



Semester Long Internship Report

On

A Comprehensive Framework for Scope 3 Data Systems, Event Reporting, and Brightway2 Modeling

Submitted by

Hakesh Kadapa

Bharatiya Engineering Science and Technology Innovation University

Under the guidance of

Prof. Kannan M. Moudgalya

Chemical Engineering Department, IIT Bombay

Mentors

Mr. Shubham Sonkusare

Mr. Nikhil Sharma

Under Supervision of

Mr. Sumanto Kar

(Asst. Project Manager)

July 8, 2025

Chapter 1

Acknowledgment

First and foremost, I would like to take this opportunity to express my sincere gratitude to the **FOSSEE team at IIT Bombay** for granting me the privilege of participating in the Semester-Long Internship Program. This internship has been a valuable experience that strengthened my technical knowledge and helped me explore the intersection of open-source technology and sustainability.

I am deeply thankful to my mentors — **Mr. Shubham Sonkusare, Mr. Sumanto Kar, and Mr. Nikhil Sharma** — for their unwavering support, timely feedback, and continuous encouragement throughout the internship. Their guidance was instrumental in shaping my project work and enhancing my learning.

I also extend my gratitude to my home institution, **Bharatiya Engineering Science and Technology Innovation University**, for the academic and logistical support that enabled me to participate in this internship.

Lastly, I am grateful to my peers, colleagues, and the wider FOSSEE community, whose collaborative spirit and dedication inspired me to contribute effectively to this impactful initiative.

Contents

1	Acknowledgment	2
2	Introduction	5
2.1	About FOSSEE	5
2.2	Importance of Open-Source Tools	5
2.3	Overview of Environmental Emissions	6
3	About The Internship	7
3.1	Internship Title and Domain	7
3.2	Selection Process	7
3.3	Internship Duration	8
3.4	Objectives of the Internship	8
4	Internship Project-1 (Realtime Emissions Dashborad for Events)	9
4.1	Research and Data Collection	9
4.1.1	Emission Factor's Research	9
4.1.2	Data Collection	10
4.2	Database and Product design	10
4.2.1	Product Design	11
4.3	Product Development	12
4.3.1	Main dashboard development	13
4.3.2	Scope3 Emission calculator and Dashboard development	15

4.3.3	Form Development(Travel-Form and Dashboard	16
4.4	Testing	19
4.5	Deployment	20
4.5.1	Deployment Using Streamlit	20
4.5.2	Other Deployment Considerations	21
5	Project-2 EcoInsight (LCA Predictions Using Brightway)	22
5.1	Project Work Flow	22
5.2	Understanding About LCA, Ecoinvent, Brightway2 . .	22
5.3	Loading Ecoinvent3.9 into Brightway2	23
5.4	Performing LCA operations	26
5.5	Displaying All Widgets	29
5.6	Performing Dynamic LCA Operations	29
6	Key Learnings	38
6.1	Technical Skills Gained	38
6.2	Domain Knowledge Acquired (Climate Science & Emis- sions)	38
6.3	Communication and Team Collaboration	39
7	Conclusion	40
8	References	41

Chapter 2

Introduction

2.1 About FOSSEE

The **Free/Libre and Open Source Software for Education (FOSSEE)** project is an initiative by the **Indian Institute of Technology (IIT) Bombay**, supported by the **National Mission on Education through Information and Communication Technology (NMEICT)**, Ministry of Education, Government of India.

FOSSEE promotes the use of **open-source software** in education and research to reduce dependency on proprietary tools and to provide high-quality alternatives accessible to all. It supports a variety of open-source platforms including **Python**, **Scilab**, **eSim**, **OpenFOAM**, **R**, and **DWSIM**, among others.

FOSSEE also runs **Semester-Long Internships (SLI)**, **SummerFellow ships**, **Winter Internships Textbook Companions**, and **Lab Migration** projects to provide hands-on learning opportunities for students and educators across the country.

For more information, visit: <https://fossee.in>

2.2 Importance of Open-Source Tools

Open-source tools play a vital role in democratizing access to technology. They are cost-effective, highly customizable, and supported by active communities. In academic and sustainability projects, open-source platforms enable trans-parency, reproducibility, and flexibility. During this internship, tools like **Python**, **Streamlit**, **SQLite**, and **Brightway2** were utilized extensively to build and visualize emission tracking systems and perform life cycle assessments.

2.3 Overview of Environmental Emissions

Environmental emissions refer to the release of pollutants into the air, water, and soil from natural or human activities. These emissions include greenhouse gases (GHGs) such as carbon dioxide (CO₂), methane (CH₄), and nitrous oxide (N₂O), as well as air pollutants like sulfur dioxide (SO₂) and particulate matter (PM).

Human-induced emissions primarily originate from industrial processes, transportation, agriculture, and energy production. These emissions contribute to global warming, acid rain, smog formation, and degradation of air and water quality. To evaluate their impact, emissions are typically categorized into three scopes:

Scope 1: Direct emissions from owned sources (e.g., company vehicles).

$$\text{Scope 1 Emissions} = \sum_{i=1}^n (A_i \times EF_i)$$

where:

- A_i = Activity data (e.g., liters of fuel, tons of waste)
- EF_i = Emission factor for activity i (e.g., kg CO₂e per liter of diesel)

Scope 2: Indirect emissions from purchased energy (e.g., electricity use).

$$\text{Scope 2 Emissions} = E \times EF$$

where:

- E = Electricity consumption (in kWh)
- EF = Emission factor of electricity (in kg CO₂e/kWh)

Scope 3: All other indirect emissions (e.g., supply chain activities).

$$\text{Scope 3 Emissions} = \sum_{j=1}^m (A_j \times EF_j)$$

where:

- A_j = Activity data for Scope 3 category j (e.g., distance traveled, money spent)
- EF_j = Emission factor for activity j

Chapter 3

About The Internship

3.1 Internship Title and Domain

Title: “**Scope 3 Emission Data Collection, Analysis, and Dashboard Development using Open Source Tools**”

This internship was offered under the **Sustainable Computing domain** of the FOSSEE project at **IIT Bombay**. It focused on the analysis and visualization of greenhouse gas emissions, particularly **Scope 3 emissions**, using **Python** and open-source libraries. The project also included the design and development of a real-time **web dashboard** using **Streamlit**, alongside work related to **Life Cycle Assessment (LCA)** using **Brightway2**.

3.2 Selection Process

Stage	Description
Application Submission	Interested candidates applied through the FOSSEE internship portal.
Screening Task	Applicants completed a task focused on Scope 3 emissions data collection, analysis using Python , and report submission .
Review and Evaluation	Submissions were evaluated based on code quality, data handling, documentation, and understanding of emission concepts.
Final Selection	Selected students were notified via email and onboarded to begin the internship.

3.3 Internship Duration

Start Date: February 2025.

End Date: July 2025.

Mode: Remote (Online).

Type: Semester-long Internship.

3.4 Objectives of the Internship

The internship aimed to build a strong understanding of Scope 1, 2, and 3 emissions, emphasizing the significance of structured data collection for Scope 3 categories. It involved analyzing emission data using Python and presenting insights through effective visualizations. A key objective was to develop a Streamlit-based web application for tracking event-related emissions. The work also included exploring Brightway2, an open-source LCA tool, and contributing to open-source solutions promoting environmental sustainability.

Tools and Technologies Used:

Python

Core programming language used for data analysis and application development.

Install: `sudo apt install python3` or download from <https://www.python.org>

SQLite3

Lightweight SQL database engine storing data as a single file on disk.

Install: `pip install sqlite3` or <https://sqlite.org/download.html>

Pandas and NumPy

Used for data manipulation and numerical computation.

Install: `pip install pandas numpy`

Streamlit and seaborn

Web framework for interactive dashboards.

Install: `pip install streamlit seaborn`

Brightway2 and matplotlib

Open-source LCA tool used for life cycle assessment modeling.

Install: `pip install brightway2 matplotlib,` `setup:`
`https://docs.brightway.dev`

Chapter 4

Internship Project-1 (Realtime Emissions Dashborad for Events)

Work Flow of Implementation

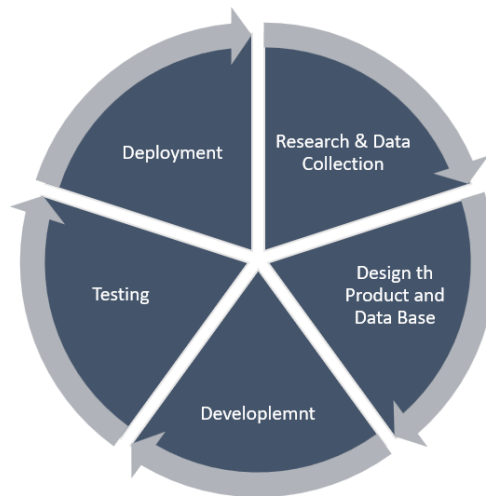


Figure 4.1: Project implementation workflow

4.1 Research and Data Collection

4.1.1 Emission Factor's Research

: The project began with a comprehensive study of Emission Factors (EFs), which represent the average emission rate of a given pollutant relative to a specific activity. Scope 3 emissions, being the most complex and indirect in nature, were prioritized. Research

focused on identifying standard EFs from trusted sources such as the IPCC, GHG Protocol, India articles, Indian Blogs and publicly available academic databases. Special emphasis was given to categories like business travel, accommodation, commuting, and procurement. Differences in methodologies and units required careful normalization to ensure consistency and comparability.

