DATA 219 Final Project Summary

1. Overview of the Dataset

- **Dataset Name**: Stress Detection Dataset
- **Description**: The dataset contains participant-level data on stress indicators, including variables such as PSS_score (Perceived Stress Score), screen_on_time, sleep efficiency, heart rate, and personality traits like Neuroticism.
- **Data Quality Issues**: Identified issues include missing values, potential outliers, formats that are not consistent, and invalid ranges.

2. Question to Answer

- **Question**:

  How do sleep efficiency and screen-on time impact stress levels, and can we efficiently rank participants with the highest stress levels?

  - This question requires determining correlations, ranking, and efficiently maintaining top-k participants with the highest stress scores.
- **Answer**:
  - Sleep efficiency reduces stress levels if there's a negative correlation.
  - Screen-on time increases stress levels if there's a positive correlation.
  - Correlation and scatter plots provide evidence for these relationships.
  - Using a heap or sorting allows efficient identification of participants with the highest stress levels, enabling targeted stress management strategies.

3. Chosen Data Structures

1. NumPy Array

- **Justification**:
    - Efficient for numerical computations, matrix operations, and element-wise transformations.
    - Allows us to handle large, numerical datasets compactly.
- **Planned Operations**:
    - Correlation computation (e.g., between PSS_score and sleep efficiency).
    - Feature normalization for consistent comparison across participants.

2. Heap (Heapq)

- **Justification**:
    - Optimized for finding and dynamically maintaining extreme values like the top-k participants with the highest stress levels.
    - Suitable for priority-based operations.
- **Planned Operations**:
    - Efficient retrieval of the top-k participants with the highest stress scores.
    - Sorting and maintaining ranked lists dynamically.

3. Comparison of Data Structures

- Although both NumPy arrays and heaps are well-suited for their specific tasks, I will conduct a brief performance comparison. For example, I could compare the time it takes to compute correlations with NumPy versus other possible methods or test the time taken to retrieve the top-k participants using heaps compared to sorting the entire dataset. This

will allow me to highlight the effectiveness of the chosen data structures in terms of speed and efficiency.

4. Data Cleaning Steps

- **Completeness**:
    - Impute missing sleep efficiency and screen_on_time values using median values to minimize data loss.
- **Accuracy**:
    - Remove duplicate entries and handle conflicting values in features.
- **Validity**:
    - Address invalid ranges (e.g., negative values in heart rate).
- **Consistency**:
    - Ensure consistent formats for all columns (e.g., ensure screen_on_time is uniformly measured in minutes).

5. Operations and Implementation

- **Custom Algorithms Implemented**:
    - **Correlation Computation**: Implement a manual algorithm to calculate the Pearson correlation coefficient without built-in library functions.
    - **Ranking**: Use a custom heap-based algorithm to dynamically maintain the top-k stress scores.
    - **Sorting and Filtering**: Develop algorithms to sort and filter subsets of data for specific analyses.

6. Performance Analysis

- **Comparison Metrics**:
    - Execution time for correlation computation using NumPy arrays vs. raw Python lists.
    - Retrieval time for top-k participants using a heap vs. sorting methods.
    - Test with varying dataset sizes (e.g., 100, 1,000, 10,000 rows) to compare scalability.
- **Big O Analysis**:

1. Arrays:
    - O(n) for iteration-based operations.
    - The operations for data manipulation (such as slicing, correlation, and element-wise transformations) generally operate in linear time, O(n), making NumPy an efficient choice for handling large datasets.
2. Heaps: O(log k) for insertion and extraction, where k is the heap size.
    - As mentioned earlier, the heap operations for insertion and deletion are O(log n), which is more efficient than sorting an entire list (which has a time complexity of O(n log n)).

- Outputs for data structures:
    - NumPy Array:

Correlation between sleep duration and PSS score: -0.014899485704908467

Correlation between screen on time and PSS score: 0.004376872540594537

Correlation between sleep duration and screen on time: -0.006658611465802431

- Heap (Heapq):

Participant ID: 81, PSS Score: 39.0

Participant ID: 71, PSS Score: 39.0

Participant ID: 48, PSS Score: 39.0

Participant ID: 84, PSS Score: 39.0

Participant ID: 38, PSS Score: 39.0

Participant ID: 51, PSS Score: 39.0

Participant ID: 5, PSS Score: 39.0

Participant ID: 18, PSS Score: 39.0

Participant ID: 8, PSS Score: 39.0

Participant ID: 93, PSS Score: 39.0