Assessing Sustainability of Software Libraries: A Framework inspired by Nutri-Score

Sven Butzelaar [1], Julian Hirschler [1,2], Shantanu Jare [1],
Patrick Krumpl [1,3] & Thomas Verwaal [1]

[1]Delft University of Technology

[2]Vienna University of Technology

[3]Graz University of Technology

CS4415 Sustainable Software Engineering*

Abstract

As the information and communication technology sector continues to grow, so does the environmental impact of software. However, assessing the sustainability of software libraries remains a complex and often low-priority task for developers. To support more informed and sustainable decision making, we present a classification framework that evaluates software libraries across multiple sustainability dimensions. Inspired by the Nutri-Score nutritional rating system, our approach translates complex sustainability data into a simple and intuitive label. We apply this framework to Python visualisation libraries and conduct experiments to evaluate and classify them. The results show significant differences in the sustainability of the libraries evaluated. Our framework provides developers with a simple and accessible way to compare software libraries, providing a valuable starting point for more sustainable software development.

1 Introduction

Over the past decade, the information and communications technology (ICT) sector has experienced rapid growth, expanding at an average annual rate of 6.3% between 2013 and 2023, three times faster than the overall economy of the 27 countries that make up the Organisation for Economic Co-operation and Development (OECD) [1]. This growth is also reflected in the sector's environmental footprint, with ICT accounting for 7% of global electricity consumption in 2023, with a potential to cause 14% of global emissions by 2040 [2]. This development demonstrates the need for more sustainable and energy-efficient software.

However, despite ongoing research in the field of green software engineering, developers still face challenges in assessing the sustainability of particular software and software libraries. The measurement of software energy consumption is inherently complex, as many influencing factors can lead to different results in different measurements [3]. Moreover, there are no standardised evaluation methods and no agreed benchmarks to assess the sustainability of software. As a result, developers tend to focus on aspects such as functionality and performance rather than the sustainability of software.

The general problem of simplifying complex information into an easily understandable format for end users extends beyond software engineering. In the food industry, consumers are often overwhelmed by detailed nutritional information and lengthy ingredient lists, making it difficult to determine how healthy a product is quickly. To address this problem, the Nutri-Score nutritional rating system has been introduced in parts of the European Union. The front-of-pack labelling system makes complex information, such as nutritional value, easily understandable to the consumer, supporting better decision-making for healthy diets [4]. This approach demonstrates how compressing complex information into a simple classification system can influence decision-making. Inspired by how Nutri-Score supports easier decision-making, we propose a similar classification system to assess the sustainability of software libraries. We focus specifically on visualisation libraries for Python to demonstrate the usefulness of our framework. This allows developers to easily assess the sustainability of visualisation libraries without the need for complex and time-consuming evaluations.

Our proposed classification system focuses on three main areas to assess the sustainability of a software library: *Energy Efficiency*, *Maintainability* and *Sociotechnical Factors*. To demonstrate the effectiveness of our approach, we apply our framework to selected widely used visualisation libraries. The results are visualised with a label indicating the sustainability of a particular software library, inspired by the Nutri-Score system. This clear, standardised and accessible framework aims to guide software developers to make more informed and environmentally friendly decisions when integrating third-party libraries into their projects, thus reducing the overall environmental footprint of the ICT sector.

When we applied our framework to selected Python visualisation libraries, we found significant differences in their sustainability. We also found notable dis-

*Summer Term 2025

crepancies between the different metrics, with some libraries performing well in energy efficiency but poorly in maintainability and vice versa. Although there were some challenges during the experiment and the framework requires some future work, it is a good starting point for developers to assess sustainability more easily.

2 Background

2.1 The History of Nutri-Score

Diet and nutrition have a major impact on health. However, people struggle to make healthy food choices due to a lack of knowledge about the healthiness of many ingredients and their complex composition. In 2017, around 11 million people worldwide died from diseases related to unhealthy diets [5].

In 2014, front-of-pack labelling was first proposed to the French Minister of Health, followed by the voluntary implementation of front-of-pack Nutri-Score labelling in France in 2017 [6]. The labelling system has been the subject of numerous scientific studies, with the objective of developing a straightforward and comprehensible system for customers.

In the following years, other European countries such as Belgium, Switzerland, Germany, Spain, the Netherlands and Luxembourg introduced the Nutri-Score label [4]. The food scoring methodology is still being refined and has undergone some changes in 2024 [7]. Despite criticism from some opposition countries and companies [8], Nutri-Score remains one of the leading labels in Europe, with ongoing discussions about its potential EU-wide adoption [9].

