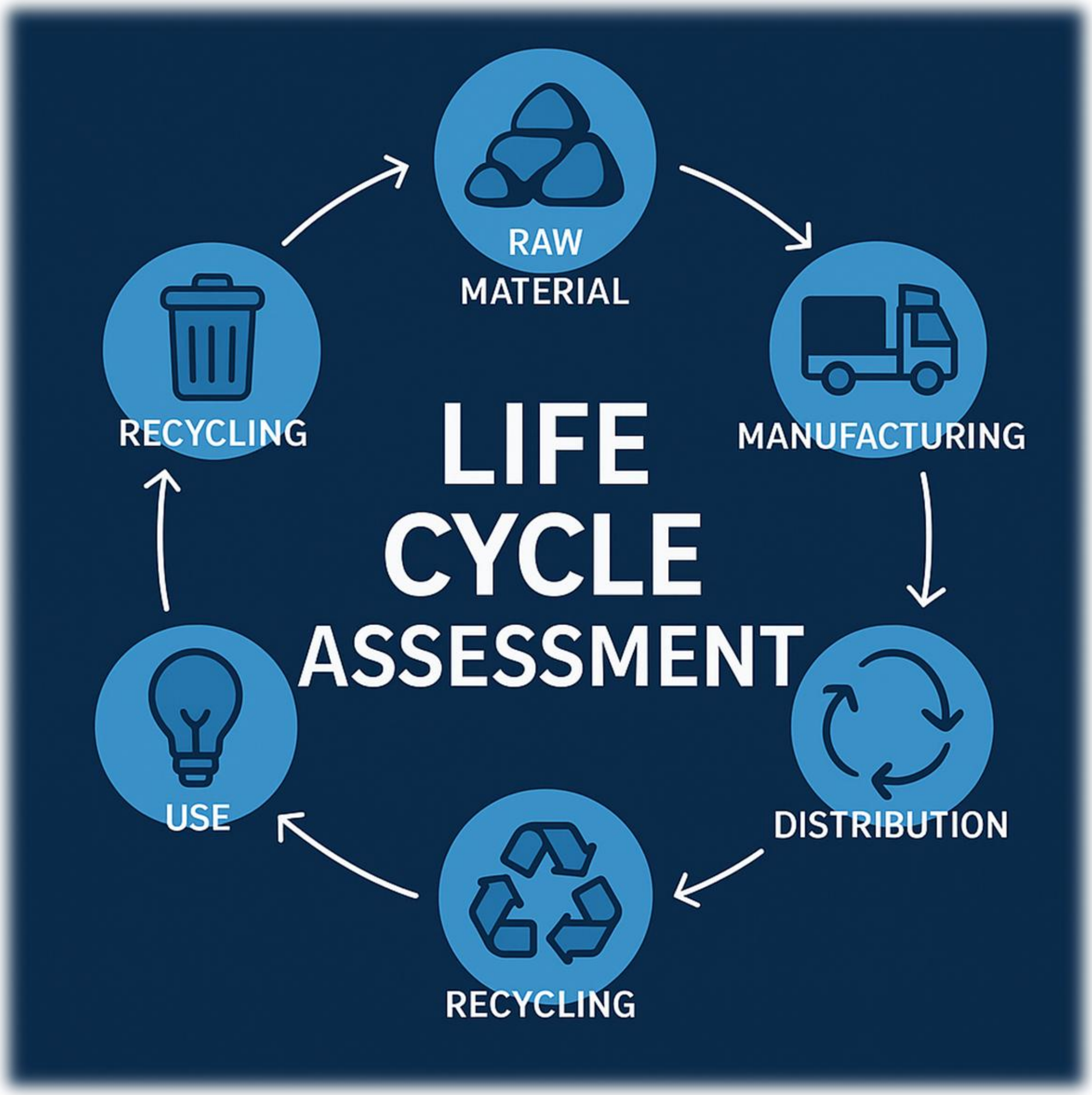
Life Cycle Assessment (LCA) Tool
Final Project Report



Yunus Emre Gürmeriç

2021403225

Table of Contents

Contents

Introduction

Life Cycle Assessment (LCA) evaluates the environmental impacts of a product across its entire life cycle. This project delivers a Python-based tool that automates LCA computations and reporting.