4.1.2 Data Collection

: structured data collection templates were designed to capture real-world activity data. Includes travel details, vendor purchases, event participation, and electricity usage. A key challenge was the variation in data availability and format across sources. CSV templates were created for users to enter data, which were later cleaned and validated using Python scripts. Data validation involved checks for missing fields, outliers, and inconsistencies to ensure high data integrity before further analysis.

4.2 Database and Product design

3.2.1 Database Design and Architecture Planing: A modular design approach was adopted to make the system scalable and maintainable. The architecture was divided into four layers: data input, processing, EF mapping, and visualization. The backend focused on robust processing logic while the frontend emphasized usability. Database design was created in form of snowflake scheme to handle to connections between tables for entered data. and output in detailed reports and charts.

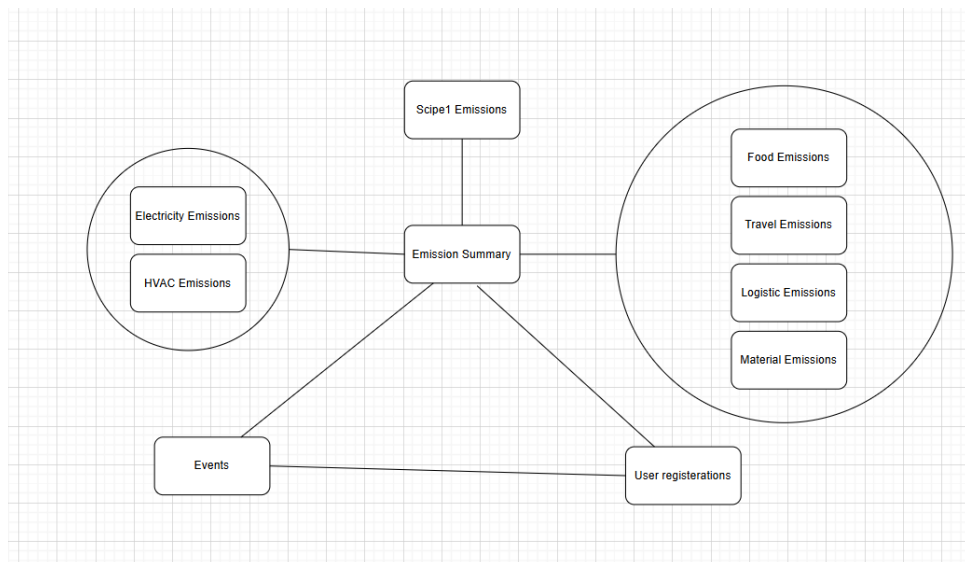


Figure 4.2: Database schema Design

Most of the required tables were created using standard `CREATE TABLE` syntax, with

customized entity names and attributes based on the context of emissions tracking.

Below is an example of the schema for the **EmissionsSummary** table:

```
CREATE TABLE EmissionsSummary (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    Event TEXT NOT NULL,  
    Category TEXT CHECK (Category IN ('Scope_1', 'Scope_2', '  
        ↳ Scope_3')) NOT NULL,  
    SourceTable TEXT NOT NULL,  
    Emission REAL NOT NULL,  
    FOREIGN KEY (Event) REFERENCES Events(name) ON UPDATE CASCADE  
);
```

To automate data flow and ensure consistency across tables, **triggers** were implemented. These triggers transfer data from category-specific emission tables into the centralized **EmissionsSummary** table upon insertion.

The following is an example trigger that inserts data from the **logistics_emissions** table into the **EmissionsSummary** table upon new data entry:

```
CREATE TRIGGER insert_scope3_logistics  
AFTER INSERT ON logistics_emissions  
FOR EACH ROW  
BEGIN  
    INSERT INTO EmissionsSummary (Event, Category, SourceTable,  
        ↳ Emission)  
    VALUES (NEW.event, 'Scope_3', 'logistics_emissions', NEW.  
        ↳ total_emission);  
END;
```

To enhance the performance of both insertion and retrieval operations, **indexes** were created on key columns. Indexing allowed for faster lookups, particularly in reporting modules that require grouped or filtered emissions data.

Example syntax for creating an index:

```
CREATE INDEX index_name ON table_name (column1, column2, ...);
```

4.2.1 Product Design

: The below diagram represents the structural flow and functional design of a web-based Emissions Dashboard application. It is divided into two main modules: the Main Dashboard and the Form Module.

The user journey begins at the 1st Interface, which introduces emissions and educates users through a chatbot. From there, the user proceeds to the Login Page, accessible via a Sidebar, enabling user authentication. Once logged in, users are redirected to the Home Page, which includes Top and Bottom Navigation panels. The top navigation facilitates event creation, while the bottom navigation provides access to the Scope 1, Scope 2, and Scope 3 Emission Calculators and Dashboards.

From the Home Page, users can navigate to Home Page 2, which focuses on dashboard visualizations and report generation. This ensures that users can view and export emission data based on their inputs.

The second major section, the Form Module, handles data entry. The Home Screen includes user registration and a sidebar. Users can navigate to various Form Screens such as Travel Data, Food Data, a Dashboard, and a Contact Us page. All forms are directly linked to the emission calculators in the main module, enabling real-time updates and seamless data integration.

Overall, the design emphasizes modularity, user guidance, and data-driven navigation.

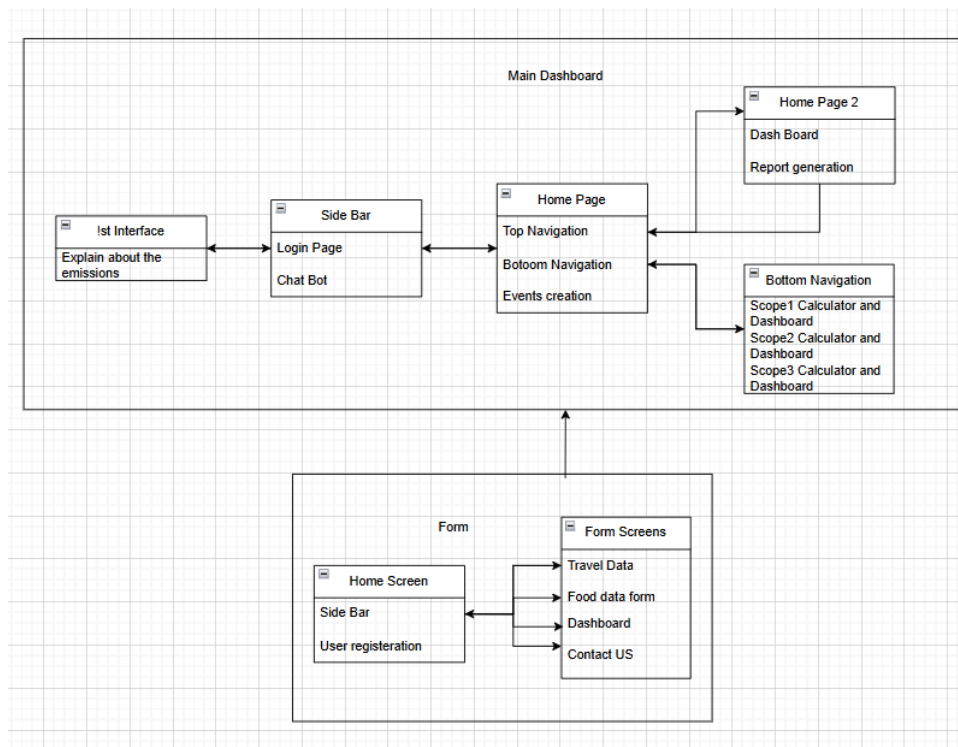


Figure 4.3: Product Design

4.3 Product Development

My Contribution: my main contributions in this phase was developing the **Main dashboard**, **scope3 emission calculator and visualization**, **report generator**, **database design and implementation**, **Travel data collection form**, **Form dashboard**.,

which formed the backbone of data collection.

4.3.1 Main dashboard development

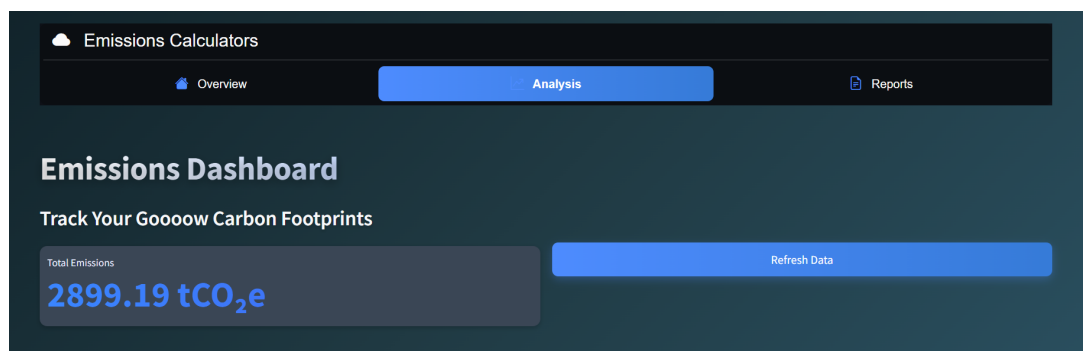
: The Main Dashboard serves as the central interface of the emissions tracking system, allowing users to visualize, monitor, and analyze emission data interactively. It aggregates data collected from various user-submitted forms (e.g., travel, food) and presents it through dynamic visual elements like bar charts, pie charts, and tables.

