BIN ROUTES COLLECTION STREAMLIING USING WEIGHT, PROXIMITY AND TEMPERATURE SENSORS CONNECTED TO A RASPBERRY PI

AUTHORS:
ANDREA BUTERA- C2653 67
MARK DITCHBURN - C2932952
KUNO DE-LEEUW KENT - B1033634

MODULE: INTERNET OF THINGS
DATE: 03/04/2025
NO GENERATIVE AI WAS USED FOR THIS PROJECT

Contents

Artifact Link:

https://github.com/KunoDLK/IoTGroupProject

Introduction

The integration of Internet of Things (IoT) technologies into urban infrastructure is revolutionising the way public services are delivered, improving both efficiency and sustainability. Among the services poised to benefit most from these innovations is waste management.

This report presents a smart waste management solution developed as part of the Teesside University Internet of Things project. The system combines a Raspberry Pi with a suite of sensors—including pressure, proximity, and environmental sensors—to monitor bin usage in real time.

Data collected from these sensors is analysed using lightweight machine learning algorithms to predict bin fill levels. This predictive insight enables the system to generate optimised collection routes based on location, helping reduce unnecessary trips, lower operational costs, and minimise environmental impact.

Developed collaboratively by Andrea Butera, Kuno De-Leeuw Kent, and Mark Ditchburn, the project demonstrates the potential of combining embedded IoT hardware with intelligent software to address real-world urban challenges. The following sections explore the technical architecture, implementation process, and evaluation of the prototype.

Problem Identification

Traditional municipal waste collection relies on fixed schedules and routes that do not reflect actual bin usage. This often leads to inefficiencies such as bins being emptied when only partially full, or remaining full and overflowing between collections.

These issues result in wasted fuel, increased emissions, higher labour costs, and negative public health and environmental outcomes—especially in densely populated areas where waste accumulates quickly.

Another major limitation of current systems is the absence of real-time data and predictive capabilities. Waste management authorities are unable to respond dynamically to varying usage patterns, seasonal changes, or localised events that affect bin usage.

There is a clear need for a smarter, adaptive system that uses data to inform collection timing and routing. IoT-enabled solutions—combined with predictive analytics—can offer this intelligence, providing municipalities with a more efficient, responsive, and environmentally sustainable approach to waste management.

Proposed IoT-based Solution

To address the inefficiencies inherent in traditional waste collection systems, this project proposes a smart bin monitoring and route optimisation solution based on Internet of Things (IoT) technology and Machine Learning. The system is designed to provide real-time data collection, predictive analytics, and intelligent routing for waste collection services.

The core of the solution is built around a Raspberry Pi, which serves as the central processing unit. It is connected to three key sensors:

- Pressure Sensor: Mounted at the base of the bin, this sensor measures the weight of the waste, providing a reliable indication of how much material has accumulated.
- Proximity Sensor: Positioned near the lid, this sensor detects the distance between the top of the waste and the lid itself, offering an additional metric for estimating how full the bin is.
- Temperature and Humidity Sensor: Installed outside the bin, this sensor records ambient weather conditions. Environmental factors can influence waste decomposition rates and may also affect collection strategies.

The data gathered from these sensors is continuously monitored and analysed. Using Machine Learning algorithms, the system predicts the estimated time until each bin reaches full capacity.

These predictions enable proactive alerts to be generated, notifying waste management teams when a bin is likely to require servicing soon.

The system also incorporates a smart routing feature. By analysing the predicted fill levels and total weight of bins within specific postcodes, it generates optimised collection routes.

This approach prioritises areas with heavier waste loads, ensuring resources are allocated efficiently and reducing unnecessary trips.

Overall, the proposed IoT solution enhances waste collection by making it more data-driven, sustainable, and responsive to real-time conditions.

Data Flow and Integration Architecture

Effective data storage and accessibility are fundamental to the performance of any IoT solution, particularly when real-time analysis and decision-making are involved.

In this project, data generated by the pressure, proximity, and temperature and humidity sensors is transmitted and stored using HiveMQ, a cloud-based MQTT broker designed to handle high volumes of sensor data with low latency.

The stored data is then accessed by a custom-built web application, which retrieves the latest sensor values through MQTT subscriptions or via an integrated backend service.

The application provides a user-friendly interface for monitoring current bin conditions and viewing historical data trends.

This centralised approach ensures that the system remains responsive, robust, and capable of supporting advanced features such as route optimisation and environmental analysis. By leveraging HiveMQ, the project benefits from secure, efficient, and real-time data transmission, forming the backbone of the intelligent waste management system.

Design of the Solution

The design of the IoT-based smart bin system was structured into three key phases: data collection, data analysis and model creation, and data visualisation. Each phase was developed with the goal of demonstrating the feasibility and effectiveness of a smart, data-driven waste management solution.

Data Collection

Due to hardware limitations, only a single Raspberry Pi was available for this project. This device was configured to function as one smart bin, collecting real-time data from connected sensors measuring waste weight (via a pressure sensor), fill level (via a proximity sensor), and environmental conditions (via a temperature and humidity sensor).

To simulate a realistic deployment across multiple bins, mock data was generated to represent sensor readings from additional bins located in different postcodes. This approach allowed for comprehensive system testing and demonstration without requiring a full-scale hardware rollout. The combination of live and simulated data provided a diverse dataset for subsequent analysis and modelling.

Data Analysis and Model Creation

The collected data—both real and simulated—was aggregated and processed using a custom algorithm designed to predict when bins in each postcode would require emptying. The model analyses trends in bin fill levels, considering variables such as waste weight, proximity to the lid, and weather conditions, to estimate the time remaining until each bin reaches capacity.

Using this information, the system determines when the total waste load in a specific postcode crosses a predefined threshold, at which point it triggers a collection recommendation. This allows for smarter scheduling of bin collections and ensures that areas with higher waste accumulation are prioritised.

Data Visualisation

To enhance usability, all data is presented visually through an intuitive web application.

Live graphs display key metrics such as bin fill levels, weight progression, and environmental data.

These dynamic visualisations update in real time, providing users with a clear overview of system performance and current bin statuses.

In addition to monitoring, the web app also displays alerts and suggested collection routes, making it a central hub for managing and interacting with the smart bin system.

Bin Prototype (By Kuno de Leeuw-Kent)

Sensing

The sensing process is responsible for collecting real-time physical and environmental data from the smart bin using a set of dedicated sensors connected to a Raspberry Pi. These include:

Hardware Used

Ultrasonic Distance Sensor HC-SR04:

This sensor provides 2cm to 400cm of non-contact measurement functionality with a ranging accuracy that can reach up to 3mm. Each HC-SR04 module includes an ultrasonic transmitter, a receiver and a control circuit.

