Topic 8: KPI Calculation Engine

Introduction

Our team has implemented an initial version of a calculation engine capable of responding to queries with a high level of stability, based on the dataset available to us, which we get in the form of a `.pkl` file. It is able to calculate KPIs based on the definitions selected by Topic 1 (Knowledge Base & Taxonomy). We chose to work in Python due to the wide availability of libraries and high compatibility, as our calculation engine will need to communicate with most of the other modules. Our initial program version is capable of handling some basic errors.

Dataframe Filtering

A first class, called `kpi_dataframe_filter`, serves the purpose of selecting only certain rows of the dataset, in order to pass them on to further filtering or data extraction operation. This operation is preliminar and could technically be avoided, instead filtering appropriately every time a value has to be selected. Our motivations to move these operation in their own class are as follows:

- **Code modularity:** by calling class methods we avoid a lot of code repetition.
- **Code readability** is very improved. The syntax of the filtering operations can be overwhelming when paired with more complex KPI aggregation, and a single line would be hard to interpret.
- **Debugging** is made easier.

In particular, we found that just three filtering operations are sufficient to compute all complex KPIs defined by the taxonomy. These are:

- `filter_kpi_by_machine(df, machine_id)`: The rows are filtered based on a string value compared with the `asset_id` column values. Many KPIs are computed on a single machine, so the importance of this method is paramount. Passing `'all_machines'` as an argument produces no further filtering and all machines are selected.
- `filter_dataframe_by_kpi(df, kpi)`: A single machine has many atomic KPIs. For most calculations, only one of these is required, and this method selects it by taking its name in string form as an argument.

- `filter_dataframe_by_time(df, start_time, end_time)`: All KPIs are relative to a time period, and this method takes a `start_time` and an `end_time` as strings and compares them with the `time` column in the dataset. Basic error handling has also been implemented so that the start of the period does not come strictly after the end of the period.

Data Extraction

Once all the filtering has been done, we notice that each KPI has more than one value. In particular we have the maximum, minimum, sum and average over periods of 24 hours. Operations to extract meaningful information (again, as informed by the taxonomy) have been implemented in the class `kpi_dataframe_data_extraction`:

- `sum_kpi(kpi, df, machine_id, start_time, end_time)`:a sum over the `'sum'` column is returned.
- `avg_kpi(kpi, df, machine_id, start_time, end_time)`: the average of the values in the `'avg'` column is returned. The number of rows post-filtering is used as the denominator of the division.
- `max_kpi(kpi, df, machine_id, start_time, end_time)`: the maximum over the `'sum'` column is returned. This indicates the most (or least, based on the atomic KPI to be considered) productive day of the time period.
- `min_kpi(kpi, df, machine_id, start_time, end_time)`: equivalent to the above, but the minimum value is returned instead.

For all the methods, the same filtering operations are executed before returning the value, which are:

```
fd = df     # fd = filtered dataframe
fd = kpi_dataframe_filter.filter_dataframe_by_machine(fd, machine_id)
fd = kpi_dataframe_filter.filter_dataframe_by_kpi(fd, kpi)
fd = kpi_dataframe_filter.filter_dataframe_by_time(fd, start_time, end_time)
```

KPI Calculation

Once the methods described in the previous section have been defined, implementing the KPI calculation is almost trivial: a typical KPI formula consists of time period filtering, machine filtering (with the possibility of considering all machines) and a series of operations over values for certain atomic KPIs.

Power Consumption Efficiency

This KPI is calculated as $\frac{TotalWorkingTime}{TotalPowerConsumption}$ for a specific machine over a given interval.

So, the calculation is as follows:

```
def power_consumption_efficiency(df, machine_id, start_time,
↪  end_time):
fd = df
total_working_time =
↪  kpi_dataframe_data_extraction.sum_kpi(kpi='working_time',
↪  df=fd, machine_id=machine_id, start_time=start_time,
↪  end_time=end_time)
total_power_consumption =
↪  kpi_dataframe_data_extraction.sum_kpi(df=fd,
↪  kpi='consumption', machine_id=machine_id,
↪  start_time=start_time, end_time=end_time)
return total_working_time / total_power_consumption
```

Testing

We were unable to conduct rigorous testing due to the lack of an oracle, which is a reference system that provides reliable and validated output values to compare against the results from our calculation engine. In its absence, we tested all the methods by comparing them with manually calculated values derived from the dataset file, converted into .csv format. The values calculated by the engine match the manually calculated ones, so, within the context of this informal testing, we have deduced that the methods are functioning correctly.