The Emissions Dashboard is built using Streamlit, SQLite3, Pandas, and Plotly. It connects to a local SQLite database (`emissions.db`) and retrieves event-specific data from the `EmissionsSummary` table:

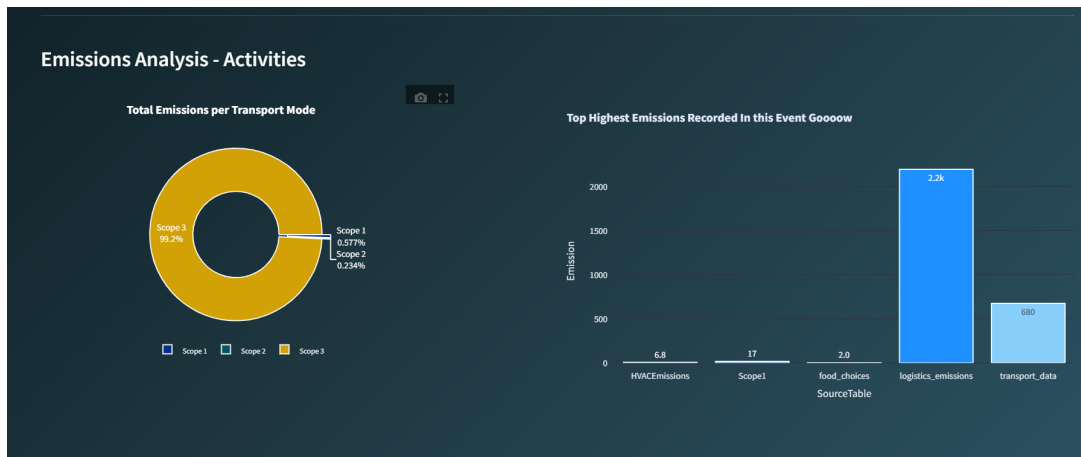
```
conn = sqlite3.connect(DB_PATH)
df = pd.read_sql_query("SELECT_*_FROM_EmissionsSummary_WHERE_Event=_?", conn,
    ↪ params=(event_name,))
```

The dashboard displays:

The below image shows the “Emissions Dashboard” interface of a Streamlit web application designed to track carbon footprints. It features a dark-themed navigation bar with three tabs: Overview, Analysis (active), and Reports. Below, the dashboard displays the total emissions in bold on a styled card to the left. A prominent “Refresh Data” button on the right allows users to reload data dynamically. The layout is clean and responsive, with visually distinct sections that enhance usability and help users monitor and manage their environmental impact effectively.



- **Metrics:** Total CO2 emissions in tCO2e in above image.
- **Donut Chart:** Distribution of emissions by Scope categories (Scope 1, 2, 3).
- **Bar Chart:** Top 5 source tables with the highest emissions.
- **What-If Simulator:** Sliders simulate CO2 reduction scenarios using:
 - HVAC, Electricity (Scope 2)
 - Direct emissions (Scope 1)

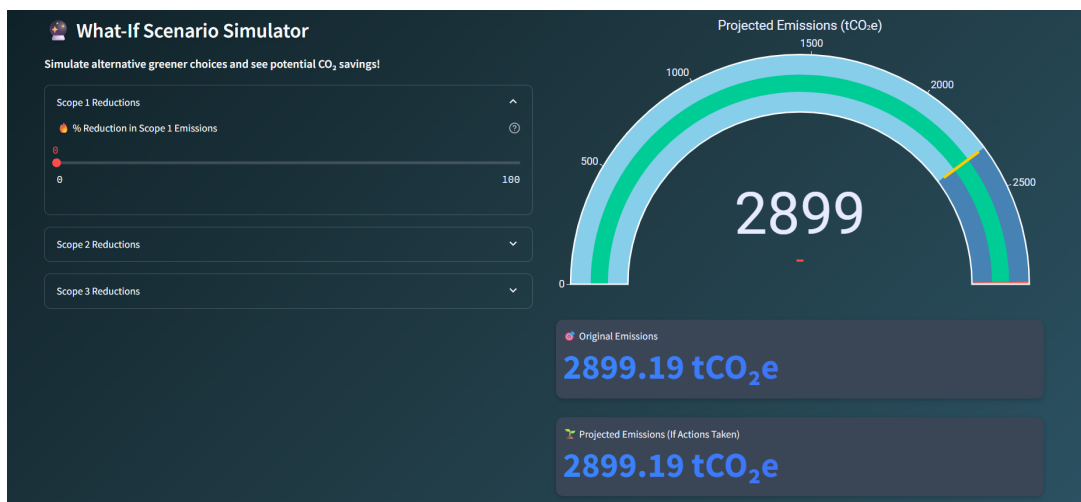


– Materials, Transport, Logistics, Food (Scope 3)

Emissions are recalculated:

```
savings = scope1 * (slider1 / 100.0) + ...
new_total = actual_total - savings
```

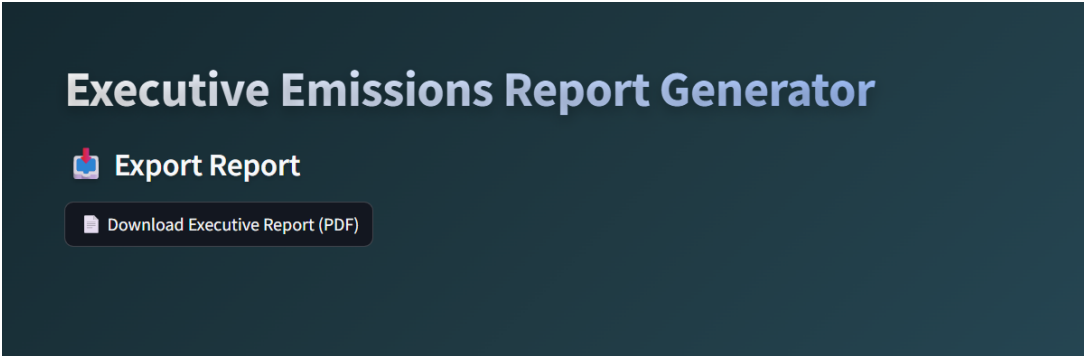
- **Gauge Chart:** Real-time projection of reduced emissions using `plotly.graph_objects`.



The UI is styled using custom embedded CSS, enhancing readability and aesthetics. A refresh button is used to update data dynamically, and the dashboard ends with an embedded PDF report trigger:

```
from visualizations.report import report
report()
```

This modular, interactive dashboard offers a real-time, customizable interface to monitor event-based carbon emissions effectively.



The Main Part of Report:

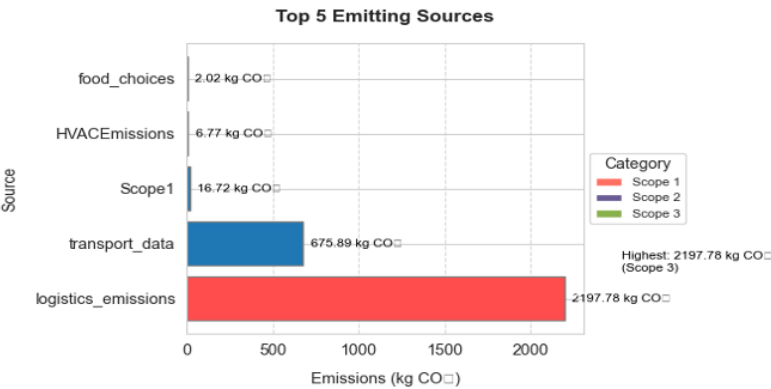
Emissions Summary

Category	Emission (kg CO ₂ e)
Scope 1	16.72
Scope 2	6.77
Scope 3	2875.70

Impact Equivalent

- Trees needed to offset: 138.1
- Homes powered for 1 month: 7.5
- Kilometers driven by a car: 14495.9 km

Top 5 Emitting Sources



Carbon Reduction and Sequestration Strategies

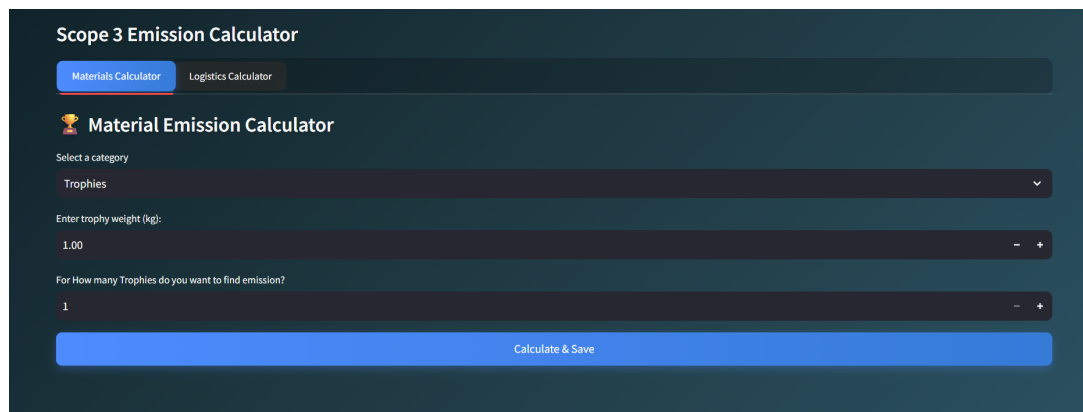
Total report can be generated from the dashboard.Above image shows the main part of the report.

