**Program Documentation – Aver\_each\_gen\_per\_attempt\_and\_sim (2 genes)**

Royal Truman

**Overview**

The Find\_max\_generations.py script processes a semicolon-delimited input file containing simulation data storage in file results\_data\_per\_generation.txt, to identify the maximum generation number for each unique combination of simulation parameters (SimNr, attempt, Rep). The results are written to an output file in a structured format.

**Purpose**

The purpose of this script is to help analyze the distribution of max generations when simulation runs completed. Typically, the output is used to create histograms of the generation number to show the probability curse does not have a Gaussian shape.

The program:

* Extracts the highest generation number for each unique combination of SimNr, attempt, and Rep (Repetition).
* Ensures reliable processing by handling missing columns, empty lines, and malformed data.
* Produces a clean, semicolon-delimited output file for downstream analysis or reporting.

**Code Structure and Functionality**

**Function: process\_max\_generation**

* **Signature**:
* def process\_max\_generation(input\_file='results\_data\_per\_generation.txt', output\_file='Max\_generations.txt'):
* **Parameters**:
  + input\_file (str, default: 'results\_data\_per\_generation.txt'): Path to the input file containing simulation data.
  + output\_file (str, default: 'Max\_generations.txt'): Path where the output file will be written.
* **Purpose**: Reads the input file, processes it to find the maximum generation for each unique combination of SimNr, attempt, and Rep, and writes the results to the output file.

**Key Components**

1. **Import and Data Structure**:
   * Uses defaultdict from the collections module to create a dictionary (max\_gen\_per\_group) that stores the maximum generation for each unique key, defaulting to 0 for new keys.
   * **Why**: defaultdict simplifies the logic by eliminating the need to initialize dictionary entries, making the code more concise and efficient.
2. **Input File Processing**:
   * Opens the input file and reads the header line, splitting it by semicolons to identify column indices for SimNr, attempt, Rep, and generation.
   * Uses try-except to catch ValueError if any required column is missing, raising a RuntimeError with a descriptive message.
   * **Why**: Ensures the script fails gracefully if the input file lacks expected columns, aiding debugging and robustness.
3. **Line-by-Line Processing**:
   * Skips empty lines to avoid processing errors.
   * Splits each line by semicolons and extracts values for SimNr, attempt, Rep, and generation using the column indices.
   * Converts generation to an integer and handles IndexError or ValueError by skipping malformed lines.
   * Creates a tuple key = (simnr, attempt, rep) to uniquely identify each group.
   * Updates max\_gen\_per\_group[key] if the current generation is greater than the stored value.
   * **Why**: The tuple key ensures uniqueness for each combination, and skipping malformed lines prevents crashes on invalid data, making the script robust for real-world datasets.
4. **Output File Writing**:
   * Opens the output file in write mode and writes a header line: SimNr;attempt;Rep;generation.
   * Iterates over max\_gen\_per\_group to write each key-value pair as a semicolon-delimited line.
   * **Why**: The structured output format matches the input, ensuring compatibility with downstream tools, and the header clarifies the data structure.
5. **Main Execution**:
   * The if \_\_name\_\_ == '\_\_main\_\_': block calls process\_max\_generation() with default arguments.
   * Prints a confirmation message: "Results stored in file Max\_generations.txt."
   * **Why**: Allows the script to be run standalone or imported as a module, and the confirmation message informs the user of successful execution.

**Input and Output**

**Input File Format**

* **File**: results\_data\_per\_generation.txt (default).
* **Format**: Semicolon-delimited text file with a header row containing at least SimNr, attempt, Rep, and generation columns.

**Output File Format**

* **File**: Max\_generations.txt (default).
* **Format**: Semicolon-delimited text file with a header row and one row per unique (SimNr, attempt, Rep) combination, showing the maximum generation.

**Error Handling**

* **Missing Columns**: Raises RuntimeError if any required column (SimNr, attempt, Rep, generation) is missing in the header.
* **Empty Lines**: Skips empty lines to prevent processing errors.
* **Malformed Lines**: Skips lines with invalid data (e.g., non-integer generation or missing fields) to ensure robustness.
* **Why**: These checks make the script resilient to common data issues, reducing the need for manual data cleaning.

**Design Decisions**

* **Semicolon Delimiter**: Matches the input format for consistency and compatibility with CSV-like data processing tools. This avoid inconsistent use of decimals (. Vs ,) for the USA and Europe.
* **Defaultdict**: Simplifies tracking maximum generations without explicit initialization for each key.
* **Tuple Key**: Ensures uniqueness for grouping by SimNr, attempt, and Rep, leveraging Python’s hashable tuples.
* **Error Skipping**: Skips malformed lines instead of raising errors to allow processing to continue with valid data.
* **Default File Names**: Provides sensible defaults to simplify usage while allowing customization via parameters.

**Usage**

To run the script:

python Find\_max\_generations.py

* Uses default input (results\_data\_per\_generation.txt) and output (Max\_generations.txt) files.
* Alternatively, import the function and call it with custom file paths:

from Find\_max\_generations import process\_max\_generation

process\_max\_generation('custom\_input.txt', 'custom\_output.txt')

**Limitations**

* Assumes semicolon-delimited input; other delimiters require code modification.
* Does not validate the content of SimNr, attempt, or Rep (e.g., could be strings or numbers).
* Overwrites the output file without warning, potentially losing previous data.