MONASH University

MALAYSIA

Non-Intrusive Load Monitoring (NILM)

Project Initial Concept, Design, and Data Analysis

Chong Ming Sheng, Martin Ung Chee Hong, Pang Wai Qi, Wang Kai Jie

September 23, 2023

Abstract

Rising energy costs have become a point of concern in the modern era. Coupled with increasing environmental consciousness, there is an exigent need to monitor energy consumption more closely. This is essential not just for easing the financial burden on consumers, but also for addressing the environmental implications of inefficient energy practices and streamlining the energy production and distribution for a sustainable future.

In this paper, we introduce a model for Non-Intrusive Load Monitoring (NILM). This concept uses centralised meters to track the energy consumption of particular household appliances using time-series disaggregation techniques. In NILM, disaggregation is frequently accomplished using clustering algorithms that find recurring patterns in the time-series data of energy usage. NILM tries to extract or distinguish energy consumption of distinct appliances from their combined energy profile by utilising advanced clustering techniques such as k-means algorithms, spectral cluster mean (SC-M), spectral cluster eigenvector (SC-EV) or others. Moreover, NILM can also help in identifying energy wastage or overconsumption in households via outlier detection in clustering.

Both the f-measure score (F1) and accuracy in identifying particular appliances from the aggregated energy data as well as the Mean Squared Error (MSE) in predicting energy usage would be the three key metrics used to critically assess the performance of the NILM model. A successful or reliable NILM model should have low MSE in energy consumption projections in addition to high F1 score and accuracy in determining appliances in use. This knowledge empowers consumers to make informed decisions about their energy consumption, fostering an environment of efficiency and sustainability.

Contents

Chapter 1

Introduction

In today's world of rising energy consciousness, understanding and controlling electricity consumption has become crucial for individuals as well as companies. The importance of tracking energy usage cannot be overstated, as it aids in reducing expenses for energy users, mitigating the adverse environmental effects stemming from ineffective energy production or wastage, and finally, identifying opportunities to increase efficiency in energy production and distribution [1]. Conventionally, keeping tabs on the energy consumption of connected appliances would involve placing sensors in every energy outlet to track their energy usage. Nevertheless, this traditional method is very resource-demanding, particularly when contemplating its implementation for all households across the country.

To achieve this without the use of additional meters or unwanted facilities, an innovative system should be developed to allow for the breakdown of overall energy consumption into individual appliance-level usage patterns [2]. Therefore, this paper outlines our project's initial concept and design, with Chapter 2 focusing on software design, Chapter 3 on the data analysis report, Chapter 4 providing details on hardware specifications, and Chapter 5 outlining software specifications to be used throughout our project.

This innovative approach strives to fundamentally change the methods in which we oversee and control energy consumption, ultimately providing increased efficiency and effectiveness in our pursuit to meet energy conservation objectives.

Chapter 2

Software Design

2.1 Level 0 Data Flow Diagram

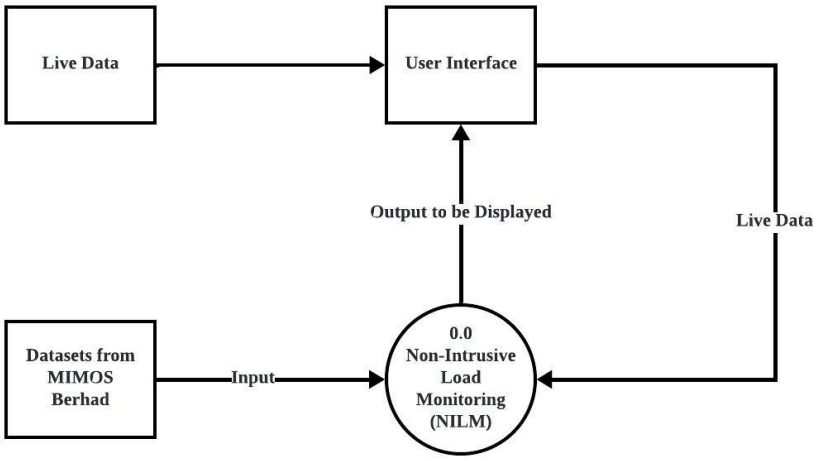

Figure 2.1: Level 0 Data Flow Diagram

In the Level 0 Data Flow Diagram (Figure 2.1), we present a high-level overview of Non-Intrusive Load Monitoring (NILM). The NILM software can seamlessly integrate with hardware components, notably centralised smart meters. This project is undertaken in collaboration with MIMOS Berhad, safeguarded by a Non-Disclosure Agreement (NDA) we have mutually endorsed. They have graciously provided datasets derived from their laboratory experiments. The final deliverable of this project is a user interface that displays electrical consumption patterns within a residential or industrial setting. Crucially, it identifies the household appliances in use and alerts users of any unwanted energy usage.

Table 1: Description of Level 0 Data Flow Diagram

Process	**Description**
0.0 Non-Intrusive Load Monitoring (NILM)	A NILM program that integraets multiple processes such as data preprocessing and model training

2.2 Level 1 Data Flow Diagram

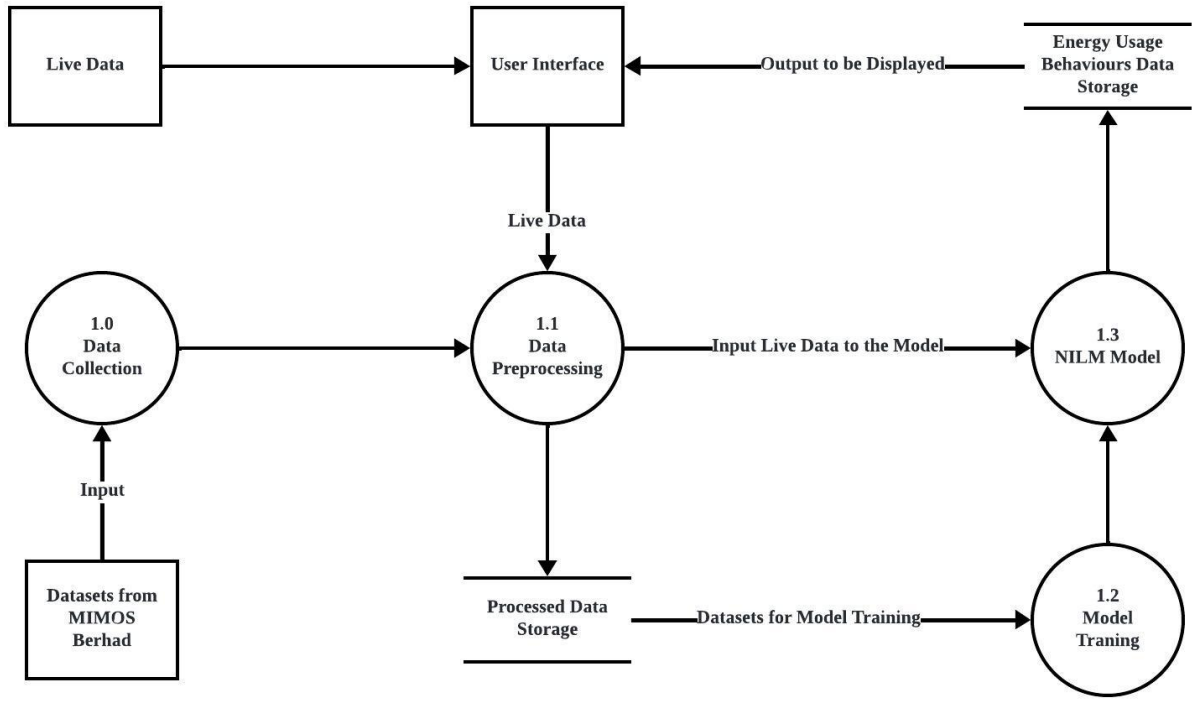

Figure 2.2: Level 1 Data Flow Diagram

In the Level 1 Data Flow Diagram (Figure 2.2), we delineate the system into four primary processes - **Data Collection**, **Data Preprocessing**, **Model Training**, and the **NILM Model Execution**. Our system architecture includes two distinct databases - one for storing preprocessed data and another for the results generated by the NILM model. Real-world readings from the centralised meter will undergo data preprocessing before being directly input into the NILM model for the final output generation. Conversely, the model data provided by MIMOS Berhad first undergoes preprocessing and subsequently aids in training our data science model, culminating in the finalised NILM model.

Table 2: Description of Level 1 Data Flow Diagram

Process	**Description**
1.0 Data Collection	Collect data from the users and also MIMOS Berhad
1.1 Data Preprocessing	Preprocess the collected data and store them in the preprocessed data storage
1.2 Model Training	Train the model and input the preprocessed data into the model to get the output
1.3 NILM Model	Execute the final NILM model on live data

2.3 Level 2 Data Flow Diagram

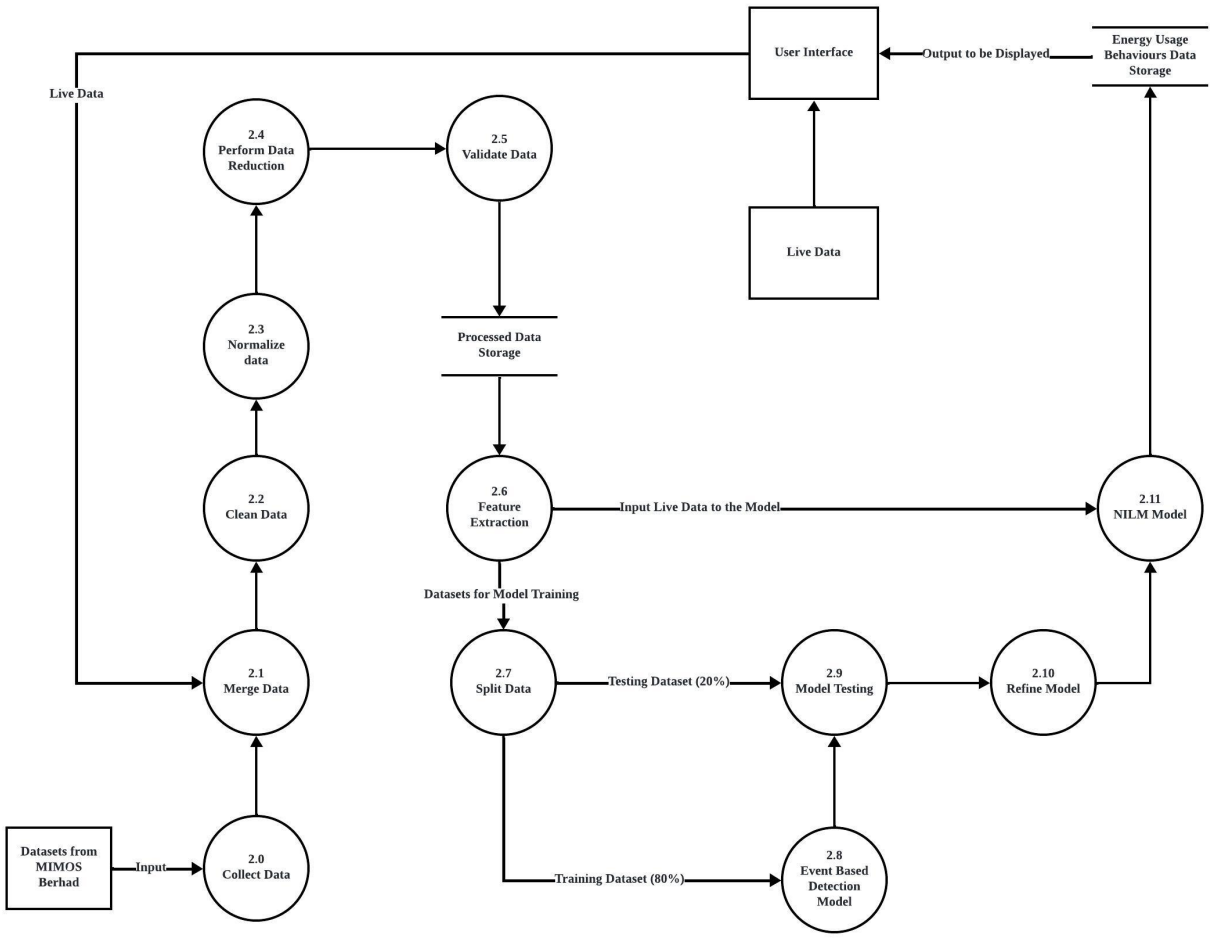

Figure 2.3: Level 2 Data Flow Diagram

In the Level 2 Data Flow Diagram (Figure 2.3), we delve deeper into two main processes - **Data Preprocessing** and **Model Training**. To prepare data for the machine learning model, it undergoes a structured transformation process, transitioning from raw data to a feature-extracted format. When training our model, the data is strategically split into separate training and testing sets, ensuring robust model evaluation. During the model training phase, this training data fuels the event-based detection model. If the NILM model's accuracy does not meet our set benchmark, iterative model refinements are made until the desired accuracy is achieved. In the real-world scenario, given that the NILM model has been pre-established, the live data from users will be directly inputted into the NILM model, bypassing the data splitting and training stages. The NILM model output, which provides insights into appliance energy usage behaviours, is archived in a dedicated storage system and visualised on a user interface.

Table 3: Description of Level 2 Data Flow Diagram

Process	**Description**
2.0 Collect Data	Collect datasets regarding energy disaggregation from different sources
2.1 Clean Data	Remove the noise and inconsistency in the raw data
2.2 Merge Data	Merge the datasets from different appliances based on timestamps
2.3 Normalise Data	Normalise the data to the range of 0 to 1
2.4 Perform Data Reduction	Simplify and compress the electrical consumption data
2.5 Validate Data	Ensure the data is accurate, consistent and no missing values
2.6 Feature Extraction	Identify and isolate unique patterns from the data
2.7 Split Data	Divide data into distinct sets for training and testing
2.8 Event Based Detection Model	Identify appliance switching events in the power signal
2.9 Model Testing	Evaluate the performance and accuracy of load disaggregation process on the model
2.10 Refine Model	Tweak and optimise the algorithm parameters to enhance the model performance and accuracy
2.11 NILM Model	Execute the final NILM model on live data

2.4 Hidden Markov Models (HMM)

Hidden Markov Models (HMM) is our choice of model to identify the appliance switching event in the power signal. The recent trend of employing the Hidden Markov Model (HMM) and its variants within smart NILM solutions are explored, spotlighting their contemporary applications in both non-event-based and event-based frameworks [3]. In our NILM project, we utilise HMM's principles of hidden and observed events to discern the appliances operational state (ON and OFF) and their energy consumption, including both the apparent and active power [4]. HMMs excel in their capability of distinguishing and tracing transitions between states, where the actual state is "hidden" and can only be inferred indirectly from the emitted data. This proficiency is particularly useful for detecting brief periods of stationary signal segments defined by distinct parameters. These hidden states correlate to appliance state transitions (ON and OFF) in the context of NILM, while power measurements are the observations. HMM can be combined with a variety of advanced algorithms to produce accurate results in NILM. HMM scales appliance consumption values and, combined with the AFAMAP algorithm and appliance footprints, offers a refined approach to energy disaggregation challenges [5]. As the result, HMMs are particularly well-suited to accurately capture changes in power consumption during transient phases and expedite the process of classifying and recognising appliances under diverse operational conditions.

