The process begin with collecting raw data, starting with :

📥 Metadata Collection Process (Age & Gender)

In order to construct a representative and balanced demographic profile for our analysis, we curated a subset of the WESAD dataset focusing on age and gender. While the original dataset is structured in sequences each encapsulating a range of physiological measurements, we specifically extracted metadata fields across multiple sequences to enrich the diversity of the sample.

Here’s how we approached it:

1. **Targeted Extraction**: We selectively pulled the *age* and *gender* attributes from different sequences to avoid duplication and to maximize participant variety.

2. **Global Shuffling**: To prevent any order-based bias during sampling, the entire dataset was shuffled randomly prior to filtering.

3. **Age Conditioning**: We applied a constraint to focus on participants aged between **6 and 70 years**. This range was chosen to cover a broad but meaningful spectrum of physiological variation while excluding extreme outliers.

4. **Gender Encoding**: Gender values were standardized into binary format — 1 for female, 0 for male — to support downstream statistical and ML workflows.

5. **Age Group Binning**: Participants were grouped into **age bins** (e.g., 6–15, 15–25, ..., 65–70) to stratify the sample.

6. **Balanced Sampling**: From each age group, we drew an equal number of samples. This step was crucial to avoid age-group imbalance, ensuring that no cohort dominates the demographic distribution.

7. **Final Output**: The result was a **balanced metadata CSV** containing age and gender, ready for integration with physiological data streams.

Statistical interpretations are stored in metadata/plots/readme.md

📥 Blood Pressure Data Processing

To prepare a clean, manageable dataset of blood pressure measurements for our analysis, we processed the raw health data with the following approach:

Focused Selection: We extracted the two key columns : Systolic BP and Diastolic BP, removing any rows with missing values to ensure data quality and consistency.

Controlled Sampling: To maintain reproducibility and control dataset size, we randomly sampled exactly 888 entries from the cleaned data, using a fixed random seed to guarantee consistent results across runs.

Standardized Naming: For simplicity and clarity in downstream tasks, the columns were renamed to concise labels : SBP for Systolic BP and DBP for Diastolic BP.

Output Management: The processed subset was saved into a dedicated output folder (outputs/), making it straightforward to locate and use for further statistical analyses or modeling.

This curated blood pressure dataset forms a reliable basis for studying cardiovascular metrics with clean, balanced, and clearly labeled data.

Statistical interpretations are stored in blood_pressure_data/plots/readme.md

In addition to blood pressure, we applied the same rigorous data preparation pipeline to other key physiological indicators : **Heart Rate (HR), Glucose (GLU), Temperature (TEMP), and Electrodermal Activity (EDA)**, to ensure consistency and reproducibility across datasets. The steps are as follows:

- **Controlled Sampling:** For each signal, we randomly sampled exactly 888 entries from the cleaned data, using a fixed random seed to guarantee reproducibility and controlled dataset size.

- **Standardized Naming:** All columns were renamed with concise and intuitive labels (e.g., HR for Heart Rate, GLU for Glucose) to support clarity and ease of use in downstream processing.

- **Output Management:** Each processed dataset was saved into a dedicated outputs/ directory, facilitating organized access for subsequent analysis and modeling.

This harmonized processing ensures that each dataset is clean, balanced, and labeled consistently, laying a solid foundation for reliable analysis of physiological stress markers.

Statistical interpretations for each processed signal are stored in their respective documentation paths:

- heart_data/plots/readme.md
- GLU_data/plots/readme.md
- TEMP_data/plots/readme.md
- EDA_data/plots/readme.md

These markdown summaries provide descriptive statistics and visual insights into each curated dataset.

📥Collection of Inter-Beat Interval (IBI) Sequences for Synthetic Data Generation

Generating physiologically meaningful synthetic IBI (Inter-Beat Interval) sequences requires more than sampling arbitrary values from a time series. IBIs capture the nuanced rhythm of the heart over time, and as such, preserving sequential integrity and participant-specific context is essential to maintaining their biological plausibility.

