Project Overview
Over the course of the semester in PANSLAB, I developed a real-time environmental monitoring system integrating both air quality and vibration sensing. The system evolved from early experiments with machine learning models to a robust, interactive frontend paired with a live Python backend. The overarching goal was to develop a tool that visualizes patterns in indoor spaces using sensors, with a focus on both static and live data processing. I also had practice using Iris datasets from earlier in the semester.

Early Semester Progress – ML Foundations
In the first part of the semester, I explored data classification and signal analysis through Python scripts. The script vib.py extracted basic features from vibration data and helped shape my approach to waveform normalization and live data filtering. I also created two scripts, iris.py and irisCross.py, to work with the Iris dataset, implementing decision tree classification and K-Fold cross-validation. Users can fetch random samples until they are exhausted. These earlier projects output in the console terminal. These projects strengthened my understanding of machine learning and performance evaluation, skills that would later inform how I interpreted sensor outputs in real time.

Main System – Real-Time Dashboard & Visualization
Building on these foundations, I developed a cross-platform dashboard using React Native and Expo. The app displays four air quality sensor nodes over a blueprint of a home. The first four graphs are static graphs with JSON format. Each node is draggable and renameable, providing a flexible, interactive interface. However, these changes can only be done in edit mode. The app features two primary data tabs: one for historical CSV-based graphing and another for live vibration waveform visualization.

The terminal tab is designed to show live vibration data collected via UDP or WebSocket. This data is normalized using Z-score filtering and displayed as a waveform. I also implemented PNG rendering of the waveform using matplotlib, which the backend sends to the frontend via Base64. Although PNG display in the app is currently inconsistent due to frontend rendering issues, the live matplotlib graph still successfully displays locally the live waveform once data collection is complete. This ensures that users can still view and analyze the signal structure even if the PNG preview fails to load.

Backend & Signal Processing
The backend is composed of two Python scripts. `capture_udp_to_csv.py` logs real-time UDP sensor data into CSV files, while `plot_waveform.py` plots the waveform in real time as a PNG and encodes the graph image for frontend transmission. This combination enables both live monitoring and retrospective graphing using saved data. The backend design was influenced by the earlier ML work and is structured to support signal normalization, flexible input handling, and continuous visualization.

Challenges & Learnings
Challenges for me included managing UDP limitations on mobile platforms, ensuring reliable WebSocket communication, and fixing image rendering in React Native. I also encountered difficulties in syncing

real-time data feeds with frontend display updates. Despite these challenges, I gained valuable experience in mobile development, cross-platform UI design, real-time data flow, and system integration.

Conclusion

These semester assignments and projects improved my skills in machine learning, signal processing, and real-time application design. I developed a sensing platform that started from offline modeling and practice with datasets to a live, user-interactive system. This helped expand my technical abilities and provided a strong learning base for my future work.