There are three main goals we want to achieve by having our HMM model:

1. Identify the appliances in use
    - In HMMs for NILM, a state could represent a particular appliance being ON (e.g. a refrigerator) or a combination of appliances being ON (e.g. television and refrigerator). The number of states in HMM will depend on the number and combination of appliances tracked.
    - For every state, there is an associated probability distribution over the main meter readings. Once the HMM is trained, and given a new sequence of main meter readings, it can determine the most probable sequence of states that produce those readings. This indicates which appliances are likely active at each timestamp.
2. Learn and optimise energy usage behaviours
    - Using a dataset of energy consumption readings of main meter and each appliance, the HMM's parameters (transition and emission probabilities) can be adjusted to better model the data.
    - The household's typical energy usage behaviour will then be learned by HMM. For example, the air conditioner will always be switched on during nighttime.
    - With this ability, the HMM model can identify patterns and behaviours that could be optimised for better and more efficient energy usage. When the energy usage is not optimal, the suggestion of behaviour should then be proposed to the household which could help to optimise their energy usage behaviours.
3. Detect unwanted energy wastage
    - HMM model can compute the likelihood of new sequence of main meter readings. By examining the decoded state sequences, we can identify appliances that remain ON for prolonged periods, potentially indicating wastage. For instance, a light that remains ON for 24 hours continuously might be unintentional.
    - Sometimes, malfunctioning appliance could draw more power than usual. If the power readings for a particular state consistently fall outside the expected range, it might suggest that the appliance is not operating efficiently.

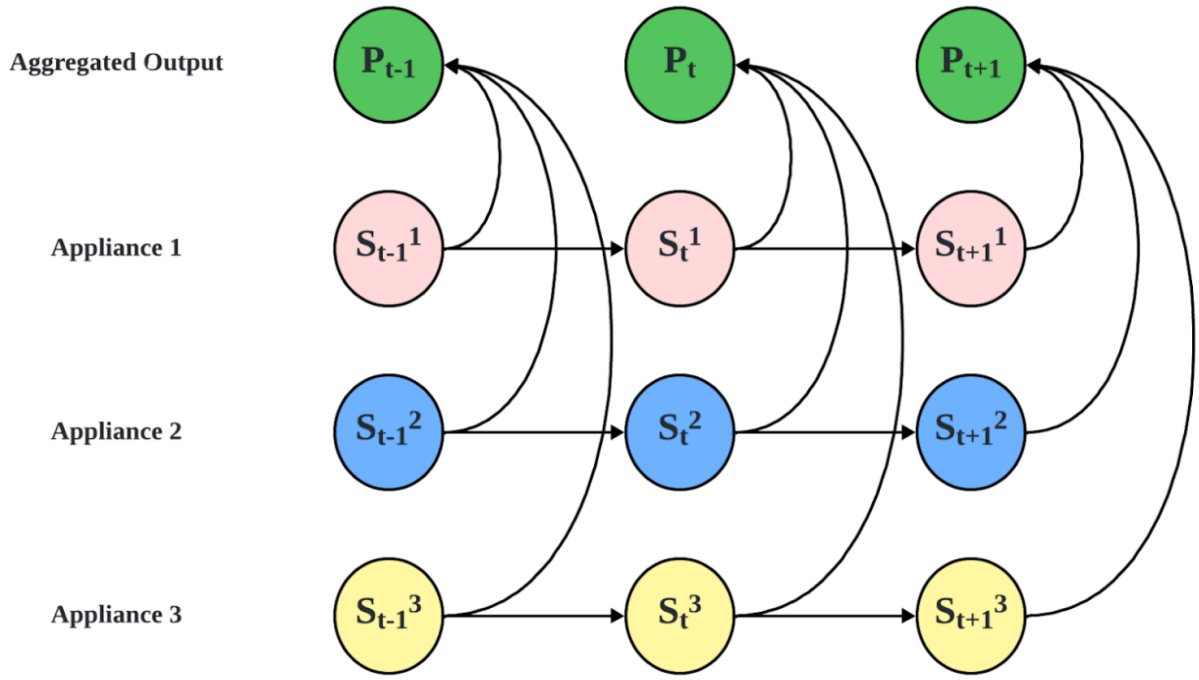

Figure 2.4: Working principle of an HMM model

2.5 Data Preprocessing and Analysis

The windowing technique, which is a well-known technique utilised in NILM, plays a pivotal role in processing and analysing time series energy consumption data [6]. This method involves segmenting the continuous stream of aggregated energy data into smaller, fixed-size intervals (windows), either overlapping or non-overlapping, to facilitate more granular analysis. In NILM preprocessing, raw samples are refined and segmented into frames using a **sliding time window** of **variable length** and **overlap**, with both causal and non-causal approaches, which are then subjected to feature extraction to produce feature vectors for further analysis [6]. It aids in capturing transient events or short-lived appliance activations, ensures temporal consistency, and often simplifies the complexity of the datasets, making them more manageable for algorithms to process. Moreover, when used in real-time NILM applications, windowing allows for near-instantaneous feedback on appliance states, as each window's data can be processed sequentially, thereby supporting timely appliance state detection [7]. By utilising this technique, we can better handle the inherent variability and intricacies of household energy consumption patterns, enhancing the accuracy and reliability of disaggregation algorithms.

During **feature extraction**, the segmented windows are then channelled into three parallel sub-processes, each responsible for extracting features from a different domain - **time**, **frequency**, and **transient**. The time-domain focuses on statistical attributes of the power signal, such as its mean or variance. In contrast, the frequency-domain delves into the spectral characteristics using methods like the Fast Fourier Transform. In [8], frequency-domain feature calculation is emphasised because, despite commercial smart meters for homes typically supplying low-frequency sampling, extracting electric signal characteristics around significant power variations (switching events of appliances) serves as distinctive features, enhancing the accuracy of NILM disaggregation algorithms. The transient analysis identifies and characterises rapid shifts in power consumption, indicative of appliance state changes. The proposed NILM system in [9] uses the transient response of the power signal for identifying appliances, which is based on the processing of the turn-on transient responses triggered by an appliance being switched-on, and can be integrated into a low-cost chip-set or smart meter. Once these features are extracted, they will be converged into a unified feature aggregation process, producing a comprehensive feature vector for each data window.

In the study presented in [10], the authors used a **multiple overlapping sliding window** approach to segment sequence data and a shortcut through one Conv1D to address the vanishing gradient problem in NILM. Their findings concluded that the recognition accuracy is higher when the slide step is shorter. Overlaps between windows occur when the slide step is smaller than the window size. This underscores the superiority of the overlapping windows technique over its non-overlapping counterpart. Upon reviewing several scholarly articles [9][11], we are contemplating setting the **window size** in the range of **20 to 100 seconds** while implementing a **moving average filter** spanning **40 ms** intervals.

Chapter 3

Data Analysis Report

3.1 Background Information

This data analysis report outlines the comprehensive analysis of the dataset provided by MIMOS Berhad, which is the National Applied Research and Development Centre in Malaysia. The provided dataset comes with a reliable source, in which the energy consumption of different household appliances is monitored by myriad IoT sensors installed in the MIMOS laboratory. This report examines the data collection processes, data preprocessing steps, data analysis methodologies, identification of potential relationships between data, as well as summary and key findings derived from the given dataset. With the data analysis performed, it helps our group to draw meaningful conclusions and sophisticated recommendations in making more informed decisions and determining well-suited solutions during the project execution phase in the future.

3.2 Problem Statement

The primary goal of this data analysis report is to aid our team in looking into the provided dataset as a whole by understanding the format and structure of the data. Also, it is aimed to preprocess the data in order to address certain data issues that are identified at this stage of the project. Through the preliminary analysis of data, we seek to discover some unexpected yet interesting underlying data patterns via various visualisation techniques. In summary, this report serves as an essential segment in our project which provides the team with a clear overview of the existing data, meanwhile helping the team to gain more specific knowledge and expertise regarding our domain, i.e. NILM.

3.3 Significance of Data Analysis

Several chief benefits are expected to be yielded as the result of this data analysis report. Essentially, a thorough understanding of the data helps our team to identify correct ways for data conversion and efficient access to certain attributes, especially when we are dealing with the data during the project execution phase. Furthermore, cleaned and preprocessed data ensures consistency in data representation and improves the interpretability of data, which helps the team to work on the data more easily. This can also lead to enhanced model performance as cleaner and more reliable input can potentially reduce model training time, better generalize the model, and increase the model accuracy in identification. Plus, informed decision-making and viable future actions could be driven by the data analysis made, with the help of any possible data

correlations discovered from the data. Overall, the primary importance of this data analysis report is to aid the team in achieving better outcomes via its capability of transforming existing data into actionable or practical insights.

3.4 Data Collection

3.4.1 Data sources

The dataset is sourced from MIMOS Berhad. It is provided in CSV (Comma-separated values) format and arrives within a compressed Zip file. Upon extraction, it reveals a hierarchical structure with multiple sub-folders contained within the main directory called "Data", as shown in Figure 3.1. One of these sub-folders, which is "main", is designated for the total or aggregated meter reading, while the remaining sub-folders represent various household appliances such as "aircond", "dryer" and "fridge". Each of these sub-folders includes multiple datasets, storing the readings of energy consumption of that particular household appliance at different timestamps.

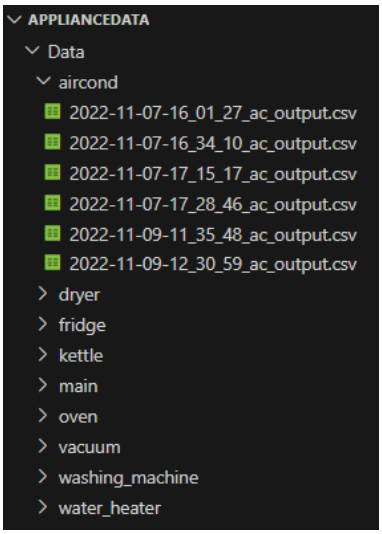

Figure 3.1: Dataset structure

3.4.2 Limitations and Biases Associated with Data

It is important to acknowledge certain limitations inherent to this data source. The energy consumption data collected within the laboratory of MIMOS Berhad in Malaysia attempts to emulate real household setups. Consequently, it may not precisely reflect the energy usage patterns of individual households or other types of buildings, even in different countries.

Moreover, since the dataset was collected in a controlled laboratory environment to measure the energy consumption patterns of household appliances, it is worth noting that the dataset may exhibit some bias as it may not fully mirror the household energy usage behaviours in the real world. For instance, the use of limited appliance types in the lab setup might potentially lead to underrepresentation of the other household appliances. Another valid example would be the controlled lab–based environment. It might not be able to capture the full range of unexpected variability and energy usage patterns discovered in real homes.

3.5 Data Preprocessing

All the CSV files consist of three columns, namely "Timestamp", "Apparent (VA)", and "Active (W)", as shown in Figure 3.2. Table 4 below shows the total number of records after concatenating all the CSV files available in the respective subfolders.

	Timestamp	Apparent (VA)	Active (W)
0	2022-11-07 16:01:27	0.05	0.01
1	2022-11-07 16:01:28	0.00	0.00
2	2022-11-07 16:01:29	0.00	0.00
3	2022-11-07 16:01:30	0.00	0.00
4	2022-11-07 16:01:31	0.00	0.00

Figure 3.2: Display of columns in CSV files

Table 4: Total records after concatenating all CSV files available in respective sub-folders

Subfolder Name	**Total Records**
aircond	8691
dryer	16197
fridge	9535
kettle	7967
main	36463
oven	7199
vacuum	6793
washing_machine	20949
water_heater	6855

The data type of the "Timestamp" column is Python object, which theoretically may consist of either text or a mix of numeric and non-numeric values [12]. After further inspection as shown in Figure 3.3, it was found that this column has a data type of "str". The data type of both the "Apparent (VA)" and "Active (W)" columns is "float64".

```
ac_combined.dtypes
```

```
Timestamp         object
Apparent (VA)    float64
Active (W)       float64
dtype: object
```

```
print("Data value:", ac_combined['Timestamp'][0])
print("Data type:", type(ac_combined['Timestamp'][0]))
```

```
Data value: 2022-11-07 16:01:27
Data type: <class 'str'>
```

Figure 3.3: Python code to inspect data types of columns

3.6 Data Analysis

3.6.1 Limitation 1 - Low Granularity of Timestamp Attribute

The "Timestamp" column itself contains all information about the date and time of each record. It makes it difficult to extract the date or time information at a granular level.

- *Checking:*
  Refer to Figure 3.2. Python code *df.head()* is used to display the first 5 records of any dataframe.
- *Preprocessing steps:*
  Create new columns, namely "Year", "Month", "Day", "Hour", "Minute" and "Second", by extracting the corresponding values from the "Timestamp" column. To efficiently apply this change to all the records across all CSV files available in the main directory "Data", we create a new dataframe concatenating all the CSV files. Code execution is shown in Figure 3.4, and the result is shown in Figure 3.5.

  (a) Create a new column, namely "Type", in all the CSV files, and assign values indicating the correct appliance type to this "Type" column.

  (b) Concatenate all the CSV files into a new dataframe called "df".