2.2 How Nutri-Score works

For each 100 g or 100 ml of a product, a score is calculated for the nutrients to be promoted (fibres, proteins, fruits, vegetables and pulses). This score is then subtracted from the score of the nutrients to be limited (energy, saturated fatty acid, sugars, salt and non-nutritive sweeteners) [4]. The result is the so called FSAm-NPS score, which is a modification of the British Food Standards Agency nutrient profiling system (FSA-NPS) [5]. Based on this total score, the Nutri-Score labelling system classifies foods into five classes, from A for the healthiest to E for the least healthy, as shown in Figure 1.

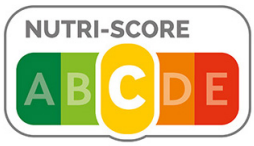

Figure 1: Nutri-Score label with a colour-coded gradient from dark green (A) for the healthiest to red (E) for the least healthy, indicating the nutritional quality of foods [7].

It is important to note that the comparison between different products is only valid within a specific product group. For example, due to the methodology described above, it is not meaningful to compare pizza and bread, even though the calculations are the same [10].

3 Methodology

3.1 Visualisation Libraries as Use Case

As with the Nutri-Score, the sustainability score we calculate with our framework is only applicable within a certain group of software libraries. This is due to the fact that different functionalities have a huge impact on the sustainability and especially on the energy efficiency of a software library. For example, machine learning libraries used to train deep neural networks consume more energy than visualisation libraries used to plot graphs. Therefore, we focus on only one group of software libraries to demonstrate our framework. Furthermore, the different libraries within a group should be comparable in terms of features and functionality.

In this article, we have chosen to present our framework applied to visualisation libraries for Python, since there are many libraries with similar functionalities. In addition, we believe that this choice has a large impact on the awareness of sustainability in software, since visualisation libraries are used by a large number of software developers, including non-professional ones. Unlike performance-critical domains such as machine learning, visualisation libraries may not always be optimised for efficiency to the same extent. This could lead to potential differences in sustainability.

For the experiment, we use the following visualisation libraries: Matplotlib, Seaborn, Plotly, Plotnine, Pygal, Holoviews.

3.2 Design of the Classification Framework

To implement a Nutri-Score-like framework for software libraries, we use a bottom-up approach. The Nutri-Score system calculates a score by subtracting the sum of beneficial attributes from the sum of unfavourable attributes, which is then classified into different labels [4]. Following this principle, we use sustainability attributes to assess the software library. These attributes are divided into three categories: *Energy Efficiency*, *Maintainability* and *Sociotechnical Factors*. Each category gives a score between 0 (less sustainable) and 100 (more sustainable). The weighted sum of these attributes gives the final score, which is classified into different sustainability labels.

3.2.1 Energy Efficiency

The energy consumption of a software library has a huge impact on its sustainability. Since the same functionality of a library can be implemented more or less efficiently, energy efficiency becomes a critical factor

in calculating the sustainability score. This attribute shows how effectively a library performs its core functions while minimising the use of resources.

To evaluate the energy efficiency and calculate a normalised score, a reference value is needed. Due to the fact that we can only compare software libraries within the same group, we follow a best-in-class approach. We start by defining specific test cases tailored to the particular group of software libraries. Next, we measure the energy consumption during the execution of all test cases.

$$\textit{Energy Efficiency Score} = 100 * e^{-k * \frac{E_{library} - E_{min}}{E_{max} - E_{min}}} \quad (1)$$

Equation 1 shows the calculation of the normalised energy efficiency score. E_{max} refers to the highest energy consumption among all compared libraries, while E_{min} refers to the lowest. $E_{library}$ is the energy consumption of the specific library being assessed. This ensures that the software library with the lowest energy consumption in the group receives a score of 100.

Our experiments showed that there are some outliers in the class when comparing energy consumption. If we used a linear or logarithmic function, the libraries close to the optimal library would all score very high, while the outlier would be pushed down to near zero. The use of exponential decay in Equation 1 ensures that even small differences within libraries close to the optimal library are visible in the final score. At the same time, libraries that are already far from the best-in-class are penalised less severely, avoiding excessive punishment for large outliers.

3.2.2 Maintainability

The maintainability of a library can have a significant impact on its sustainability. If a library is not easy to maintain, it will take more time and energy to develop new features or fix bugs. In addition, low maintainability can lead to high technical debt and, eventually to the abandonment of the project.

An empirical study was conducted to determine if there is a relationship between energy consumption and maintainability of software and concluded that maintainability has an impact on the energy consumption of the software tested [11]. The study used lines of code (LOC) as a measure of maintainability.

