Week 9 Progress – OBD-II Physical Data Logging & Cleaning Pipeline

This presentation covers the SkyLedge Predictive Maintenance project's Week 9 progress. The focus was on physical data collection from a real vehicle using an ELM327 OBD-II adapter, building a Python pipeline for real-time logging via Raspberry Pi, merging and cleaning three datasets, and generating insights for modeling and feature engineering.

OBD-II Physical Data Collection Collection

Hardware & Vehicle

Used ELM327 16-pin Wi-Fi adapter connected to a team Dale's 2005 Subaru Liberty.

Data Files & Duration

- Data is stored in csv
- One file is created per session
- Contains all parametres

Sampling Challenges

PID's can only be sampled one at a time, no parallellisation, which presented an unknown challenge, as sample frequency was initially very slow (11 seconds)

Data Logging

Custom Script Features

Logs high-frequency PIDs every 1.5 seconds, such as RPM's Speed, Oil pressure etc. Low frequency PIDs every 2 minutes, exporting timestamped CSV files with headers.

Robust & Modular

Custom script built on top of ODB library to ensure we receive decoded data.

```python
def main():
    BASE_LOG_INTERVAL = 1.5  # for high frequency data
    LOW_FREQUENCY_GROUP_POLL_INTERVAL = 120.0  # Interval in seconds to poll one
    NUM_LOW_FREQUENCY_GROUPS = 3

    # Prepare Low-Frequency PID groups
    low_frequency_pid_groups = []
    if LOW_FREQUENCY_PIDS_POOL:
        chunk_size = (len(LOW_FREQUENCY_PIDS_POOL) + NUM_LOW_FREQUENCY_GROUPS -
        for i in range(0, len(LOW_FREQUENCY_PIDS_POOL), chunk_size):
            low_frequency_pid_groups.append(LOW_FREQUENCY_PIDS_POOL[i:i + chunk_

    if not low_frequency_pid_groups: # Handle case with no LF PIDs
        low_frequency_pid_groups.append([])
        NUM_LOW_FREQUENCY_GROUPS = 1

    last_low_frequency_group_poll_time = time.monotonic()
    current_low_frequency_group_index = 0

    current_pid_values = {pid.name: '' for pid in ALL_PIDS_TO_LOG}

    log_dir_path = os.path.join(os.getcwd(), LOG_SUBDIRECTORY)

    try:
        os.makedirs(log_dir_path, exist_ok=True)
        print(f"Logs saved in: {log_dir_path}")
    except OSError as e:
        print(f"Error creating directory {log_dir_path}: {e}")
        print("Log files will be saved in the current working directory instead.
        log_dir_path = os.getcwd()

    current_session_timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S'
    csv_file_name_only = f"{CSV_FILENAME_BASE}_{current_session_timestamp}.csv"
    CSV_FILENAME = os.path.join(log_dir_path, csv_file_name_only)
```

below or in the code.

Filter symbols

- const HIGH_FREQUENCY_PIDS
- const LOW_FREQUENCY_PIDS_...
- const ALL_PIDS_TO_LOG
- const CSV_FILENAME_BASE
- const LOG_SUBDIRECTORY
- const WIFI_ADAPTER_HOST
- const WIFI_ADAPTER_PORT
- const WIFI_PROTOCOL
- const USE_WIFI_SETTINGS
- func get_pid_value
- func main

Retrieved from: https://github.com/benty691/EAT40005/blob/main/obd_logger.py

Data cleaning steps

Remove Constant or Redundant Features
- Flag: nunique() ≤ 1 or near-perfect correlation (>98%)

Drop Columns with Excessive Missingness
- Flag: > 70 % null values in a column

Drop Rows with Too Many Missing Values
- Flag: > 40 % null values in a row

Convert & Validate Timestamps
- Action: pd.to_datetime() + drop invalid dates

Replace Known Sensor Placeholders
- Flag: Values like –22, –40, 255, 9999
- Action: Replace with NaN

- Detect & Treat Outliers

Remove Duplicate Records
- Action: drop_duplicates() on timestamp+features

Normalize Numeric Values
- Action: Scale using MinMaxScaler()

Data Cleaning Overview

Cleaning Steps

- Merged all three logs on common columns
- Dropped empty or constant columns, placeholder error values (-40, 255), and duplicate rows
- Handled missing values with column-wise mean imputation