  (c) Create new columns, namely "Year", "Month", "Day", "Hour", "Minute" and "Second", as explained above.

```
# Create new column "Type"

files_str = ['ac_combined', 'dryer_combined', 'fridge_combined', 'kettle_combined', 'main_combined', 'oven_combined',
             'vacuum_combined', 'wm_combined', 'wh_combined']
types = ["aircond", "dryer", "fridge", "kettle", "main", "oven", "vacuum", "washing_machine", "water_heater"]

for i in range(len(files_str)):
    combined = pd.read_csv(str(files_str[i]) + '.csv')
    combined = combined.iloc[: , 1:]
    combined.loc[:, 'Type'] = types[i]
    combined.to_csv(files_str[i] + '.csv')
```

```
# Concatenate all the CSV files

df1 = pd.read_csv('ac_combined.csv').iloc[: , 1:]
df2 = pd.read_csv('dryer_combined.csv').iloc[: , 1:]
df3 = pd.read_csv('fridge_combined.csv').iloc[: , 1:]
df4 = pd.read_csv('kettle_combined.csv').iloc[: , 1:]
df5 = pd.read_csv('main_combined.csv').iloc[: , 1:]
df6 = pd.read_csv('oven_combined.csv').iloc[: , 1:]
df7 = pd.read_csv('vacuum_combined.csv').iloc[: , 1:]
df8 = pd.read_csv('wm_combined.csv').iloc[: , 1:]
df9 = pd.read_csv('wh_combined.csv').iloc[: , 1:]

df = pd.concat([df5, df1, df2, df3, df4, df6, df7, df8, df9], ignore_index=True)
```

```
# Add new columns for "Year", "Month", "Day", "Hour", "Minute" and "Second" & insert them to after "Timestamp" column
df['Timestamp'] = pd.to_datetime(df['Timestamp'])

year_values = df['Timestamp'].dt.year
month_values = df['Timestamp'].dt.month
day_values = df['Timestamp'].dt.day
hour_values = df['Timestamp'].dt.hour
minute_values = df['Timestamp'].dt.minute
second_values = df['Timestamp'].dt.second

df.insert(1, 'Year', year_values)
df.insert(2, 'Month', month_values)
df.insert(3, 'Day', day_values)
df.insert(4, 'Hour', hour_values)
df.insert(5, 'Minute', minute_values)
df.insert(6, 'Second', second_values)
```

Figure 3.4: Python code to break down "Timestamp" information into more granular levels

	Timestamp	Year	Month	Day	Hour	Minute	Second	Apparent (VA)	Active (W)	Type
0	2022-09-27 15:52:32	2022	9	27	15	52	32	22.12	16.25	main
1	2022-09-27 15:52:33	2022	9	27	15	52	33	22.13	16.54	main
2	2022-09-27 15:52:34	2022	9	27	15	52	34	22.46	16.42	main
3	2022-09-27 15:52:35	2022	9	27	15	52	35	22.22	16.38	main
4	2022-09-27 15:52:36	2022	9	27	15	52	36	22.32	16.50	main

Figure 3.5: Result of data preprocessing to address Limitation 1

3.6.2 Data Inconsistency Issue 1 - Outliers

Possible outliers are detected. To elaborate, categorised by appliance types, the values in both "Apparent (VA)" and "Active (W)" columns of certain records are overly high or low, compared to their respective mean values.

- *Checking 1:*
  Plot boxplot (Figure 3.7) to visualise the overall distribution of both "Apparent (VA)" and "Active (W)" data values for each appliance type.

  (a) Create a new dataframe by extracting only "Apparent (VA)" and "Type" columns from the "df" dataframe (with all CSV files being concatenated together).

  (b) Remove all records with the value of "Apparent (VA)" being 0.

  (c) Plot the boxplots categorised by different appliance types.

  (d) Repeat steps (a) to (c) for "Active (W)" instead of "Apparent (VA)".

```
df_apparent = df[["Apparent (VA)", "Type"]]
```

```
df_removed_0_apparent = df_apparent[df_apparent.ne(0).all(axis=1)]
```

```
# boxplot of 'Apparent (VA)'
sns.set(style='whitegrid')
fig, ax = plt.subplots(figsize=(15, 8))


# Create a box plot of apparent with x axis for types
ax = sns.boxplot(x='Type',
                 y='Apparent (VA)',
                 data=df_removed_0_apparent,
                 palette='Set3',
                 ax=ax)

title = 'Boxplot of each Appliance for Apparent (VA) Values'
fig.suptitle(title, y=.97)

lines = ax.get_lines()
categories = ax.get_xticks()

for cat in categories:
    y = round(lines[4+cat*6].get_ydata()[0],1)
    ax.text(cat, y, f'{y}', ha='center', va='center', size=8, color='white',
        bbox=dict(facecolor='#828282', edgecolor='#828282')
    )

ax.set_xticklabels(ax.get_xticklabels())
# Display the plot
plt.show()
```

Figure 3.6: Python code to plot boxplot based on different appliances

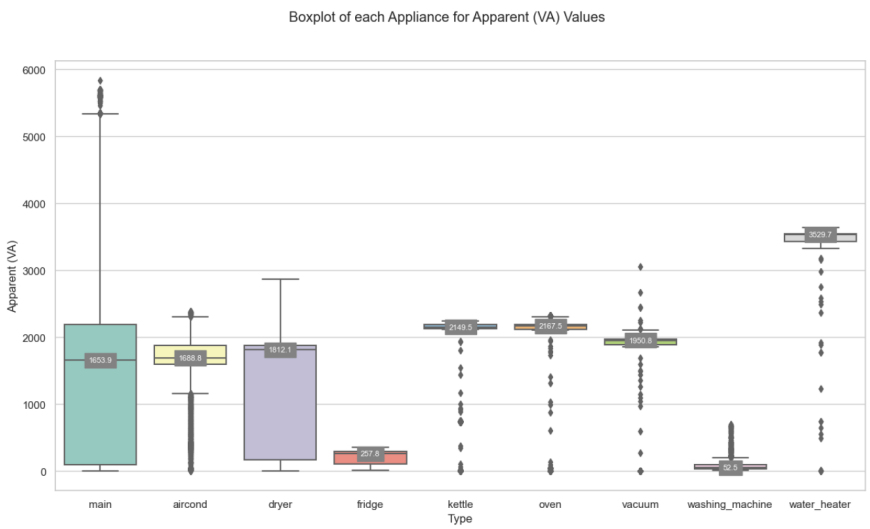

Figure 3.7: Result of data preprocessing to address Data Inconsistency Issue 1

- *Checking 2:*
  Use the formula *Lower = Q1 - 1.5 IQR* and *Upper = Q3 + 1.5 IQR* to inspect the possible outliers. To be specific, the data values smaller than "Lower" or larger than "Upper" are considered possible outliers. In this case, we only display the checking process done for the "Type" == "vacuum", on data in the "Apparent (VA)" column.

  (a) Create a new dataframe with "Type" == "vacuum" and all "Apparent (VA)" values of 0 being removed.

  (b) Calculate the values of Q1 (first quartile), Q3 (third quartile), and IQR (interquartile range = Q3 - Q1), based on the data values in the "Apparent (VA)" column.

  (c) Filter out the records with "Apparent (VA)" values being lower than the "Lower" or bigger than the "Upper". These "Apparent (VA)" values filtered, as shown in Figure 3.9, are the possible outliers.

```
df_removed_0_apparent_vacuum = df_removed_0_apparent[df_removed_0_apparent["Type"] == "vacuum"]
```

```
# Print out records containing possible outliers (for vacuum)

Q1 = df_removed_0_apparent_vacuum['Apparent (VA)'].quantile(0.25)
Q3 = df_removed_0_apparent_vacuum['Apparent (VA)'].quantile(0.75)
IQR = Q3 - Q1
lower_b = Q1 - (1.5 * IQR)
upper_b = Q3 + (1.5 * IQR)

filt = (df_removed_0_apparent_vacuum['Apparent (VA)'] < lower_b) | (df_removed_0_apparent_vacuum['Apparent (VA)'] > upper_b)

outliers = df_removed_0_apparent_vacuum[filt]
```

Figure 3.8: Python code to filter out the possible outliers of "Apparent (VA)" for "vacuum" appliance type

```
print(sorted(outliers['Apparent (VA)'].to_list()))
```

```
[0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.05, 0.05, 0.05, 0.05, 0.05, 0.06, 0.06, 0.09, 0.09, 0.09, 0.09, 0.09, 0.1, 0.11, 0.12,
0.13, 0.17, 0.18, 0.18, 0.2, 0.21, 0.22, 0.23, 0.31, 0.31, 0.32, 0.33, 0.34, 0.39, 0.41, 0.48, 0.5, 0.52, 0.58, 0.59, 0.6, 0.6
6, 0.69, 0.8, 0.87, 1.0, 1.16, 1.24, 1.29, 1.45, 1.67, 1.87, 1.89, 2.03, 2.07, 2.12, 2.23, 2.29, 2.42, 2.82, 2.93, 2.98, 3.0,
3.06, 3.11, 3.34, 3.35, 3.42, 3.42, 3.45, 3.55, 3.59, 3.61, 3.63, 3.63, 3.75, 3.77, 3.84, 3.88, 3.89, 3.98, 3.98, 3.99, 4.02,
4.02, 4.03, 4.09, 4.12, 4.14, 4.15, 4.15, 4.19, 4.19, 4.2, 4.21, 4.21, 4.24, 4.25, 4.26, 4.28, 4.33, 4.35, 4.37, 4.37, 4.37, 4.
38, 4.39, 4.39, 4.39, 4.42, 4.42, 4.43, 4.51, 4.51, 4.52, 4.53, 4.55, 4.55, 4.57, 4.57, 4.62, 4.65, 4.65, 4.67, 4.68, 4.68, 4.6
8, 4.69, 4.72, 4.75, 4.77, 4.8, 4.8, 4.82, 4.88, 4.93, 4.93, 4.95, 4.95, 4.97, 4.97, 4.98, 5.0, 5.01, 5.04, 5.08, 5.08, 5.11,
5.11, 5.11, 5.14, 5.14, 5.14, 5.14, 5.15, 5.18, 5.19, 5.19, 5.22, 5.24, 5.24, 5.25, 5.26, 5.28, 5.28, 5.31, 5.31, 5.34, 5.37,
5.38, 5.39, 5.48, 5.54, 5.59, 5.61, 5.7, 5.94, 268.52, 596.5, 966.85, 1046.15, 1095.58, 1146.47, 1263.56, 1351.31, 1442.38, 149
3.96, 1500.44, 1598.38, 1690.34, 2119.05, 2125.72, 2129.64, 2222.36, 2249.67, 2439.44, 2450.73, 2666.91, 3051.92]
```

Figure 3.9: Display of possible outliers of "Apparent (VA)" for "vacuum" appliance type in ascending order

- *Preprocessing steps:*
  From Figure 3.7 and Figure 3.9, it can be observed that the minimum and maximum potential outliers are 0.04 and 3051.92 respectively. According to [13], a normal vacuum cleaner typically consumes around 450 to 3000 Watts per second. Therefore, the only "Apparent (VA)" value larger than 3000 Watts found in Figure 3.9 (which is 3051.92) is still considered feasible and reasonable in this case. However, for the values smaller than 450 Watts, they are all generally smaller than 6 Watts, which could possibly be due to unwanted energy wastage occurring in the particular vacuum cleaner when it is not in use. Therefore, these potential outliers in the "Apparent (VA)" for the vacuum are not removed or preprocessed, as they could give certain significant features for the vacuum's energy consumption pattern, as well as show some signs of unwanted energy wastage by the vacuum cleaner.

3.6.3 Data Inconsistency Issue 2 - Duplication of Records

Based on the provided CSV files, there is more than one record having the same values in the "Timestamp" and "Type" columns (duplicate records in both "Timestamp" and "Type" columns), but giving different "Apparent (VA)" and "Active (W)" values. In this case, we only display the checking process done on data in the "Apparent (VA)" column.

- *Checking 1:*
  Plot line charts of "Apparent (VA)" and "Active (W)" against "Timestamp" on different days. When it comes to the "Day" == 29, "Month" == 9 and "Year" == 2022, there are some weird patterns as shown in Figure 3.10, particularly on the blue line (representing "main") and grey line (representing "washing_machine").

  To be specific, these two lines show some area marks with shallower colour at certain timestamps, as shown in Figure 3.10. Also, it could be observed that the data points of the grey line (representing "washing_machine") extend beyond the boundaries of the blue line (representing "main").

  Therefore, it is suspected and assumed that there is a high possibility of having duplicate records with the same "Timestamp" and "Type" values but different "Apparent (VA)" values in the "df" dataframe (with all CSV files being concatenated together).

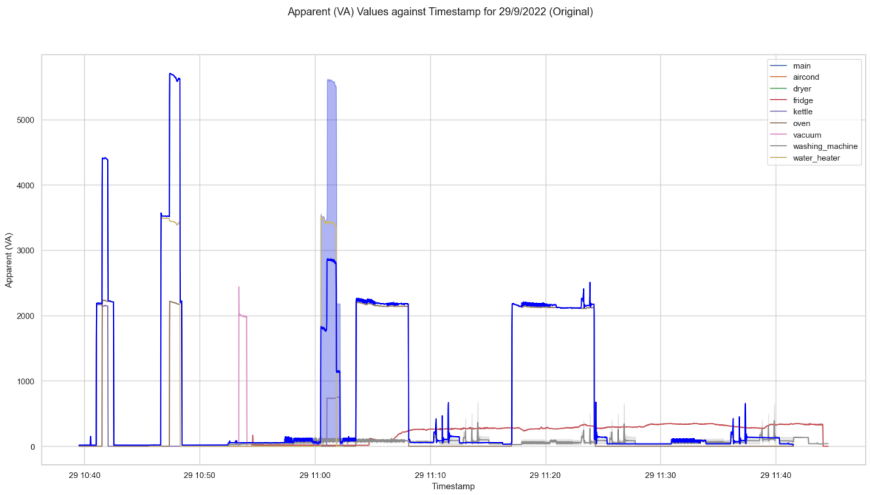

Figure 3.10: Line chart of "Apparent (VA)" against "Timestamp" on 29/9/2022

- *Checking 2:*
  Verify the assumption above to see whether or not there are such duplicate records in the "df" dataframe by using Python code, as shown in Figure 3.11.

```
# Find duplicate records based on 'Timestamp' and 'Type' == 'washing_machine' or 'main'

filt1 = df['Type'] == 'washing_machine'
filt2 = df['Type'] == 'main'
filt3 = (df['Day'] == 29) & (df['Month'] == 9) & (df['Year'] == 2022)
filt4 = df.duplicated(subset=['Timestamp', 'Type'], keep=False)

duplicates = df[(filt1 | filt2) & filt3 & filt4]

duplicates = duplicates.sort_values(by=['Type', 'Timestamp'])
```

Figure 3.11: Python code to filter out the duplicate records mentioned above

Based on Figure 3.12, it can be seen that our assumption made above is definitely true, as there are clearly 6212 records with the same "Timestamp" values but different "Apparent (VA)" and "Active (W)" values. The latest "Timestamp" value that has duplicate records is "2022-09-29 11:41:32".

```
print(duplicates[['Timestamp', 'Apparent (VA)', 'Active (W)', 'Type']])
```

```
                Timestamp  Apparent (VA)  Active (W)             Type
1477  2022-09-29 10:59:48         107.19       79.74             main
3982  2022-09-29 10:59:48          18.53       12.36             main
1478  2022-09-29 10:59:49         122.97       92.99             main
3983  2022-09-29 10:59:49          18.84       12.52             main
1479  2022-09-29 10:59:50          81.24       59.20             main
...                   ...            ...         ...              ...
98740 2022-09-29 11:41:30         133.64      100.65  washing_machine
95792 2022-09-29 11:41:31           0.00        0.00  washing_machine
98741 2022-09-29 11:41:31         137.23      102.96  washing_machine
95793 2022-09-29 11:41:32           0.00        0.00  washing_machine
98742 2022-09-29 11:41:32         135.02      101.78  washing_machine

[6212 rows x 4 columns]
```

Figure 3.12: Display of some of the duplicate records mentioned above

- *Preprocessing steps:*
  - **Part A:**

    Figure 3.13 below shows all the CSV files provided in the "washing_machine" subfolder. When inspecting carefully, only two of the CSV files are on the date "2022-09-29" and have time earlier than "11:41:32", which is the latest "Timestamp" value having duplicate records as stated previously.

