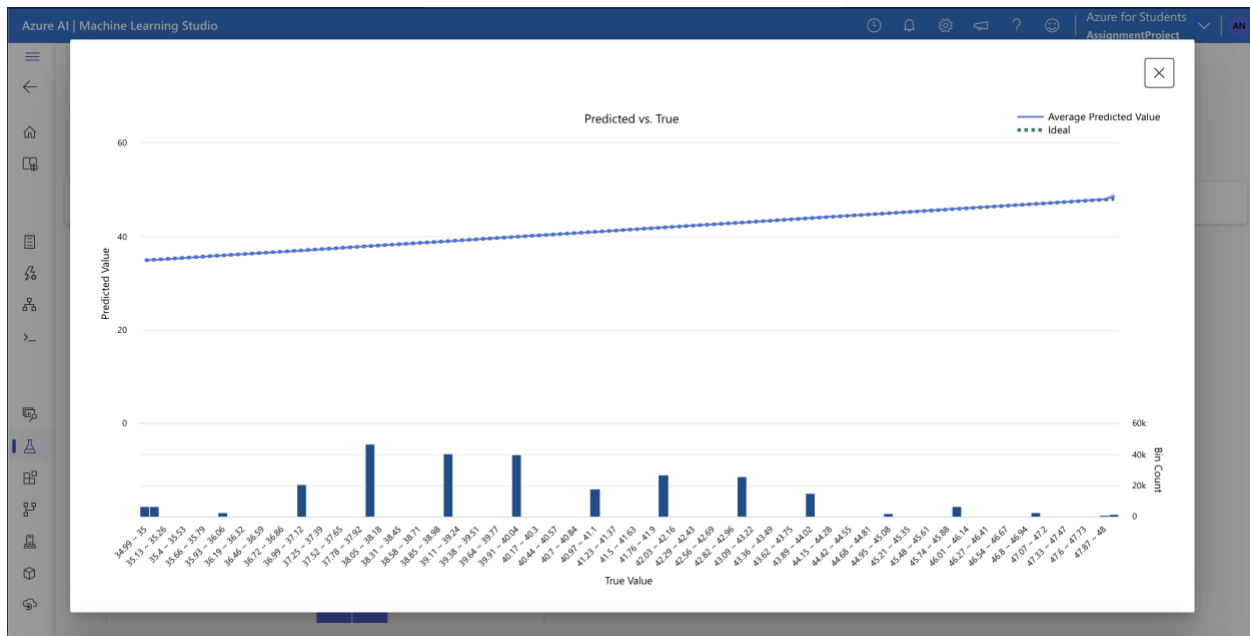


Figure 4.1 – Azure AutoML Time Series Forecasting job configuration

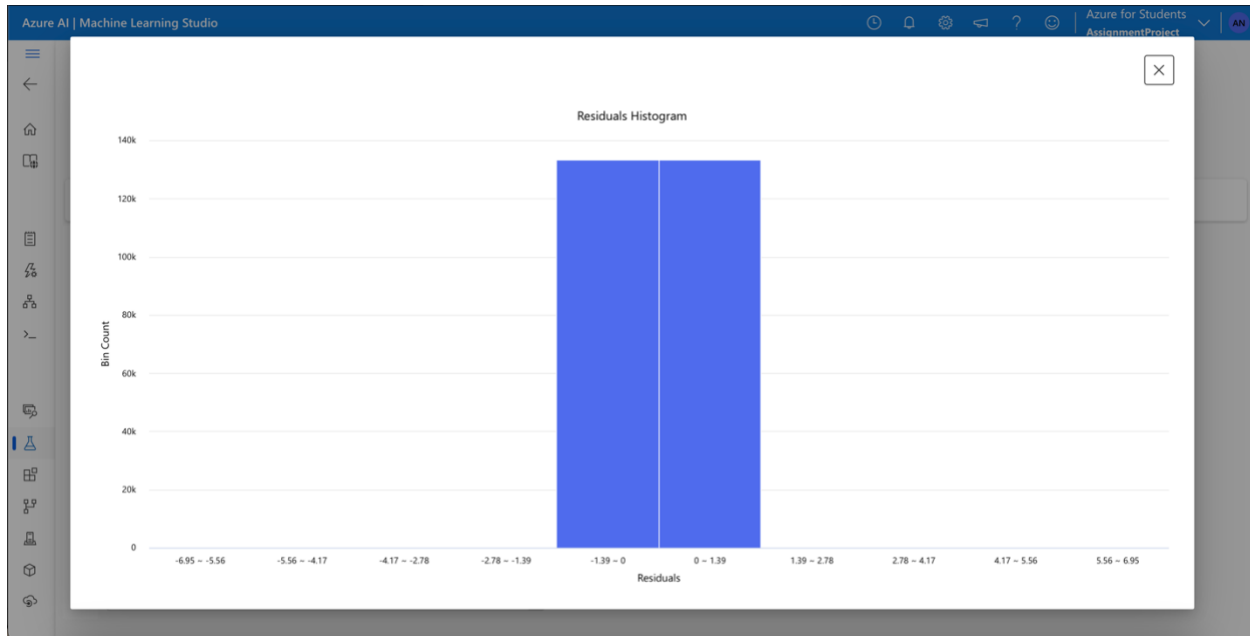




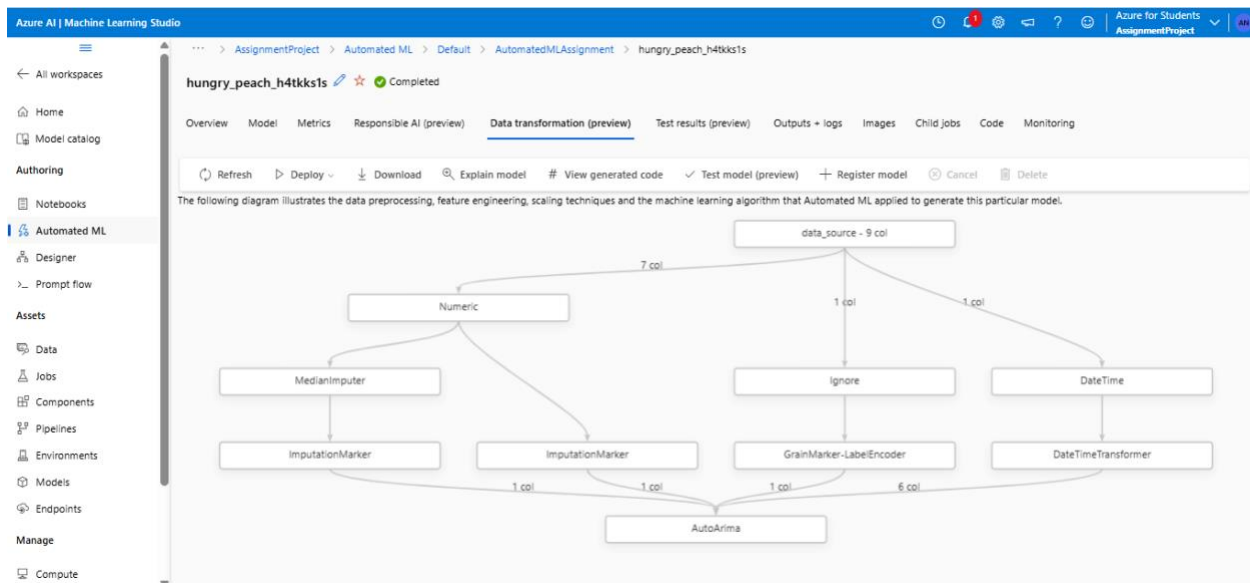
*This chart shows the model's forecast of future air quality values (purple line), compared to the actual values (blue line). The shaded purple area shows the confidence range, meaning the model is quite sure the true values will fall inside it.*