Additional Processing

- Normalized data using Min-Max scaling
- Converted timestamps and sorted data chronologically
- Added feature columns like AVG_ENGINE_LOAD and TEMP_MEAN for modeling

```python
# ----------------------------------------
# 📌 Step 2: Merge on Common Columns Only
# ----------------------------------------
common_cols = list(set(df1.columns) & set(df2.columns) & set(df3.columns))
df_merged = pd.concat([df1[common_cols], df2[common_cols], df3[common_cols]], ignore_index=True)

# Save raw merged file
df_merged.to_csv('/content/drive/My Drive/EAT40005/W9/merged.csv', index=False)
print("✅ Merged file saved!")

# ----------------------------------------
# 📌 Step 3: Data Cleaning Pipeline
# ----------------------------------------

# Parse timestamp
df_merged['timestamp'] = pd.to_datetime(df_merged['timestamp'], errors='coerce')

# Drop empty or constant columns
drop_cols = [col for col in df_merged.columns if df_merged[col].nunique() <= 1 or df_merged[col].isna().all()]
df_merged.drop(columns=drop_cols, inplace=True)

# Drop duplicate columns (identical across rows)
df_merged = df_merged.loc[:, ~df_merged.T.duplicated()]

# Drop exact duplicate rows
df_merged.drop_duplicates(inplace=True)

# Replace placeholder sensor error values
df_merged.replace([-22, -40, 255], np.nan, inplace=True)

# Drop columns with >80% missing
missing_ratio = df_merged.isna().mean()
df_merged.drop(columns=missing_ratio[missing_ratio > 0.8].index, inplace=True)

# Fill missing numeric values with median
for col in df_merged.select_dtypes(include=[np.number]).columns:
    df_merged[col].fillna(df_merged[col].median(), inplace=True)

# Clip extreme RPM outliers (beyond 100 < x < 6000)
if 'RPM' in df_merged.columns:
    df_merged['RPM'] = df_merged['RPM'].apply(lambda x: np.nan if x < 100 or x > 6000 else x)
    df_merged['RPM'].fillna(df_merged['RPM'].median(), inplace=True)

# Final sort and reset
df_cleaned = df_merged.sort_values(by='timestamp').reset_index(drop=True)

# Normalize numeric columns
scaler = MinMaxScaler()
numeric_cols = df_cleaned.select_dtypes(include=[np.number]).columns
df_cleaned[numeric_cols] = scaler.fit_transform(df_cleaned[numeric_cols])

# Save cleaned version
df_cleaned.to_csv('/content/drive/My Drive/EAT40005/W9/cleaned.csv', index=False)
print("✅ Cleaned file saved!")
```

Retrieved from: https://colab.research.google.com/drive/1-01MrHI6puMfvvfuBUIl28XM6mzR2ViX?usp=sharing

Exploratory Insights

Dataset Summary

Cleaned dataset contains 366 entries with approximately 19 features, providing a solid basis for modeling.

Correlation matrix shows SPEED, RPM, and ENGINE_LOAD are highly correlated and important for predictive modeling.