    It is also clear that the first two CSV files have the same name. Further inspection is done and it is verified that both CSV files have the same "Timestamp" values but different "Apparent (VA)" and "Active (W)" values, as shown in Figure 3.14 below. Therefore, we repeat the step of concatenating all the CSV files into the new dataframe "df". However, this time we try to exclude either of these two CSV files one by one during concatenation. We would like to see if they yield different results on the line chart.

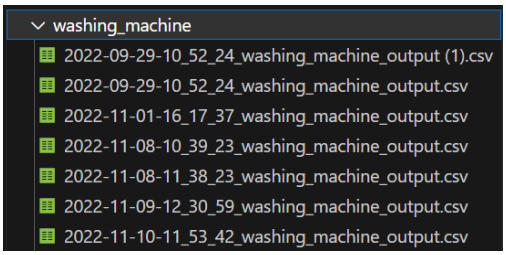

Figure 3.13: Display of all CSV files in the "washing_machine" subfolder

```
washing_machine_output_1 = pd.read_csv("washing_machine/2022-09-29-10_52_24_washing_machine_output (1).csv")
washing_machine_output_1.head(10)
```

	Timestamp	Apparent (VA)	Active (W)
0	2022-09-29 10:52:24	0.00	0.00
1	2022-09-29 10:52:25	0.00	0.00
2	2022-09-29 10:52:26	0.00	0.00
3	2022-09-29 10:52:27	0.00	0.00
4	2022-09-29 10:52:28	34.43	23.82
5	2022-09-29 10:52:29	46.02	32.37
6	2022-09-29 10:52:30	45.48	32.25
7	2022-09-29 10:52:31	45.33	32.05
8	2022-09-29 10:52:32	45.59	31.98
9	2022-09-29 10:52:33	44.97	31.91

```
washing_machine_output = pd.read_csv("washing_machine/2022-09-29-10_52_24_washing_machine_output.csv")
washing_machine_output.head(10)
```

	Timestamp	Apparent (VA)	Active (W)
0	2022-09-29 10:52:24	0.0	0.0
1	2022-09-29 10:52:25	0.0	0.0
2	2022-09-29 10:52:26	0.0	0.0
3	2022-09-29 10:52:27	0.0	0.0
4	2022-09-29 10:52:28	0.0	0.0
5	2022-09-29 10:52:29	0.0	0.0
6	2022-09-29 10:52:30	0.0	0.0
7	2022-09-29 10:52:31	0.0	0.0
8	2022-09-29 10:52:32	0.0	0.0
9	2022-09-29 10:52:33	0.0	0.0

Figure 3.14: Display of first 10 records in the first two CSV files respectively

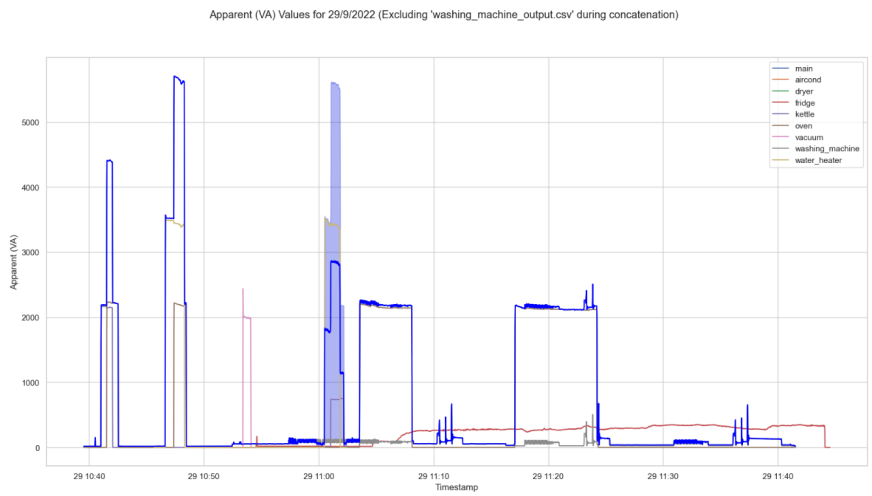

Figure 3.15: Line chart of "Apparent (VA)" against "Timestamp" on 29/9/2022, without including the "2022-09-29-10_52_24_washing_machine_output.csv" file during concatenation

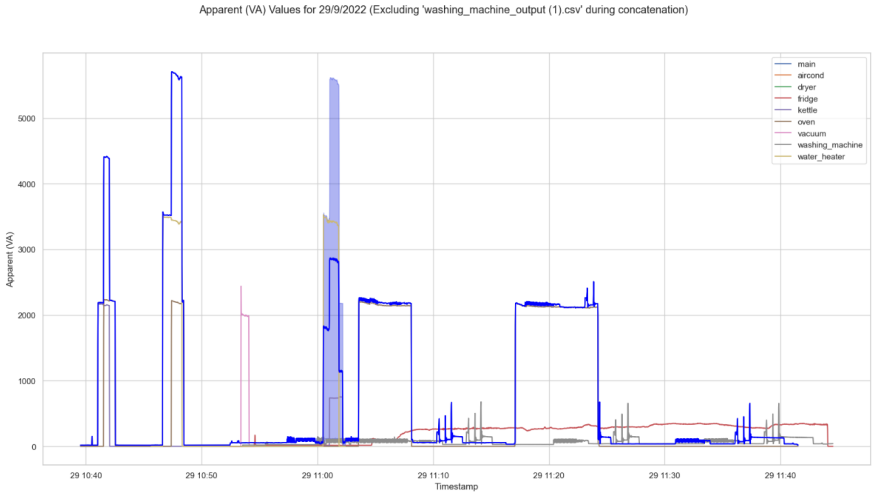

Figure 3.16: Line chart of "Apparent (VA)" against "Timestamp" on 29/9/2022, excluding the "2022-09-29-10_52_24_washing_machine_output (1).csv" file during concatenation

It could be justified that by excluding the "2022092910_52_24_washing_machine_output.csv" file in the "washing_machine" folder, the grey line becomes aligned with the pattern shown by the blue line, as shown in Figure 3.15. This is however not the case for the exclusion of "2022092910_52_24_washing_machine_output (1).csv", as shown in Figure 3.16.

Another possible reason for this misalignment of data between "main" and "washing_machine" could be a short delay of signal transmission from "washing_machine" to "main". This would result in the peak of "main" coming slightly later than the peak of "washing_machine" in the line chart. Nevertheless, as seen from Figure 3.16 above, the peak of "main" is shown earlier than the peak of "washing_machine", and this clearly does not align with the possible reason stated above.

Therefore, we remove the "2022-09-29-10_52_24_washing_machine_output.csv" file from the dataset provided, as the data in this CSV file might have been incorrectly collected or recorded based on our careful observations, possibly due to certain random errors made during the lab experiment.

– **Part B:**

Referring to Figure 3.15 above, the issue of duplicate records for "Type" == "main" has yet to be resolved. After detailed analysis, we discovered that the blue solid line within the blue area marks indicates the average values of "Apparent (VA)" for "Type" == "main", which is due to the features of the *seaborn* library used [14].

Simply taking these average values of "Apparent (VA)" when there are duplicates will definitely solve the issue, but it will lead to another issue where the energy consumption of "water_heater" (represented by yellow line) will become higher than these average values of "Apparent (VA)" for total meter reading, which is undeniably illogical and impractical in the real-world scenario.

As shown in Figure 3.17, we experiment by removing duplicate records for "Type" == "main", by taking the maximum values of "Apparent (VA)". According to the energy usage pattern of this result, as shown in Figure 3.18, we found out that taking the maximum values instead of the average can help to solve the issue of duplicate records for "Type" == "main", as the solid blue line will then always overlay the yellow line, at around the time of "11:00".

```
# Remove duplicate records for "Type" == main", by taking the max "Apparent (VA)" values

df['Timestamp'] = pd.to_datetime(df['Timestamp'])

# Group by 'Timestamp' and 'Type', then find the index of the maximum 'Apparent' value
idx = df.groupby(['Timestamp', 'Type'])['Apparent (VA)'].idxmax()

# Select the rows with the maximum 'Apparent' values
df_remove_duplicates_main = df.loc[idx]
```

Figure 3.17: Python code to remove the duplicate records for "Type" == "main", by taking the maximum values of "Apparent (VA)"

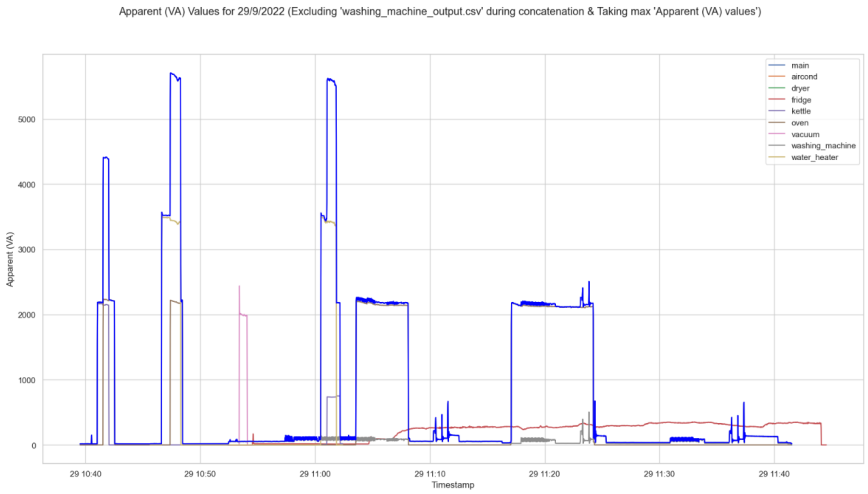

Figure 3.18: Result of data preprocessing to address Data Inconsistency Issue 2

3.6.4 Challenge 1 - Data Reading Error

Some appliances have their "Apparent (VA)" or "Active (W)" data values higher than that of "main", in which "main" represents the total meter reading.

- *Checking:*
  Referring to Figure 3.18 above, we can obviously observe that the energy consumption of both "vacuum" (represented by pink line) and "fridge" (represented by red line) are higher than that of "main" (represented by blue line). This is because both the pink line and red lines overlay the blue line. However, in the real-world scenario, this must not be the case, as it is never possible for the total meter reading to show a value that is smaller than the value of real energy usage by a particular appliance.

- *Preprocessing steps:*
  We assume that this issue is very likely due to certain random errors when conducting the lab experiment. For instance, the sensors tracking the energy consumption of both "vacuum" and "fridge" might be disconnected from the main circuit, in which the main circuit is supposed to be connected with all the existing appliances in use. Due to this unexpected disconnection, the readings from the sensors did not contribute to the total meter reading, resulting in such an issue we met in this case.

  Thus, we decided to simply remove the records of both "Type" == "vacuum" and "Type" == "fridge" on this particular date which is "29/9/2022". This is because the energy consumption of both these appliances were not recorded by the total meter, which is represented by "main" in this case.

  Not only applicable to this particular date, the same preprocessing steps taken here are also applied whenever we see such an issue happening in the line chart of other date(s) as well. As an illustrative example, we present the demonstration performed on the date "29/9/2022".

  After performing data preprocessing, the result in Figure 3.19 shows that the blue line perfectly overlays all the other lines throughout the entire line chart, which is a lot more practical, compared to the previous one (Figure 3.18).

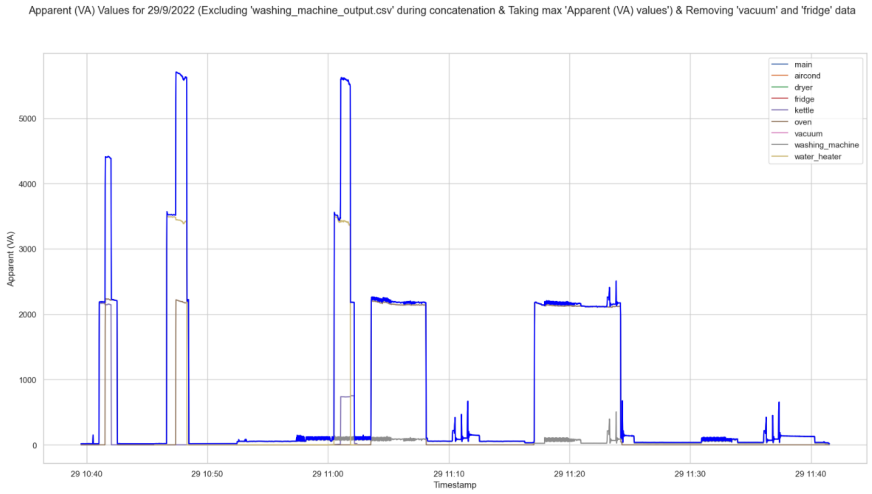

Figure 3.19: Result of data preprocessing to address Challenge 1

3.6.5 Challenge 2 - Non-continuous Timestamp Data

The data values in the "Timestamp" column are not in a continuous sequence, in which certain times/dates are unavailable in between.