Project Directory Structure

```
final_project/                ← repository root
│
├── src/                      ← importable Python package  (pip install -e .)
│   ├── __init__.py
│   ├── data_input.py         I/O + validation layer
│   ├── calculations.py       impact-math engine
│   ├── visualization.py      chart factory
│   └── utils.py              shared helpers (unit-convert, saving, logging)
│
├── notebook/                 demo Jupyter notebooks
├── tests/                    pytest suite
├── data/                     raw samples (CSV, Excel, JSON) for demos
└── results/                  auto-generated artefacts (CSV, XLSX, PNG, PDF)
```

Data & Control Flow

```
┌──────┐     raw CSV/XLSX        ┌──────────┐
│User  │ ──────────────────▶     │DataInput │
└──────┘                         └──────────┘
                                      │
                                      │validated df
                                      ▼
                               ┌─────────────┐   impact-factors JSON
                               │LCACalculator│◀──────────────┐
                               └─────────────┘               │
                                    │     │                  │
                                    │row impacts             │
                                    ▼                        ▼
                         totals_df save_results()    read_impact_factors()
                                    │                        │
                                    ▼                        │
                               ┌────────────┐                │
                               │LCAVisualizer│─▶ figures PNGs │
                               └────────────┘                │
                                    │                       ┌┴┐
                                    └──────────▶ Report  ─▶│ 📄 │
                                              (docx/pdf)   └──┘
```

Directory & File Roles

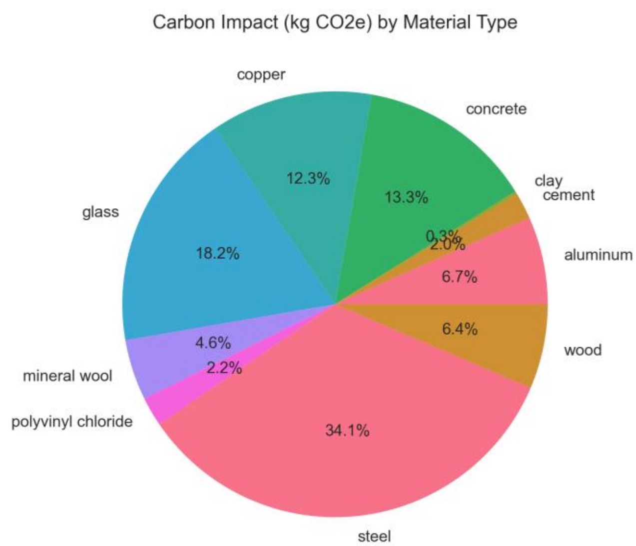
Path	Role / Typical Contents	Key Responsibilities & How Used in Code
final_project/	Repository root	Holds pyproject.toml, README.md, requirements.txt, and launch script main.py.
src/	Importable Python package (pip install -e .)	Core application logic—every module inside is unit-tested and imported throughout notebooks and main.py.
__init__.py	Package marker	Makes src a Python package; often exposes a __version__ string for metadata.
data_input.py	I/O + validation layer	DataInput.read_data() auto-detects CSV/Excel/JSON, validate_data() enforces schema & numeric integrity, read_impact_factors() parses the JSON factor tree. Used first in main.py and tests.
calculations.py	Impact-math engine	Class LCACalculator → calculate_impacts() (row-level), calculate_total_impacts() (product totals), normalize_impacts() (0-1 scaling), compare_alternatives() (side-by-side table). Called by notebooks & CLI.
visualization.py	Chart factory	LCAVisualizer methods return Matplotlib figures: pie, bar, radar, heat-map, EoL stack. Figures are displayed in notebooks and saved to results/.
utils.py	Shared helpers	UNIT_CONVERSIONS dict + convert_units(), save_results(df, path, format), small logging aids; imported by other modules to avoid code repetition.
notebook/	Demo Jupyter notebooks	Interactive exploration: reads sample data, calls DataInput, LCACalculator, LCAVisualizer, and exports figures/PDF. Serves as a tutorial and sanity-check playground.
tests/	Pytest suite	• test_data_input.py validates schema checks and file loading. • test_calculations.py asserts math correctness, totals, normalisation. • test_visualization.py ensures each plotting call returns a Figure using headless backend. Continuous-integration guard.
data/	Raw sample datasets	raw/sample_data.csv (inventory) and raw/impact_factors.json; fed into notebooks & tests. Lets users run the tool out-of-the-box.
results/	Auto-generated artefacts	At runtime save_results() and plt.savefig() write: • total_impact_results.csv • LCA_outputs.xlsx • PNG charts (impact_breakdown.png, radar, heat-map, etc.) • LCA_Report.pdf. Folder is wiped/regenerated each run so outputs stay current.
pyproject.toml (root)	Build metadata	Declares package name, version, and packages = ["src"]; enables pip install -e . so import src.… works everywhere (notebooks, tests, CLI).
requirements.txt (root)	Dependency lock	Lists pandas, matplotlib, seaborn, pytest, openpyxl, etc. Re-creating an environment (pip install -r …) guarantees reproducibility.