To determine the maintainability of software, we calculate the Maintainability Index (MI) of the particular software library. The MI is composited of the Halstead Volume (HV), Cyclomatic Complexity (CC) and LOC [12]. For the calculation, we use an adopted formula introduced in the Radon library [13].

The CC is defined as measuring "the amount of decision logic in a source code function" [14]. It's looking at how many independent paths can be made through the control-flow graph of the file or program. The formula is given by,

$$CC = E - N + 2 * P \quad (2)$$

E is the total number of edges in the control flow graph of a program or function. **N** is the total number of nodes (representing statements or decisions) in the control flow graph. **P** represents the number of connected components in the graph. A connected component is a sub graph where all nodes are reachable from any other node within that component [15]. The Halstead Volume, on the other hand, looks at two things: the program vocabulary and the program length. The program vocabulary (η), which calculates the number of unique operands in the codebase, is calculated using the following formula:

$$\eta = \eta_1 + \eta_2 \quad (3)$$

The other part of the Halstead volume is the program that calculates the total number of operands in the codebase or the Program Length(N) using the formula,

$$N = N_1 + N_2 \quad (4)$$

These two are combined to calculate the final value for the Halstead volume(V),

$$HV = N * \log_2(\eta) \quad (5)$$

[16]
Thus, the overall formula given by Microsoft for the maintainability index (MI) can be calculated by,

$$\text{MI} = 100 * \max\left(0, \frac{(171 - 5.2 * \ln(\text{HV}) - 0.23 * (\text{CC}) - 16.2 * \ln(\text{Loc}))}{171}\right) \quad (6)$$

3.2.3 Sociotechnical Factors

For the sociotechnical factors, we look at three factors: documentation quality, knowledge retention and issue resolution metrics. The evaluation of the documentation quality and of the knowledge retention are done manually, whereas a script is used for the issue resolution metrics.

A library with sufficient documentation leads to long-term usability, maintainability, and community adoption, and thus to a more sustainable product. We rate the documentation quality of a library based on the presence of the following documentation aspects: an API reference, an installation and setup guide, examples or tutorials, an FAQ page, and a contribution guide. Each aspect is worth 20 points. This means that a total score of 100 is reachable.

To quantify knowledge retention in a library, we consider the number of active contributors and community engagement. A higher number of contributors indicates a lower risk of knowledge loss, making the library more sustainable over time. The score is based on the total number of different contributors with more than 20 commits in the library's repository. We define the upper threshold (100 points) as more than 20 such contributors, representing strong knowledge retention, and the lower threshold (0 points) as only one contributor, indicating high risk. Libraries with several contributors between these thresholds are scored proportionally on a linear scale between 0 and 100 points.

For the issue resolution metrics, we measure the average issue closure time and average first response time. These metrics are crucial indicators of a library's sustainability. A short average issue closure time indicates that issues are being resolved quickly, encouraging continued use and contributions from the community. Similarly, a quick average first response time

demonstrates active maintainers who are responsive to user needs, which can help prevent frustration and abandonment of the library. Libraries that excel in these metrics are more likely to retain users and contributors, leading to a stronger and more sustainable ecosystem. Therefore, these issue resolution metrics provide valuable insights into the long-term health of the library and its ability to evolve and maintain a thriving user and developer community. To calculate the score for these metrics we use Equation 7, where T_{min} is the library with the lowest time.

$$Score = 100 * \frac{T_{min}}{T_{library}} \tag{7}$$

For calculating the Sociotechnical score, we use a weighted sum since each category has a value between 0 (less sustainable) and 100 (more sustainable). We assigned different weights to the categories based on our assessment of their relative importance to the sustainability of a software library. We experimented with different weights to capture meaningful differences in the software libraries we examined. Equation 8 shows the final weights.

$$\begin{aligned} \textit{Sociotechnical Score} = 0,4 * \textit{Documentation Quality} \\ \textit{Score} + 0,3 * \textit{Knowledge Retention Score} \\ + 0,15 * \textit{Issue Closure Score} \\ + 0,15 * \textit{Issue Response Time Score} \end{aligned} \tag{8}$$

3.2.4 Final score

Since each category has a value between 0 (less sustainable) and 100 (more sustainable), the final score must be calculated as the weighted sum of the scores for each category. We assigned different weights to the categories based on our assessment of their relative importance to the sustainability of a software library. Rather than an analytical approach, we focused on an empirical experiment with different weights to capture meaningful differences in the software libraries we examined.

$$\begin{aligned} \textit{Sustainability Score} = 0,5 * \textit{Energy Efficiency Score} + \\ + 0,3 * \textit{Maintainability Score} + 0,2 * \textit{Sociotechnical Score} \end{aligned} \tag{9}$$

Equation 9 shows the calculation of the final *Sustainability Score*. This score is then divided into different labels to make it more intuitive and quick to interpret. We have chosen to use three labels instead of five, as in the original Nutri-Score system. This decision was based on practical insights we gained during the course of the experiment, which showed that three categories were sufficient to meaningfully distinguish between more and less sustainable software libraries. Using five categories would have unnecessarily increased complexity without adding significant value to the system, especially given the relatively small number of libraries we evaluated.