(https://www.sparkfun.com/ultrasonic-distance-sensor-hc-sr04.html)

Interlink Electronics Force Resisting sensor:

When the sensor is in a neutral state, the circuit remains open, and electricity is unable to pass from one wire to the other.

A spacer is then affixed to this substrate to create a small separation between it and the second substrate, which is coated with a proprietary conductive ink. When force is applied to the sensor, its conductive substrate contacts printed circuit substrate, allowing electricity to flow from one wire to the other.

The amount of electricity that can flow within the circuit depends on the pressure exerted on the FSR, as greater pressure brings more of the conductive material in contact with the wires and ups the electrical output in a predictable way, allowing them to detect changes in force as well.

(https://www.interlinkelectronics.com/force-sensing-resistor)

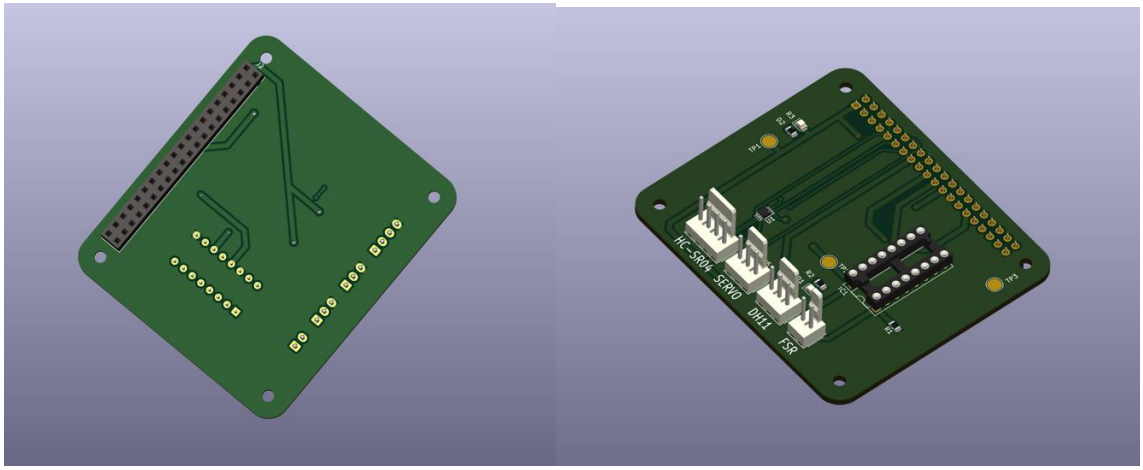
DHT11 Temperature and Humidity Sensor:

DHT11 module featuring a temperature and humidity sensor with a calibrated digital output suitable for use with Arduino, Raspberry Pi and other development boards.

(https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf?srsltid=AfmBOoq-38GUVw2WFBwF1AULK5ilAaBuFfqZ5X4woLUHhhVj8-Uth07q)

Raspberry PI Integration:

To ensure reliable sensor integration, a custom-designed PCB Raspberry Pi shield was developed. This shield performs signal conditioning and provides clearly labelled and convenient connectors for each sensor (HC-SR04, DHT11, FSR), reducing wiring complexity and improving robustness. It also helps isolate and manage signal paths, ensuring accurate readings and easier deployment.

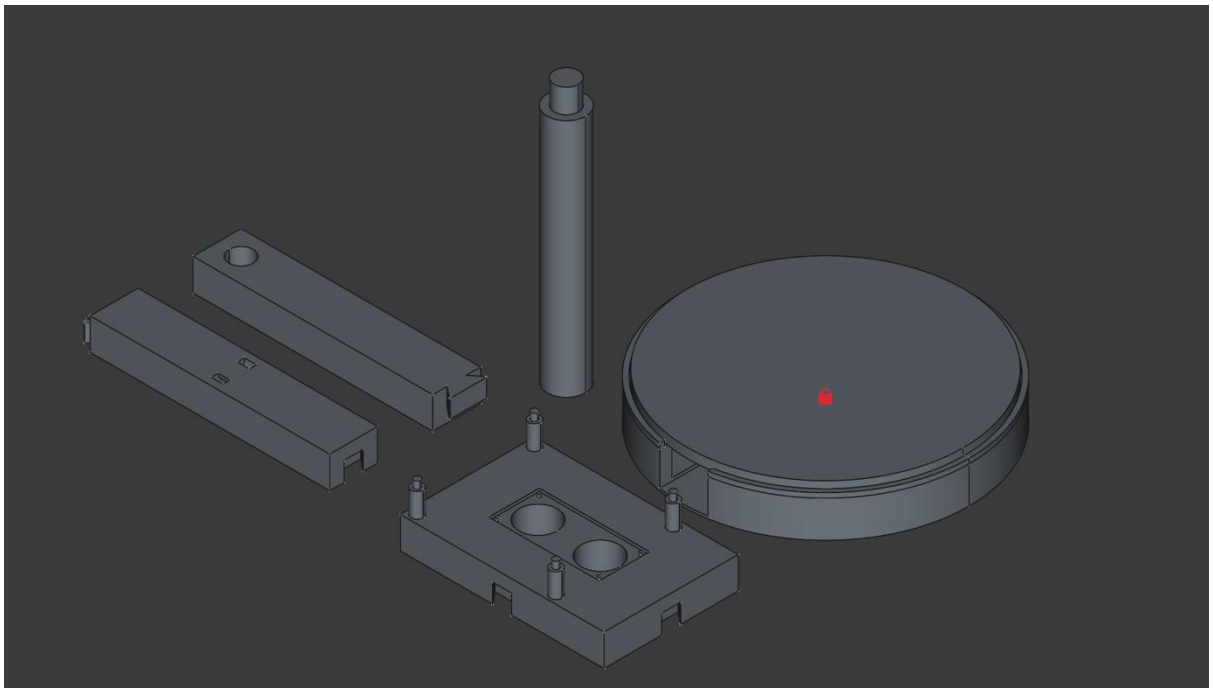


Sensor readings are structured using Python `dataclasses` (`SensorData`, `EnvironmentData`) and collected in two concurrent loops:

- `sensor_loop()`: Sends bin fill level, weight, and calculated density every 5 minutes.
- weather_loop()`: Sends temperature and humidity every 15 minutes.