- *Checking 1:*
  Based on Figure 3.20 below, when plotting the similar line chart for the date of "27/9/2022", the straight slanting blue line (representing "main") in the middle of the chart appears to be weird as it is very impossible for the total meter reading ("main") to experience such a constant decline in its "Apparent (VA)" values in the real-world scenario. After careful analysis, it is revealed that such an unexpected pattern could be due to the characteristics of seaborn library in which it will directly connect all the existing data points available in the provided dataset [14].

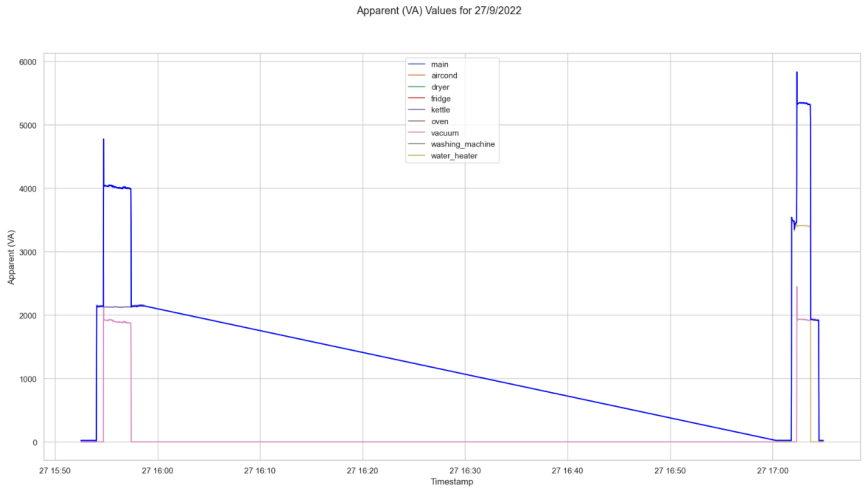

Figure 3.20: Line chart of "Apparent (VA)" against "Timestamp" on the date of "27/9/2022"

- *Checking 2:*
  Based on Figure 3.21, it is suspected that the dataset does not contain the "Timestamp" values with the time data at around "15:59" on the date of "27/9/2022". We have verified this issue by using the Python code as shown in Figure 3.21. The result shows that the time data of "Minute" == 59 when "Hour" == 15 is not available on this particular day.

```
temp_df_ = df[(df['Day'] == 27) & (df['Month'] == 9) & (df['Year'] == 2022) & (df['Hour'] == 15)]

temp_df_["Minute"].unique()
```

```
array([52, 53, 54, 55, 56, 57, 58], dtype=int64)
```

Figure 3.21: Python code to check the missing time data in the "Timestamp" column

- *Preprocessing steps:*
  Break the dataframe into multiple separate dataframes, in which each separate dataframe contains a sequence of continuous date/time data values in the "Timestamp" column. In other words, each separate dataframe consists of records at different time sessions, which are separable by unavailable "Timestamp" data. This is acceptable as practically there is no relationship between one session and another.

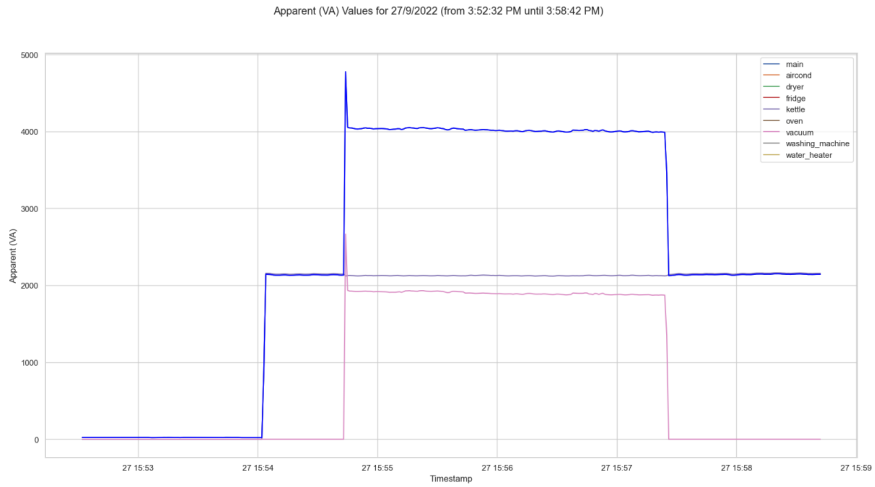

Figure 3.22: Result of data preprocessing to address Challenge 2 (Part 1). This dataframe contains records within a session (consisting of continuous timestamps) from "3:52:32 PM" until "3:58:42 PM" on "27/9/2022"

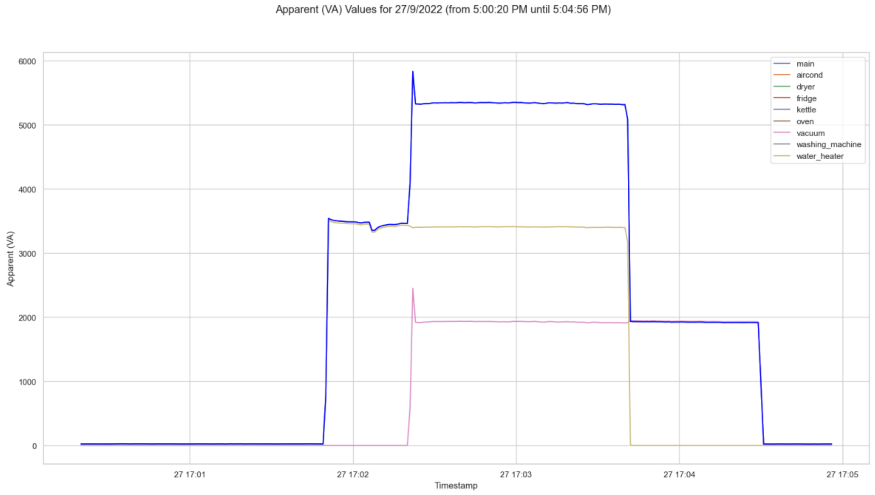

Figure 3.23: Result of data preprocessing to address Challenge 2 (Part 2). This dataframe contains records within a session (consisting of continuous timestamps) from "5:00:20 PM" until "5:04:56 PM" on "27/9/2022"

3.6.6 Data Missing Issue 1 - Missing of Total Meter Reading

The data records for "Type" == "main" at certain timestamps are missing or unavailable, although there exist records for the other appliance types at these particular timestamps.

- *Checking:*
  Plot the line chart of "Apparent (VA)" against "Timestamp" for "Day" == 9, "Month" == 11 and "Year" == 2022. It is clearly evident that there are data available at the "Timestamp" before around "11:30", but these data are only for both the "dryer" and "fridge" appliances (represented by green and red line respectively), but not for the "main" (represented by blue line).

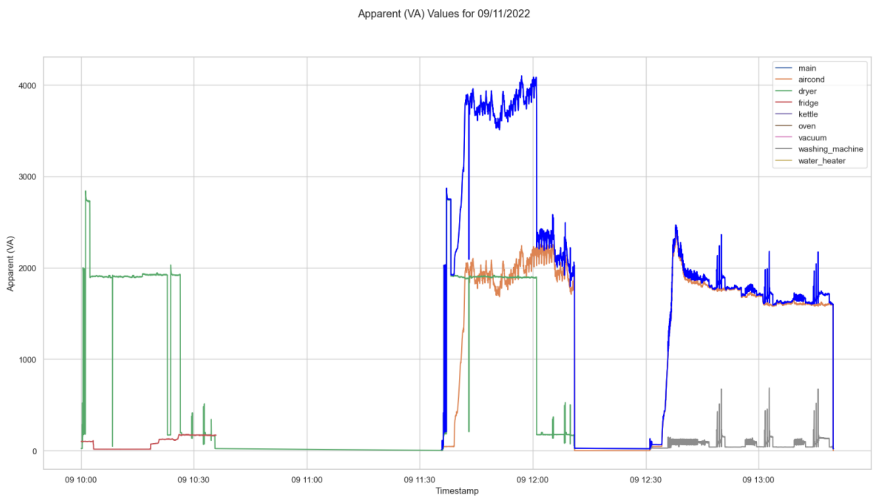

Figure 3.24: Line chart of "Apparent (VA)" against "Timestamp" on the date of "09/11/2022"

- *Preprocessing steps:*
  We assume that this issue is very likely due to missing CSV file(s) in the "main" subfolder within the dataset provided by MIMOS Berhad. Thus, we decided not to impute or create any data values for "Type" == "main" at these timestamps, as this would cause data authenticity issues. Nevertheless, we made the decision to simply remove the records for "Type" == "dryer" and "Type" == "fridge" at these timestamps, which is from "10:00:06 AM" until "10:35:52 AM" on the date of "09/11/2022". This could be done by removing the data records in both "2022-11-09-10_00_06_dryer_output.csv" and 2022-11-09-10_00_06_fridge_output.csv" files from the "df" dataframe before plotting the line chart.

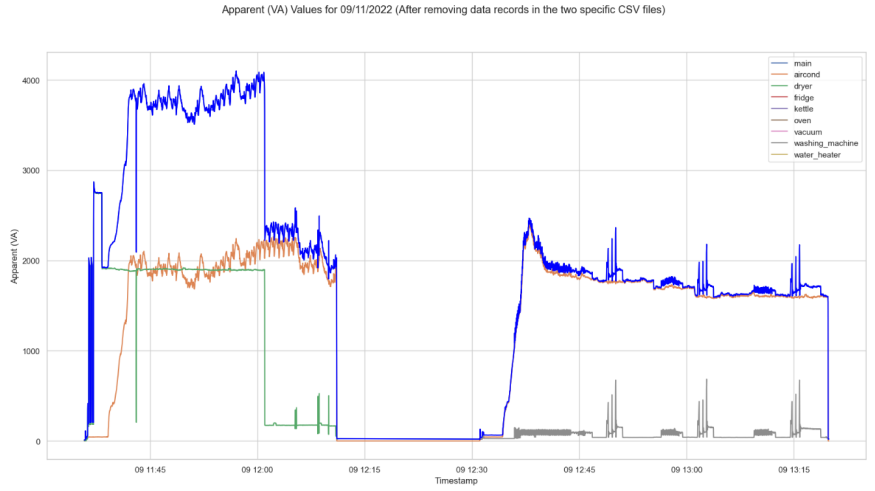

Figure 3.25: Result of data preprocessing to address Data Missing Issue 1

Definitely, based on Figure 3.25, a similar issue as what we have discussed previously in Section 3.6.5 Challenge 2 has been encountered again. Therefore, further data preprocessing steps similar to that in the Section 3.6.5 Challenge 2 will be carried out to separate the dataframe into different sessions (each consists of a sequence of continuous timestamps) on this particular day.

3.7 Descriptive Statistics

```python
# Remove all rows containing value of 0
df_removed_0 = df[df.ne(0).all(axis=1)]

# Extract useful columns only
df_removed_0 = df_removed_0[['Apparent (VA)', 'Active (W)', 'Type']]

# Group the dataframe by 'Type' and apply 'describe()' for each group
grouped = df_removed_0.groupby('Type')
for name, group in grouped:
    print(f"Describe the Type '{name}':")
    print(group.describe())
    print("\n")
```

Figure 3.26: Python code to print out a summary of statistics based on different appliance types

Describe the Type 'aircond':

	Apparent (VA)	Active (W)
count	8323.000000	8323.000000
mean	1476.409795	1374.446206
std	669.471213	655.729785
min	0.040000	0.010000
25%	1588.845000	1499.135000
50%	1685.620000	1580.070000
75%	1874.625000	1765.745000
max	2391.680000	2333.770000

Describe the Type 'kettle':

	Apparent (VA)	Active (W)
count	2163.000000	2163.000000
mean	1874.668488	1873.121993
std	716.245651	718.515349
min	0.040000	0.010000
25%	2135.735000	2135.615000
50%	2153.280000	2153.160000
75%	2187.870000	2187.730000
max	2245.850000	2245.710000

Describe the Type 'vacuum':

	Apparent (VA)	Active (W)
count	2040.000000	2040.000000
mean	1773.431500	1746.048505
std	544.480619	536.563573
min	0.040000	0.010000
25%	1882.290000	1857.065000
50%	1951.030000	1919.975000
75%	1970.822500	1939.847500
max	3051.920000	2941.690000

Describe the Type 'dryer':

	Apparent (VA)	Active (W)
count	15574.000000	15574.000000
mean	1042.372205	1039.995602
std	881.555841	883.300231
min	0.120000	0.010000
25%	163.810000	159.870000
50%	1810.355000	1809.545000
75%	1880.675000	1880.065000
max	2866.750000	2866.580000

Describe the Type 'main':

	Apparent (VA)	Active (W)
count	35177.000000	35177.000000
mean	1428.711140	1392.465708
std	1410.637777	1406.342416
min	1.670000	0.260000
25%	102.260000	77.050000
50%	1657.300000	1548.850000
75%	2202.090000	2185.320000
max	5834.800000	5805.660000

Describe the Type 'washing_machine':

	Apparent (VA)	Active (W)
count	20091.000000	20091.000000
mean	69.403829	51.338133
std	56.618147	42.561806
min	12.440000	6.920000
25%	27.520000	20.900000
50%	52.790000	38.350000
75%	95.910000	71.325000
max	697.640000	504.950000

Describe the Type 'fridge':

	Apparent (VA)	Active (W)
count	9110.000000	9110.000000
mean	198.951963	113.444819
std	109.835692	60.758711
min	13.770000	5.940000
25%	114.032500	73.945000
50%	258.135000	143.435000
75%	292.127500	159.197500
max	353.030000	189.510000

Describe the Type 'oven':

	Apparent (VA)	Active (W)
count	3917.000000	3917.000000
mean	1709.795183	1707.881764
std	894.112905	896.721485
min	0.040000	0.010000
25%	2116.800000	2116.740000
50%	2167.910000	2167.280000
75%	2190.830000	2190.150000
max	2325.290000	2325.160000

Describe the Type 'water_heater':

	Apparent (VA)	Active (W)
count	2177.000000	2177.000000
mean	3473.171994	3466.482540
std	252.809884	253.555837
min	0.100000	0.020000
25%	3433.060000	3426.700000
50%	3530.510000	3524.840000
75%	3545.210000	3539.370000
max	3642.280000	3634.620000

Figure 3.27: Summary of statistics for each appliance

- *Measures of central tendency:*

  The summary of statistics shown in Figure 3.27 can help us understand some of the main characteristics of data, such as its measures of central tendency and variability.