The classification ranges were determined empirically based on experimentation with different thresholds, selecting the ranges that provided the clearest and most meaningful differentiation between the visualisation libraries. Table 1 shows the classification into the different labels.

Sustainability Score	Rating	Meaning
75 - 100	A	More Sustainable
50 - 74	B	Moderate Sustainability
0 - 49	C	Less Sustainable

Table 1: Classification of the *Sustainability Score* into different labels.

3.3 Experiment Setup

To evaluate the sustainability of the plotting libraries, we gather data from their repositories and conduct experiments. We focus on three categories: Energy Efficiency, Maintainability and Sociotechnical Factors. For each category, we define a measurement procedure for collecting relevant data. Energy consumption is measured using automated scripts, the maintainability score is based on source code metrics and sociotechnical factors are based on statistics of the repository.

3.3.1 Hardware Setup

The experiments were conducted on a **DELL XPS 15 9560** with **16GB RAM** running **Windows 10 Pro Version 22H2**.

To ensure consistent and reproducible results, we took the following steps before conducting the experiment:

- Let the device run for at least **one hour** as warm-up.
- Charge the battery of the device to **100%** and **plug it in**.
- Close all applications except of the console running the test script.
- Disable all unnecessary background services (cloud synchronization and automatic software updates).
- Turn off all notifications.
- Unplug unnecessary connected hardware like a mouse or headphones.
- Go into **Airplane mode**.
- Set the screen brightness to 50%.
- Disable all power-saving modes.
- Maintain a stable room temperature during the experiment (avoid open windows).

3.3.2 Software Setup

For the energy measurement, we used EnergiBridge and Python 3. The specific versions of the plotting libraries can be found in Table 2.

Library	Version
Matplotlib	3.10.1
Seaborn	0.13.2
Plotly	6.0.1
Plotnine	0.14.5
Pygal	3.0.5
Holoviews	1.20.2

Table 2: Plotting libraries and versions used in the experiment.

3.3.3 Energy Consumption Measurements

For our experiments, it was important to achieve statistical significance in the energy consumption measurements, so we designed the experiments to minimise the influence of external factors.

Each experiment was executed in isolation for one plotting library at a time and repeated across 30 runs to ensure a sufficient sample size. The order of the libraries was randomized in every iteration. Before measurements began, the script executed a five minute warm-up phase by calculating Fibonacci numbers.

The input data was randomly generated once and then reused across all libraries and iterations to ensure consistency. We created data frames for every plot type (line, bar, scatter, box & heat map). Details about these plotting types can be found in section 3.3.4. We scaled the complexity and size of the dataframes to achieve comparable rendering times across libraries to ensure that the measurement focuses on library efficiency rather than performance differences due to data volume.

Some libraries, such as Plotly, produce HTML files which get rendered in the browser and some libraries render the plots differently to this approach. To enforce a uniform approach, we convert all plots into binary PNG representations in memory. These PNGs were generated but not written to disk to avoid I/O-related energy consumption.

Each run recorded the total energy consumed using EnergiBridge. We used the Mann-Whitney U test to assess the statistical significance between libraries. All libraries were evaluated under consistent system conditions, with a 60 second break between runs.

3.3.4 Visualisation Tasks

To systematically assess the energy consumption of different plotting libraries, we evaluate their performance across a diverse set of Visualisation tasks. Each task corresponds to a commonly used plot type and is tested with an appropriately structured dataset:

- **Line Plot**: Simulates time-series data, representing a cumulative sum of daily temperature variations. The dataset consists of 10^4 points with dates on the x-axis and temperature values on the y-axis.
- **Scatter Plot**: Represents randomly distributed points categorized into three groups. The dataset contains 100 points with x and y coordinates uniformly sampled from a range of 0 to 10.
- **Bar Plot**: Displays categorical data with ten distinct categories and corresponding numerical values.
- **Box Plot**: Illustrates the distribution of values within four groups, using 10^7 data points drawn from a normal distribution.
- **Heatmap**: Computes a correlation matrix from 10^7 samples, measuring the relationships between three normally distributed variables.
- **Contour Plot**: Visualises a two-dimensional function based on a fine grid of 10^7 points, capturing continuous variations in z-values.