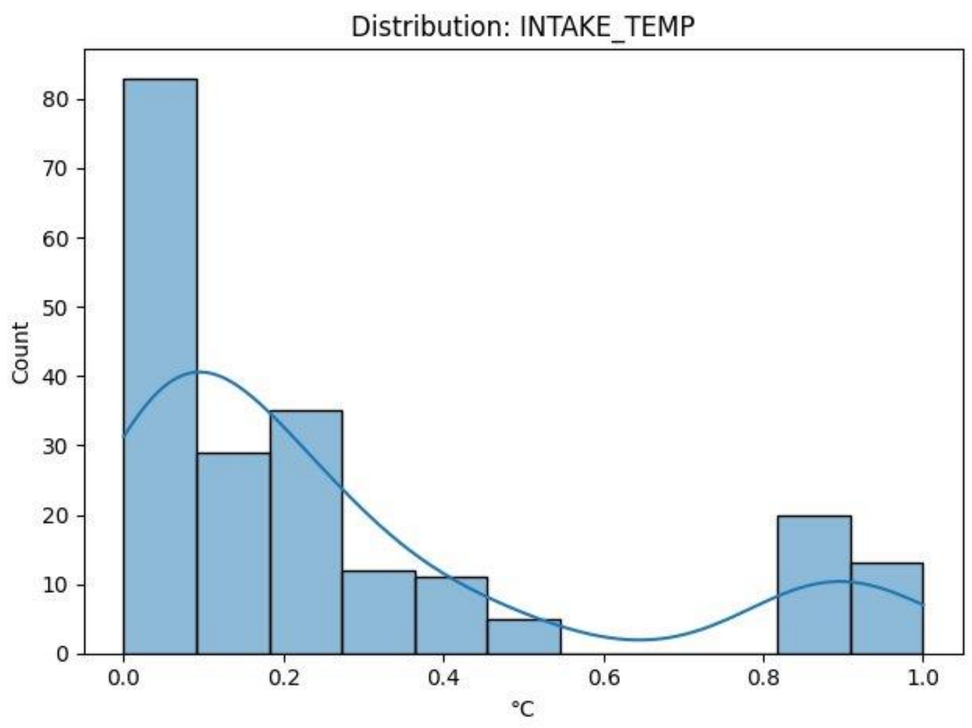
Driving States

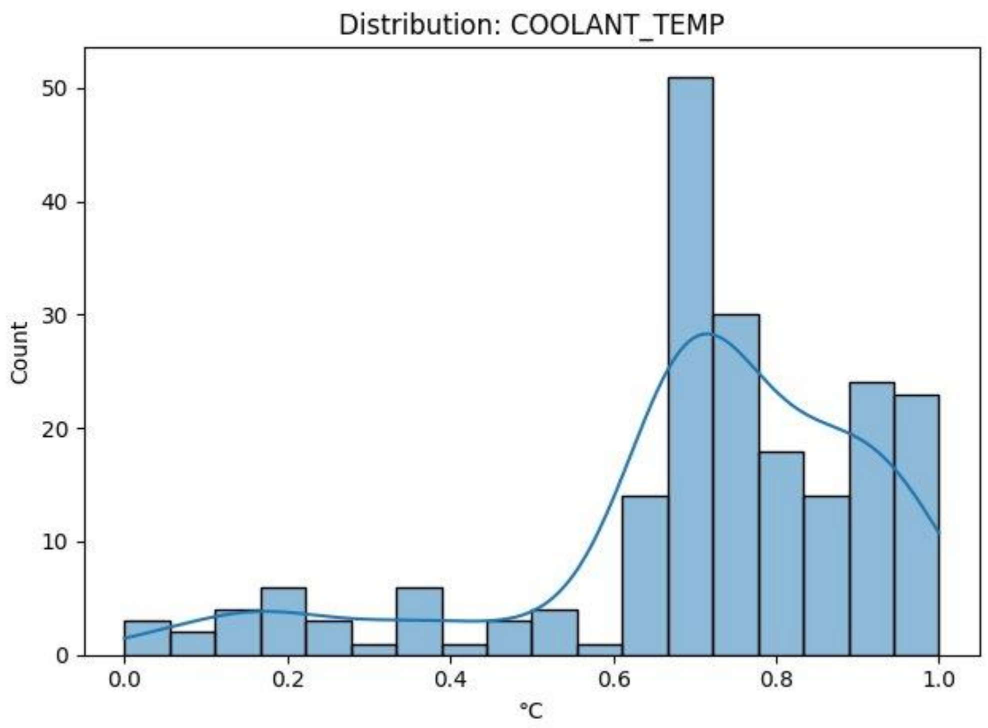
4 driving trials was conducted.

Index-based plots reveal flat lines during idle or parked states and spikes during acceleration bursts.

Suggests labeling data with states like Idle, Cruise, and Acceleration for better model training.

Data storage on Google Drive is accessible via:
https://drive.google.com/drive/folders/1rtY2whQttXFBOO2cz51xOsd-OQ7dcBts?usp=sharing





📊 Summary Statistics of Cleaned Data:

	THROTTLE_POS	RPM	SHORT_FUEL_TRIM_1	LONG_FUEL_TRIM_1 \
count	208.000000	208.000000	208.000000	208.000000
mean	0.131203	0.295964	0.611357	0.430889
min	0.000000	0.000000	0.000000	0.000000
25%	0.016129	0.026028	0.540541	0.125000
50%	0.080645	0.274379	0.675676	0.312500
75%	0.197581	0.517329	0.675676	0.750000
max	1.000000	1.000000	1.000000	1.000000
std	0.161595	0.273885	0.136744	0.328553

	timestamp	SPEED	COOLANT_TEMP \
count	208	208.000000	208.000000
mean	2025-05-07 17:23:04.451040768	0.359375	0.707062
min	2025-05-07 15:24:03.071318	0.000000	0.000000
25%	2025-05-07 17:15:33.379538688	0.000000	0.689655
50%	2025-05-07 17:20:28.951006464	0.333333	0.724138
75%	2025-05-07 18:29:12.089195520	0.700000	0.862069
max	2025-05-07 18:34:12.509935	1.000000	1.000000
std	NaN	0.353066	0.222107