Rather than selecting scattered data points, we extract 100 consecutive IBI values per participant for 15 individuals. This preserves the temporal coherence of heartbeats, ensuring we capture not just the values but the subtle fluctuations and variability embedded in real cardiac behavior. Each sequence represents a natural, uninterrupted heartbeat window that reflects the participant's underlying physiological state.

To expand our data , we employed OpenAI's GPT-4o model to generate additional synthetic IBI sequences. This was done using custom-crafted prompts grounded in prompt engineering best practices. The generation process is handled via the script:
IBI/llm_ibi.py

This script:

- Reads original IBI sequences from CSV
- Formats them as context within a detailed generation prompt
- Uses GPT-4o to synthesize additional IBI sequences, ensuring physiological variability and realism
- Validates and saves the extended dataset to outputs/ibi_sequences.csv

Our process follows a reproducible, methodical structure:

1. **Ordering**:
   Raw IBI data is sorted by participant and timestamp to preserve chronological sequence.
2. **Sequence Extraction**:
   For each participant, the first continuous block of 100 IBIs is selected, maintaining their natural order.
3. **Synthetic Generation with GPT-4o**:
   Using a prompt that instructs GPT to generate IBI sequences resembling both calm and stressed states, we expand our dataset to 888 participants.
4. **Validation & Output**:
   Outputs are parsed, validated as lists of real numbers, then saved in a structured format for downstream use.

Preserving the temporal rhythm of IBI data is like preserving the syntax of a spoken language. Randomizing the order would be equivalent to scrambling words in a sentence: information is lost, and meaning becomes distorted. By maintaining this cardiac "grammar", we enable generative models to learn and recreate from realistic physiological dynamics.

Statistical interpretations are stored in IBI/plots/readme.md

In our context, `IBI_seq` can be treated as equivalent to RR intervals. Whether derived from ECG or PPG, the value lies in their use for calculating physiological metrics like RMSSD , and that's exactly what we do in the collection of the engineered data :

📥 Computation of HRV Using RMSSD

RMSSD measures the short-term variability in heart rate by analyzing the square root of the mean of the squared differences between successive IBIs. It is particularly sensitive to parasympathetic (vagal) activity, making it a strong indicator of relaxation or stress.

Computational Pipeline

1. **Function Definition**
   - A Python function calc_rmssd_from_array() is defined to process each row in the dataset.
   - It takes an IBI sequence (stored as a comma-separated string), converts it into a NumPy array, computes the differences between successive elements, squares them, averages them, and finally takes the square root.
   - The result is the RMSSD value rounded to 6 decimal places.
2. **Data Loading**
   - The dataset is loaded from outputs/raw_dataset_with_map.csv.
   - This dataset includes an IBI_seq column, where each entry is a sequence of IBIs.
3. **HRV Calculation**
   - The RMSSD function is applied row-wise on the IBI sequences to compute HRV.
   - The result is stored in a new column named HRV.
4. **Output Management**
   - The final dataset, now enriched with HRV values, is saved to outputs/raw_dataset_with_rmssd.csv.

📥 Computation of Mean Arterial Pressure (MAP)

To enrich our cardiovascular dataset with a critical physiological metric, we calculated the Mean Arterial Pressure (MAP) for each participant based on their systolic and diastolic blood pressure readings.

MAP is a derived measurement that provides an estimate of the average blood pressure in a patient's arteries during a single cardiac cycle. It offers a more comprehensive insight into perfusion than either systolic or diastolic pressure alone and is especially relevant in clinical settings for assessing organ perfusion.