This sensing process forms the foundation of the system's data pipeline, enabling real-time waste monitoring and environmental tracking.

Physical Build of the System



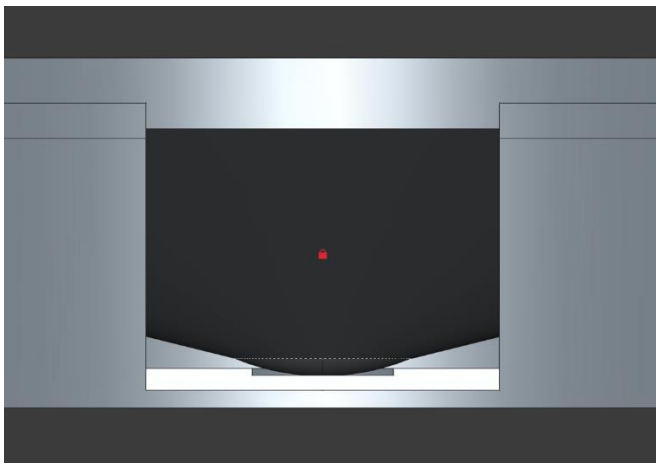
The smart bin's sensing system is housed within a custom-designed modular supports, as illustrated in the exploded CAD view above. The components are designed to be 3D-printable for ease of prototyping and replication, and they ensure secure mounting. The physical build includes the following key parts:

Main Compute Platform (Bottom Middle)

This serves as the foundation for mounting the Raspberry Pi, custom PCB shield and for holding the ultrasonic sensor. The Platform is designed with external dovetail mounting points to securely hold platform in place.

FSR Platform Assembly (Top Right)

This circular base provides a platform for the bin to sit on. The platform ensures even weight distribution from the bin contents above to be concentrated on the FSR. So that no matter the centre of mass of the bin all the force will be registered.



Notice in the image above the top platform concentrates its force on the centre point where FSR would mount.

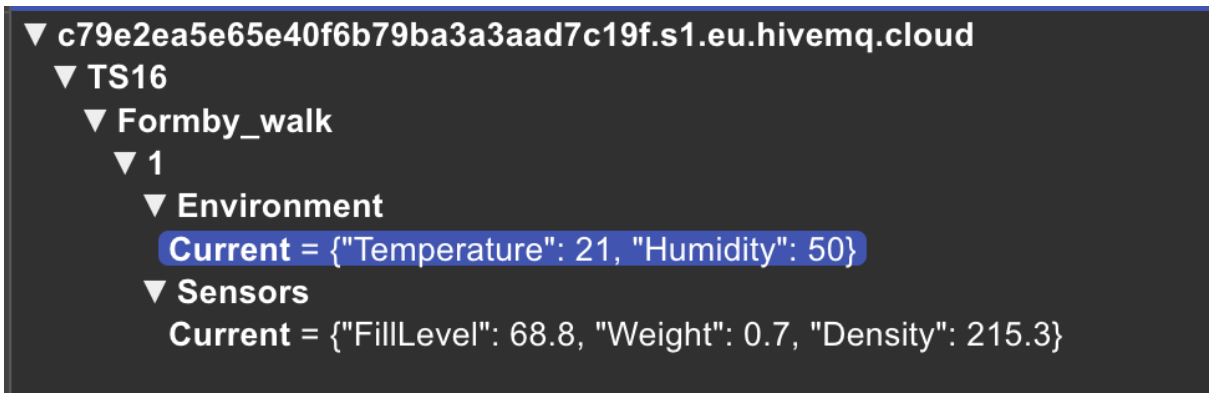
Support Column (Other Item in Photo)

Each part is designed with snap-fit joints for tool-free assembly, making the build process intuitive and modular. This physical design not only supports accurate sensor data acquisition but also ensures durability and ease of deployment.

Transport Layer:

The transport layer facilitates efficient and secure transmission of sensor data from the edge device (Raspberry Pi) to the cloud. This is achieved using the MQTT protocol, which is ideal for IoT due to its low bandwidth requirements and publish/subscribe architecture. These include:

- MQTT Client: Configured with TLS encryption and authentication credentials to connect securely to a public HiveMQ broker.
- Dynamic Topic Generation: MQTT topics are generated based on bin location (`Area/Street/House`) and data type (`Sensors`, `Environment`), allowing scalable multi-bin deployment.
- Real-Time Publishing: JSON payloads of sensor data are published to MQTT topics from within the concurrent loops using `client.publish(...)`.



The system runs MQTT communication alongside sensor reading using Python threading, ensuring real-time data transmission without blocking execution. The transport layer enables seamless integration with cloud services for storage, monitoring, and visualization of bin conditions.

Scaling to production:

To support large-scale, real-world deployment, especially in outdoor or remote urban environments, power efficiency becomes a critical consideration.

The system must be capable of operating on the limited power provided by a compact solar panel mounted on each bin, supported by an energy buffer such as a rechargeable battery

To achieve this, the sensor collection component should be re-engineered to run on a bare-metal microcontroller instead of a Raspberry Pi, significantly reducing idle power consumption.

A custom embedded PCB should be developed to allow precise power management, including the ability to shut down sensors when not actively collecting data.

Efficient battery charging circuits and high-efficiency DC-DC power conversion will further optimize energy use. Additionally, to reduce power demands from network communication, the system should transition from Wi-Fi to a low-power LoRa-based transport layer. (Appl. Sci. 2019)

A central, mains-powered LoRa-to-MQTT bridge would relay data to the cloud, with the placement of LoRa routers tailored to the urban topology and signal coverage needs.

This approach ensures that the smart bin system remains both energy-efficient and easy to deploy without requiring a dedicated power infrastructure.

Web Application, Integration with Data Layer and Cloud Architecture (Led by Mark Ditchburn)

Design of the Web Solution, the Bindicator Web Application

The design of our smart bin solution focuses on creating a modular, scalable, and resilient system that moves sensor data efficiently through the IoT stack, from collection to actionable insights. Below is a recap of the application layers and their integration with the web application. The web application is built is ASP.NET Core.

Device Layer

- Physical sensor array using Raspberry Pi (Raspberry Pi Foundation, 2024).
- Sensors measuring fill level (ultrasound), weight (load cell), temperature, and humidity.
- Data published over MQTT to a cloud broker.

Transport Layer

- MQTT used to transmit sensor data efficiently.
- Ingestion handled by an ASP.NET Core background service (MqttSubscriberService) running inside Azure App Service.