	INTAKE_PRESSURE	ENGINE_LOAD	INTAKE_TEMP	time_diff
count	208.000000	208.000000	208.000000	207.000000
mean	0.181070	0.164374	0.270879	55.118061
min	0.000000	0.000000	0.000000	5.473024
25%	0.062500	0.037975	0.057143	5.567791
50%	0.075000	0.088608	0.142857	5.617369
75%	0.175000	0.189873	0.342857	6.148060
max	1.000000	1.000000	1.000000	6270.703584
std	0.227422	0.200123	0.299554	509.084626

Data Distributions & Patterns

🔎 Distribution Observations

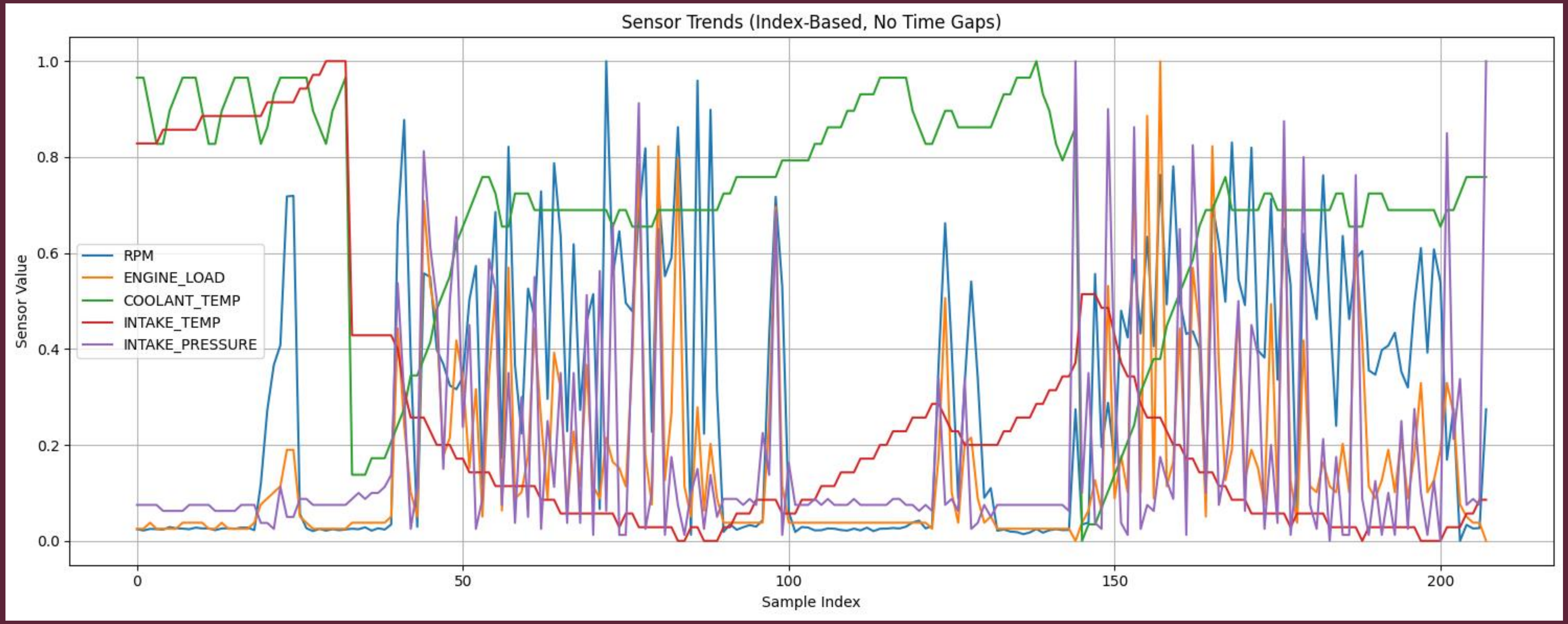
Plots of INTAKE_TEMP, COOLANT_TEMP, RPM, and other variables reveal peaks at fixed thresholds, indicating idle engine periods.