3.3.5 Maintainability Index Measurement

The experiment for the Maintainability Index of each source codebase of each plotting library was done as follows. First, the source code of each of the repos was cloned from Github. Then, a script was run that would calculate the maintainability index of each file in the repo of a plotting library using the following formula that was obtained from Microsoft mentioned before.

The average of all the calculated maintainability indices of all the files was taken, and this was the maintainability index of that repo.

3.3.6 Evaluating Sociotechnical Factors

For the issue resolution metrics, we used a script to determine the average time to close issues and the average first response time. We use the PyGitHub [17] library to interact with the GitHub API and collect the relevant statistics.

3.3.7 Replication

To replicate this experiment, check out our public GitHub repository. It contains all the necessary scripts as well as instructions on how to set up the experiments [18].

4 Results

For each library, the scores for the different categories, the final sustainability score and its corresponding rating can be found in Table 3. To calculate the final sustainability score, we applied Equation 9 and used the classification scheme described in Table 1. Matplotlib receives the highest energy efficiency score and thus the highest sustainability score and an A rating. Matplotlib had the highest Sociotechnical score as well

Library	Energy Eff. Score	Maintainability Score	Sociotechnical Score	Sustainability Score	Rating
Matplotlib	**100.00**	53.82	**76.64**	**81.47**	**A**
Seaborn	63.29	55.59	64.89	61.30	B
Holoviews	70.69	38.22	60.09	58.83	B
Plotly	13.53	**91.38**	72.41	48.66	C
Plotnine	30.23	54.94	60.65	43.73	C
Pygal	27.93	47.83	53.33	38.98	C

Table 3: Comparison of Plotting Libraries Across Different Metrics

amongst all the libraries. Seaborn and Holoviews perform average on all categories and receive a B rating. Plotly, Plotnine and Pygal all receive a C rating since their energy efficiency score is quite low. Plotly was the library with the highest maintainability score amongst all the libraries.

4.1 Energy Consumption

The results of the energy efficiency score can be found in Table 4.

Library	Mean Energy (J)	Energy Eff. Score
Matplotlib	58.24	100.00
Holoviews	182.01	70.69
Seaborn	221.49	63.29
Plotnine	485.12	30.23
Pygal	513.39	27.93
Plotly	771.89	13.53

Table 4: Mean energy consumption and energy efficiency score per library.

Matplotlib was the most energy-efficient library with a mean energy consumption of **58.24 J** and an energy efficiency score of 100.00. `Holoviews` and `Seaborn` have an energy usage of 182.01 J and 221.49 J, with energy efficiency scores of 70.69 and 63.29. `Plotnine` and `Pygal` have a higher mean energy usage of 485.12 J and 513.39 J and energy efficiency scores of 30.23 and 27.93. `Plotly` has the lowest energy efficiency score of 13.53 and the highest mean energy consumption of 771.89 J.

To assess the statistical validity of our energy measurements, we performed a Shapiro-Wilk test for normality on each library's data after removing the outliers. The results showed that `Seaborn`, `Plotnine`, `Pygal`, and `Holoviews` followed a normal distribution, while `Matplotlib` and `Plotly` did not. Since these libraries were not normally distributed, we used the non-parametric Mann-Whitney U test to evaluate the significance of differences.

A detailed visualisation of the Energy Consumption for each library is provided in the appendix in Figure 2.

4.2 Maintainability

The maintainability index for the libraries can be found in Table 5. These values correspond to the maintainability scores for the libraries. The computation of the maintainability scores has been explained in the previous section. The scores are between 0 and 100. The library with the highest maintainability index was **Plotly**, with a value of **91.38**. The library that had the lowest maintainability index was Holoviews, with a value of 38.22.

Library	Maintainability Index
Plotly	91.38
Seaborn	55.59
Plotnine	54.94
Matplotlib	53.82
Pygal	47.83
Holoviews	38.22

Table 5: Maintainability Index of Various Libraries

4.3 Sociotechnical Factors

The results and scores for the different Sociotechnical Factors can be found in Table 6, Table 7, Table 8 and Table 9. The results for the issue resolution metrics where collected on 25-03-2025.

All libraries perform good on documentation quality, although some are missing an FAQ or contribution guide.