4.3.2 Scope3 Emission calculator and Dashboard development

: Below Figure illustrates User Interface of Scope 3 Material Emission Calculator. This module allows users to input material-specific data such as category (e.g., trophies), weight in kilograms, and quantity. Upon clicking **Calculate & Save**, the application

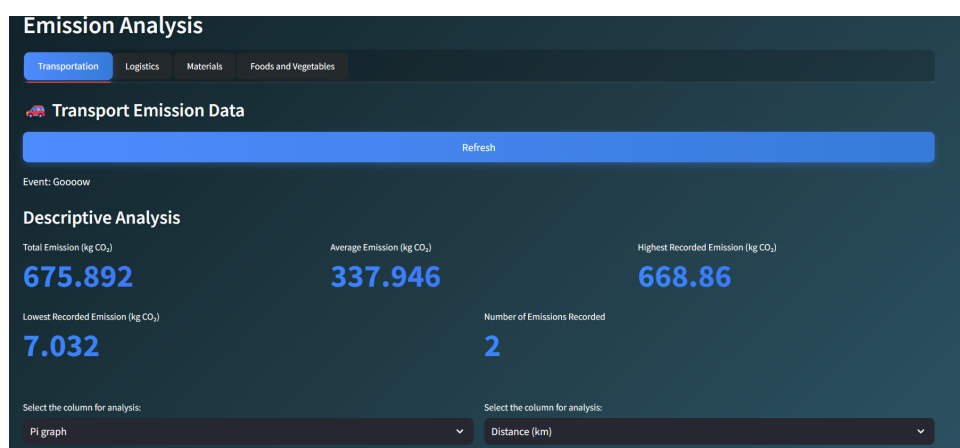
computes the carbon emission using pre-defined emission factors and stores the result in the database. The tabs allow switching between material-based and logistics-based Scope 3 sources and other scope3 data can be collected from **Form**. All emissions are linked to the latest event using:

```
def get_latest_event():
    conn = sqlite3.connect(DB_PATH)
    ...
```



The stored data is then visualized under the “Visualization” analysis tab using pie charts and tables, enabling real-time insights.

The Emission Analysis User Interface (UI) is designed to provide insights into the carbon footprint generated through transportation activities during an event. It allows users to track, analyze, and visualize emissions data interactively. The UI displays real-time statistics such as total, average, and peak emissions. It also supports data refreshing and dynamic graph generation based on user-selected metrics (e.g., distance). This enables organizations to make informed decisions for emission reduction and promotes sustainability by identifying emission hotspots and optimizing travel plans accordingly.



4.3.3 Form Development(Travel-Form and Dashboard

This is a web-based emission calculator that estimates carbon emissions for multi-modal travel using Python and Streamlit. The calculator integrates Google Maps API to compute

real-time distances and matches them to emission factors for various transport modes.

It supports: Road, Rail, Air and Manual Distance entry, Real-time geolocation and route queries via Google Maps APIs, Pie-chart based visual analysis of emissions and distances, Persistent storage using SQLite **Sample Code: Getting Latitude and Longitude**

```
1 def get_lat_lon(city):
2     url = f"https://maps.googleapis.com/maps/api/geocode/json?address={city}&key={
3         ↪ GOOGLE_MAPS_API_KEY}"
4     response = requests.get(url).json()
5     if response["status"] == "OK":
6         location = response["results"][0]["geometry"]["location"]
7         return location["lat"], location["lng"]
8     return None, None
```

Listing 4.1: Function to get coordinates using Google Maps API

Distance Calculation The application uses geodesic distance for air routes and the Google Distance Matrix API for road and transit modes:

```
1 def get_distance(origin, destination, mode):
2     url = f"https://maps.googleapis.com/maps/api/distancematrix/json"
3     params = {
4         "origins": f"{origin[0]},{origin[1]}",
5         "destinations": f"{destination[0]},{destination[1]}",
6         "mode": mode,
7         "key": GOOGLE_MAPS_API_KEY
8     }
9     response = requests.get(url, params=params).json()
10    if "rows" in response:
11        return response["rows"][0]["elements"][0]["distance"]["text"]
12    return "N/A"
```

Listing 4.2: Calculate Distance Between Two Coordinates

Visualization Using Plotly Once calculations are complete, emissions are visualized using Plotly pie charts.

```
1 fig = px.pie(
2     emission_df,
3     names="Mode",
4     values="Emission (kgCOe)",
5     title="Emission Distribution by Mode",
6     hole=0.7
7 )
8 st.plotly_chart(fig, use_container_width=True)
```

Listing 4.3: Pie Chart for Mode-wise Emissions

Air Distance Calculation Nearest Airports: For both origin and destination cities, the app uses Google Maps API to find the nearest airports within a 50 km radius. Air Distance Options:

- – It first tries to compute the air distance using city names and a real-world aviation database (airports.csv from OpenFlights).

- If that fails, it falls back to calculating the geodesic (straight-line) distance between the two airport coordinates using the geopy library.
- Additional Road Distances: The app adds driving distances from:
 - Origin → Origin Airport
 - Destination Airport → Destination
- Emission Calculation: Final emission is calculated using the core air segment distance with a factor of 1.58 kg CO per km.

```

1 air_distance_result = get_air_distance_by_city(origin_city, dest_city)
2 if air_distance_result is None:
3     air_distance = geodesic(origin_airport_coords, dest_airport_coords).km
4 else:
5     air_distance = air_distance_result
6
7 # Add driving to/from airport
8 to_airport = extract_distance(get_distance(origin_coords, origin_airport_coords, "
    ↳ driving"))
9 from_airport = extract_distance(get_distance(dest_airport_coords, dest_coords, "
    ↳ driving"))
10
11 distance_value = round(to_airport + air_distance + from_airport, 2)
12 emission_dist = round(air_distance * 1.58, 2)

```

Listing 4.4: Air Distance Calculation Logic

User Interface:

The ‘visualize data’ function is a crucial component of the application that presents carbon emissions data for a particular user during a specific event. It fetches transport and food-related data from an SQLite database and uses modern data visualization techniques to provide meaningful insights.

Key Features Displays **total emissions** from transport and food choices, Provides interactive **bar and pie charts** for transport data (distance or emission-wise), Generates **word clouds** that visually emphasize key emission-related terms, Supports selection of specific columns (e.g., Distance vs Emission) and graph type, Gracefully handles missing data and errors

```

1 c.execute("SELECT mode, distance, Emission FROM transport_data WHERE Event = ? AND
   ↳ name = ?", (Event, name,))
2 transport_data = c.fetchall()

```

Listing 4.5: Transport Data Fetch from SQLite

```

1 fig_transport = px.bar(
2     transport_df,
3     x="Mode",
4     y=columns,
5     color="Mode",
6     title=f"Transport {columns} of {name} at {Event}",
7     text=columns
8 )

```

Listing 4.6: Plotly Bar Graph

```

1 wordcloud = WordCloud(
2     mask=cloud_mask,
3     contour_color='steelblue',
4     colormap="viridis"
5 ).generate_from_frequencies(words)

```

Listing 4.7: WordCloud Visualization

Output and Usage The user selects the type of graph (Pie or Bar) and metric (Distance or Emission) to analyze their transportation footprint. The tool also visualizes food choices using pie charts categorized by frequency and emission value. A customized word cloud is generated to highlight the overall carbon footprint with a cloud shape.

4.4 Testing

:

Unit Testing:

Each function such as `get_distance()`, `get_lat_lon()`, and `insert_transport_into_db()` was tested in isolation using test data. The correctness of outputs was manually verified.

Integration Testing:

The full data flow—from user input in Streamlit, through API interactions and database storage, to visualization—was tested with realistic end-to-end scenarios. Focus was placed on ensuring consistency between calculation and chart data.

Boundary Testing: Edge cases such as:

- Blank or malformed origin/destination
- Extremely small/large distances
- Unsupported countries (for road/rail modes)

were tested to check system stability and appropriate warning handling.

API Response Testing: Google Maps and OpenFlights-based functions were tested with both valid and invalid queries to handle cases like:

- No nearby stations
- Rate limiting
- Unavailable distances

Database Verification: The data inserted into `emissions.db` was validated using SQLite viewers to ensure proper logging of.

UI Testing: Various components such as dropdowns, graphs, and word clouds were tested interactively through Streamlit to verify responsiveness and correctness.

Testing Outcomes All major functionalities behaved as expected under standard and edge-case conditions.

No data loss or misrepresentation occurred in visual outputs.

Word clouds and plots scaled correctly for different emission sizes.

User warnings and messages helped avoid miscalculations.

Tools Used for Testing

- Streamlit rerun and debug console
- SQLite database browser
- Postman for testing external API endpoints
- pytest (optional for unit tests)

Conclusion The system passed all functional and visual validations. It is reliable for educational and semi-formal use cases involving awareness of carbon emissions.

4.5 Deployment

4.5.1 Deployment Using Streamlit

The project was deployed using **Streamlit**, a Python framework that transforms data scripts into shareable web applications. Deployment involved converting the local development version into an accessible online tool for users to input travel and food data and view their carbon emissions interactively.

The deployment steps followed were:

1. **Project Structure:** All Python scripts, supporting image files, and datasets were organized into a single directory with proper relative paths. The SQLite database (`emissions.db`) was placed in a `data/` folder.
2. **Requirements File:** A `requirements.txt` file was created using:

```
pip freeze > requirements.txt
```

to ensure all Python dependencies like `streamlit`, `geopy`, `plotly`, and `pandas` are installed during deployment.

