**1. Problem Definition**

* **Objective:** Optimize coffee recipes (combination of ingredients and proportions) to maximize a taste rating.
* **Evaluation Criterion:** A numerical taste rating provided by a user or automated system.

**2. Coffee Recipe Representation**

Determine how the coffee recipe will be encoded for the algorithm:

* **Parameters:**
  + Coffee beans type (categorical variable, e.g., "Arabica," "Robusta").
  + Grind size (Numbers from 1 to 30 being 1 is the most fine and 30 is the most coarse)
  + Water-to-coffee ratio (continuous variable).
  + Brewing temperature (continuous variable).
  + Brewing time (continuous variable).
  + Additives (categorical, e.g., milk, sugar, syrups, with corresponding proportions).

**3. Fitness Function**

* **Input:** A coffee recipe.
* **Output:** A taste rating (numerical score, e.g., 0–10).
* Can be a user-provided score.
* Example fitness function:

**4. Evolutionary Components**

Define the key components of the evolutionary algorithm:

* **Population:** A collection of recipes to evaluate and evolve.
* **Selection:** Choose the best-performing recipes based on their fitness scores.
* **Crossover:** Combine features of two parent recipes to create offspring.
* **Mutation:** Introduce random changes to a recipe to maintain diversity.

**5. Evolutionary Algorithm Settings**

Specify parameters for the evolutionary process:

* **Population size:** Number of recipes in each generation.
* **Generations:** Total number of iterations to run the algorithm.
* **Mutation rate:** Probability of altering a recipe parameter.
* **Crossover rate:** Probability of combining recipes during reproduction.
* **Convergence criteria:** Stop the algorithm when a certain fitness score is reached or after a fixed number of generations.

**6. Constraints**

List any restrictions on recipe components:

* **Range limits:** Ensure ingredient proportions, temperatures, and times remain within feasible limits (e.g., temperature: 80–100°C, brewing time: 1–10 minutes).
* **Compatibility checks:** Avoid incompatible ingredient combinations (e.g., milk with certain brewing methods).

**Example Flow**

1. **Initialize Population:** Randomly generate a set of coffee recipes.
2. **Evaluate Fitness:** Assign taste ratings to each recipe.
3. **Selection:** Pick top recipes based on ratings.
4. **Crossover and Mutation:** Generate new recipes.
5. **Repeat:** Iterate over generations until the stopping criteria are met.
6. **Output:** Return the best recipe.

Enhancement

### 1. Save and Resume Progress

For long optimization runs, saving progress is crucial:

* Save the population and their fitness scores to a file (e.g., using pickle or JSON).
* Allow the algorithm to resume from the last saved state.

### 2. Add a Graphical User Interface (GUI)

Replace the terminal-based interaction with a GUI for better user experience:

* Use libraries like Tkinter, PyQt, or Streamlit to create an interactive application.
* Allow users to view recipe details and rate them via buttons or sliders.

3. Data Preference report at the end

### 3. Include Constraints

Some recipes might be impractical or unpalatable:

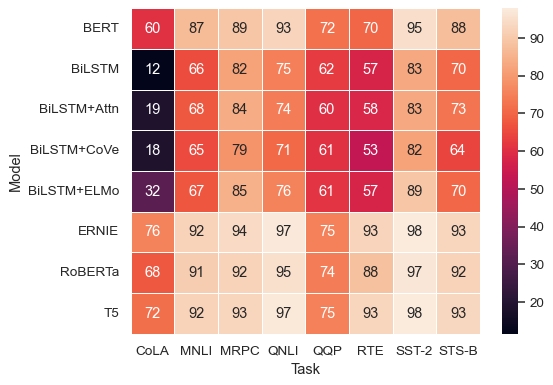
* Add constraints to prevent impossible combinations (e.g., excessive brewing time or very low temperatures).
* Clamp additive quantities to a reasonable range (e.g., sugar <= 10 grams).

### 4. Allow Feedback on Specific Features

Instead of rating the entire recipe, let users provide feedback on individual components:

* Ask the user to rate "bitterness," "sweetness," "aroma," etc.
* Use these ratings to compute an overall fitness score.

### 5. Create a coffee preference report

After each generation create a report that shows how to rating relate to other features using Heatmap or any other statistical tool

### 6. Make additives optional