The knowledge retention of the libraries is less good, only Matplotlib and Plotly have 20 or more contributors with more than 20 commits. The other libraries have less than 10 such contributors and thus have low knowledge retention.

The issue resolution metrics results vary quite a lot. Matplotlib, Plotly and Holoviews perform quite poorly on both metrics, whereas Seaborn and Plotnine perform well on both metrics. Pygal has a high issue closure score but a low issue response score. Overall **Matplotlib** has the highest Sociotechnical Score of **76.64**.

Library	API Ref.	Install Guide	Examples Tutorials	FAQ	Contrib. Guide	Score
Matplotlib	✓	✓	✓	✓	✓	100
Seaborn	✓	✓	✓	✓	×	80
Plotly	✓	✓	✓	✓	✓	100
Plotnine	✓	✓	✓	×	×	60
Pygal	✓	✓	✓	×	✓	80
Holoviews	✓	✓	✓	✓	✓	100

Table 6: Documentation Quality Evaluation

Library	Number of Contributors with 20+ commits	Score
Matplotlib	20+	100
Seaborn	6	30
Plotly	20+	100
Plotnine	6	30
Pygal	4	20
Holoviews	9	45

Table 7: Knowledge Retention Evaluation

Library	Average Issue closure time (hours)	Score
Matplotlib	7648.08	23.26
Seaborn	1779.17	100.00
Plotly	15005.96	11.86
Plotnine	2109.99	84.32
Pygal	1920.80	92.63
Holoviews	4635.09	38.38

Table 8: Issue Closure Evaluation

Library	Average First Response Time (hours)	Score
Matplotlib	1199.57	21.03
Seaborn	425.73	59.25
Plotly	6011.02	4.20
Plotnine	252.25	100.00
Pygal	2628.57	9.60
Holoviews	4536.73	5.56

Table 9: Issue Response Evaluation

5 Discussion

5.1 Summary of Findings

The classification of the different Python libraries according to their sustainability using a comprehensive Nutri-Score-like system leads to the following insights.

1. **Energy Efficiency vs Sociotechnical Factors:** The library has the highest sustainability rating, namely Matplotlib with a classification of A. The energy efficiency affects the score the Matplotlib the most. It is the best score in Sociotechnical as well. However, the second-best scoring library on Sociotechnical scores, namely, Plotly, is the one that has the lowest energy efficiency, showing that Socio Technical factors have almost no tangible relationship with energy efficiency.

2. **Energy Efficiency vs maintainability:** The most maintainable library, on the other hand, is Plotly, which has the lowest energy efficiency. This could be because libraries which are energy efficient have been heavily optimised and this might increase the logical complexity and the number of operators to check for edge conditions increasing consumptions. There is always a trade-off between performance and maintainability and performance. A classic case of this is Python vs C++. This could be an interesting point for further research.

3. **Maintainability vs Sociotechnical factors:** An interesting pattern observed here is that the library with the highest maintainability score, Plotly, also has a high Sociotechnical score of 72.24. This suggests that Plotly benefits from regular contributors and strong documentation, resulting in a well-maintained codebase. In contrast, libraries like Holoviews and Pygal, which have poor documentation and fewer contributors, demonstrate lower maintainability due to these factors.

5.2 Limitations

While our study provides a comparative sustainability analysis of various plotting libraries, it is important to acknowledge several limitations:

1. **Scope of Evaluation**: The experiment only assesses energy consumption in the context of simple plot types (e.g., line, bar, scatter). Many libraries are designed for specific use cases—such as interactive visualisations (Plotly, Bokeh) or statistical graphics (Seaborn, ggplot/Plotnine). As a result, our findings may not generalize to other visualisation tasks where certain libraries might excel.

2. **Data Handling Capabilities**: Not all plotting libraries are designed to handle large datasets efficiently. For example, Altair was excluded because of its data size limitations. This restricts the comparability of our results, as some libraries may be optimized for different scales of data processing.

3. **Feature Discrepancies and Workarounds**: Some plotting libraries do not support all the plot types we tested. In such cases, we implemented workarounds to generate comparable results, but these may introduce inconsistencies. For instance, certain libraries do not natively support heatmap plots, requiring alternative methods that may not be fully representative of their intended usage.

4. **Differences in Rendering Techniques**: The underlying rendering mechanisms vary significantly between libraries. Some use CPU-based rasterization (Matplotlib, Seaborn), while others leverage GPU acceleration (VisPy) or web-based rendering (Plotly, Bokeh). These fundamental differences make direct energy comparisons challenging, as they reflect different computational strategies rather than raw efficiency alone.