3. **Streamlit Sharing:** The project was uploaded to a GitHub repository and linked to **Streamlit Cloud** (<https://streamlit.io/cloud>). The app was configured with the entry-point script and deployed with a click. Secret keys such as the Google Maps API key were managed securely using Streamlit's `secrets.toml` configuration.
4. **Testing Post Deployment:** The deployed app was tested for responsiveness, API reliability, and database operations. Visualizations rendered accurately and data inserted into the hosted database successfully.

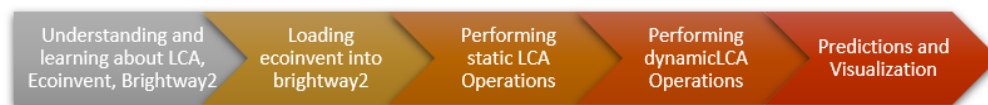
4.5.2 Other Deployment Considerations

- **Docker:** For consistent local and cloud environments, Docker containers can encapsulate the app and dependencies.
- **Heroku/Render/Vercel:** Other platforms like Heroku or Render can be used for more flexible server control or background tasks.
- **Custom Domain:** A domain can be mapped for a professional interface.
- **Security:** Sensitive keys should never be hardcoded. Use environment variables or encrypted secret managers.
- **Database Hosting:** SQLite can be replaced with PostgreSQL or Firebase for multi-user scalability.

Chapter 5

Project-2 EcoInsight (LCA Predictions Using Brightway)

5.1 Project Work Flow



AboveFigure illustrates the workflow of comparison between energies in terms of their Global Warming Potential (GWP) over a span of years. The analysis was performed using the Brightway2 framework, leveraging the ecoinvent 3.9 life cycle inventory database. This dynamic evaluation model was created by integrating scenario data (such as SSP2-19) and modifying the background database.

5.2 Understanding About LCA, Ecoinvent, Brightway2

During the project, significant effort was invested in understanding and working with Brightway2, a Python-based life cycle assessment (LCA) framework. Unlike GUI-based tools, Brightway2 provides full flexibility and control over datasets, calculation methods, and scenario modeling.

We began by setting up the Brightway2 environment and importing the `ecoinvent 3.9` database. This database served as the foundation, providing high-quality, regionally differentiated inventory data for products and processes across industries.

Key learning outcomes include:

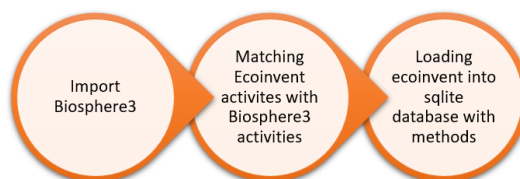
- Understanding how to initiate and manage Brightway2 projects using `bw2data`.
- Querying and modifying datasets and processes using Python code, enabling efficient preprocessing and data exploration.
- Using `LCA()` and `Method()` functions to compute environmental impact scores for specific functional units.
- Integrating scenario data (e.g., SSP2-19) to modify background datasets dynamically.
- Handling uncertainty and performing Monte Carlo simulations for probabilistic LCA.
- Visualizing output impacts over time, especially for sectors like energy, transportation, and materials.

Compared to traditional LCA tools, Brightway2 offered greater learning potential by requiring direct interaction with data structures and computations. We used it to model systems like EV vs ICEV, varying grid mix, and country-specific emissions, linking data with real-world policy scenarios.

Overall, this technical exposure enhanced our skills in environmental data science, reproducible research, and dynamic life cycle modeling—crucial for modern sustainability assessments.

5.3 Loading Ecoinvent3.9 into Brightway2

Loading ecoinvent follow the workflow as described in the below image.



Brightway2 is a modular framework for life cycle assessment in Python. During our project, we extensively used three essential modules—`bw2io`, `bw2data`, and `bw2calc`—each serving a distinct role in the LCA workflow.

Below is a sample Python code snippet demonstrating the import of these modules:

```
1 import bw2io as bi
2 import bw2data as bd
3 import bw2calc as bc
```

Setting Up the Biosphere Database in Brightway2

To begin using Brightway2 with Ecoinvent 3.9, we first set the current project and ensure the biosphere database is present. The following Python code checks if a biosphere database exists, and if not, it initializes the core data using `bw2setup()` from the `bw2io` package.

```
1 bd.projects.set_current('ecoinvent39')
2 if any("biosphere" in db for db in bd.databases):
3     print('Biosphere is already present in the project.')
4 else:
5     bi.bw2setup()
```

Upon successful execution, Brightway2 creates the `biosphere3` database and initializes 762 LCIA methods with over 227,000 characterization factors. This step is critical as it sets up the foundational flows and methods necessary for life cycle impact assessment (LCIA) calculations in Brightway2.

OUTPUT:

```
1 Creating default biosphere
2
3 Writing activities to SQLite3 database:
4 Applying strategy: normalize_units
5 Applying strategy: drop_unspecified_subcategories
6 Applying strategy: ensure_categories_are_tuples
7 Applied 3 strategies in 0.01 seconds
8 0% [#####] 100% | ETA: 00:00:00
9 Total time elapsed: 00:00:00
10 Title: Writing activities to SQLite3 database:
11   Started: 06/17/2025 22:20:40
12   Finished: 06/17/2025 22:20:41
13   Total time elapsed: 00:00:00
14   CPU %: 94.30
15   Memory %: 43.56
16 Created database: biosphere3
17 Creating default LCIA methods
18
19 Wrote 762 LCIA methods with 227223 characterization factors
20 Creating core data migrations
```

Note: To use the Ecoinvent 3.9 database in Brightway2, one must first obtain a valid license from the official [Ecoinvent website](<https://ecoinvent.org>). In this project Ecospol2 format is utilized.

Importing Ecoinvent 3.9.1 Cutoff Database

The next step is to import the Ecoinvent 3.9.1 cutoff version. The following code snippet ensures the database is only imported if it hasn't already been added to the project.

```
1 if 'ecoinvent-3.9.1-cutoff' in bd.databases:
2     print('Ecoinvent 3.9.1 is already present in the project.')
```



```

3 else:
4     ei = bi.SingleOutputEcospold2Importer(
5         dirpath=r'Z:\brightway2 env\ecoinvent 3.9_cutoff_ecospold2\datasets',
6         db_name='ev391cutoff'
7     )
8     ei.apply_strategies()
9     ei.statistics()
10    ei.write_database()

```

Before importing the Ecoinvent 3.9.1 cutoff database into Brightway2, it's important to check whether it has already been added to avoid duplication. The code below performs this check. If the database is not found, the `SingleOutputEcospold2Importer` is initialized with the directory path containing the dataset.

This code initializes the importer, applies standard data-cleaning strategies, checks dataset statistics, and writes the processed data into the Brightway2 project environment.

Output for importing the Ecoinvent-3.0 as Follows:

```

1 Extracting ecospold2 files:
2 0% [#####] 100% | ETA: 00:00:00 | Item ID: fff9fe74-
   ↪ f099-5
3 Total time elapsed: 00:06:52
4 ...
5 Applied 22 strategies in 7.86 seconds
6 Writing activities to SQLite3 database:
7 21255 datasets
8 676292 exchanges
9 0 unlinked exchanges
10 ...
11 Created database: ev391cutoff

```

We can check the available databases with the following command

```

1 databases

```

Output:

```

1 Databases dictionary with 2 object(s):
2 biosphere3
3 ev391cutoff

```

To use the ecoinvent database we need to set it up as a database

```

1 bd.databases['ev391cutoff']

```

Output:

```

1 {'overwrite': False,
2  'format': 'Ecoinvent XML',
3  'depends': ['biosphere3'],

```

```

4 'backend': 'sqlite',
5 'number': 21255,
6 'modified': '2025-06-17T22:28:34.758547',
7 'searchable': True,
8 'processed': '2025-06-17T22:30:52.623529'}

```

This confirms that:

- The dataset format is Ecoinvent XML.
- It contains 21,255 activities.
- It depends on the biosphere3 database.
- The import was successful and the dataset is searchable.

5.4 Performing LCA operations

Here LCA stands for **Life cycle assessment**. We use these operations on the **activities of ecoinvent** using the loaded methods to find the **Global warming potential(GWP)**. Let's explore the ecoinvent first.

The below code snippet allow us to see the first 5 activities in ecoinvent 3.9.

```

1 list(bd.Database('ev391cutoff'))[:5]

```

Output:

```

1 ['heat and power co-generation, wood chips, 6667 kW' (megajoule, IN-AP,
   ↪ None),
2 'market for waste graphical paper' (kilogram, ZA, None),
3 'market for protein pea' (kilogram, GLO, None),
4 'electricity production, natural gas, combined cycle power plant' (kilowatt
   ↪ hour, BG, None),
5 'natural gas, high pressure, import from NO' (cubic meter, GB, None)]

```

Extracting and Cleaning Ecoinvent Activities

The following Python code snippet demonstrates how to extract relevant information from the `ev391cutoff` database in Brightway2. It retrieves each activity's name, location, and reference product, and stores the data in a structured pandas DataFrame. Duplicate entries based on these three columns are removed to ensure uniqueness.

```

1 activities = [{
2     'Activity': act['name'],
3     'Location': act.get('location', 'unknown'),
4     'Product': act.get('reference product', 'unknown'),
5 }

```

```

6 } for act in db]
7 df = pd.DataFrame(activities)
8 df_unique = df.drop_duplicates(subset=['Activity', 'Location', 'Product']).
    ↪ reset_index(drop=True)

```

This process helps in:

- Extracting structured metadata from the Brightway2 database.
- Cleaning the dataset by removing redundant entries.
- Preparing data for further analysis, visualization, or filtering.

