Japan Power Data Toolkit (Python)

Loaders, Converters & Feature Engineering

August 12, 2025

Contents

1 Overview

This toolkit provides:

- a **canonical** in-memory representation for regional time series (`CanonicalNodalData`),
- converters between on-disk layouts: `multi` (MultiIndex columns), `long` (region column), `wide` (“`<region><sep><feature>`”; default `sep="-"`),
- single/all-region dataset loaders with pre-processing hooks,
- a registry of Japan column names and convenient groupings,
- feature-scaling utilities (EWMA/expanding) and per-file time-feature engineering.

Canonical representation `CanonicalNodalData` ≡ `dict[str,pd.DataFrame]`
Keys are region names; values are DataFrames indexed by time with feature columns.

2 Quick Start

Listing 1: Load a region and convert layouts

```
from jp_da_imb.data_loading.dataloader_nodal import (
    load_region_dataset, load_all_regions_dataset, canonical_to_layout
)

# 1) Load one region from mixed layouts with pre-processing
tokyo = load_region_dataset(
    region="tokyo",
    multi_paths=["/path/japan_train.parquet"],
    long_paths=["/path/imbalance_train.parquet"],
    long_region_cols=["region"], # region label column for each long file
    freq="30T", na_removal=True, add_time_feats=True,
)

# 2) Load all regions into canonical dict
all_regions = load_all_regions_dataset(
    multi_paths=["/path/japan_train.parquet"],
    wide_paths=["/path/occto_daily.parquet"], # optional
    freq="30T", add_time_feats=True,
)

# 3) Export canonical dict to wide layout
wide_df = canonical_to_layout(all_regions, layout="wide", sep="-")
```

3 Module: `dataloader_nodal_helper.py`

3.1 Types & constants

- `CanonicalNodalData=Dict[str,pd.DataFrame]`
- `NodalDataLayout=Literal["multi","long","wide"]`
- `DEFAULT_NODE_FEATURE_SEP="-"`

3.2 Function: `multi_to_dict`

Convert a MultiIndex-column DataFrame (level-0=region, level-1=feature) to canonical dict. Optional `cols` keeps a feature subset. Returns `CanonicalNodalData`.

3.3 Function: long_to_dict

Convert a long/tidy frame (must contain `region_col`, default `"region"`) to canonical dict. Drops the region column; all others are features.

3.4 Function: wide_to_dict

Convert wide columns named `"<region><sep><feature>"` to canonical dict (default `sep="-"`). Validates the name pattern.

3.5 Function: dict_to_multi

Combine a canonical dict into a MultiIndex-column DataFrame (sorted columns).

3.6 Function: dict_to_long

Combine a canonical dict into a long/tidy DataFrame, appending `region_col`; final columns are `[features...,region]`.

3.7 Function: dict_to_wide

Combine a canonical dict into a wide DataFrame with `"region<sep>feature"` columns.

3.8 Function: load_parquet_as_canonical

Read a parquet file (known `layout`) and return a canonical dict. Layout switch: `"multi"` → multi_to_dict, `"long"` → long_to_dict, `"wide"` → wide_to_dict. Raises `ValueError` for unsupported layouts.

4 Module: `dataloader_nodal.py`

4.1 Function: canonical_to_layout

Convert a canonical dict to the requested layout (`"multi"`, `"long"`, `"wide"`). Returns `pd.DataFrame`.

4.2 Function: load_region_dataset

Aggregate one region's data from any mix of `multi/long/wide` parquet files. It loads each file to canonical form, selects the region, runs preprocess_region_df on each slice, then merges:

- concatenate along columns,
- de-duplicate columns (keep last),
- sort by index.

Long layout: if `long_region_cols` is omitted, default is `"region"` per file (lengths validated). Raises `KeyError` if the region is missing from all files.

4.3 Function: load_all_regions_dataset

Load every region (or a subset) with the same pre-processing pass and return a canonical dict. It buckets frames by region, applies preprocess_region_df, merges with the same de-dup rules, and errors if requested `regions` are missing.

5 Module: `feature_engineering_nodal.py` (partial)

5.1 Function: `preprocess_region_df`

Per-file hygiene: clip to date range; resample to `freq` (mean); `dropna` if requested; `ffill(limit=1)`; convert index to `"Asia/Tokyo"`; if `add_time_feats`, call `construct_time_features` and cast common time columns (`weekday,hour,month,quarter,koma,koma_week,is_holiday,is_peak,is_weekend`) to categorical.