Cloud Platform Layer

- Web application hosted on Azure App Service.
- Sensor and environment readings stored in Azure SQL Database via Entity Framework Core.
- SignalR hub for pushing real-time updates to clients.

Application Layer

- ASP.NET Core MVC web app serving multiple views:
- Dashboard (live bin status)
- Map (bin locations and collection routes)
- Trend (historical sensor and environment data)

Decision-Making Layer

- Lightweight linear prediction (PredictionHelper) estimating future bin fill dates.
- Routing based on urgency (today vs next two weeks).

Key Design Considerations

- Modularity: Services like BinDataService and BinTrendService separate concerns cleanly.
- Scalability: Designed so additional bins or sensors can be added without changing code.
- Resilience: Background service architecture and SignalR minimize downtime risk.
- Real-Time Visibility: Dashboard and charts update instantly as data arrives.
- Fallback Ready: Dummy data seeding is available if no live sensors are active.

Database Schema

Three simple but flexible tables:

SensorReadings

- Postcode, Street, BinNumber
- FillLevel, Weight, Density
- Latitude, Longitude
- Timestamp

EnvironmentReadings

- Postcode, Street, BinNumber
- Temperature, Humidity, LowTemp, HighTemp
- Timestamp

SensorAnalysisReadings

- Postcode, Street, BinNumber
- FillLevel, Weight
- Latitude, Longitude
- Timestamp

Indexes on Postcode, Street, and BinNumber allow efficient querying and trend analysis.

Deployment Pipeline

- GitHub Actions for CI/CD.
- Automatic build, test, and deploy on push to main branch.
- Azure configuration for environment variables (e.g., database connection strings).

Our design prioritises reliability, ease of expansion, and user experience, while keeping costs low to demonstrate a practical, real-world IoT solution suitable for municipal or private-sector adoption.

Ingestion of Sensor Data via ASP.NET Core Background Service

On the application side, we implemented an "MqttSubscriberService", a hosted background service in our ASP.NET Core MVC application. This service maintains a persistent connection to the MQTT broker and listens to incoming messages on specific topics.

Upon receiving a message, the service:

- Parses the JSON payload.
- Stores the reading into a SQL Server database via Entity Framework Core.
- Broadcasts the update to connected dashboard clients using SignalR.

This transport approach allows us to achieve true real-time data flow from the sensor layer to the user interface.

SignalR for Real-Time Updates

To push data to clients instantly, we integrated SignalR hubs into the application:

- The dashboard view listens for "ReceiveBinUpdate" messages to refresh the bin status table.
- The trend view listens for "ReceiveTrendUpdate", enabling live chart updates.

This event-driven model eliminates the need for polling and enhances the user experience by reflecting changes as they happen.

Robustness and Reliability

To ensure resilience:

- The MQTT service runs as a hosted service, separate from the web thread.
- Messages are processed in a try-catch block to avoid crashing the subscriber.
- Data is validated before saving to prevent corrupt or malformed entries.

This layered approach ensures consistent data delivery even if one part of the system restarts or fails.

Cloud Integration

The cloud integration layer bridges our transport system with storage, visualisation, and decision-making components. For our smart bin solution, we deployed the application to Microsoft Azure, leveraging several platform-as-a-service (PaaS) tools.

Hosting on Azure App Service

The main web application is hosted on Azure App Service (Microsoft, 2024a). This platform was selected for its ease of deployment, scalability, and seamless integration with .NET Core applications.

We configured a CI/CD pipeline using GitHub Actions (GitHub, 2024), which builds and deploys the app on push to the main branch. This automated workflow ensures that cloud deployment is:

- Repeatable and consistent.
- Fast, with minimal manual steps.
- Safe, thanks to environment-specific configuration.

Database: Azure SQL

Sensor readings and environmental data are stored in Azure SQL Database (Microsoft, 2024b). Entity Framework Core is used to manage migrations (Microsoft, 2024d) and access data in a code-first manner. The schema includes two key tables:

- SensorReadings (for bin fill, weight, and location)
- EnvironmentReadings (for temperature, humidity, and conditions)
- SensorAnalyisReadings (for demonstration of long-time machine learning data analysis)

Azure SQL offers scalability, managed backups, and monitoring features ideal for IoT data.

Real-Time Communication with SignalR

To provide live updates, the app integrates SignalR, which is configured and hosted within the same Azure App Service. It enables the dashboard and trend pages to respond in real-time to incoming sensor data without page refreshes.

This is critical for:

- Updating fill levels on the dashboard.
- Redrawing charts on the trend page as new data arrives.
- Alerting users to warnings or sudden changes.

MQTT and Azure Interaction

Although our project uses a public MQTT broker (HiveMQ 2024) the ingestion system is tightly integrated into the Azure-hosted app through a background hosted service. This

means all data ingestion, validation, and storage happens within the cloud, reducing latency and simplifying monitoring.

Cost, Scalability, and Practicality

The web-app uses Azure's free student tier and scalable services; we maintained a cost-effective prototype while simulating a realistic cloud deployment.

The cloud platform enables:

- Scalability to thousands of bins with minimal changes.
- Easy migration to production infrastructure if adopted by a local authority.
- Future integration with Azure Maps, IoT Hub, and other Azure-native services.

This integration ensures our system is not only functional, but also portable, secure, and ready for scale.

Web App Decision-Making

The decision-making component of our IoT system focuses on using recent sensor data to predict when a bin will become full, supporting timely waste collection and route optimisation.

Fill-Level Prediction Strategy

Rather than implementing a complex machine learning model, we adopted a lightweight linear prediction approach using a helper method called PredictionHelper. This method relies only on the most recent three fill level readings for each bin.

- Using basic linear estimation, the helper:
- Checks that fill levels are increasing consistently.
- Calculates the rate of change (fill per day).
- Estimates how many days remain until the bin reaches 100%.
- Computes the predicted full date.

This prediction is embedded into both the SensorDataViewModel and BinTrendViewModel, making it accessible across the map and trend views.

This method is:

- Fast and executes in milliseconds.
- Robust to noise or gaps in data.
- Appropriate for real-time streaming data, where sensor quality or consistency may vary.

It also ensures that the app performs consistently with live data, without requiring training, tuning, or retraining of a model.

Integration with Collection Planning

Predicted dates are used on the Map view to:

- Identify priority bins that will be full within 24 hours.
- Plan a secondary collection route for bins expected to be full within 14 days.
- Calculate total weight and wagons required, using the sum of projected bin weights.

The utilise the Azure Maps service, with a map embedded in this view. The system automatically flags which bins need urgent attention creates the most efficient route for collections via the Atlas API.