  For example, if we look at the summarised statistics of Type "main" which indicates the total meter reading, we know that the average energy consumption in the MIMOS laboratory throughout all the experiment days is 1428.711140 and 1392.465708 respectively for its "Apparent (VA)" and "Active (W)" readings. Furthermore, the median of energy consumption in the MIMOS laboratory throughout all the experiment days is 1657.300000 and 1548.850000 respectively for its "Apparent (VA)" and "Active (W)" readings.

Additionally, based on the summarised statistics in Figure 3.27, it is worth noting that, overall, the "Apparent (VA)" readings are higher than that of the "Active (W)" readings on average, by comparing their mean values in every appliance. The valid reason behind this is that, usually a small portion of the original power from the "Apparent (VA)" value is not consumed by the load or simply the appliance itself, which could be due to power loss to the resistors or power loss as heat energy to the surroundings. This results in an "Active (W)" value that is slightly smaller than that of the original power, "Apparent (VA)" [15].

- *Variability:*

On the other hand, based on Figure 3.27, we can see that the standard deviation values for "Apparent (VA)" and "Active (W)" readings of the Type "main" are 1410.637777 and 1406.342416 respectively.

These standard deviation values with the mean values tell us the information such that in a normal distribution, approximately 68% of the "Apparent (VA)" readings will be within one standard deviation above (1428.711140 + 1410.637777 = 2839.348917 VA), and one standard deviation below (1428.711140 - 1410.637777 = 18.073363 VA) the average "Apparent (VA)" reading, which is within 18.073363 VA and 2839.348917 VA to be specific. The same applies to the "Active (W)" readings.

The interquartile range (IQR) is also useful as it can be used to determine the possible outliers by using a specific formula. To elaborate, from Figure 3.27, we can see that the Q1 value and Q3 value for "Apparent (VA)" of Type "main" are 102.26 and 2202.09 respectively. Therefore, the IQR is 2099.83 (calculated by Q3 - Q1 = 2202.09 - 102.26). In this case, any data in the "Apparent (VA)" column with value smaller than (Q1 - 1.5 * IQR) or larger than (Q3 - 1.5 * IQR) will be considered as possible outliers.

- *Distribution:*

Boxplots can be used to visualize the overall distribution of the data in certain columns. In the examples shown in Figure 3.28 and Figure 3.29, the two boxplots display the distributions of "Apparent (VA)" and "Active (W)" data values respectively, based on different appliance types.

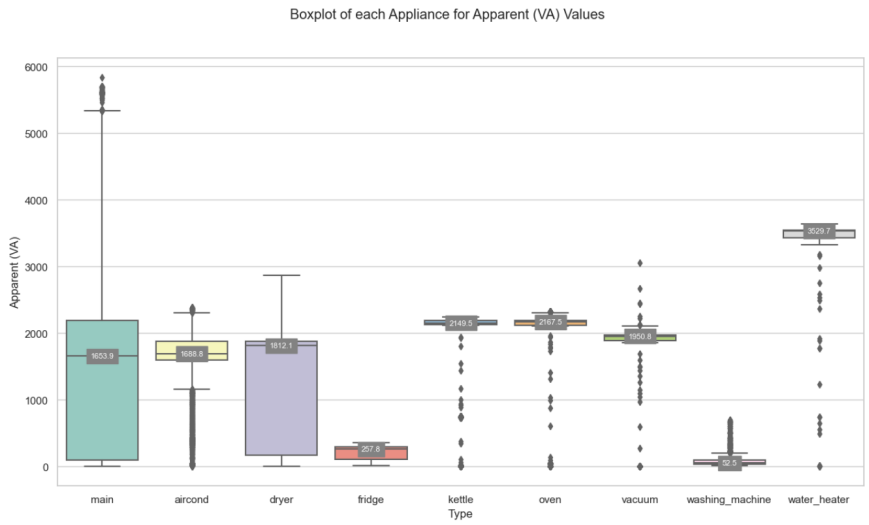

Figure 3.28: Boxplot showing the distribution of "Apparent (VA)" data values for every appliance

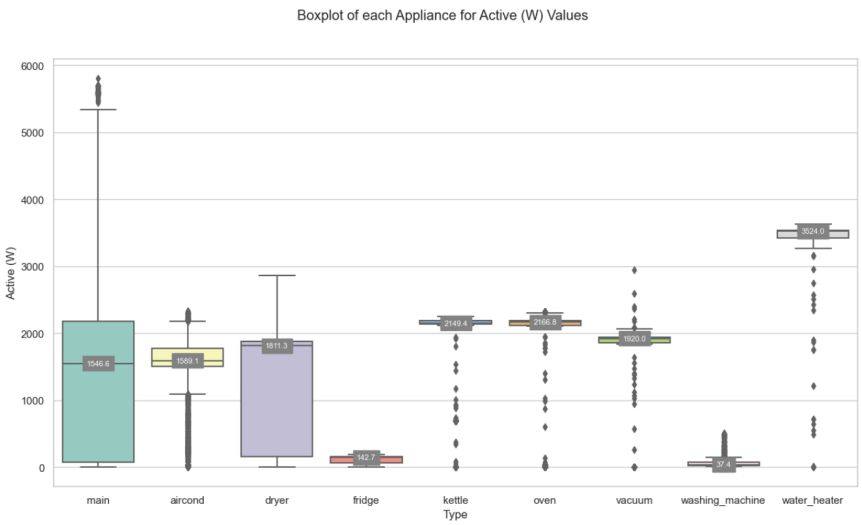

Figure 3.29: Boxplot showing the distribution of "Active (W)" data values for every appliance

Looking at the boxplot in Figure 3.28 for "Apparent (VA)" of Type "main", it could be deduced that the distribution is positively skewed. Since the whisker to the right of this boxplot is longer than that to the left, we could deduce that more extreme values exist towards the positive end. In other words, more "Apparent (VA)" readings are comparatively far from the median "Apparent (VA)" value being gathered on the right side of the boxplot. This makes the entire boxplot positively skewed.

Also, it is evident that this boxplot has a larger distance between Q1 and median, compared to that of between Q3 and median. It indicates that the "Apparent (VA)" readings within the lower 25% are relatively compact or clustered, compared to that of the upper 75%.

Instead of boxplots, histograms can also be used to visualise the overall distribution of the data, as shown in Figure 3.30. This result also equally supports our claim that both the distributions of "Apparent (VA)" and "Active (W)" data are positively skewed.

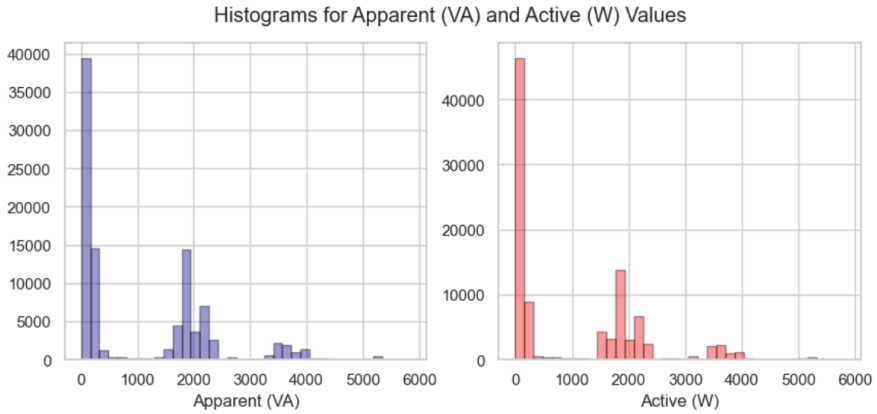

Figure 3.30: Histograms showing the distribution of both "Apparent (VA)" and "Active (W)" data values

3.8 Discussion

Since both boxplots in Figure 3.28 and Figure 3.29 above display very similar distributions for every appliance type overall, we would expect to see a strong positive linear correlation between the "Apparent (VA)" data and the "Active (W)" data.

A scatter plot is produced to observe the correlation between "Apparent (VA)" data and the "Active (W)" data of different appliances (excluding "main").

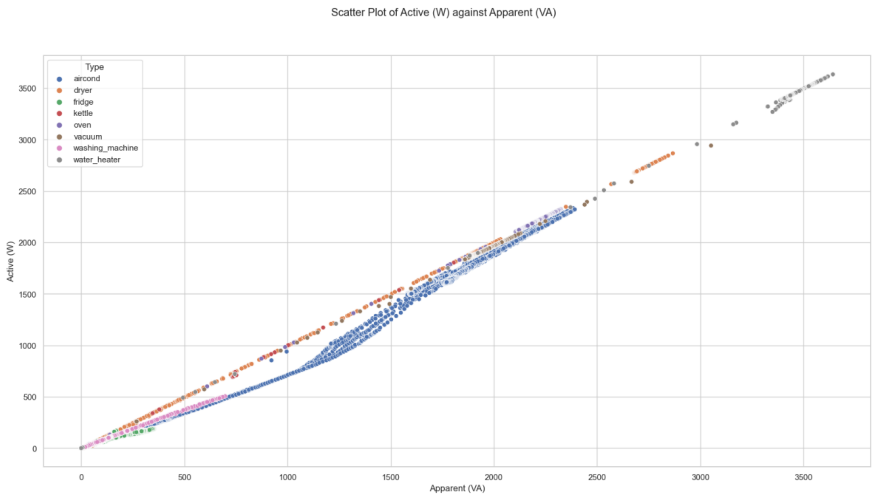

Figure 3.31: Scatter plot showing the correlation between "Apparent (VA)" data and the "Active (W)" data

```
# Calculate Pearson's correlation coefficient

import numpy as np

x = df_removed_0['Apparent (VA)']
y = df_removed_0['Active (W)']

r = np.corrcoef(x, y)[0, 1]

print("Pearson's correlation coefficient (r-value):", r)
```

```
Pearson's correlation coefficient (r-value): 0.9988375838547553
```

Figure 3.32: Python code to calculate the Pearson's correlation coefficient between "Apparent (VA)" data and "Active (W)" data

Similar to our expectation, there is a very strong positive linear correlation between the "Apparent (VA)" data and "Active (W)" data. This gives information that an increase in the "Apparent (VA)" values will very highly likely result in an increase in the "Active (W)" values. This is true and reasonable, as higher "Active (W)" values can be absorbed and used by the loads or appliances themselves if the original power provided by "Apparent (VA)" values is higher.

This strong positive linear correlation is also supported by the extremely high Pearson's correlation coefficient (r-value) which has a value of 0.9991410697023518.

Based on the clusters in Figure 3.31, the below insights and findings are derived.

- ***"aircond" cluster*** *(represented by blue data points):*

  Generally, these data points show a relatively weaker linear correlation between "Apparent (VA)" and "Active (W)" values, compared to the data points of other appliances. It could be inferred that the air-conditioner used for the lab experiment is old and outdated, resulting in inefficient energy conversion from "Apparent (VA)" values into "Active (W)" values [16]. Therefore, larger portion of original energy provided is dissipated within the air-conditioner itself, rather than being fully utilized to generate cool air. This causes "aircond" to have relatively lower "Active (W)" values than "Apparent (VA)" values.

- ***"water_heater" cluster*** *(represented by grey data points):*

  This cluster can be obviously seen at the top-right region within the scatter plot. In essence, it visually describes that the water heater is the appliance that contributes the most to the total meter reading in this lab experiment, for both "Apparent (VA)" and "Active (W)" values.

- ***"washing_machine" cluster*** *(represented by pink data points):*

  These data points only cluster at the bottom-left region within the scatter plot, which visually describes that the washing machine only consumes relatively little power in general, if compared to that of the other appliances, for both "Apparent (VA)" and "Active (W)" values.

- ***"dryer" cluster*** *(represented by orange data points):*

  These data points are widely spread across a huge range of "Apparent (VA)" and "Active (W)" values, ranging approximately from 0 to 2800. This could give us the inference that the dryer used in the lab experiment consumes a wide range of energy. This could probably be due to different settings and configurations of the dryer, such as the level of heat, wind power, as well as spinning power.

Moreover, rather than assessing the relationship between continuous variables, we would also like to inspect the discrete relationship between the frequencies of different appliances being switched on throughout different days. A grouped bar chart is produced to visualise this relationship, prior to any data preprocessing steps.

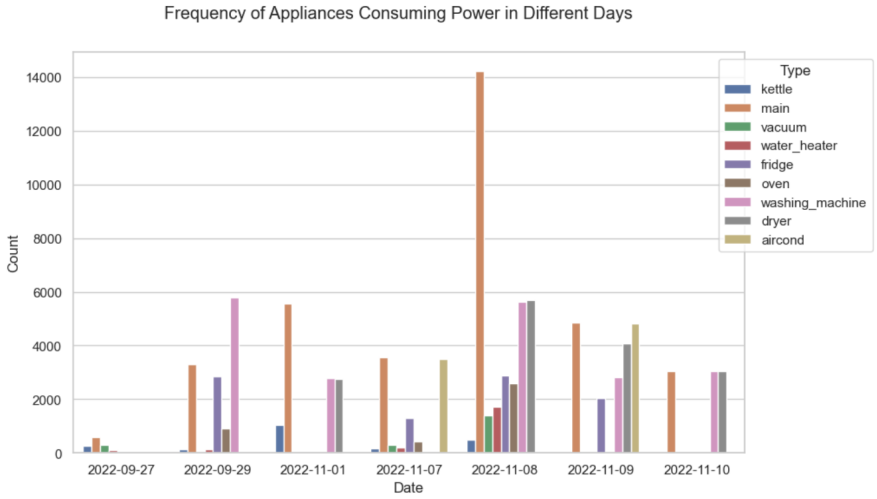

Figure 3.33: Grouped bar chart showing the frequency of different appliances consuming power based on different days

Interestingly, this grouped bar chart can also display certain interesting insights such as the obvious data issues in the provided dataset. To give an illustration, if we inspect carefully the individual group of bar charts on "2022-09-29", it is obvious that the frequency of "washing_machine" (represented by pink bar) consuming power is higher than that of "main" (represented by orange bar). As discussed previously, this situation is impractical and impossible in the real-world scenario.

