Danial Karami
Python Tool Developer Assignment
Siemens Gamesa Renewable Energy

SIEMENS Gamesa
RENEWABLE ENERGY

Part 1 - Process Automation

In the overall process, selection of the most critical loads is the step I would choose to automate first. Doing this manually is not only slow and repetitive, but also error-prone. For example, if the wrong cases are selected or some are missed, the simplified FEA model ends up with false input, which directly affects all downstream results. Hence, this task is a good candidate for automation and would save time while avoiding such mistakes.

To approach this, I would write a Python script that reads the load spreadsheets and checks for issues such as missing values, inconsistent units, or unexpected column names. Once the data is cleaned, the script applies filters (e.g. min/max shear forces) to identify the critical load cases. The reduced set of loads is then written to a new CSV so the engineer can use it directly when setting up the simplified FEA model.

Steps to implement:

1. Load the Excel files and check that the tables follow the expected format.
2. Handle special cases such as empty cells, non-numeric entries, or inconsistent units.
3. Apply the filtering rules to identify the relevant load cases.
4. Export the result to a new, formatted CSV file for the Structural Analysis engineer.

Functions to define:

- `load_data(path)` – loads the Excel files.
- `validate(data)` – checks that the tables follow the expected format.
- `clean(data)` – fixes empty cells, non-numeric values, or inconsistent units.
- `filter(data)` – applies the selection rules to identify the relevant load cases.
- `export_csv(data)` – writes the reduced load set to a formatted CSV file.

Advantages: This removes repetitive spreadsheet work and keeps the filtering process consistent between users.

Disadvantages: Changes in the Excel format or project-specific filtering rules would require updates to the script.

Robustness: Including format checks, handling unusual entries, and adding clear error messages would make the tool reliable.

Part 2 - Coordinate System Transformation

To solve the second part of the assignment, I reduced the problem to two essential ideas:

- Every coordinate system is defined by an origin O and an orthonormal rotation matrix R.
- A point expressed in CSYS1 must first be expressed in the global frame, and only then re-expressed in CSYS2.

Standard rotation matrices about the X, Y, and Z axes are:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

In the implementation, these are constructed by the three small helper functions `rot_x`, `rot_y`, and `rot_z`. A full coordinate system is then created by supplying an origin and optional rotation angles about these axes. If no angles are provided, all three matrices reduce to the identity, and the coordinate system is simply aligned with the global frame. Whenever rotations are specified, the final orientation matrix is constructed using

$$R = R_z(\theta_z)\, R_y(\theta_y)\, R_x(\theta_x),$$

which is the fixed X→Y→Z Euler rotation convention used consistently throughout the code. Since matrix multiplication is not commutative, this fixed order is essential to ensure predictable behavior.

Point transformation. Given a point $p^1 = [x_1, y_1, z_1]^T$ expressed in CSYS1, it is first represented in global coordinates:

$$p_g = O_1 + R_1^g\, p_1.$$

To re-express the same global point in CSYS2, I rearrange CSYS2's definition:

$$p_g = O_2 + R_2^g\, p_2 \implies p_2 = (R_2^g)^{-1}\,(p_g - O_2).$$

Because we are dealing with orthonormal coordinate systems, $(R_2^g)^{-1} = (R_2^g)^T$, hence, the transpose is used instead of inverse. This both simplifies the computation and improves numerical stability.

All in all, the code directly follows this formula: first convert the point from CSYS1 to the global frame, then convert it from global to CSYS2. A simple True/False test is included at the end to confirm the implementation works.

Last but not least, I defined an `Object` class, since objects have position and orientation, unlike points which only carry position. A small `transform_object` function updates the position using the same point transformation as above, and updates the orientation by applying $R_1^g\, R_{\text{obj}}$ to express the object in the global frame and then premultiplying by R_g^2 to express it in CSYS2.

Performance and Executable Size Considerations

The point transformation can be simplified by avoiding the intermediate global point. Instead of first going from CSYS1 to the global frame and then from the global frame to CSYS2, the two

steps can be merged into a single direct expression. Starting from the relations $p_g = O_1 + R_1^g p_1$ and $p_2 = R_g^2(p_g - O_2)$, substituting the first into the second gives

$$p_2 = R_g^2\big(O_1 + R_1^g p_1 - O_2\big),$$

so the transformation can be carried out in one step instead of two.

A more compact and reusable way to express the same idea is to store both rotation and translation in a single 4×4 matrix:

$$T_1^2 = \begin{bmatrix} R_g^2 R_1^g & R_g^2(O_1 - O_2) \\ 0 & 1 \end{bmatrix}.$$

Then transforming a point becomes just one matrix–vector multiplication:

$$\begin{bmatrix} p_2 \\ 1 \end{bmatrix} = T_1^2 \begin{bmatrix} p_1 \\ 1 \end{bmatrix},$$

which expands exactly back into the previously derived expression:

$$p_2 = R_g^2\big(O_1 + R_1^g p_1 - O_2\big).$$

Why this helps. By combining everything into one 4×4 transformation matrix, I only compute the rotation and translation once and reuse it. This makes repeated point or object transformations faster, because each one becomes a single matrix–vector multiplication instead of several separate arithmetic steps. In a compiled implementation this also reduces the size of the executable, since the logic is compact and does not rely on NumPy or multiple helper function calls—just small fixed-size arrays and one clean operation.

Generative AI Use Statement

Generative AI (i.e., LLMs such as ChatGPT) was only used to improve English clarity and assist with minor debugging during the preparation of this work. All mathematical derivations, Python implementations, and the overall problem-solving approach were done by me.