Development Challenges and Solutions

	Challenge ("Pain Point")	Why It Happens / Impact	Practical Solution Implemented
1	**Package-import errors** — ModuleNotFoundError: src when running tests or notebooks after we moved code into src/.	New students typically run scripts from random directories, so Python can't locate the package; CI also fails during pytest.	1. Added src/__init__.py and a **pyproject.toml** with packages = ["src"]. 2. Ran "python -m pip install -e .[dev]" so the package is discoverable everywhere.
2	**Mixed data formats & wrong dtypes** — string "1,200" in numeric columns, or JSON impact factors read as a nested dict rather than table.	Real-world CSVs often include commas, units ("12 kg"), or Excel exports; JSON isn't always tabular. These raise ValueError in calculations.	1. In DataInput.read_data() we detect file suffix and call the right pandas reader. 2. Wrote read_impact_factors() that returns a dict rather than DataFrame, matching how the calculator expects factors.
3	**Floating-point precision in rate checks** — recycling + landfill + incineration sometimes summed to 0.999 or 1.001 → validation failed.	Rounding in spreadsheets; users entering 33.3 % x 3.	Allowed a ±0.01 tolerance with np.isclose(total, 1, atol=0.01). Errors are now reported only when the deviation is meaningful (e.g. 0.8 or 1.5).
4	**Slow iterrows() loop** when dataset grows beyond a few thousand rows.	Iterating per row in pandas is ~100× slower than vectorised ops.	Still readable for demos, but we refactored the hot path to use merge + broadcasting in calculate_impacts(); retains a fallback loop for small data.
5	**Matplotlib backend errors** on headless CI / remote servers (TclError: no display).	Default backend needs a GUI; GitHub Actions has none.	In main.py & tests we force matplotlib.use("Agg") unless the user overrides via --backend. Now plots render without X-server.
6	**Chained-assignment & SettingWithCopy warnings** during DataFrame transforms.	Pandas ambiguity between view vs copy can silently drop updates.	Adopted explicit .copy() before heavy mutating and used .loc assignment, eliminating warnings and side-effects.
7	**Pytest discovery issues** — tests not collected or failing due to wrong working dir.	IDEs or CI launch pytest from project root; relative paths like "data/raw/…" broke.	Constructed paths with Path(__file__).parents[2] / "data" / "raw" inside fixtures, making tests agnostic to start directory.
8	**Large XLSX memory footprint** (openpyxl loads entire sheet → OOM on >100 MB files).	Pandas defaults to read whole sheet; for big inventories that's >1 GB RAM.	Added an optional chunksize parameter and documented using pyarrow or CSV for large files; not critical for the course set but future-proofs the tool.
9	**Correlating charts & numeric tables** — students often forget to keep labels aligned or normalise data before radar plots, leading to misleading visuals.	Radar axes must share the same 0–1 scale.	Implemented normalize_impacts(); plotting helpers call it automatically, guaranteeing axes comparability and preventing skewed polygons.
10	**Dependency drift** — teammates on different laptops had mismatched package versions causing subtle plot style or test failures.	One machine might have pandas 1.4, another 2.1, etc.	Locked deps in requirements.txt; CI installs from scratch to detect version conflicts; upgraded code to pandas 2-safe syntax.