Therefore, it could be instantly deduced that there are certain data issues existing for the records on "2022-09-29", in which preprocessing steps must be taken and more careful analysis should be done to ensure data transparency.

Furthermore, based on Figure 3.33, there also shows some cases where the frequency of "main" has a very similar value as the frequency of the other appliances, such as on "2022-11-07", "2022-11-09" and "2022-11-10". Thus, extra care must be given to these dates during the data preprocessing stage to see whether or not there is any data issue leading to such patterns in this grouped bar chart.

3.9 Results and Findings

In summary, from this data analysis report, we discovered that plotting line charts of "Apparent (VA)" and "Active (W)" against "Timestamp" via the seaborn library is extremely useful for exploring and showing the signs of certain data issues. Also, the grouped bar chart displaying the frequencies of power consumption for each appliance over different days can help us to immediately outline some interesting insights as well as identify possible data issues. From the scatter plot produced, it is particularly evident that the air-conditioner has comparatively smaller "Active (W)" values if compared to its "Apparent (VA)" values.

In addition, we retrieved an interesting insight showing that the "Active (W)" values of appliances are lower than their "Apparent (VA)" values on average. Following this, we also learned that for the case where an appliance has relatively lower "Active (W)" values than its "Apparent (VA)" values, it could be inferred that there could be some energy lost to the load or appliance itself, as well as energy being lost as heat to the surroundings.

On the other hand, there were some potential outliers determined from the statistical calculation. However, after careful analysis, we decided to not simply remove or impute these potential outliers since the values of these outliers are still within the feasible range of the typical energy consumption by the particular appliances. Hence, these extreme values might be able to give useful and insightful information for our NILM project at a later stage.

3.10 Conclusion

In summary, certain data issues have been explored and identified at this early stage of our NILM project. Appropriate data preprocessing steps have been discussed and justified to fix these data issues correctly. These data preprocessing steps can help us to speed up the execution phase of our project, as less effort would be required to preprocess the data again at a later stage. Last but not least, this data analysis report is significant for us to have a better understanding and detailed analysis of the dataset provided by MIMOS Berhad, which will prepare us to implement our NILM project excellently.

Chapter 4

Hardware Specifications

4.1 Comparison between Team Members' Laptops

Table 5: Specifications of all of our team members' laptops

	Ming Sheng	**Wai Qi**	**Martin**	**Kai Jie**
Central Processing Unit (CPU)	AMD Ryzen™ 5 4600H	AMD Ryzen™ 5 4600H	Intel Core i5-8250U	Dual-core 7th-generation Intel Core i5
CPU architecture	x86	x86	x86	x86
Virtual CPUs (vCPUs)	12 cores	12 cores	4 cores	2 cores
Random-Access Memory (RAM) Storage	16 GB	8 GB	8 GB	8 GB
Graphics Processing Unit (GPU)	AMD Radeon™ Graphics, 3000 Mhz	AMD Radeon™ Graphics, 3000 Mhz	Intel UHD Graphics 620	Intel Iris Plus Graphics 640 (Integrated)
Physical Storage Capacity	512 GB	512 GB	512 GB	256 GB

After conducting a thorough comparison, we have identified the most optimal laptop specifications among the currently available options is **Ming Sheng's laptop**, due to having the largest number of cores available (12 cores), largest RAM storage capacity (16 GB), most sophisticated GPU's performance, and lastly largest physical storage capacity (512 GB).

4.2 Comparison between Best Laptop and Cloud VM

When determining the appropriate hardware specifications to meet our computing requirements, various factors must be taken into account. These factors encompass performance prerequisites, financial limitations, and adaptability. Below, we will provide rationale for the hardware specifications in two situations: our existing best laptop and an alternative based on cloud computing.

Table 6: Hardware Specifications

	Currently available laptop specifications	**Ideal specifications of Virtual Machine (VM) from Google Cloud**
Central Processing Unit (CPU)	AMD Ryzen™ 5 4600H	Intel Cascade Lake
CPU Architecture	x86	x86
Virtual CPUs (vCPUs)	12 cores	12 to 96 cores (Based on the configuration chosen)
Random-Access Memory (RAM) Storage	16 GB	85 to 1360 GB
Graphics Processing Unit (GPU)	AMD Radeon™ Graphics, 3000 Mhz	NVIDIA A100 GPUs (Up to 16 GPUs)
Physical Storage Capacity	512GB	Maximum 3 TB of local SSD

Our current best laptop specifications, as mentioned earlier, provide a versatile computing experience suitable for everyday tasks. However, as our computing demands grow in this NILM project, we recognize the necessity for enhanced performance to effectively handle more demanding workloads. Consequently, we have chosen to acquire **Google Cloud A2 Virtual Machine (VM)** services, starting at a rate of USD$0.87 per hour [17]. This decision is strategically aimed at elevating our overall productivity and addressing the escalating demands of our tasks. Google Cloud A2 VM is particularly well-suited for deep learning tasks, such as our NILM project, supplying the computational power needed for data-intensive and compute-intensive operations. It not only enhances performance but also offers the adaptability, scalability, and cost-effectiveness required to align with our evolving computing requirements and financial constraints. This strategic investment in cloud resources ensures our competitiveness and efficiency in today's dynamic computing landscape.

4.2.1 Scalability

Our current available laptop, which features an AMD Ryzen™ 5 4600H processor and AMD Radeon™ Graphics, has fixed and limited computational capabilities. As our NILM project expands, particularly during the training of intricate PyTorch models, the laptop may encounter difficulties in handling the increased demands. Conversely, the virtual machine (VM) offers the flexibility to be configured with up to **96 cores** and **16 NVIDIA A100 GPUs**. This built-in scalability guarantees that as our project's computational requirements increase, the VM can adapt accordingly [18].

Since deep learning tasks demand substantial computational resources, a higher core count can significantly accelerate the training process [19]. Whether it's in CPUs or GPUs, an increased core count can lead to faster training and enhanced handling of extensive datasets, ultimately resulting in a reduction in the time required to train models, especially beneficial for our NILM project. This underscores there is a possibility for an environment with a greater number of CPU cores over our current laptop specification which only offers 12 cores.

Furthermore, utilising a virtual machine (VM) enables us to expand our RAM capacity significantly, going from **85 GB to an extensive 1360 GB**. This provides us with abundant memory resources capable of effectively managing large datasets and memory-intensive tasks. In terms of storage capacity, our current laptop provides a decent 512 GB of storage, but the VM can surpass it with an impressive SSD storage capacity up to a **maximum of 3 TB**. This ample storage capacity proves highly advantageous for accommodating the extensive datasets often encountered in our NILM research.

All in all, this inherent scalability ensures stable performance, even when dealing with larger datasets and complex models.

4.2.2 State-of-the-Art (SOTA) Hardware

Our current laptop, equipped with the AMD Ryzen™ 5 4600H, is suitable for various tasks but may not be considered the optimal choice for deep learning endeavours due to its GPU, the AMD Radeon™ Graphics, which, while capable for everyday tasks, might encounter limitations when handling extensive computational operations such as those in NILM. In contrast, the Google Cloud virtual machine (VM) boasts Intel Cascade Lake CPUs and NVIDIA A100 is a forefront of computational hardware specifically designed for AI and deep learning.

We opted for the **NVIDIA A100 GPU** for our project because of its robust architecture, highlighted by its impressive count of **6912 CUDA cores**. These cores, specifically designed for parallel processing, make the A100 exceptionally adept at handling intensive deep learning computations. Moreover, the A100 not only boasts up to **80GB of memory** but also provides an astounding **19.5 teraflops of FP32 performance** and **1.6TB/s** of graphics memory bandwidth. This combination of features, spearheaded by the CUDA cores, makes the **world's fastest** deep learning GPU [20].The VM setup's capability to utilise multiple A100 GPUs means that the aggregated CUDA cores (**maximum of 16 GPUs**) can significantly expedite PyTorch computations, resulting in substantial reductions in both the time required for model training and inference in the context of NILM. This configuration establishes it as state-of-the-art hardware ideally suited for tasks like NILM.

4.2.3 Flexibility

The hardware capabilities of our current laptop are fixed, which means that making adjustments or upgrades can be a convoluted process limited by the laptop's inherent design constraints. In contrast, cloud infrastructure provides considerable flexibility. We aren't bound to a single hardware choice and can tailor our resources to match the precise phase and computational demands of our project as needed [18].

4.2.4 Collaboration

Collaborating on a local machine often involves manual sharing of data and code, which can be cumbersome and lead to challenges in version control and concurrent access. Conversely, cloud computing offers the capacity to streamline collaboration. Within a cloud-based environment, we can concurrently access the same virtual machine (VM), collaborate on shared datasets, and leverage integrated tools like version control [21]. This setup proves particularly beneficial for our project, given our extensive need for collaborative work within the group.

In conclusion, while the local laptop provides a starting platform, the Google Cloud VM offers superior scalability, access to state-of-the-art hardware, unparalleled flexibility, and seamless collaboration capabilities—attributes that are paramount for complex tasks such as NILM using PyTorch.

Chapter 5

Software Specifications

5.1 Programming Language

Table 7: Comparison between Programming Languages

	R	**Python**
Primary Usage	Statistics and data analysis	General purpose and data science
Popularity in Deep Learning	Low	High
Flexibility	Moderate	High
Ecosystem	Nearly 19,000 packages available in the Comprehensive R Archive Network (CRAN)	+300,000 available packages in the Python Package Index (PyPi)
Performance in Deep Learning	Not optimised	Very Good
Integration with NILM	Rare	Common

After careful analysis, we have selected **Python** as the main programming language for our NILM project. Justifications are stated below:

- *Python*:
  Python has become the primary language for many data science tasks, including NILM. This is because of its extensive library support, readability, and the versatility of the language. Its deep learning ecosystem (TensorFlow, PyTorch, Keras, Scikit-learn, Matplotlib) is mature and well-supported by the community [22]. For a project like NILM, Python offers a robust set of tools for data preprocessing, feature extraction, model training, evaluation, and deployment.

- *R*:
  R is historically a language for statistical analysis and visualization. It's well-regarded in the statistics community, and its package ecosystem is geared towards statistical models and tests [22]. While R has packages like Keras and mxnet to support deep learning, its ecosystem for DL is not as mature or extensive as Python's. For tasks like NILM, the availability of tools and community support in R is limited compared to Python.

5.2 Deep Learning Frameworks

Table 8: Comparison between Deep Learning Frameworks

	PyTorch	**TensorFlow**	**Keras**
API Level	Low	High and Low	High
Graph Type	Dynamic	Static	Depends
Debugging	Easy (due to Dynamic Graph)	Complex	Simplified (due to high API level)
Performance	Excellent	Excellent	Slow
Programming Language	Python, LUA	Python, C++, CUDA	Python
Datasets	Large	Large	Small
Flexibility	High	High	Low
Research vs Production	Research	Production	Early prototyping
Community Support	Excellent	Excellent	Excellent
Learning Curve	Moderate to steep	Steep	Low to moderate

- *PyTorch*:
  - Offers a dynamic computation graph, which is intuitive and allows more flexibility during model creation, especially for research-based tasks [23].
  - Well-suited for tasks that need to change the architecture frequently (experimental phase).
  - Its pythonic nature and straightforward approach are beneficial for NILM, where the model could need frequent adjustments.
- *TensorFlow*:
  - TensorFlow provides a mature platform and has been adopted widely in the industry.
  - The introduction of TensorFlow 2.0 provides a more user-friendly approach combining Keras' ease with TensorFlow's power [23].
  - For large-scale projects or when deployment is a priority, TensorFlow offers various tools and optimizations.
- *Keras*:
  - Keras is a high-level API designed for simplicity and ease of use. It acts as a wrapper around TensorFlow [24].
  - Ideal for beginners or for projects where a prototype needs to be developed quickly.
  - Given its high-level nature, it might not offer as much flexibility for intricate NILM tasks as PyTorch or pure TensorFlow, but it can serve as a good starting point.

In the process of selecting the most suitable framework for our NILM project, we've gravitated towards **PyTorch**. This decision stems from several of its standout features. Firstly, its dynamic computation graph not only simplifies the debugging process but also allows for rapid model iteration, a critical advantage when dealing with research-driven tasks. Additionally, PyTorch's inherently pythonic design ensures that the code remains clean, readable, and aligns seamlessly with our programming environment. This characteristic is further emphasised by its strong orientation towards the research community, enabling a level of flexibility that's paramount for our experimental endeavours [25]. Another compelling factor is the extensive community support behind PyTorch, which becomes invaluable when navigating challenges. Lastly, with the advent of tools like TorchServe, the path from research prototypes to deployable models becomes more straightforward. Taking these considerations into account, PyTorch distinctly aligns with the goals and requirements of our NILM project.

5.3 Software Libraries

Table 9: Software Libraries

Libraries	**Justification**
NILMTK 0.4.3	NILMTK (Non-Intrusive Load Monitoring Toolkit) is a Python library designed for energy disaggregation and load monitoring tasks. It provides a variety of tools and algorithms for NILM.
matplotlib 3.7.2	Matplotlib is a powerful data visualisation library in Python, offering a wide range of plotting functions for creating visualisations that help in understanding data patterns and trends.
numpy 1.25.0	NumPy is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions.
sklearn 1.3.0	sklearn is a machine learning library for Python, offering a wide range of machine-learning algorithms and tools for data preprocessing, model evaluation, and model selection.
pandas 2.1.0	pandas is an open-source data analysis and manipulation tool for Python. It simplifies data handling and preparation tasks.
seaborn 0.12.2	Seaborn is a Python library, built on top of Matplotlib, for creating attractive statistical visualisations with ease.

While the software libraries listed in Table 9 above serve as our primary tools, it's important to note that our toolkit extends beyond these selections. These libraries form the cornerstone of our approach, providing essential functionality for our project. However, we maintain the flexibility to incorporate additional libraries and tools as needed to address specific requirements and challenges, ensuring the versatility and effectiveness of our solution.