Interactive LCA Tool Using IPyWidgets

This section describes the development of an interactive Life Cycle Assessment (LCA) interface using Brightway2, pandas, and ipywidgets. The goal is to allow users to select a life cycle activity and multiple LCIA methods, and then compute environmental impacts in real-time.

Step 1: Generating Activity Dropdown

We first build a list of user-readable activity options by combining the activity name, location, and reference product.

```

1 activity_options = [f"{i}: {row['Activity']} ({row['Location']}) - {row['Product']}"
    ↪ for i, row in df_unique.iterrows()]
2
3 activity_dropdown = widgets.Dropdown(
4     options=activity_options,
5     description='Activity:',
6     layout=widgets.Layout(width='90%')
7 )

```

Step 2: Preparing Method Selector

A list of all available LCIA methods is fetched from Brightway2. These methods are displayed in a SelectMultiple widget to allow multi-selection.

```

1 methods_list = list(bw.methods)
2 method_labels = [f"{i}: {m}" for i, m in enumerate(methods_list)]
3 method_selector = widgets.SelectMultiple(
4     options=method_labels,
5     description='Methods:',
6     rows=10,
7     layout=widgets.Layout(width='90%', height='300px')
8 )

```

Step 3: Adding a Button and Output Area

We include a button to trigger the LCA calculation and an output area to print the results.

```
1 run_button = widgets.Button(description="Run LCA")
2 output = widgets.Output()
```

Step 4: Callback Function for LCA Computation

Upon clicking the button, the following function:

- Retrieves the selected activity.
- Parses the selected methods.
- Runs LCI and LCIA for each method.
- Prints the LCA results (impact scores).

```
1 def on_button_clicked(b):
2     with output:
3         clear_output()
4
5         activity_index = int(activity_dropdown.value.split(":")[0])
6         selected_activity_info = df_unique.iloc[activity_index]
7         selected_activity = [act for act in db
8                               if act['name'] == selected_activity_info['Activity']
9                               and act.get('location', 'unknown') ==
10                                  ↳ selected_activity_info['Location']
11                               and act.get('reference product', 'unknown') ==
12                                  ↳ selected_activity_info['Product']] [0]
13
14         selected_method_indexes = [int(m.split(":")[0]) for m in
15                                   ↳ method_selector.value[:20]]
16         selected_methods = [methods_list[i] for i in selected_method_indexes]
17
18         for method in selected_methods:
19             try:
20                 lca = LCA({selected_activity: 1}, method)
21                 lca.lci()
22                 lca.lcia()
23                 print(f"{method[0]} ({method[1]} {method[2]}) = {lca.score:.6f}"
24                       ↳ )
25             except Exception as e:
26                 print(f"{method} Error: {e}")
```

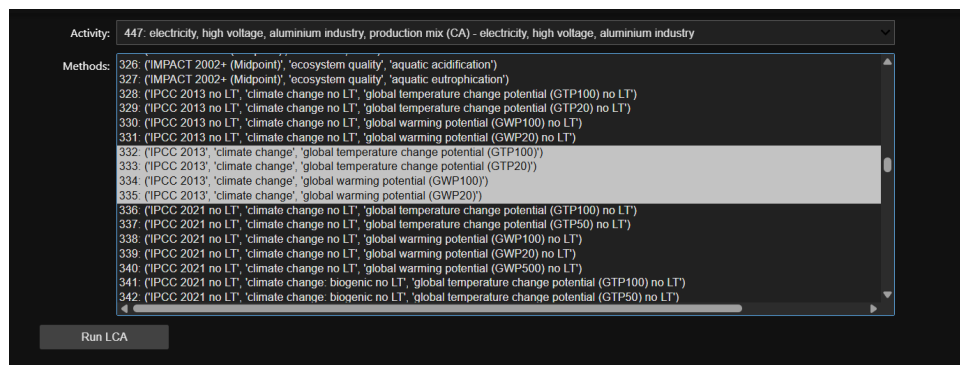
5.5 Displaying All Widgets

Finally, the UI is rendered with the dropdown, multi-selector, and output.

```
1 run_button.on_click(on_button_clicked)
2 display(activity_dropdown, method_selector, run_button, output)
```

This tool streamlines the LCA process, enabling users to interactively explore multiple environmental impact categories for selected supply chain activities.

Output:



Then the Final output after clicking the on the button **Run LCA**

```
IPCC 2013 (climate change global temperature change potential (GTP100)) = 0.036775
IPCC 2013 (climate change global temperature change potential (GTP20)) = 0.043801
IPCC 2013 (climate change global warming potential (GWP100)) = 0.039480
IPCC 2013 (climate change global warming potential (GWP20)) = 0.045642
```

These scores represent the environmental burden (in kg CO₂-eq) of producing 1 unit of the selected activity. The values help compare short-term vs long-term climate impacts and show how the choice of assessment method can influence the reported result. The closer the score is to 1, the greater the climate impact of the activity.

5.6 Performing Dynamic LCA Operations

Activity Selection for Electricity Sources in Brightway2

To perform a comparative Life Cycle Assessment (LCA) of electricity generation from different sources, we begin by retrieving relevant datasets from the Ecoinvent 3.9.1 database. The following Python code uses the Brightway2 framework to search for electricity production activities using specific keywords.

```

1 import pandas as pd
2 db = Database("ev391cutoff")
3
4 def safe_search(db, keyword):
5     results = db.search(keyword)
6     if not results:
7         print(f"No match found for: {keyword}")
8         return None
9     return results[0]
10
11 sources = {
12     'Coal': safe_search(db, "electricity production, hard coal"),
13     'Gas': safe_search(db, "electricity production, natural gas, at power plant
14         ↪ "),
15     'Solar': safe_search(db, "photovoltaic, 3kWp slanted-roof installation"),
16     'Wind': safe_search(db, "electricity production, wind, onshore"),
17     'Hydro': safe_search(db, "hydro, run-of-river"),
18     'Nuclear': safe_search(db, "nuclear, pressure water reactor"),
19 }

```

This function `safe_search()` searches for datasets containing the given keyword and returns the first match found. The `sources` dictionary maps common electricity generation types (e.g., Coal, Gas, Solar, etc.) to their corresponding Brightway2 activity. These activities can later be used to compute environmental impacts using various LCIA methods.

to see the sources just run the

```

1 sources

```

Output as follows

```

1 {'Coal': 'hard coal mine operation and hard coal preparation' (kilogram,
2     ↪ RoW, None),
3 'Gas': 'electricity production, natural gas, conventional power plant' (
4     ↪ kilowatt hour, CO, None),
5 'Solar': 'photovoltaic slanted-roof installation, 3kWp, a-Si, panel, mounted,
6     ↪ on roof' (unit, RoW, None),
7 'Wind': 'market for wind turbine, 4.5MW, onshore' (unit, GLO, None),
8 'Hydro': 'market for hydropower plant, run-of-river' (unit, GLO, None),
9 'Nuclear': 'market for nuclear power plant, pressure water reactor, 650MW' (
10     ↪ unit, GLO, None)}

```

SSP IAM Dataset Integration: The Shared Socioeconomic Pathways (SSP) database provides scenario-based projections for demographic, economic, and energy variables. The CSV file used in our project, named `SSP_IAM_V2_201811.csv`, contains global model output data under various SSP scenarios, such as SSP1–SSP5.

```

1 spp = pd.read_csv(r"C:\Users\hakes\Downloads\SSP_IAM_V2_201811.csv\
2     ↪ SSP_IAM_V2_201811.csv")

```

This file can be downloaded from the official IIASA SSP Database at:

<https://tntcat.iiasa.ac.at/SspDb/dsd?Action=htmlpage&page=welcome>

It is primarily used to analyze long-term emission trends and energy demand scenarios and can be integrated into Brightway2 models to align LCA results with future climate pathways.

Filtering SSP2-Baseline Electricity Data from IMAGE Model

To extract relevant data from the SSP IAM dataset, we filtered the dataset to include only projections from the IMAGE integrated assessment model, under the SSP2-Baseline scenario, and for the World region. This allows for a focused analysis on global electricity generation trends under a moderate development pathway.

```
1 # Filter just IMAGE model, SSP2-Baseline, and World region
2 filtered = ssp[
3     (ssp["MODEL"] == "IMAGE") &
4     (ssp["SCENARIO"] == "SSP2-Baseline") &
5     (ssp["REGION"] == "World")
6 ]
7
8 # Show available electricity variables
9 print(filtered[filtered["VARIABLE"].str.contains("Secondary Energy\\|
    ↳ Electricity", regex=True)]["VARIABLE"].unique())
```

This step helps identify the available electricity-related variables, such as different sources (coal, wind, solar, etc.), within the filtered dataset. These variables can then be aligned with Brightway2 electricity production activities for scenario-based LCA simulations.