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



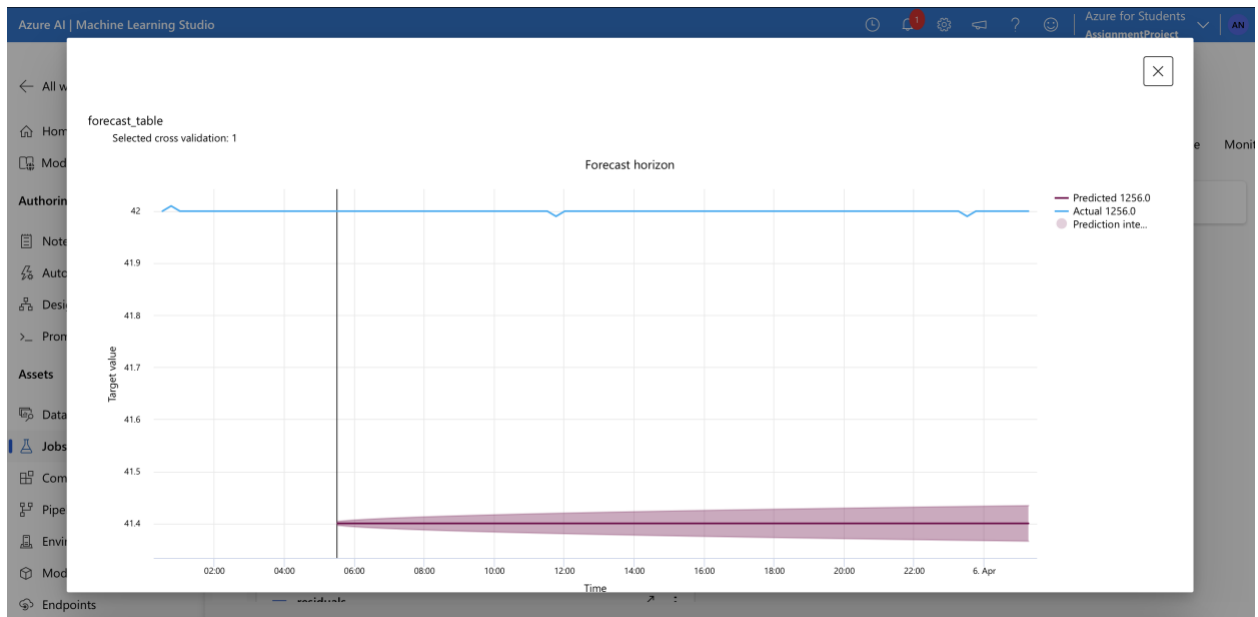
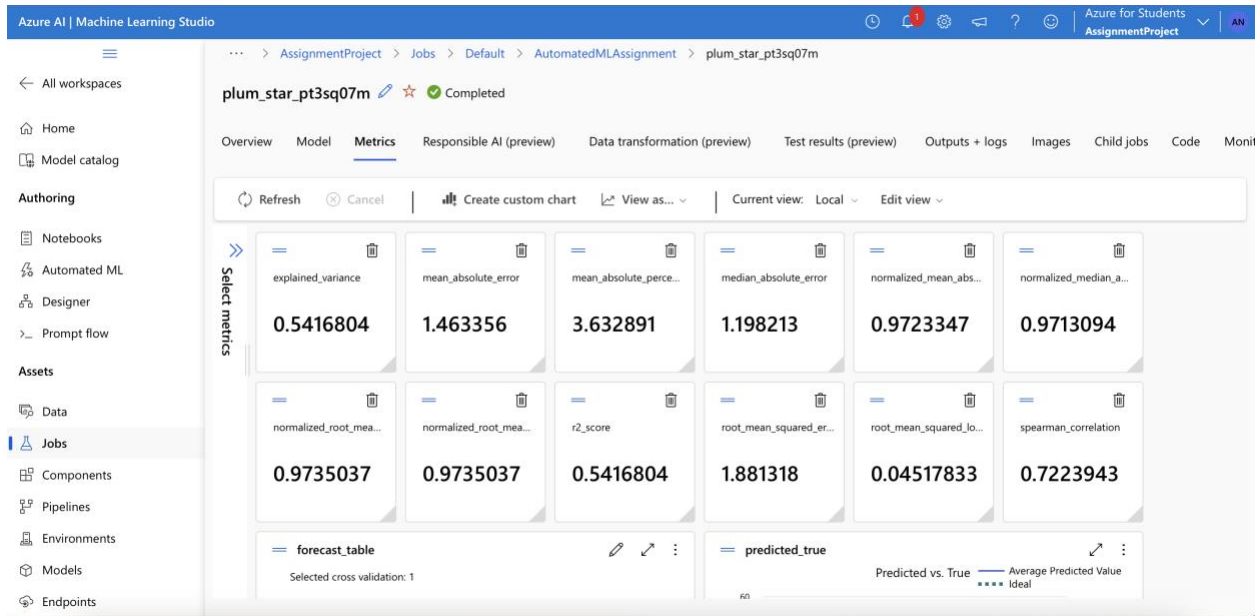
*This histogram shows the distribution of prediction errors (residuals). Most errors are small and centered around zero, meaning the model made very few big mistakes.*