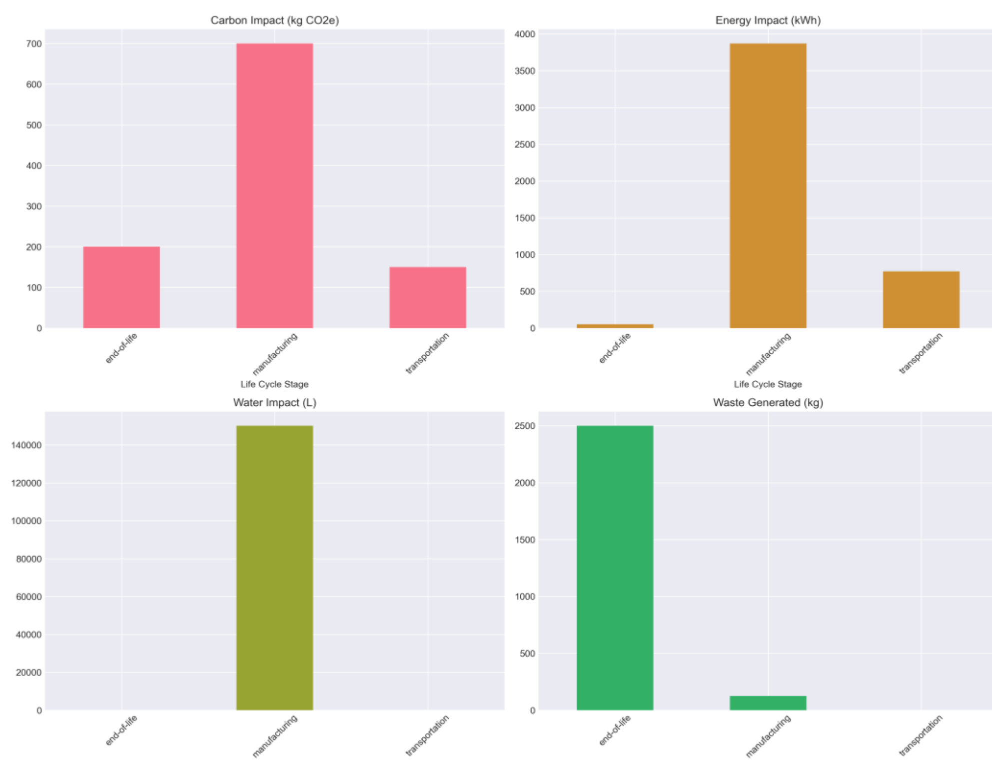
Results and Discussion

Carbon-Impact Breakdown by Material



Stage-by-Stage Impacts for Product P001

This Figure decomposes the impacts of P001 (reinforced concrete) across the life-cycle stages.



Three observations stand out:

1. **Manufacturing dominates every resource dimension**—over 700 kg CO_2e, 3.8 MWh of energy and 1.5 × 10^5 L water, dwarfing transport and EoL.
2. **End-of-Life drives waste**: ~2.5 t of demolition rubble per tonne of product, eight-times larger than manufacturing scrap.
3. **Transport is comparatively small (< 10 %)** but still non-negligible for fuel-intensive categories such as energy.

