Sprint Report

PORTFOLIO TASK 2/3/4

Unit code: EAT40005
Unit Name: Engineering Project A
Submission date:

[This is a template, remove any text within third brackets before submission]

Student Name	Student Id	Statement of contribution to the report
Dang Khoa Le	103844421	

ACKNOWLEDGMENT OF COUNTRY

[Add a statement on acknowledgment of the country.]

Each team member identifies:

> the Traditional Owners of the land they lived on while completing this work (if living in Australia)…

1. SPRINT PLAN

[A detailed plan for the sprint. A list of items selected for the sprints should be presented in a sprint backlog along with the estimation. This section should give a clear pisture of what team wanted to deliver at the end of the sprint.]

2. SPRINT PROGRESS

[This section describes the progress of the sprint with relevant evidence (e.g. screenshots of repository, task board etc). The evidence should indicate team members' contributions. The evidence can be discussed with the supervisor.]

3. SPRINT DEMONSTRATION

I. Week 8 Demonstration:

Overview: Week 8 was originally intended for physical OBD-II data collection using the ELM327 device. Due to delays in obtaining the device and vehicle access, physical logging was rescheduled. Instead, the team focused on preparing detailed setups, installation guides, and comprehensive strategies for data structure and logging. Initial tests were performed using mobile applications (Car Scanner, Torque) with data transmission to a Raspberry Pi-based logging server.

Client Feedback: Client acknowledged the team's proactive planning during the hardware delay and approved the interim logging approach and preparation steps.

Action:

- Used Car Scanner and Torque App for initial logging.
- Setup Raspberry Pi HTTP server with MySQL database for logging.
- Defined sampling configurations (2 Hz, 500 ms intervals), essential sensor parameters, and rolling-window processing plan.

II. Week 9 Demonstration:

Overview: Successfully conducted physical OBD-II data collection using an ELM327 device on a 2005 Subaru Liberty. The 'obd_logger.py' script, built using the python-OBD library, managed real-time data collection. It categorized PIDs into high-frequency (RPM, SPEED, THROTTLE_POS) and low-frequency groups, polled cyclically. Custom CLI input allowed on-the-fly tagging with 'driving_style' and 'road_type'. Data was written to CSV with timestamps.

Key Technical Features of obd_logger.py:

- Modular PID polling using obd.commands, split into high-frequency (every 0.5s) and low-frequency (45s rotation per group).
- CLI-based real-time tagging using stdin for driving context updates like "style aggressive".
- Dynamic CSV file creation using session-based timestamps.
- Intelligent fallback and chunking of low-frequency PIDs to reduce lag.
- Logs sensor data, timestamp, driving_style, and road_type per entry.

Cleaning Pipeline from W9.ipynb:

- **Step 1:** Drop constant or redundant features (nunique() <= 1, high correlation).
- **Step 2:** Remove columns with >70% nulls and rows with >40% nulls.
- **Step 3:** Validate and parse timestamps.
- **Step 4:** Replace placeholders like 255, -22, -40 with NaN.
- **Step 5:** Impute missing values using mean.
- **Step 6:** Normalize numeric features using MinMaxScaler.
- **Step 7:** Add features: AVG_ENGINE_LOAD, TEMP_MEAN, AIRFLOW_PER_RPM.
- **Step 8:** Visualize trends, generate heatmaps, and identify outliers.
- **Exploratory Insights:**
- Detected strong correlation between SPEED, RPM, and ENGINE_LOAD.
- Time-index plots revealed driving behavior transitions.
- Highlighted idle periods and acceleration bursts for potential driving style labeling.

Client Feedback: Client was pleased with the hands-on data collection and modular Python script. Suggested refinements in missing value strategies and prompted us to consider dropping or merging highly correlated features.

Action:

- Applied standard cleaning pipeline (placeholder filtering, MinMax scaling, timestamp validation).
- Explored data patterns and confirmed feature correlations.
- Proposed merging of correlated features like RPM and SPEED or retaining only one.III.

III. Week 10 Demonstration:

Overview: A full cloud-based FastAPI server (app.py) was developed, containerized via Docker, and hosted on Hugging Face Spaces (free-tier).
Accessible via: https://huggingface.co/spaces/BinKhoaLe1812/OBD_Logger.

This system automates the ingestion, cleaning, and storage of OBD-II logs either from streaming (via POST /ingest) or batch upload (/upload-csv/). Cleaned outputs are saved to Google Drive.