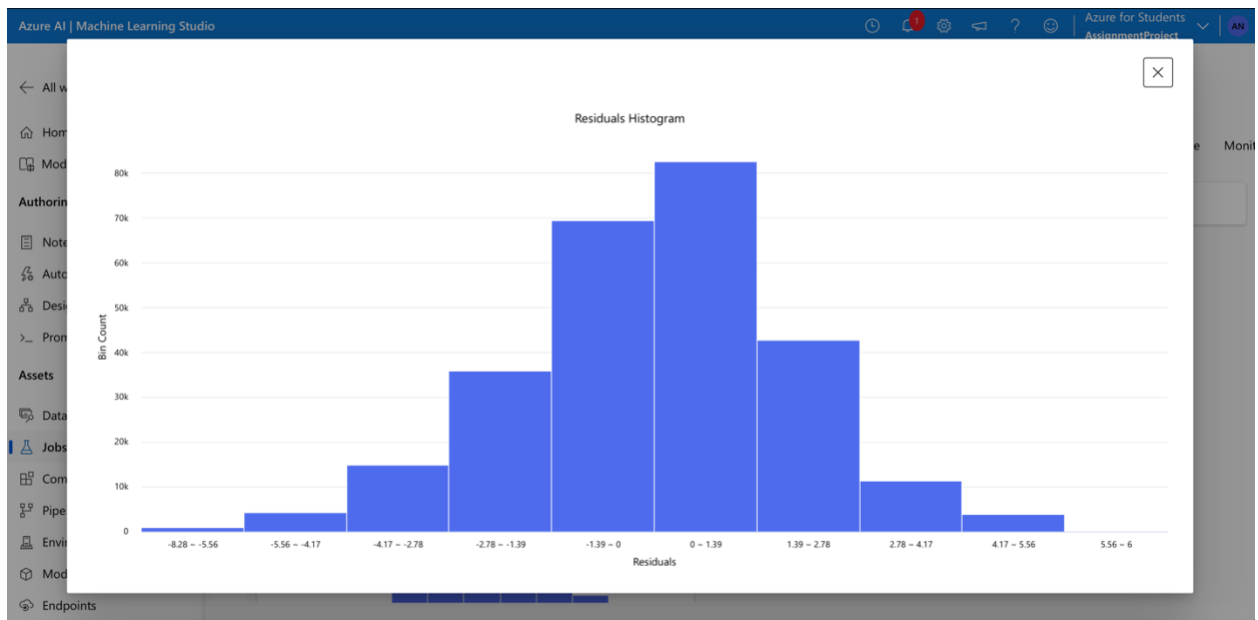


**Figure 4.2 – AutoARIMA Model Evaluation Summary**

*AutoARIMA model performance including metrics ( $R^2$ , RMSE, MAE), forecast chart, residual histogram, predicted values comparison, and preprocessing steps.*