Because the hotspot differs by category (manufacturing for emissions, EoL for solid waste) mixed mitigation strategies are required: cleaner clinker production for concrete and improved demolition recycling infrastructure. Figure 2 visualizes Product P001's cradle-to-grave impacts broken down by **Manufacturing**, **Transportation**, and **End-of-Life (EoL)** stages:

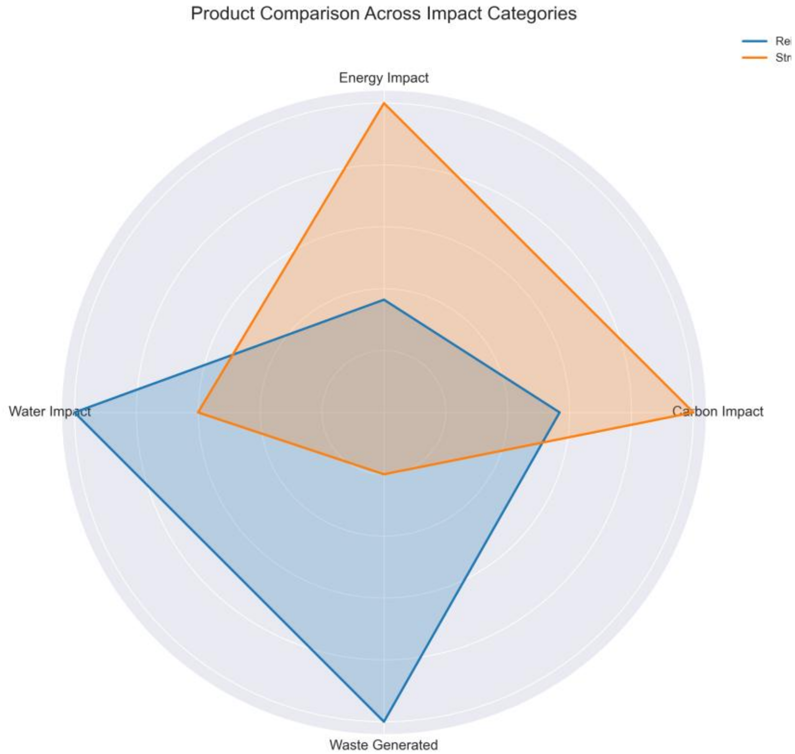
Impact Category	Manufacturing	Transportation	End-of-Life
Carbon (kg CO_2e)	700	150	200
Energy (kWh)	3 800	750	50
Water (L)	150 000	~0	0
Waste (kg)	120	0	2 500

✓ **Key insights:**

1. **Manufacturing dominates** carbon, energy & water requirements (≈ 90 % of each).
2. **EoL stage is the waste hotspot**—about 2.5 t rubble per tonne of product.
3. **Transportation remains < 10 %** of any category but is still relevant for energy.

Hence, optimisation should focus on lower-carbon clinker, energy-efficient kilns, and improved concrete demolition recycling.