Key Technical Features of app.py:

- Uses FastAPI + Pydantic for real-time validation of OBDEntry data.
- Implements BackgroundTasks to asynchronously process logs post-ingestion.
- Cleaning logic includes:
    - Dropping constant/empty columns.
    - Replacing placeholder values (255, -40, -22) with NaN.
    - Outlier filtering: RPM kept between 100–6000.
    - Median imputation of missing values.
    - MinMax scaling of all numeric columns.
    - Feature engineering:
        - AVG_ENGINE_LOAD = mean of ENGINE_LOAD & ABSOLUTE_LOAD.
        - TEMP_MEAN = mean of INTAKE_TEMP, COOLANT_TEMP, OIL_TEMP.
        - AIRFLOW_PER_RPM = MAF / RPM.
- CSVs are stored in timestamped directories and uploaded to Google Drive via googleapiclient.

Client Feedback: Client appreciated the centralized automation and encouraged labeling driving style based on acceleration metrics instead of manual tagging. Suggested Streamlit for building an interactive frontend.

Action:

- Finalized cloud logging pipeline.
- Logged and processed CSVs uploaded from Raspberry Pi.
- Planned Streamlit (client recommended) for dashboard and automated label generation or develop full web-stack (HTML, CSS, JS) for robustness.

4. SPRINT REVIEW (CRITICAL REVIEW OF THE PRODUCT)

I. What Was Demonstrated to the Client:

- **Week 8:** Setup preparations for logging pipeline; interim mobile logging solutions with custom Raspberry Pi server.
- **Week 9:** Physical data logging with Python-based script (obd_logger.py), robust data cleaning and exploratory analysis pipeline (W9.ipynb), detailed correlation analysis.
- **Week 10:** Cloud-based automated FastAPI server deployment (hosted on Hugging Face Spaces), streamlined data processing, and automated upload to Google Drive.

II. Feedback Client Provided:

- **Week 8 Feedback:** Appreciated team's flexibility in dealing with hardware delays; supported interim logging approach.

- **Week 9 Feedback:** Positive on data logging and comprehensive data cleaning; recommended further review of missing data handling and addressing highly correlated features – the team insists on merging highly-correlated features.
- **Week 10 Feedback:** Highly satisfied with automated data ingestion and cleaning via the cloud pipeline. Suggested leveraging Streamlit for UI visualization and automatic labeling of driving style from acceleration data.

III. Critical Analysis of Progress and Feedback:

Progress Against Objectives: The team showed effective adaptation to initial delays in Week 8, rapidly transitioning into successful physical logging and analysis in Week 9. By Week 10, we had built a fully automated cloud-based pipeline significantly exceeding original project milestones.

Team's Adaptation and Improvement:

- **Khoa:** Undergone data cleaning strategies and data science tasks. Create FastAPI server for automated workflow.
- **Dale: Barsat:**
- **Sadman:**

Effective integration of feedback resulted in professional presentations, refined technical pipelines, and robust automation of data handling.

Challenges Faced:

- Initial hardware and vehicle access delays necessitated adjustments to the project timeline.
- Technical complexity in real-time data logging and automated pipeline creation was addressed successfully.
- Ensuring consistent data cleaning and handling missing or corrupted values required iterative refinements.

Key Achievements:

- Successful physical OBD-II data logging.
- Implementation of an effective Python logging script (obd_logger.py) supporting comprehensive PID management.
- Robust data cleaning pipeline refined and automated (via W9.ipynb).
- Deployment of a scalable, automated FastAPI service for real-time data ingestion, cleaning, and storage, significantly improving workflow efficiency.

Reflection: Despite early setbacks, we quickly refocused, making considerable progress in physical data collection, cleaning methodologies, and creating automated workflows. Client feedback remains highly positive, emphasizing continued excellence in pipeline automation and visualization. The next sprint should focus on automated driving style labeling and UI development for improved client interaction and insights.

5. RETROSPECT (CRITICAL REVIEW OF THE PROCESS)

[This section should present a summary on how the overall team process was followed. Challenges, bottlenecks found during the sprint, how dealt with etc.]

6. LESSONS LEARNED (CRITICAL REVIEW OF SPRINT ONE EXPERIENCE AND FUTURE PLAN)

[Add a list of lessons learned from working together on the particular project in the group. List a clear set of to-do/recommendations for improvement in the future. It is strongly recommended that you review and reflect (based on experience) on your team plan, communication plan, quality plan, and risk registry. Teams should be referring to the ethical considerations for any data collection, and/or any development that took place in the sprint.]