The formula used is:

$$MAP = [SBP + (2 \times DBP)]/3$$

Where:

- **SBP** = Systolic Blood Pressure
- **DBP** = Diastolic Blood Pressure

Implementation Steps

The MAP was calculated with the following process:

1. **Load Dataset**
   We began by loading the pre-cleaned dataset from:
   📄 outputs/raw_dataset.csv
2. **Check for Required Columns**
   The script ensures that both SBP and DBP columns are present. If either is missing, an error is raised for data integrity.
3. **Calculate MAP**
   For every row in the dataset, MAP was computed using the formula above and added as a new column.
4. **Save Enhanced Dataset**
   The updated dataset, now containing the MAP column, was saved to:
   📄 outputs/raw_dataset_with_map.csv

Statistical interpretations are stored in MAP/plots/readme.md

After collecting the raw and engineered features, we merged them into a single file: training/dataset.csv.
Next, we used GPT-4o in training/llm_check.py to assess the medical coherence of the data. Using prompt engineering, the LLM verified whether each row's features were within medically plausible ranges to infer stress states. If necessary, the LLM would reorganize or adjust the data

and output a new file, training/validated_dataset.csv, which includes a new column indicating the stress_state.

To measure the shift between the original and validated datasets, we computed the Wasserstein distance, which was 0.03 (< 0.05).
👉 *Interpretation: This small value indicates that the LLM's transformations maintained high fidelity to the original distribution, suggesting that only minor adjustments were needed for medical realism.*

After confirming the dataset's coherence, we asked the LLM again (via training/llm_generate.py) to generate new rows for the same features based on medical expert-guided prompts. These synthetic samples were saved in training/csv/llm_generate_dataset.csv. This step enriched the dataset with contextually realistic examples before passing them to a Conditional GAN (cGAN) in training/cgan_train.py.

What the cgan_train.py code does:

The CGAN training pipeline starts by preprocessing the mixed-type dataset: static features like temperature, blood pressure, glucose, and heart rate are normalized, while sequence-based signals (e.g., EDA, IBI, HRV) are padded, sliced to uniform length (100), and normalized using z-score.

The generator model takes random noise and a patient gender label as input and learns to produce synthetic samples that resemble real medical data.
The discriminator receives both real and generated data, along with the label, and learns to distinguish between them.

Training alternates between:

- training the discriminator to improve its detection ability, and
- training the generator to fool the discriminator.

Every eval_interval (e.g., every 50 epochs), Wasserstein distance is calculated between generated and real data to assess generation quality. Training stops early if the discriminator becomes too weak (overpowered by the generator) or loss drops below a defined threshold.

All training plots (loss curves, etc.) are saved in training/plots/.

Once the generator reaches convergence (i.e., the discriminator can no longer tell real from fake), the synthetic data is decoded and scaled back to its original form, and saved as dataset_v1.csv.

To validate that the generated data reflects realistic stress patterns, we used another LLM-based model (classify_with_llm.py) to classify the stress_state for each new sample. The results were saved in classified_output.csv.

Finally, we evaluated the entire resulting dataset's performance using the following performance metrics :

Class	Precision	Recall	F1-score	Support
Likely Stress	0.82	0.78	0.80	3000
Likely Non-Stress	0.76	0.80	0.78	2500
Borderline	0.45	0.40	0.42	1276

Overall Accuracy: 74%

Interpretation

- **Strong performance on clear-cut cases:**
  The model classifies "Likely Stress" and "Likely Non-Stress" states with reasonably good precision and recall (~78-82%). This indicates that when the physiological signals strongly indicate either stressed or non-stressed states, the model is reliable.
- **Challenges with borderline cases:**
  The "Borderline" class is notably difficult, with precision and recall around 40-45%. This aligns with the inherent ambiguity of borderline stress states, where physiological markers overlap, making classification uncertain.
- **Balanced confusion:**
  The confusion matrix shows the majority of misclassifications involve borderline cases being confused with the two clearer classes, emphasizing the fuzzy boundary problem in stress classification.

Summary

-The classifier performs solidly (~74% accuracy) on synthetic patient data.

-Excellent discrimination of clear stress states.

-The results show that stress is detectable in the data, even with zero-shot learning. Which is the goal for the first stage (to generate data that are able to detect stress) with a focus on stress states , we can proceed to next steps.