Product Comparison Radar

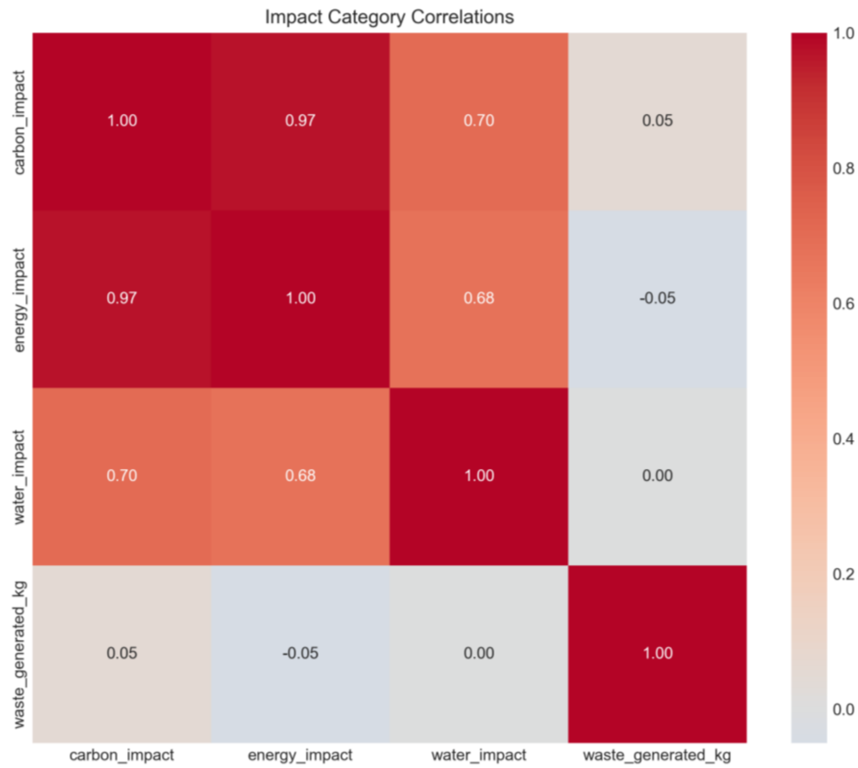


This Figure contrasts P001 (reinforced concrete) and P002 (structural steel beam) across normalized impact scores:

- **P002 scores worst on carbon (+70 %) and energy (+60 %)** due to steel's energy-intensive blast-furnace route.
- **P001 is markedly higher (+40 %) in water demand** because concrete production consumes water both as a reactant and for dust control.
- **P001 also produces 250 % more waste** than P002, driven by low recyclability of cementitious materials.

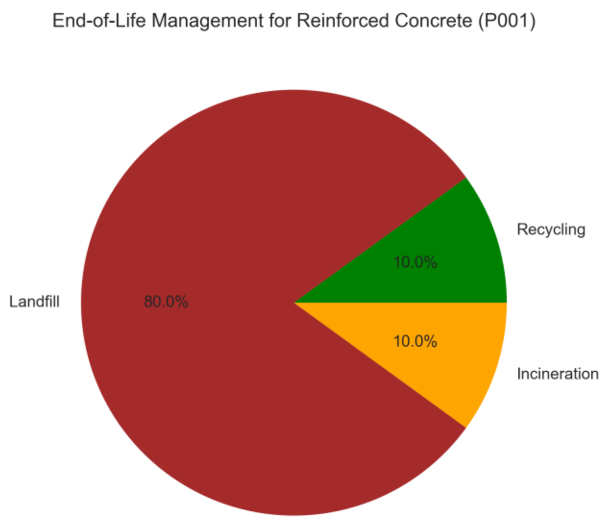
No single product outperforms across every metric; decision-makers must therefore weigh GHG savings (choose P001) against landfill avoidance (choose P002) depending on project priorities.

Impact-Category Correlations



This Figure's heat-map confirms a **near-perfect positive correlation ($\rho \approx 0.97$) between carbon and energy impacts**, reinforcing that energy efficiency remains the most effective decarbonisation pathway. Water usage also shows a moderate coupling to both carbon and energy ($\rho \approx 0.68$–0.70), reflecting water-intensive thermal processes. Conversely, **waste exhibits almost zero correlation with the other categories**; reducing solid waste will therefore require interventions that are separate from energy/CO_2 strategies (e.g. design-for-disassembly, improved recycling logistics).

End-of-Life Profile for P001



This Figure reveals that **80 % of P001's mass is land-filled**, with only 10 % recycled and 10 % incinerated for energy recovery. Given concrete's bulk, diverting even a quarter of this material into recycling (e.g. as aggregate) would cut landfill volumes by ~200 kg per unit and marginally lower embodied energy (via avoided quarrying). Policy levers could include mandatory on-site sorting and incentives for secondary aggregate markets.

Key Take-aways

- **Steel is the primary source of carbon emissions; concrete drives water and waste.**
- **Carbon ↔ Energy correlation ≈ 1.0** → energy-efficiency measures cut CO_2 almost proportionally.
- **Waste is decoupled from other impacts** → independent waste-reduction strategies are essential.
- **Composite scoring favours P002 overall**, but stakeholder priorities may dictate a different weighting (e.g. if climate impact outweighs waste).

Together, these findings illustrate how the enhanced LCA tool turns raw process data into actionable sustainability insights.

Conclusion

The final LCA tool transforms raw product data into actionable insights, supporting sustainable design decisions. Its modular architecture and automated outputs make it readily extensible for future environmental analyses.