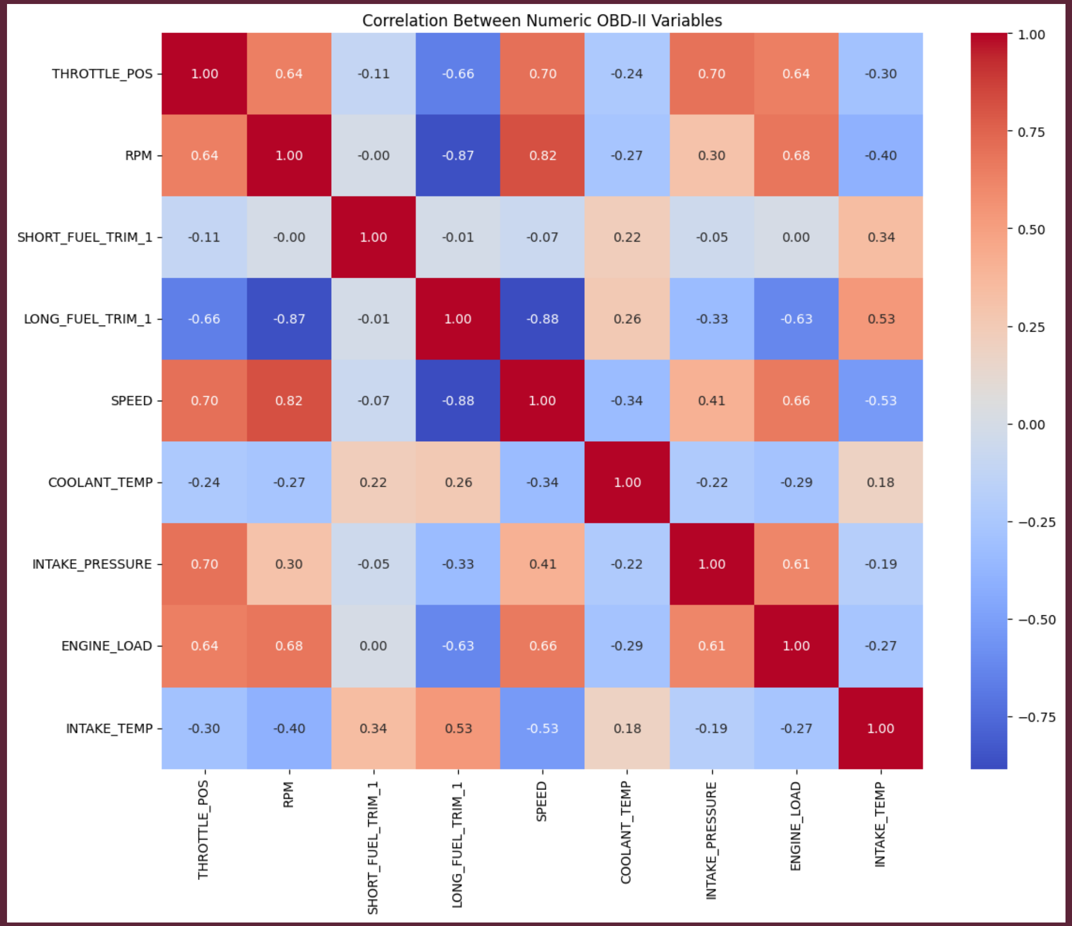
🔗 Correlations

Strong correlations found between RPM, SPEED, and THROTTLE_POS, highlighting key relationships in vehicle behavior.

📈 Data Quality

Sensor saturation was avoided by removing placeholder values like 255, ensuring cleaner data for analysis.





Visualizing Sensor Trends

Index-Based Trend Plot

Shows continuous sensor readings without gaps, helping identify clusters and patterns unaffected by timestamp irregularities.

Correlation Heatmap

Visualizes relationships between variables, highlighting key sensor interactions for feature selection.

Challenges Highlighted

Sparse data intervals posed challenges, but index plotting helped overcome bias from irregular timestamps.

Next Steps & Recommendations Recommendations

Label Data Sessions

Begin annotating current data with driving states such as idle, acceleration, and deceleration to improve model accuracy.

Improve Logging

Reduce PID count per session and better separate high versus low relevance PIDs to optimize data collection.

Explore Synthetic Labeling

Investigate annotation tools or synthetic labeling methods to enhance dataset quality and coverage.

Prepare for Modeling

Use the cleaned dataset to develop baseline machine learning models such as XGBoost or LSTM for predictive maintenance.