Electricity Mix Extraction from SSP2-Baseline Scenario

To understand future electricity production shares, we extracted and transformed specific variables from the filtered SSP2-Baseline scenario data. The goal was to isolate projections for electricity generation from major sources and convert them into a normalized share (

```
1 print("Available electricity variables:")
2 print(filtered[filtered["VARIABLE"].str.contains("Electricity")]["VARIABLE"].
    ↳ unique())
3
4 # Extract rows for selected electricity types and years
5 selected = filtered.set_index("VARIABLE").loc[
6     [
7         "Secondary Energy|Electricity|Coal",
8         "Secondary Energy|Electricity|Gas",
9         "Secondary Energy|Electricity|Solar",
10        "Secondary Energy|Electricity|Wind",
11        "Secondary Energy|Electricity|Hydro",
12        "Secondary Energy|Electricity|Nuclear"
13    ],
```

```

14     ["2020", "2030", "2040", "2050", "2060", "2070", "2080"]
15 ]
16
17 # Transpose and rename columns
18 mix = selected.T
19 mix.columns = ["Coal", "Gas", "Solar", "Wind", "Hydro", "Nuclear"]
20
21 # Normalize each row to sum to 100%
22 mix_percent = mix.div(mix.sum(axis=1), axis=0) * 100
23 mix_percent.insert(0, "Year", mix_percent.index.astype(int))

```

This code prepares a normalized electricity mix for each decade from 2020 to 2080, useful for dynamic LCA studies using models like Brightway2. Each row of `mix_percent` shows the percentage contribution of different electricity sources, enabling scenario-based environmental impact forecasting. Output as follows:

```

1     Available electricity variables:
2     ['Capacity|Electricity' 'Capacity|Electricity|Biomass'
3     'Capacity|Electricity|Coal' 'Capacity|Electricity|Gas'
4     'Capacity|Electricity|Hydro' 'Capacity|Electricity|Nuclear'
5     'Capacity|Electricity|Oil' 'Capacity|Electricity|Other'
6     'Capacity|Electricity|Solar' 'Capacity|Electricity|Solar|CSP'
7     'Capacity|Electricity|Solar|PV' 'Capacity|Electricity|Wind'
8     'Capacity|Electricity|Wind|Offshore' 'Capacity|Electricity|Wind|Onshore'
9     'Final Energy|Electricity' 'Secondary Energy|Electricity'
10    'Secondary Energy|Electricity|Biomass'
11    'Secondary Energy|Electricity|Biomass|w/ CCS'
12    'Secondary Energy|Electricity|Biomass|w/o CCS'
13    'Secondary Energy|Electricity|Coal'
14    'Secondary Energy|Electricity|Coal|w/ CCS'
15    'Secondary Energy|Electricity|Coal|w/o CCS'
16    'Secondary Energy|Electricity|Gas'
17    'Secondary Energy|Electricity|Gas|w/ CCS'
18    'Secondary Energy|Electricity|Gas|w/o CCS'
19    'Secondary Energy|Electricity|Hydro'
20    'Secondary Energy|Electricity|Non-Biomass Renewables'
21    'Secondary Energy|Electricity|Nuclear' 'Secondary Energy|Electricity|Oil'
22    'Secondary Energy|Electricity|Solar' 'Secondary Energy|Electricity|Wind'
23    'Secondary Energy|Hydrogen|Electricity']

```

Transforming Electricity Mix Data

The following Python code performs transformation and normalization of the electricity generation data across various sources (Coal, Gas, Solar, Wind, Hydro, Nuclear) for selected years. It prepares the dataset for visualization or further analysis.

```

1 mix = selected.T # Transpose to get years as rows
2 mix.columns = ["Coal", "Gas", "Solar", "Wind", "Hydro", "Nuclear"]
3
4 # Normalize each row so that values sum to 100%
5 mix_percent = mix.div(mix.sum(axis=1), axis=0) * 100

```



```

6
7 # Add the Year column to the front
8 mix_percent.insert(0, "Year", mix_percent.index.astype(int))

```

Explanation:

`mix = selected.T` transposes the DataFrame so that each year becomes a row and each energy source becomes a column. Column names are explicitly assigned to reflect the energy types.

`mix.div(...)` calculates the percentage share of each energy source per year by normalizing the row values to 100%.

The **Year** column is inserted at the start for clarity in further plotting or reporting.

This normalized mix is suitable for energy scenario modeling and time-series visualization of global electricity production trends under SSP2-Baseline projections.

Now Let's see the mix data:

```

1 mix_data = pd.DataFrame(mix_percent)
2 mix_data

```

Output as Follows

	Year	Coal	Gas	Solar	Wind	Hydro	Nuclear
2020	2020	41.648488	23.364785	1.744802	4.662266	17.311218	11.268441
2030	2030	43.807002	21.339435	3.500143	5.911775	15.244947	10.196699
2040	2040	47.297169	18.829873	5.307967	6.383325	14.069778	8.111887
2050	2050	48.767720	17.928141	7.320965	7.400336	12.533527	6.049311
2060	2060	48.312122	18.112909	9.299205	7.019769	11.131398	6.124597
2070	2070	49.262517	16.159974	12.310516	6.457119	9.914510	5.895363
2080	2080	50.765236	12.968368	15.488641	6.473705	8.941186	5.362863

Running LCA on Electricity Technologies and Grid Mix (2020–2080)

The following Python code evaluates life cycle impacts of electricity generation technologies across multiple years using Brightway2. It utilizes the IPCC 2021 GWP100 LCIA method, focusing on global warming potential.

```

1 from bw2calc import LCA
2 from bw2data import Method
3
4 # Filter only relevant IPCC 2021 GWP100 methods
5 ipcc = [m for m in bd.methods if 'IPCC' in str(m) and '2021' in str(m)
6         and 'GWP100' in str(m) and 'LT' not in str(m)
7         and 'fossil' not in str(m) and 'biogenic' not in str(m)
8         and 'land use' not in str(m) and 'SLCFs' not in str(m)]
9
10 results = []
11
12 for _, row in mix_data.iterrows():
13     year = int(row['Year'])
14     print(f"\n--- Year {year} ---")
15
16     for method in ipcc:
17         # 1. LCA for each individual technology (e.g., Coal, Wind)
18         for tech, activity in sources.items():
19             fu_single = {activity: 1}
20             lca = LCA(fu_single, method)
21             lca.lci()
22             lca.lcia()
23             results.append({
24                 "Year": year,
25                 "Technology": tech,
26                 "Impact Category": method[1],
27                 "Type": "Per Technology",
28                 "Impact (per kWh)": lca.score
29             })
30
31         # 2. LCA for the full grid mix for the year
32         fu_mix = {sources[tech]: row[tech] / 100 for tech in sources}
33         lca_mix = LCA(fu_mix, method)
34         lca_mix.lci()
35         lca_mix.lcia()
36         results.append({
37             "Year": year,
38             "Technology": "Grid Mix",
39             "Impact Category": method[1],
40             "Type": "Grid Mix",
41             "Impact (per kWh)": lca_mix.score
42         })
43 results_df = pd.DataFrame(results)
44 print(results_df)

```

Explanation ipcc filters for the appropriate climate change impact methods from IPCC 2021 (GWP100).

The code loops through each year (2020–2080) and:

- Calculates LCA for each energy source individually.

- Calculates LCA for the total energy mix using fractional contributions.

Each result stores the impact in kg CO₂-eq/kWh. Output as Follows:

Year	Technology	Impact Category	Type	Impact (per kWh)
2020	Coal	climate change	Per Technology	2.877946e-01
2020	Gas	climate change	Per Technology	5.778449e-01
2020	Solar	climate change	Per Technology	6.749898e+03
2020	Wind	climate change	Per Technology	4.166939e+06
2020	Hydro	climate change	Per Technology	5.571410e
2020	Nuclear	climate change	Per Technology	6.330581e+08
2020	Grid Mix	climate change	Grid Mix	1.036009e+09
2030	Coal	climate change	Per Technology	2.877946e-01
2030	Gas	climate change	Per Technology	5.778449e-01
2030	Solar	climate change	Per Technology	6.749898e+03
2030	Wind	climate change	Per Technology	4.166939e+06
2030	Hydro	climate change	Per Technology	5.571410e+09
2030	Nuclear	climate change	Per Technology	6.330581e+08
2030	Grid Mix	climate change	Grid Mix	9.141561e+08
2040	Coal	climate change	Per Technology	2.877946e-01
2040	Gas	climate change	Per Technology	5.778449e-01
2040	Solar	climate change	Per Technology	6.749898e+03
2040	Wind	climate change	Per Technology	4.166939e+06
2040	Hydro	climate change	Per Technology	5.571410e+09
2040	Nuclear	climate change	Per Technology	6.330581e+08
2040	Grid Mix	climate change	Grid Mix	8.355044e+08
2050	Coal	climate change	Per Technology	2.877946e-01
2050	Gas	climate change	Per Technology	5.778449e-01
2050	Solar	climate change	Per Technology	6.749898e+03
2050	Wind	climate change	Per Technology	4.166939e+06
2050	Hydro	climate change	Per Technology	5.571410e+09
2050	Nuclear	climate change	Per Technology	6.330581e+08
2050	Grid Mix	climate change	Grid Mix	7.368987e+08
2060	Coal	climate change	Per Technology	2.877946e-01
2060	Gas	climate change	Per Technology	5.778449e-01
2060	Solar	climate change	Per Technology	6.749898e+03
2060	Wind	climate change	Per Technology	4.166939e+06
2060	Hydro	climate change	Per Technology	5.571410e+09
2060	Nuclear	climate change	Per Technology	6.330581e+08
2060	Grid Mix	climate change	Grid Mix	6.592412e+08
2070	Coal	climate change	Per Technology	2.877946e-01
2070	Gas	climate change	Per Technology	5.778449e-01
2070	Solar	climate change	Per Technology	6.749898e+03
2070	Wind	climate change	Per Technology	4.166939e+06
2070	Hydro	climate change	Per Technology	5.571410e+09
2070	Nuclear	climate change	Per Technology	6.330581e+08
2070	Grid Mix	climate change	Grid Mix	5.899690e+08