Future Enhancements

This prediction approach is modular and can be upgraded to use:

- Rolling average smoothing or moving windows.
- More advanced linear regression over 5+ days.
- A machine learning model (e.g., time series forecasting) if higher accuracy is needed.

For now, our simple approach meets the needs of a responsive, real-time IoT dashboard while being transparent and easy to validate.

Historical Data Analysis using Machine Learning

As part of enhancing our smart bin application, we integrated a machine learning feature that evaluates long-term bin usage trends using historical sensor data. This analysis allows decision-makers to assess how effectively each bin is used and whether adjustments to size or collection frequency are needed. Currently the prototype analysis feature focuses on weigh and fullness.

Objective

Rather than predicting when a bin will be full on a specific date, this analysis focuses on how consistently a bin reaches high fill levels over time. It enables data-driven decisions about resource allocation and bin sizing by highlighting overused or underused bins.

Data Preparation

We used a synthetic dataset generated via the application's SeedAnalysisDataAsync method. This creates realistic data for a set of bins over a 28-day period, simulating varying fill levels and weight accumulation with collection resets every 14 days.

Each record includes:

- Postcode, Street, BinNumber

- FillLevel, Weight, Timestamp

The data is downloaded via the DownloadCsv action and analysed offline using a Python script.

Analysis Methodology

The Python script uses pandas, scikit-learn, and numpy to:

- Group historical readings by bin.
- Train a linear regression model on fill level over time.
- Predict daily fill levels, calculate average predicted fill per day, and count the number of days each bin was "near full" (≥85%).
- Generate tailored usage recommendations based on thresholds and usage patterns.

Each bin receives one of the following insights:

- ⚠️ Bin has been near full for most of the period – Consider larger bin or more frequent collection.
- ℹ️ Bin rarely exceeds 30% fill – Consider a smaller bin.
- ✅ Bin usage is balanced – No action required.

Output Integration

The script outputs a structured JSON file (analysis_output.json) which is uploaded through the UploadAnalysisJson endpoint in the web app. The ASP.NET Core MVC view then displays the results in a user-friendly format, with badges indicating recommended actions.

Sample output:

{

"bin_id": 3,
"average_fill_percent": 82.75,
"near_full_days": 17,
"total_days_monitored": 28,
"recommendation": "Bin has been near full for 17 out of 28 days monitored. ⚠️ Consider a larger bin or more frequent collection."

}

This approach focuses on bin utilisation patterns rather than immediate predictions. It provides operational insights that are:

- Easy to interpret
- Backed by linear regression
- Helpful for long-term planning

The simplicity of the model ensures quick execution, explainability, and robustness in a prototype or production environment.

Future Enhancements

- Automate daily analysis using Azure Functions.
- Include environmental context (e.g. weather, events) in future models.
- Include analysis of remaining sensor data using suitable algorithms.
- Link with route planning features.

Considerations for Scaling to Production

Cloud Infrastructure Resilience

Deploy the application using Azure App Service with multi-region redundancy managed via Azure Traffic Manager to maintain uptime during outages.

Migrate real-time messaging to Azure SignalR Service, allowing seamless scaling of client connections.

Data Storage & Processing

- Extend from Azure SQL to Azure SQL Elastic Pools or Azure Cosmos DB to handle increased read/write loads and globally distributed data.
- Partition data by geographic region (e.g., postcode or bin cluster) for improved performance and maintainability.

Secure and Reliable Data Ingestion

- Transition to Azure IoT Hub for secure device provisioning (Microsoft, 2024f), telemetry routing, and lifecycle management.
- Introduce Azure Event Grid or Azure Service Bus for loosely coupled message ingestion (Microsoft, 2024g) pipelines, ensuring message durability and reducing system bottlenecks.

DevOps and Deployment Automation

- Maintain GitHub Actions for CI/CD with staging slots in Azure App Service to support zero-downtime deployments.
- Secure secrets and settings via Azure Key Vault and automate environment provisioning using infrastructure-as-code (e.g., ARM/Bicep).

Observability and Diagnostics

- Use Azure Monitor, Log Analytics, and Application Insights to track telemetry, application health, and detect anomalies.
- Implement dashboards and alerting to monitor ingestion failures, latency spikes, or downtime risks.

Security and Compliance

- Enforce TLS across all communication channels (MQTT, APIs, SignalR).
- Enable role-based access control (RBAC) for administrative functions and user roles.
- Encrypt data at rest and in transit; define data retention policies aligned with GDPR for personal or locational data.

Machine Learning Integration

- To enhance the intelligence of the system beyond simple linear prediction:
- Use Azure Machine Learning (Azure ML) or AutoML to train and deploy models (Microsoft, 2024h) predicting bin fill times, unusual usage patterns, or seasonal trends.
- Expose trained models via REST APIs, enabling predictions to be consumed by the ASP.NET Core application in real time.
- Automate periodic retraining using Azure ML Pipelines or Azure Functions (Microsoft, 2024i), sourcing data from historical fill and environmental records.
- Store model outputs and decisions in the database for auditing, explainability, and dashboard integration.
- Optionally integrate Azure Synapse Analytics or Power BI Embedded for advanced reporting (Microsoft, 2024j) (Microsoft, 2024k).

Web App Performance and UX

- Optimise frontend responsiveness using asynchronous data loading, signal throttling, and progressive rendering.
- Continue to use Chart.js (Chart.js, 2024) or D3.js with lazy-loaded data to reduce initial page load time.
- Provide offline fallback and cache-first strategies using service workers (if later upgraded to a PWA model).

Source Code and Live Web Application

The source code is included in the submission. It can be run locally with dummy data seeded from within the application.

The live web application can be viewed at: https://bindicator-web.azurewebsites.net/

The Python script for the Analysis page can be found inside the BindicatorPythonFolder.

Run this script with your chosen Python compiler, using the dummy data generated and output to an analysis_data.csv, from the button on the Analysis page.

Upload the resulting JSON file, using the uploader on the same page, to see long term predictions.