5. **GitHub Metrics Limitations**: Our sustainability assessment includes GitHub-based metrics (e.g., issue response time, maintenance activity). However, repositories may differ in how they handle issues—some may close outdated ones automatically, while others maintain them for documentation purposes. This variability makes direct comparisons between libraries less reliable.

6. **Empirical Weighting of Metrics**: Our final scoring formula was derived empirically and reflects our own assumptions about the relative importance of energy efficiency, performance, and maintenance. While we designed the weights to be as fair as possible, they may not generalize to other contexts, and different weightings could lead to significantly different rankings.

5.3 Future Work

Despite the limitations, our study provides an initial framework for evaluating the energy efficiency of plotting libraries. There are multiple avenues in which the study can be extended in further work:

1. **Broader range of visualisations tasks**: We assessed the energy consumption by reviewing six visualisation tasks. These could be extended with other tasks, such as plotting violin plots, radar charts, and density plots. However, this could exclude libraries that cannot create them.

2. **Vary workloads**: Libraries might have different computational efficiencies for different workloads, which is not something that our current system does not take into account. Therefore, measuring the energy consumption under different workloads could lead to significantly different results.

3. **Refine the weighting system**: The current weighting system has been empirically defined. It could possibly be improved with further empirical validation or an analytical approach.

4. **Energy consumption differences between different rendering techniques**: We converted all plots into binary PNG representations to enforce a uniform approach. However, it could be interesting to explore the differences in energy consumption between rendering plots dynamically in an HTML-based environment versus generating static PNG images.

5. **Evaluate based on common use cases**: Instead of requiring each library to generate all possible plots even when workarounds are necessary, future work could focus on assessing libraries based on their common use cases. Identifying key visualisation tasks that users typically perform would provide a more practical and insightful evaluation of the library's typical energy consumption.

6. **Apply sustainability score to other library types**: The concept of a sustainability score could also be applied to different libraries, such as graph or machine learning libraries.

7. **Rethink number of labels**: We currently have only three labels (A-C). However when assessing more libraries, it is likely distinguishing libraries will be harder since many of them will get the same label. Therefore the number of labels could be increased.

6 Conclusion

In summary, the main goal of this project was to make the assessment of the sustainability of software libraries easier and more accessible for developers. We therefore focused specifically on Python visualisation libraries as a use case. We looked at the Nutri-Score system, which has been successfully introduced in the food industry in parts of Europe to help consumers make healthier choices about their diet. We then developed a methodology to assess the sustainability of visualisation libraries and divided the assessment into three groups: Energy efficiency, maintainability and socio-technical factors. Based on these dimensions, we calculated a final sustainability score and ranked each library accordingly.

Our results showed that there are significant differences in the sustainability of popular Python visualisation libraries. The final result placed one library in class A (more sustainable), two libraries in class B (moderate sustainability) and three libraries in class C (less sustainable). The classification system provides a clear and accessible starting point for identifying which software libraries are more sustainable than others.

However, our framework is based on empirical methods and test cases tailored to specific library types, which may limit its generalisability. Designing fair comparisons was particularly challenging due to functional differences between libraries. Adapting the methodology to other types of software libraries may require further refinement.

Nevertheless, we believe that our framework provides a valuable foundation for future efforts to promote sustainability in software development.

References

[1] Organisation for Economic Co-operation and Development. Growth of digital economy outper-

forms overall growth across oecd, May 2024. Accessed: 2025-03-25.

[2] Pablo Valerio. Energy efficiency is crucial for the ict infrastructure. *EE Times*, 2023. Accessed: 2025-03-25.

[3] Luís Cruz, Carolin Brandt, and Enrique Barba Roque. Scientific guide for reliable energy experiments. Lecture slides, Sustainable Software Engineering (CS4575), TU Delft, 2025. Lecture 3, available through course materials.

[4] Santé Publique France. Nutri-score, 2025. [Online; accessed 17-March-2025].

[5] International Agency for Research on Cancer (IARC) - WHO. The nutri-score: A science-based front-of-pack nutrition label, 2021. [Online; accessed 17-March-2025].

[6] Chantal Julia, Fabrice Etilé, and Serge Hercberg. Front-of-pack nutri-score labelling in france: an evidence-based policy. *The Lancet Public Health*, 2018.

[7] Santé publique France. Nutri-score : le point sur les nouveautés 2024, 2024. [Online; accessed 17-March-2025].

[8] Chantal Julia and Serge Hercberg. Big food's opposition to the french nutri-score front-of-pack labeling warrants a global reaction. *American Journal of Public Health*, 108(3):318–320, 2018.