Continued on next page

Table 5.1 – continued from previous page

Year	Technology	Impact Category	Type	Impact (per kWh)
2080	Coal	climate change	Per Technology	2.877946e-01
2080	Gas	climate change	Per Technology	5.778449e-01
2080	Solar	climate change	Per Technology	6.749898e+03
2080	Wind	climate change	Per Technology	4.166939e+06
2080	Hydro	climate change	Per Technology	5.571410e+09
2080	Nuclear	climate change	Per Technology	6.330581e+08
2080	Grid Mix	climate change	Grid Mix	5.323710e+08

Visualization of Climate Impact Trends Over Time

To analyze the evolution of climate change impacts over time, the following Python code snippet is used. It iterates through each unique impact category and distinguishes between two types of emissions data: *Per Technology* and *Grid Mix*. For each case, it generates a line plot of impact values across different years using the `seaborn` and `matplotlib` libraries.

```

1 for category in results_df['Impact Category'].unique():
2     for result_type in ["Per Technology", "Grid Mix"]:
3         plt.figure(figsize=(14, 8))
4         df_plot = results_df[
5             (results_df['Impact Category'] == category) &
6             (results_df['Type'] == result_type)
7         ]
8         sns.lineplot(
9             data=df_plot,
10            x="Year",
11            y="Impact (per kWh)",
12            hue="Technology",
13            marker="o"
14        )
15        plt.title(f"{category} ({result_type}) Impact per kWh (20252075)")
16        plt.ylabel("Impact per kWh")
17        plt.grid(True)
18        plt.tight_layout()
19        plt.show()

```

This code allows us to visually compare the climate impact of various energy technologies over a span of decades (2020–2080). The use of colored lines and markers helps to distinguish each technology's trajectory. By separating the two result types, it effectively contrasts the individual technology emissions versus the aggregated grid mix emissions over time.

Output as Follows:

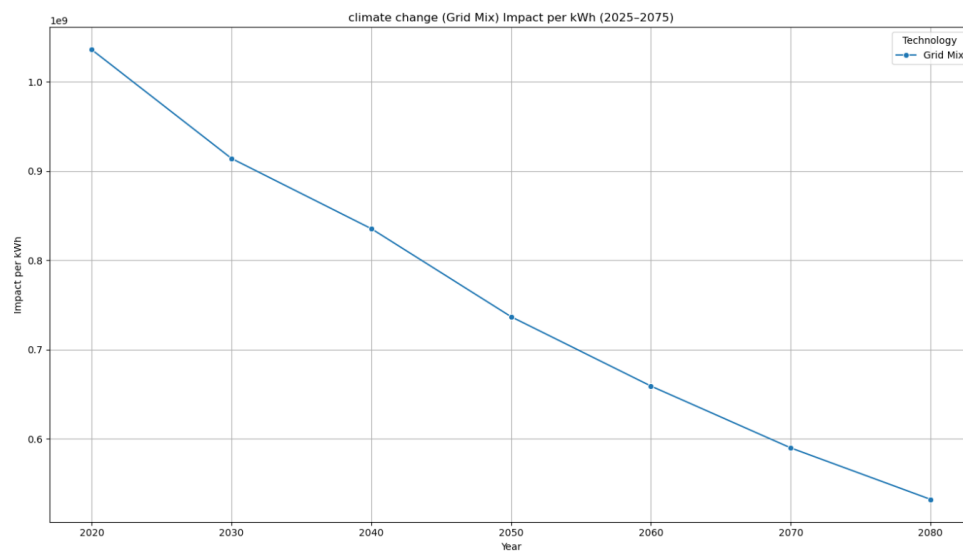


Figure 5.1: Grid Mix Impact Graph

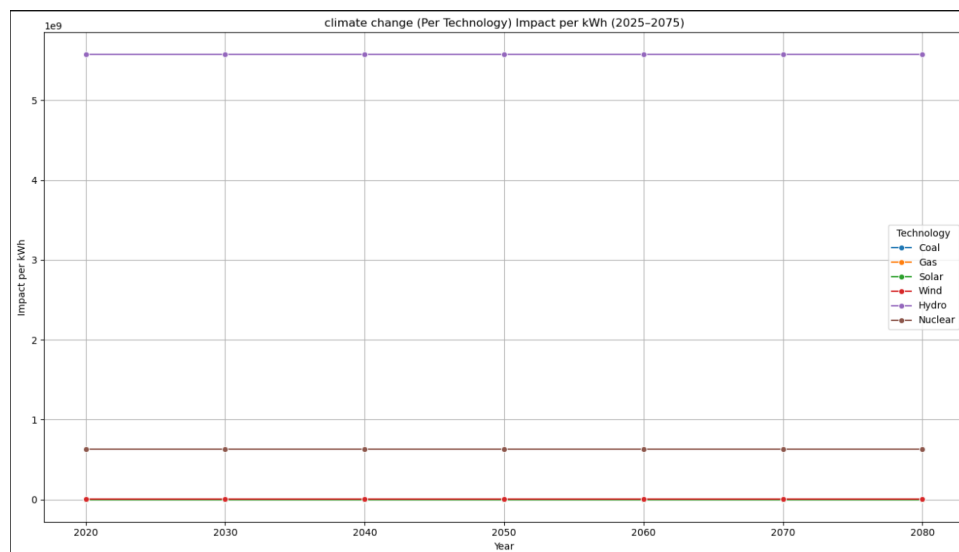


Figure 5.2: Technology Impact Graph

Chapter 6

Key Learnings

6.1 Technical Skills Gained

During this internship, I acquired and strengthened several technical skills:

1. Developed scripts for data analysis, emission calculations, and backend logic using **Python**.
2. Built interactive dashboards with forms, charts, and real-time data display with **Streamlit**.
3. Used **SQLite** for storing and retrieving structured data with CRUD operations.
4. Created visual outputs (bar charts, pie charts, tables) using **Matplotlib and Plotly**.
5. Configured, used, and interpreted life cycle models and outputs using the **Brightway2** framework in Python.
6. **Pandas** to clean, preprocess, and standardize raw datasets for emission calculations.

6.2 Domain Knowledge Acquired (Climate Science & Emissions)

This internship provided in-depth exposure to the domain of climate science, carbon emissions, and sustainable computing:

- **Understanding of Scope 1, 2, and 3 Emissions**
- **Life Cycle Assessment (LCA)**
- **product lifecycles**
- **Emission Factors and Reporting**

- Gained familiarity with open-source tools and principles through FOSSEE's workflow.
- Contributed to an open-source aligned dashboard tool for emission tracking. –Followed reproducible and transparent coding practices suitable for public sharing.

6.3 Communication and Team Collaboration

1. Participated in regular progress check-ins and coordinated with mentors and peers.
2. Maintained task documentation and version control to stay aligned with shared goals.
3. Improved clarity in reporting technical and environmental data for both technical and non-technical audiences.

Chapter 7

Conclusion

6.1 Overall Experience

The FOSSEE internship gave me a meaningful opportunity to **merge tech-nology with climate action**. Through this project, I was able to go beyond basic development and explore how digital tools can contribute to real-world sustainability.

From understanding emissions to actually calculating and visualizing them, every step of the internship felt connected to a larger environmental goal.

6.2 What Worked Well

- Building a working **dashboard from scratch** using open tools
- Creating a **user-friendly form system** that functioned both online and on mobile
- Mapping real data to **emission factor models**
- Exploring the potential of **carbon sequestration** in future tools
- Independently managing code, research, and reporting.
- LCA calculations using brightway2.
- Predictions and visualizations.

6.3 Scope for Future Work

- Add **user login system** and allow attendee profile editing
- Fetch emission factors from **real-time APIs**
- Use carbon sequestration offsets to calculate **net emissions**
- Expand dashboard to include **more Scope 3 categories**
- Incorporate national policy targets (e.g., India's National Electricity Plan or Net Zero targets) to replace or supplement SSP projections.
- Provide error bars or distributions instead of point estimates to improve decision robustness
- Design the dashboard to act as a learning tool for students and policymakers to understand the role of lifecycle assessment in energy planning.

Chapter 8

References

Data Sources

1. Intergovernmental Panel on Climate Change (IPCC) Emission Factor Guidelines
2. Food and Agriculture Organization (FAO) – Global Emission Statistics
3. UK Department for Environment, Food & Rural Affairs (DEFRA) – GHG Conversion Factors
4. Government of India – Ministry of Road Transport and Highways (MoRTH) Reports
5. Indian Railways Passenger Emission Estimations
6. Ecoinvent Database Documentation (as applicable)
7. National Renewable Energy Laboratory (NREL) – LCA Data Sources
8. Brightway2 and OpenLCA Toolkits
9. Academic Journals on Carbon Sequestration (Elsevier, Springer)

Libraries / Tools

1. Python 3.10
2. Streamlit
3. Pandas, NumPy, Matplotlib, Plotly
4. SQLite
5. Brightway2 (bw2data, bw2calc, bw2io)
6. OpenLCA (optional reference)
7. Qrcode, UUID, Time libraries for session and form utilities

Research Papers and Links

1. GHG Protocol Corporate Standard: **Click Here**
2. IPCC Reports Archive: **Click Here**
3. Brightway2 Documentation: **Click Here**
4. Activity Browser: **Click Here**
5. OpenLCA Software: **Click Here**
6. Ecoinvent: **Click Here**
7. Life Cycle Assessments **Click Here**
6. FOSSEE Resource Portal**Click Here**