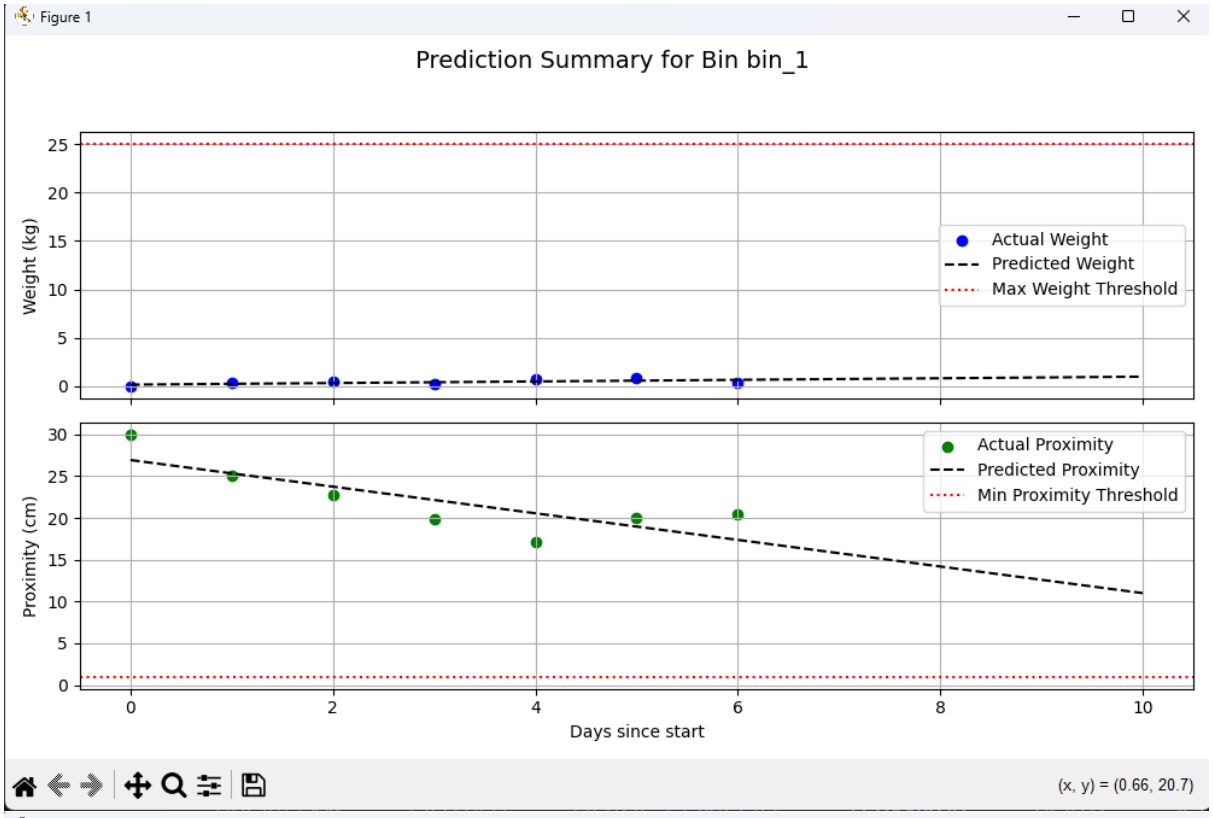
Machine Learning Visualisation and Decision Making (Led by Andrea Butera)

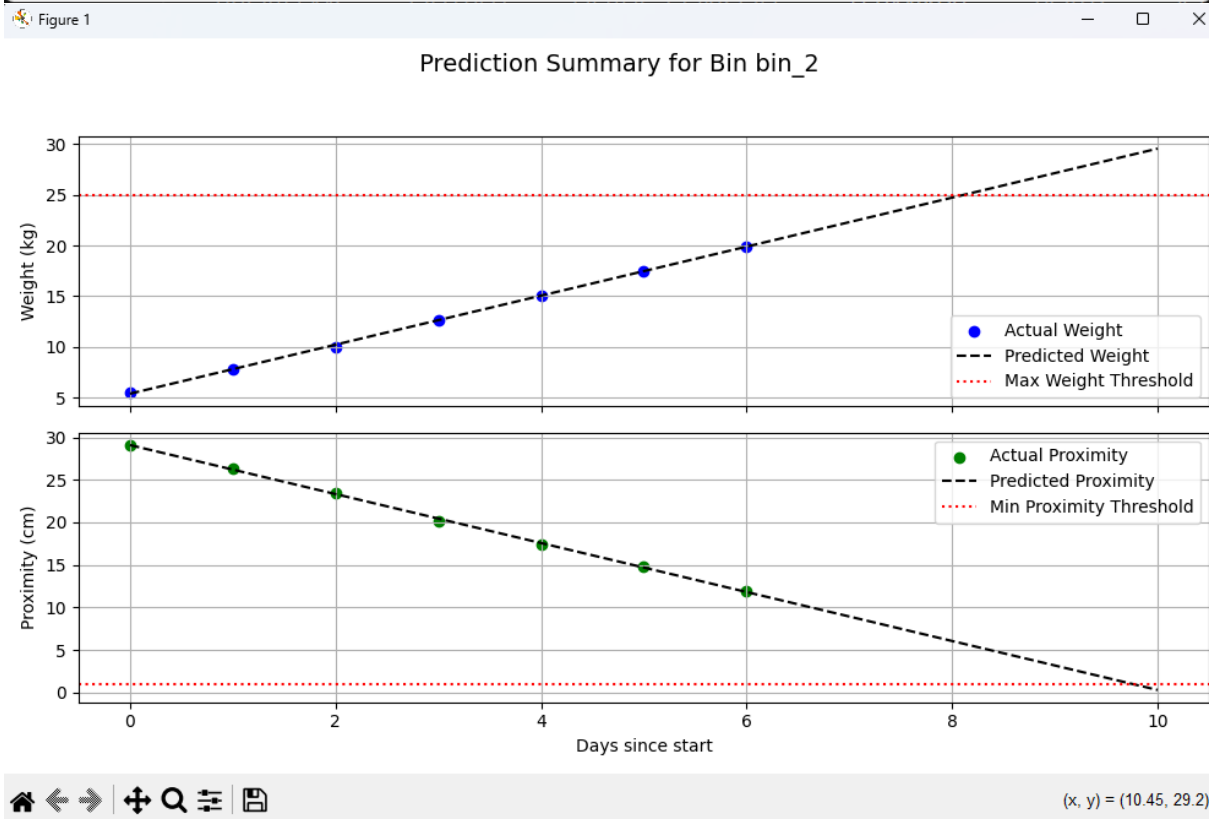
The initial script idea is to predict when bins will reach full capacity based on historical data. Predictions are made using linear regression to estimate the fill rate and proximity to the bin lid over time.