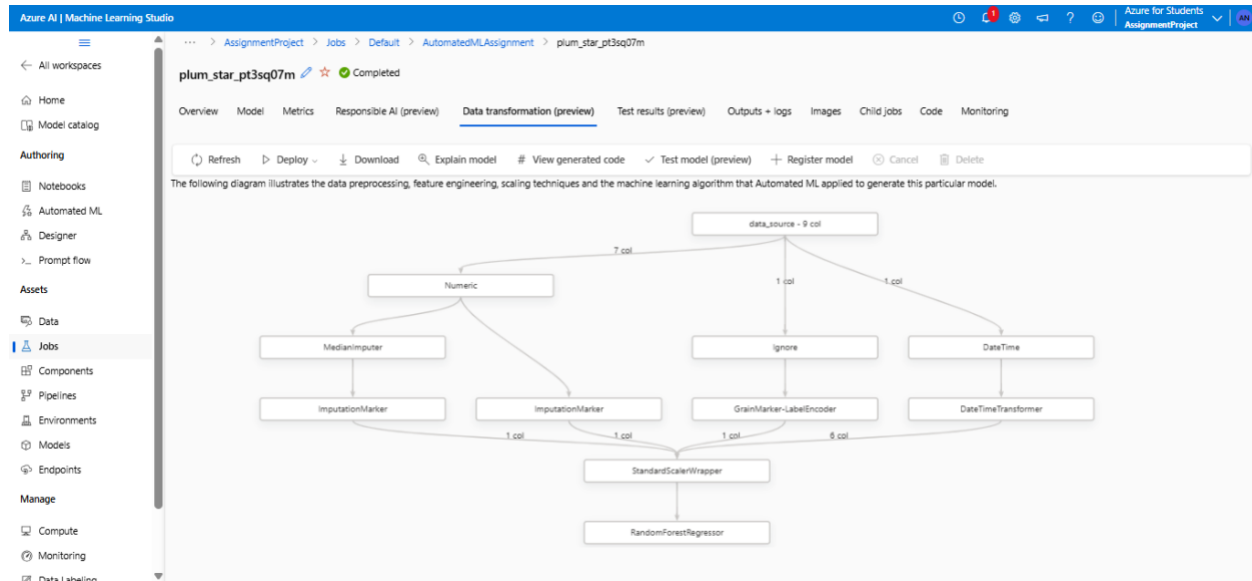


Figure 4.3 – RandomForest Model Evaluation Summary

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  $R^2$  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  $R^2$  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.