5.2 Function: `add_target_column`

Create `df[target]=df[minuend]-df[subtrahend]`. Optional trim to the valid span (`first_valid_index()..last_valid_index()`).

5.3 Function: `add_target_to_canonical`

Apply the same target construction across a canonical dict. Validates required columns; optional per-region trimming.

5.4 Function: `combine_regions`

Collapse several regions into a single aggregate frame: `add` (sum), `average` (weighted), `keep_first` (copy from first region), `drop` (discard). `weights` must be non-negative and normalized.

Note More feature-engineering helpers live here beyond the excerpts shown.

6 Module: `feature_engineering_df.py` (partial)

6.1 Function: `__prep_cols`

Split columns into numeric (non-target), categorical, and target; validate `target` exists.

6.2 Function: `scale_df_ewm`

EWMA standardisation for numeric columns; targets untouched. Uses `ewm(halflife,adjust=False).mean()/std()`, masks first `burnin_steps` to `NaN` then back-fills; re-assembles original order as float; optional `dropna()`.

6.3 Function: `scale_df_expanding`

Expanding-window standardisation (cumulative mean/std) with the same burn-in and re-assembly behaviour as EWMA.

7 Module: `japan_col_helper.py`

7.1 Class: `DataCols` (frozen dataclass)

Registry of Japan column names and logical groupings for fast selection. Examples: `con_mwh_h_jst_min15_{a,n}`, `pro_{spv,wnd}_mwh_h_jst_min15_{a,n}`, `cap_{spv,wnd}_mw_jst_min15_a`, temperature/derived indices, OCCTO reserve/demand/supply, hour dummies `hour_dummy_18..23`.

Common group properties `all`, `forecasts`, `ec00_forecasts`, `imbalance`, `outage`, `reserve_daily`, `reserve_snooping` — each returns a Python list of column names.

8 Module: quick_load_japan.py

8.1 Function: load_japan

Convenience loader for a “minimal, day-ahead” dataset: assembles parquet paths (multi/long), selects features via DataCols groupings, calls load_all_regions_dataset with `freq="30T"`, `na_removal=False`, `add_time_feats=True`, and returns a `CanonicalNodalData` dict keyed by region.

9 Examples

All regions, subset of features

```
from jp_da_imb.data_loading.japan_col_helper import DataCols
from jp_da_imb.data_loading.dataloader_nodal import load_all_regions_dataset

COLS = DataCols()
keep = COLS.forecasts + COLS.ec00_forecasts + COLS.imbalance

canon = load_all_regions_dataset(
    multi_paths=["/data/japan_train.parquet"],
    long_paths=["/data/imbalance_train.parquet"],
    long_region_cols=["region"],
    cols=keep,
    freq="30T", add_time_feats=True,
)
```

Convert to wide columns for modeling

```
from jp_da_imb.data_loading.dataloader_nodal import canonical_to_layout
wide = canonical_to_layout(canon, layout="wide", sep="-")
wide.to_parquet("/tmp/japan_wide.parquet")
```

10 Design Notes

- Converters are symmetric: any supported on-disk shape ↔ canonical dict.
- Region merges preserve column order; duplicates resolved by “keep last read”.
- A single pre-processing pass centralises resampling, NA rules, timezone and calendar features.

11 API Index

Module	Functions / Classes
`dataloader_nodal_helper`	`multi_to_dict`, `long_to_dict`, `wide_to_dict`, `dict_to_multi`, `dict_to_long`, `dict_to_wide`, `load_parquet_as_canonical`
`dataloader_nodal`	`canonical_to_layout`, `load_region_dataset`, `load_all_regions_dataset`
`feature_engineering_nodal` (partial)	`preprocess_region_df`, `add_target_column`, `add_target_to_canonical`, `combine_regions`, . . .
`feature_engineering_df` (partial)	`_prep_cols`, `scale_df_ewm`, `scale_df_expanding`, . . .
`japan_col_helper`	`DataCols` (group properties: `all`, `forecasts`, `ec00_forecasts`, `imbalance`, `outage`, `reserve_daily`, `reserve_snooping`)
`quick_load_japan`	`load_japan`