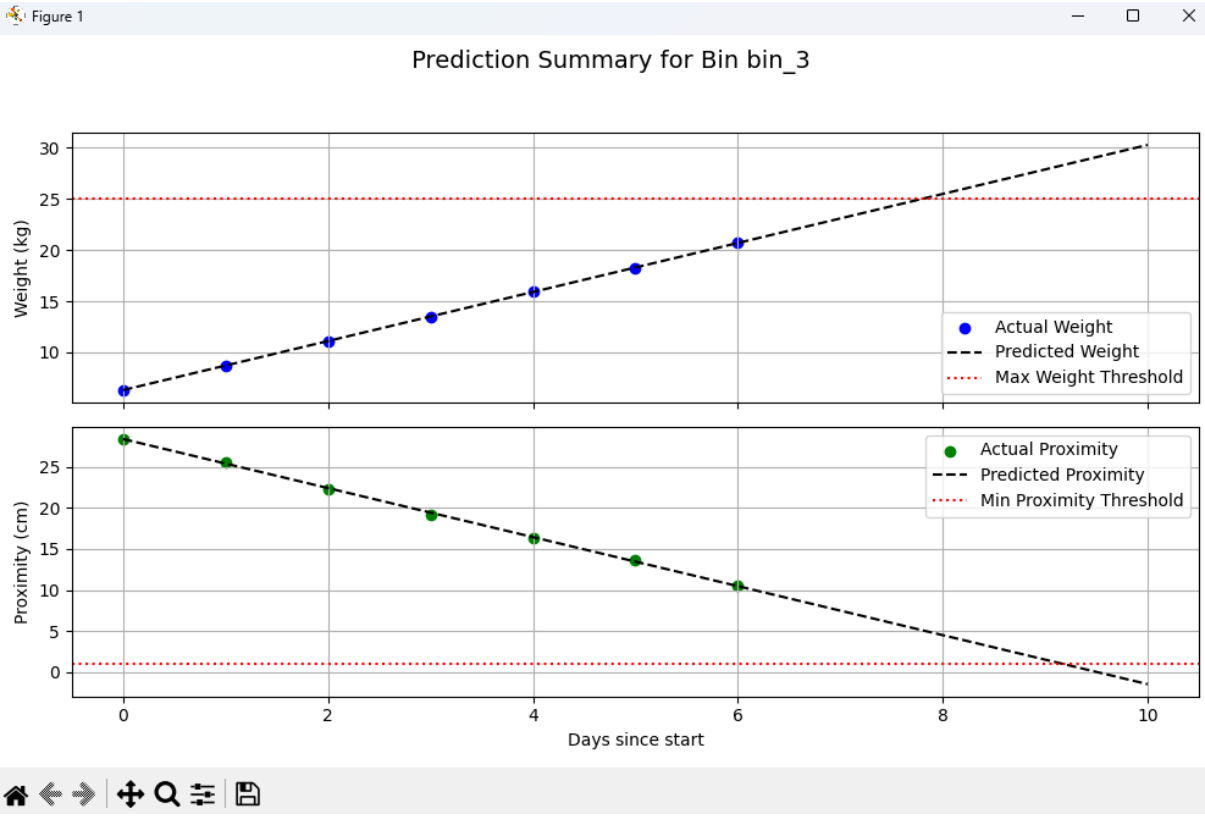
This algorithm turned out to be too complex to incorporate to our online webpage, but we consider implementing it in future iterations.

We have used mock data to train the model as we only had one raspberry Pi available. These are the results of the trained model before the webapp implementation:

This document was classified as: NOT WORK RELATED





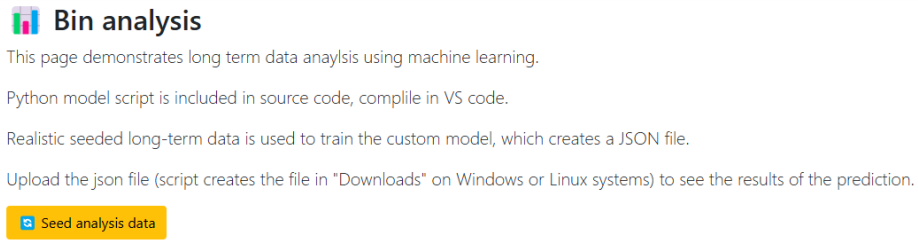



```
 bin_1:
- Current weight: 0.3 kg → Predicted fill rate: 0.08 kg/day.
- Est. time to full (25kg): 300.7 days.
- Current proximity: 20.4 cm → Predicted change rate: -1.59 cm/day.
- Est. time to lid closure (1 cm): 12.2 days.
- Summary:  Collection not yet urgent.

 bin_2:
- Current weight: 19.9 kg → Predicted fill rate: 2.42 kg/day.
- Est. time to full (25kg): 2.1 days.
- Current proximity: 11.9 cm → Predicted change rate: -2.88 cm/day.
- Est. time to lid closure (1 cm): 3.8 days.
- Summary:  Needs urgent collection.

 bin_3:
- Current weight: 20.7 kg → Predicted fill rate: 2.40 kg/day.
- Est. time to full (25kg): 1.8 days.
- Current proximity: 10.5 cm → Predicted change rate: -2.99 cm/day.
- Est. time to lid closure (1 cm): 3.2 days.
- Summary:  Needs urgent collection.
```

- The blue points represent the actual observed weight.
- The green points represent the actual proximity values.
- The black lines show the predicted values for the related measurement.

- The red dotted lines represent the maximum weight and proximity allowed before sending a warning.
- The actual vs predicted values of allow us to assess how well the model is performing.

This has then evolved into the analysis function we now have on the website. Due to the restrictions of only having a single raspberry Pi able of transmitting data, in this page we have a function that creates random mock data and downloads it to our computer.