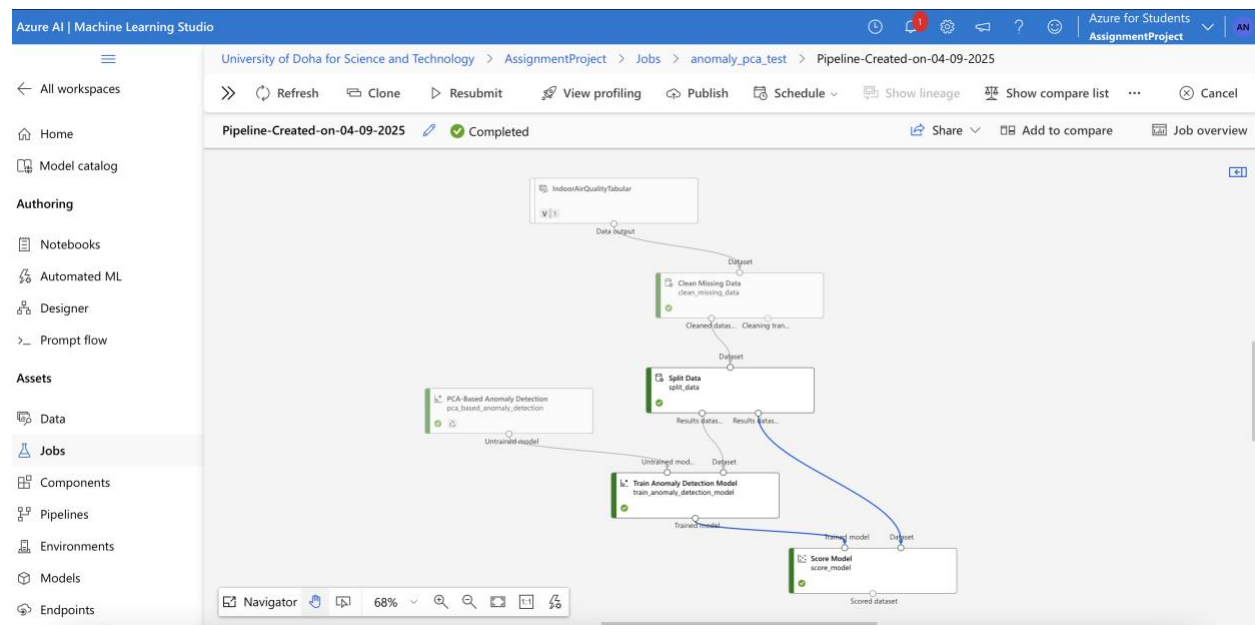


Figure 4.6 – PCA pipeline in Azure ML Designer

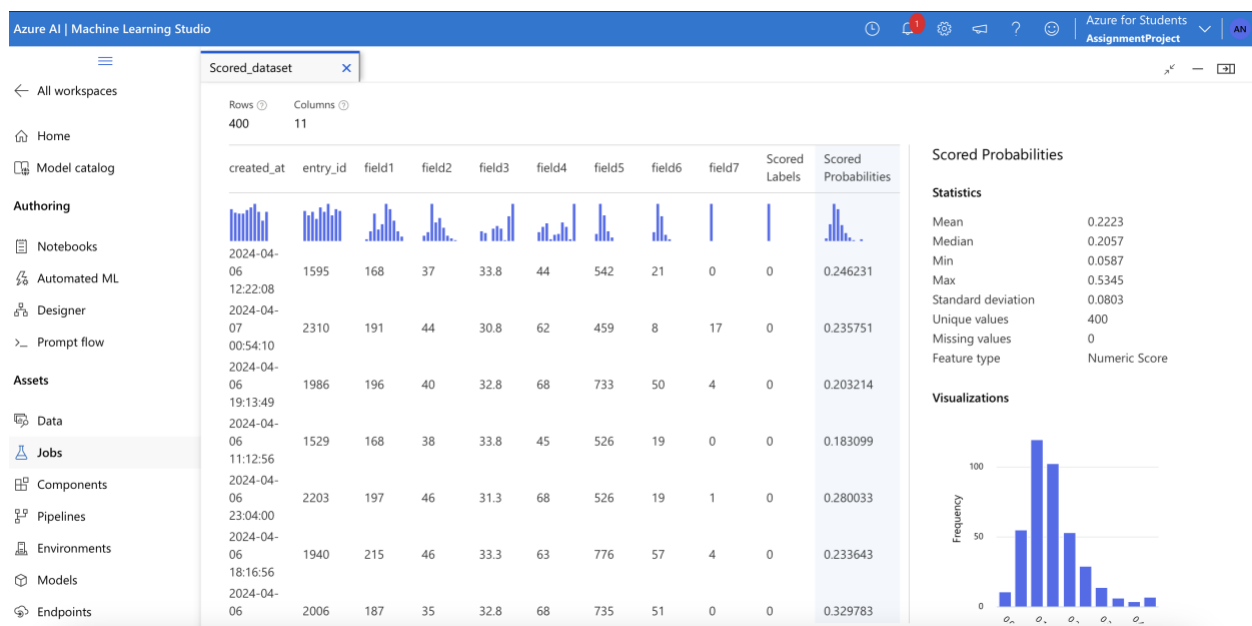
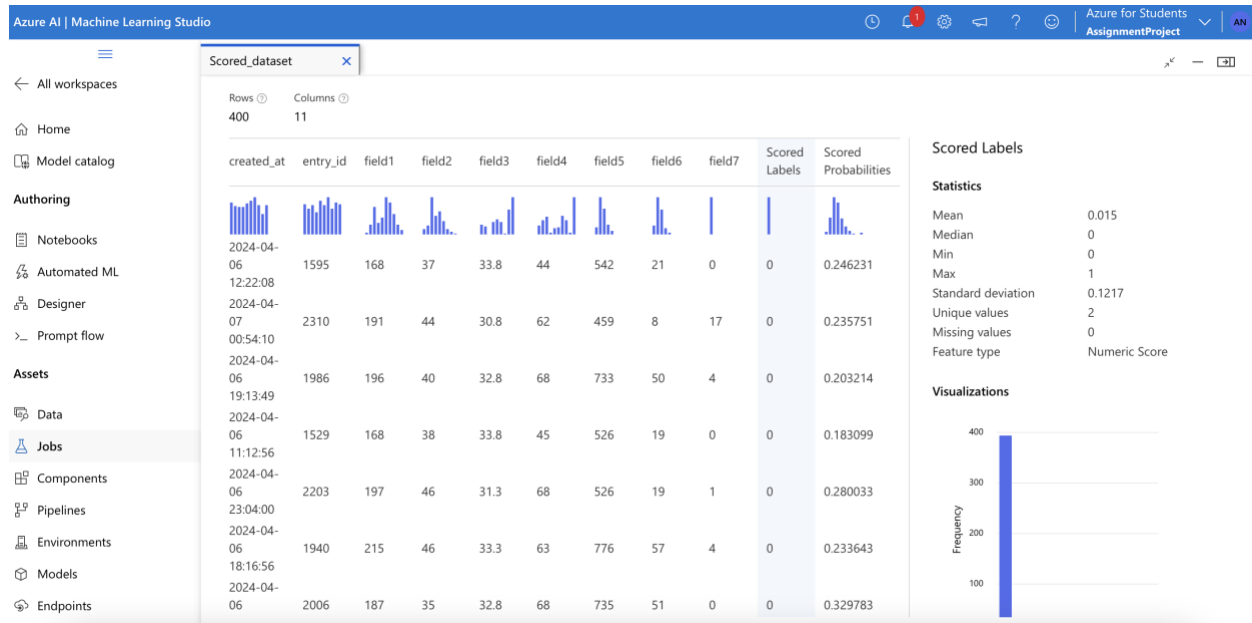


Figure 4.7 – Scored output from PCA anomaly detection

## 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  $\approx$  0.97, MAE  $\approx$  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](https://en.wikipedia.org/wiki/Air_quality_index#Overview)

## GitHub Repository:

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