1. Case explanation

Visualizing time-series usually requires compromises between the amount of data being analyzed and the amount of information one can actually retrieve. When one wants to visualize values over large periods of time, it is often useful to show a down-sampled version of the data.

It is possible to down-sample time-series in multiple ways (e.g., removing every other sample), but here we will be focusing on **aggregation**.

Aggregation works by dividing the requested time-range into smaller portions, each containing a subset of all samples and then “summarizing” those portions in some way. If one, for example, wants to aggregate by **mean value**, one calculates the mean sample value for each portion, effectively down-sampling the original time-range into a number of samples equal to the number of portions.

We want you to implement a simple aggregation algorithm that when given a dataset and a time range, down-samples the time-series within that range to a predefined number of points.

2. What we expect

- Your code should come in the form of a python package that:
    - We should be able to install with the command "pip install ." (called inside the package folder).
    - Can rely on dependencies (public packages) from pypi. Please assure that your installation process installs all needed dependencies.
    - After installation, runs by simply calling its name from the command line.
- You should make use of numpy's parallelization properties (through vectorization) to accelerate the aggregation process.
- Your program should:
    - Ingest the data from the parquet we gave you;
    - Calculate the Z-score value for each timestamp of each variable;
    - Ask the user for the range and the type of aggregation to do;
    - Output (print) the two arrays that resulted from the aggregation. The first array has the Z-score values (for each variable) and the second array has the timestamps.
- **You should ONLY deliver a .zip file containing your package. That’s the only thing we need. We will extract the contents, navigate to the package folder and install through pip.**
- You have to deliver **up to 48h after** the moment you received the case.

3. Main ideas:

You received a parquet file containing the output from a model that predicts, for every timestamp, a normal distribution represented by the μ (mean) and σ (standard deviation). This particular example has predictions for **three variables** (sensors from wind turbines). From these we want you to calculate the Z-score for each timestamp of each variable, effectively creating three new time-series. **It is to these three Z-score time-series that you should apply the aggregator.**

Think of the output we require as the arrays you would feed to a tool like “plotly” if you wanted to plot a time-series:

- One array with the Z-scores values. It can be multidimensional, containing the points for the three variables.

- One array with time information (in this case, the timestamps).

4. The aggregator:

The aggregation should work by down-sampling the user selected range of the Z-score time-series **to exactly 200 points** (for each variable). You are free to implement the types of aggregation you find relevant, e.g.: **maximum value aggregation** works by dividing the range of each of the three Z-score time-series into 200 intervals. Each interval of each variable will have a given number of samples. For each interval you find the maximum Z-score value. Those are the values you need to create the down-sampled Z-score and timestamp arrays.

5. Inputs

- Ask the user for the path to the parquet.

- Ask the user what is the time-range (first day and last day) and what type of aggregation to do (mean value, max value, etc).

BONUS POINTS:

- The user can input this through a GUI (e.g., *streamlit*).

6. Outputs

- Print to the command line the Z-score and timestamp arrays that resulted from the aggregation.

BONUS POINTS:

- The user can visualize graphically the aggregation result (e.g., *plotly*).

7. Final thoughts:

- The expected dimension of the final **Z-score array is (3, 200)**. The expected dimension of the final **timestamp array is (200,)**.

- Here you are aggregating just three time-series, but remember that in a real world application the user would be interested in visualizing simultaneously, and while changing the time ranges very frequently, dozens of variables. Make sure to optimize the calculations so that these processes are scalable.