5.4 Database system

Table 10: Comparison between Database Systems

	MongoDB	**PostgreSQL**
Data Model	NoSQL (Document-based)	SQL (Relational)
Supported Data Types	JSON, BSON	All major data types
Query Language	JSON-like queries and aggregation pipeline	SQL (Structured Query Language)
Complex Queries	Well-suited for simple queries and fast reads	Excellent support for complex queries
Joins	No native support for joins	Supports joins and complex relationships
Performance	High read and write throughput for simple queries	Excellent performance for complex queries and transactions
Security	Secure, supports encryption and authentication	Secure, supports encryption and authentication
Cost	Free and open-source	Commercially available, with a free open-source version

In our choice of a database system for NILM dataset storage, **PostgreSQL** has emerged as the preferred option, despite MongoDB sharing similar advantages. PostgreSQL's robust support for all major data types, excellent handling of complex queries, and the ability to perform joins, which is crucial for merging our NILM datasets, played a pivotal role in this decision [26]. Additionally, our team's familiarity with PostgreSQL further reinforces its suitability for our project, ensuring seamless integration and efficient utilisation of the database system.

5.5 Operating System

The available operating systems within our group are MacOS and Windows. However, it is worth noting that **any OS is acceptable** for our project as we will be using the Google Cloud A2 Virtual Machine (VM) services. Using the Cloud A2 VM for our operating system environment offers a number of other benefits, including scalability and security. We can easily scale our resources up or down as needed, without having to worry about managing physical hardware. This is important for our NILM project, as we may need to scale up our resources to handle large datasets or complex computations. Furthermore, the Google Cloud providers offer a wide range of security features to protect our data and applications. This is important for our NILM project, as we will be handling sensitive data. Additionally, using a cloud-based operating system environment can make it easier for us to collaborate on our NILM project. We can access our environment and data from anywhere in the world, and we can easily share our work with each other.

5.6 Visualisation Tools

We will be utilising both **matplotlib** and **seaborn** libraries as the main visualisation tools in our chosen Python programming environment. This is because our group is familiar with the functionalities of these two libraries. They are two powerful and versatile data visualisation libraries in Python that offer a wide range of charts and graphs for us to clearly demonstrate our data analysis capabilities which is extremely useful in our data analysis stage. We utilise matplotlib for creating and customising our plots whereas seaborn offers a user-friendly interface with a number of features for creating statistical visualisations, thus making it easy for us to create appealing and informative visualisations.

5.7 Programming Language Environment (IDE)

Table 11: Comparison between Programming Language Environment

	Jupyter Notebook	**Google Colab**
Platform	Local installation required	Cloud-based, no installation needed
Accessibility	Locally only	Across multiple devices
Collaboration	Limited sharing capabilities	Offers real-time collaboration and easy sharing
Cost	Free to use	Free access to basic features, but offers paid options for more resources
Programming Languages	Supports multiple programming languages	Primarily designed for Python, with limited support for other programming languages
Hardware Resources	Limited by local hardware, but can utilise cloud resources if configured	Access to Google's cloud infrastructure, including GPUs and TPUs
Code Execution	Local execution	Cloud-based execution
Performance	Dependent on the local device's resources	High performance due to running on cloud
Integration	Can be integrated with various tools and platforms	Well integrated with Google Drive, GitHub, and other Google services
Data Storage	Either save locally or with cloud storage if configured	Saves in Google Drive only
Offline Usage	Can work offline	Requires an internet connection
Runtime	No restrictions	Maximum of 12 hours of runtime

We have opted for **Jupyter Notebook** over Google Colab for our programming language environment. Although the accessibility and real-time collaboration of Google Colab are undeniably better than Jupyter Notebook, Jupyter Notebook provides us full control over our development environment. Using Jupyter Notebook, we are free to do any configuration, package installation, integration, and customization according to our preferences which expands our possibilities whereas Google Colab has limitations in these aspects as they come pre-installed, even with the paid options, there is no explicit assurance of the specific resources you will receive [27]. Furthermore, Google Colab imposed a restriction of a maximum of 12 hours of runtime per day, which may potentially disrupt our workflow, particularly when dealing with long-running tasks such as model training [27]. By choosing Jupyter Notebook, we circumvent this limitation and ensure that our work continues uninterrupted, regardless of the duration or complexity of our projects.

5.8 Front-End Framework

For our front-end development, we have made a strategic decision to use **Angular** as our front-end framework, along with RxJS and NGRX libraries. Angular is a framework that offers a robust and well-defined methodology for constructing a web-based application. With its component-based architecture, we have the capability to create reusable and encapsulated user interface components, thereby promoting modularity and maintainability across our project [28].

We have decided to incorporate the RxJS library into our development. RxJS allows us to handle complex asynchronous operations as observables, a core concept in RxJS that is being used to facilitate reactive programming, including user interactions, data retrieval, and real-time updates [29].

For handling our web-based application's state, we have chosen NgRx, a state management library purpose-built for Angular applications. NgRx is capable of centralising and maintaining a predictable data flow. We can specify actions, reducers, selectors, and effects to manage state alterations, address side effects, and manage asynchronous tasks [30]. This approach guarantees the coherence and long-term maintainability of our web-based application, even as it expands in complexity.

5.9 Project Management Tools

Table 12: Project Management Tools

	Alternatives
Version Control System	Git
Online Communication Channel	WhatsApp
	Microsoft Teams
	Discord
Project Planning	Jira
	Trello
	Google Calendar
Time Tracking	Clockify
Document Sharing Platform	Google Drive
	Microsoft Teams
Reference Management Tool	Zotero
	Overleaf
Document Processing/Text Editing Software	Overleaf
	Microsoft Word
	Google Docs

5.9.1 Version Control System

A version control system is an essential component in a software development project. In our software development, we opted for **Git** as our version control system. Git offers us a robust framework to track the changes we make, ensuring all group members stay in sync with the latest updates. It facilitates collaborative efforts, as it provides a platform allowing multiple developers to work independently on the same project simultaneously, with the assurance that Git will help merge our contributions. Besides, Git plays a pivotal role in maintaining and managing our codebase effectively since the support it provides for branching is another compelling feature for our development workflow. It allows us to create different branches for specific phases such as the development phase, staging phase, and production phase. These branches keep our work separate until its code is reviewed and tested by another member of the group to be integrated into the main codebase.

5.9.2 Online Communication Channel

Our group highly prefers **WhatsApp** and **Microsoft Teams** as our online communication channels. Since we are all accustomed to using WhatsApp for our daily communication, its convenience factor becomes even more pronounced due to our familiarity with the platform. We are able to exchange real-time information among our group members, considering the ubiquitous presence of mobile phones and laptops in our lives today. Our main online communication channel, as recommended by our supervisor, is Microsoft Teams. Through this platform, we can efficiently schedule meetings and seek clarification on any questions or concerns we may have with our supervisor. Utilising both of these platforms allows us to enhance our team's communication, coordination, and collaboration efforts, whether for immediate communication needs or well-organised project workflows. On the other hand, Discord is rarely used within our group, and as the result, it is not a preferred option for us even if it could be an effective online communication channel.

5.9.3 Project Planning

Jira has been our chosen project planning tool over Trello, primarily due to its capability to handle greater complexity. For instance, Trello primarily offers Kanban boards, while Jira extends its capabilities to include not only Kanban but also features like Sprint and Scrum. Admittedly, Jira's broader range of functionalities can necessitate a steeper learning curve. However, the trade-off is the substantial advantage it brings in terms of reporting and analytics, which are notably absent in Trello and we believe will prove invaluable in the long term. This is particularly significant for our project which is using agile management as Jira supports Agile reporting comprehensively. All in all, Jira is our preferred choice, especially for larger and more intricate software development projects with specific workflow requirements. Its extensive feature set and reporting tools make it well-suited for these scenarios. Conversely, Trello remains an excellent option for smaller teams and less complex projects, thanks to its simplicity and visual appeal.

5.9.4 Time Tracking

We are going to use **Clockify** to track the duration of our completion of assignments and tasks. By utilising Clockify, we would gain a deeper understanding of how the group allocates the time. This knowledge, in turn, promotes constructive routines and enhances our overall efficiency, enabling us to effectively plan and accomplish tasks within specified timeframes.

5.9.5 Document Sharing Platform

For document-sharing platform, we have chosen two platforms which are **Microsoft Teams** and **Google Drive**. We use Microsoft Teams primarily to share documents directly with our supervisor, simplifying the process of collaboration and feedback with our supervisor. Google Drive acts as a central hub for document management within our team. It allows us to grant access to team members and organise documents into folders with a hierarchical structure. Google Drive is not only practical but also provides an organised and visually appealing way to present documents, making them easily accessible to the team.

5.9.6 Reference Management Tool

Overleaf is our group's choice for reference management tool due to its streamlined integration and ease of use. While Zotero can be used in conjunction with Overleaf to manage references, it involves additional steps. To use Zotero, we first need to create a Zotero account and generate a Zotero file in bib format to serve as our reference resource. Subsequently, we must add this file to our Overleaf project. In contrast, Overleaf simplifies the process significantly. It allows us to directly download the bib resource and seamlessly integrate it into our Overleaf documents. This direct integration not only saves time but also reduces the complexity of managing references within our collaborative writing environment.

5.9.7 Document Processing/Text Editing Software

In order to control the quality, we strategically chose **Overleaf** over Google Docs and Microsoft Word, especially since our documents are academic and research-oriented. Overleaf is based on LaTeX, thus being particularly good at typesetting complicated mathematical formulae, scientific notations, and technical papers compared to Words and Google Docs. Our work may include various complicated typesettings which is why Overleaf is preferred over the limited capabilities of Words and Google Docs. Overleaf also provides a wide range of quality LaTeX templates that support various academic publishing requirements. This would save us time and increase our productivity since we can omit the process of formatting our documents based on the guidelines.

References

[1] I. Abubakar, S.N. Khalid, M.W. Mustafa, Hussain Shareef, and M. Mustapha. Application of load monitoring in appliances' energy management – a review. *Renewable and Sustainable Energy Reviews*, 67:235–245, 2017.

[2] Arpit Jain, Abhinav Sharma, Vibhu Jately, and Brian Azzopardi. *Sustainable Energy Solutions with Artificial Intelligence, Blockchain Technology, and Internet of Things*. CRC Press, 2023.

[3] Yassine Himeur, Abdullah Alsalemi, Faycal Bensaali, Abbes Amira, and Ayman Al-Kababji. Recent trends of smart nonintrusive load monitoring in buildings: A review, open challenges, and future directions. *International Journal of Intelligent Systems*, 37(10):7124–7179, 2022.

[4] R. Gopinath, Mukesh Kumar, C. Prakash Chandra Joshua, and Kota Srinivas. Energy management using non-intrusive load monitoring techniques – state-of-the-art and future research directions. *Sustainable Cities and Society*, 62:102411, 2020.

[5] Roberto Bonfigli and Stefano Squartini. *HMM Based Approach*, pages 31–90. Springer International Publishing, Cham, 2020.

[6] Pascal A. Schirmer and Iosif Mporas. Non-intrusive load monitoring: A review. *IEEE Transactions on Smart Grid*, 14(1):769–784, 2023.

[7] Dongsheng Yang, Xiaoting Gao, Liang Kong, Yongheng Pang, and Bowen Zhou. An event-driven convolutional neural architecture for non-intrusive load monitoring of residential appliance. *IEEE Transactions on Consumer Electronics*, 66(2):173–182, 2020.

[8] Barbara Cannas, S. Carcangiu, Daniele Carta, A. Fanni, Carlo Muscas, Giuliana Sias, Beatrice Canetto, Luca Fresi, and Paolo Porcu. Nilm techniques applied to a real-time monitoring system of the electricity consumption. *ACTA IMEKO*, 10:139, Jun 2021.

[9] Christos L. Athanasiadis, Theofilos A. Papadopoulos, and Dimitrios I. Doukas. Real-time non-intrusive load monitoring: A light-weight and scalable approach. *Energy and Buildings*, 253:111523, 2021.

[10] Dong Ding, Junhuai Li, Kuo Zhang, Huaijun Wang, Kan Wang, and Ting Cao. Non-intrusive load monitoring method with inception structured cnn. *Applied Intelligence*, 52(6):6227–6244, Apr 2022.

[11] Kanghang He, Dusan Jakovetic, Bochao Zhao, Vladimir Stankovic, Lina Stankovic, and Samuel Cheng. A generic optimisation-based approach for improving non-intrusive load monitoring. *IEEE Transactions on Smart Grid*, 10(6):6472–6480, 2019.

[12] Chris Moffitt. Overview of Pandas data types, Mar 2018.

[13] Rinkesh. How Many Amps and Watts Does a Vacuum Use? (A Comprehensive Guide), Oct 2022.

[14] seaborn.lineplot - seaborn 0.12.2 documentation.

[15] James Cerwinski. Active power and apparent power, Jul 2009.

[16] Does changing home appliances help reduce your electricity bills?, Apr 2023.

[17] Bharath Parthasarathy and Chris Kleban. A2 VMs with NVIDIA A100 GPUs are GA, Mar 2021.

[18] GeeksforGeeks. Difference between Cloud Computing and Traditional Computing. *GeeksforGeeks*, May 2023.

[19] Mayur Badole. Train Machine Learning Models Using CPU Multi Cores, Apr 2021.

[20] The complete guide to NVIDIA A100: concepts, specs, features.

[21] GatherContent. What is Cloud Collaboration? Here's what you need to know. *GatherContent*, June 2023.

[22] Javier Canales Luna. Python vs R for Data Science: Which Should You Learn?, Dec 2022.

[23] John Terra. Keras vs Tensorflow vs Pytorch: Key Differences Among Deep Learning. *Simplilearn.com*, Aug 2023.

[24] Simplilearn. What Is Keras: The Best Introductory Guide To Keras. *Simplilearn.com*, Jul 2023.

[25] GeeksforGeeks. Deep Learning with PyTorch An Introduction. *GeeksforGeeks*, Jan 2023.

[26] AWS. Mongodb vs postgresql - difference between databases, 2023.

[27] Ruslan Osipov. Google Colab vs Jupyter Notebook: Which is Better? *Medium*, Jun 2023.

[28] Itesh Sharma. 9 Reasons why Angular front end development is the best. *Software and Technology Blog - TatvaSoft*, Jul 2023.

[29] Maciej Treder. Asynchronous JavaScript: introducing ReactiveX and RXJS observables. *Medium*, Dec 2021.

[30] Thomas Laforge. NgRx Effect vs Reducer vs Selector. *Medium*, Dec 2023.