[9] European Commission. Front-of-pack nutrition labelling: Summary report. Technical report, European Commission, 2020. [Online; accessed 17-March-2025].

[10] German Federal Ministry of Food and Agriculture (BMEL). Nutri-score explained: Consumer information, 2025. [Online; accessed 17-March-2025].

[11] Javier Mancebo, Coral Calero, and Félix García. Does maintainability relate to the energy consumption of software? a case study. *Software Quality Journal*, 29(1):101–127, March 2021.

[12] Microsoft. Code metrics - maintainability index range and meaning. `https://shorturl.at/7AsmK`, 2022. Accessed: 2025-03-28.

[13] Radon Developers. Introduction to code metrics. `https://radon.readthedocs.io/en/latest/intro.html`, 2020. Accessed: 2025-03-28.

[14] Microsoft. Cyclomatic complexity. `https://shorturl.at/nZf1K`, 2022. Accessed: 2025-03-31.

[15] Arthur H. Watson and Thomas J. McCabe. Structured testing: A testing methodology using the cyclomatic complexity metric. Technical Report NIST Special Publication 500-235, National Institute of Standards and Technology (NIST), Gaithersburg, MD 20899-0001, September 1999. Prepared under NIST Contract 43NANB517266, Dolores R. Wallace, Editor.

[16] Rafa E. Al-Qutaish and Alain Abran. An analysis of the designs and the definitions of the halstead's metrics. In *Proceedings of the 15th International Workshop on Software Measurement (IWSM'2005)*, pages 337–352, Montreal, Canada, September 12-14 2005. École de Technologie Supérieure, University of Québec.

[17] Pygithubs. `https://github.com/PyGithub/PyGithub`, 2025.

[18] Sven Butzelaar, Julian Hirschler, Shantanu Jare, Patrick Krumpl, and Thomas Verwaal. Open source replication package for sustainability measurements. `https://github.com/svenbutzelaar/SSE_plot`, 2025.

A Appendix

A.1 Energy Measurements

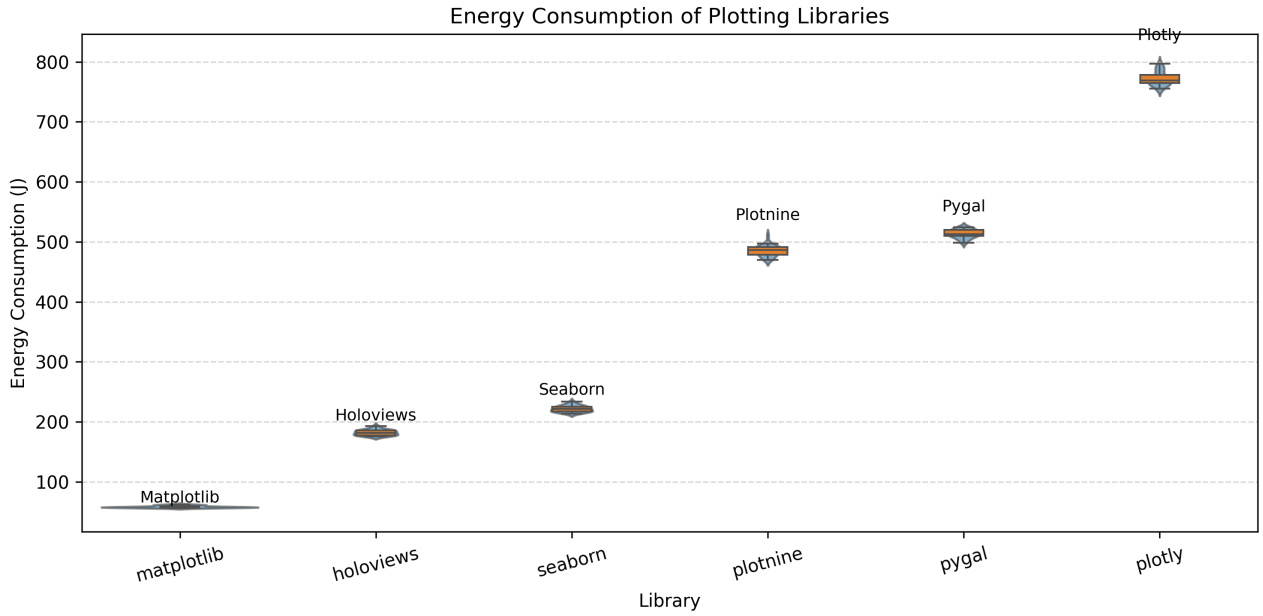

Figure 2: Energy Consumption comparison of Plotting Libraries.