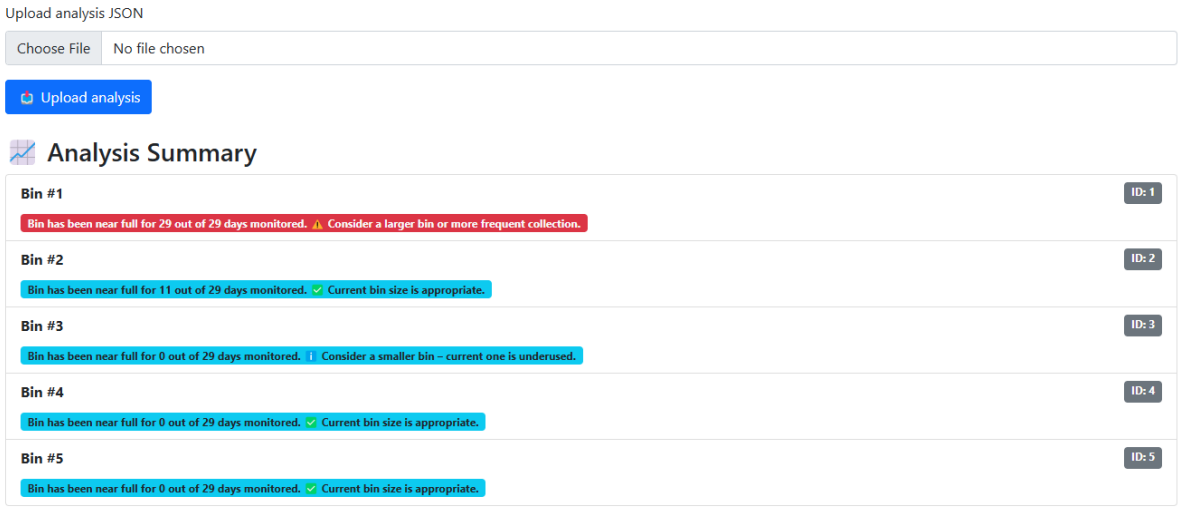
This mock csv file gets then fed into a python script that uses a linear regression algorithm and creates a json file in the download folder based on the data.


```
[
    {
        "bin_id": 1,
        "average_fill_percent": 90.0,
        "near_full_days": 29,
        "total_days_monitored": 29,
        "recommendation": "Bin has been near full for 29 out of 29 days monitored. \u26a0\ufe0f Consider a larger bin or more frequent
collection."
    },
    {
        "bin_id": 2,
        "average_fill_percent": 77.59,
        "near_full_days": 11,
        "total_days_monitored": 29,
        "recommendation": "Bin has been near full for 11 out of 29 days monitored. \u2705 Current bin size is appropriate."
    },
    {
        "bin_id": 3,
        "average_fill_percent": 25.0,
        "near_full_days": 0,
        "total_days_monitored": 29,
        "recommendation": "Bin has been near full for 0 out of 29 days monitored. \u2139\ufe0f Consider a smaller bin \u2013 current on
e is underused."
    },
```


This is then fed back to the website, which is going to show recommendations based on the data collected.

If the bin has at a fill level of 85% or higher for longer than 50% of the analysed period, the website will then recommend getting a bigger bin.

If it has been at a level below 30% for longer than the same period, it will recommend using a smaller bin; otherwise, it will just ascertain that the bin size is appropriate for the use.



Video Demonstration

A video demonstration can be find attached to the submission for the group.

Conclusion

This project demonstrates the potential of integrating Internet of Things (IoT) technology with Machine Learning to create a more efficient, responsive, and sustainable waste management system.

By combining sensor data from a Raspberry Pi-based prototype with simulated inputs, the team successfully developed a smart bin solution capable of predicting fill times and optimising collection routes based on real-time and historical data.

The system leverages a pressure sensor, a proximity sensor, and an external temperature and humidity sensor to gather comprehensive insights into bin usage and environmental conditions.

Data is transmitted and stored via HiveMQ, then accessed and analysed through a web application that provides live visualisations and intelligent alerts.

Although limited by hardware constraints, the project effectively used simulated data to demonstrate the full functionality of a multi-bin deployment.

The resulting predictive model and web interface showcase a practical and scalable approach to tackling inefficiencies in traditional waste collection methods.

We hope this the project highlights the power of IoT and data-driven decision making in addressing everyday urban challenges.

With further development and deployment, the system has the potential to reduce operational costs, minimise environmental impact, and improve the quality of public sanitation services.

References

Appl. Sci. 2019: A Comprehensive Study of the Use of LoRa in the Development of Smart Cities Available at: https://www.mdpi.com/2076-3417/9/22/4753 (Accessed: 29 April 2025)

HiveMQ (2024) *MQTT Essentials: A Lightweight IoT Messaging Protocol*. Available at: https://www.hivemq.com/mqtt (Accessed: 29 April 2025).

Learn: Machine learning in python - scikit-learn 0.16.1 documentation scikit. Available at: https://scikit-learn.org/ (Accessed: 29 May 2025).

Microsoft (2024a) *Azure App Service documentation*. Available at: https://learn.microsoft.com/en-us/azure/app-service/ (Accessed: 29 April 2025).

Microsoft (2024b) *Azure SQL Database documentation*. Available at: https://learn.microsoft.com/en-us/azure/azure-sql/database/ (Accessed: 29 April 2025).

Microsoft (2024c) *Azure SignalR Service documentation*. Available at: https://learn.microsoft.com/en-us/azure/azure-signalr/signalr-overview (Accessed: 29 April 2025).

Microsoft (2024d) *Entity Framework Core documentation*. Available at: https://learn.microsoft.com/en-us/ef/core/ (Accessed: 29 April 2025).

Microsoft (2024e) *Azure Monitor and Application Insights documentation*. Available at: https://learn.microsoft.com/en-us/azure/azure-monitor/overview (Accessed: 29 April 2025).

Microsoft (2024f) *Azure IoT Hub documentation*. Available at: https://learn.microsoft.com/en-us/azure/iot-hub/ (Accessed: 29 April 2025).

Microsoft (2024g) *Azure messaging services overview*. Available at: https://learn.microsoft.com/en-us/azure/architecture/guide/technology-choices/messaging (Accessed: 29 April 2025).

Microsoft (2024h) *Azure Machine Learning documentation*. Available at: https://learn.microsoft.com/en-us/azure/machine-learning/ (Accessed: 29 April 2025).

Microsoft (2024i) *Azure Functions documentation*. Available at: https://learn.microsoft.com/en-us/azure/azure-functions/ (Accessed: 29 April 2025).

Microsoft (2024j) *Azure Synapse Analytics documentation*. Available at: https://learn.microsoft.com/en-us/azure/synapse-analytics/ (Accessed: 29 April 2025).

Microsoft (2024k) *Power BI Embedded documentation*. Available at: https://learn.microsoft.com/en-us/power-bi/developer/embedded/ (Accessed: 29 April 2025).

Pandas documentation. Available at: https://pandas.pydata.org/ (Accessed: 29 May 2025).

Raspberry Pi Foundation (2024) *Raspberry Pi Documentation*. Available at: https://www.raspberrypi.com/documentation (Accessed: 29 April 2025).

Welcome to Python.org (no date) Python.org. Available at: https://www.python.org/ (Accessed: 29 May 2025).