Quick start

`nemseer` lets you download raw historical forecast data from the MMSDM Historical Data SQLLoader, cache it in the parquet format and use `nemseer` to assemble and filter forecast data into a `pandas.DataFrame` or `xarray.Dataset` for further analysis. Assembled queries can optionally be saved to a processed cache.

Core concepts and information for users

Glossary

Refer to the glossary for an overview of key terminology used in `nemseer`. This includes descriptions of datetimes accepted as inputs in `nemseer`:

- run_start
- run_end
- forecasted_start
- forecasted_end

Note

AEMO ahead process tables with forecasted results typically have *three* datetime columns:

1. A forecasted time which the forecast outputs pertain to
2. A nominal run time. For most forecast types, this is reported in the `RUN_DATETIME` column.
3. An actual run time
   - The *actual* run time can differ from the *nominal* time. For example:
     - The 18:15 `P5MIN` run (`RUN_DATETIME`) may actually be run/published at 18:10 (`LASTCHANGED`)
     - The 18:30 `PREDISPATCH` run (`PREDISPATCHSEQNO`, which is parsed into `PREDISPATCH_RUN_DATETIME` by `nemseer`) may actually be run/published at 18:02 (`LASTCHANGED`)

The glossary also provides an overview of the various ahead processes run by AEMO, including:

- P5MIN
- PREDISPATCH
- PDPASA
- STPASA
- MTPASA

Parquet

Parquet files can be loaded using data analysis packages such as pandas, and work well with packages for handling large on-memory/cluster datasets (e.g. polars and dask). Parquet offers efficient data compression and columnar data storage, which can mean faster queries from file. Parquet files also store file metadata (which can include table schema).

Types of compiled data

`nemseer` has functionality that allows a user to compile data into two types of in-memory data structures:

- pandas DataFrames. Pandas is a widely-used Python package for manipulating data.
- Multi-dimensional xarray Datasets. xarray is intended for handling and querying data across multiple dimensions (e.g. the regional price forecast for a particular forecasted time from a range of run times)
  - For more information, refer to the *Getting started* section of the xarray documentation. The xarray tutorial is also an excellent resource.
  - Converting to xarray can be memory-intensive.

Managing memory

Some queries via `nemseer` may require a large amount of memory to complete. While memory use is query-specific, we suggest that `nemseer` be used on a system with at least 8GB of RAM. 16GB+ is preferable.

However, there are some things you can try if you do run into issues with memory. The suggestions below also apply to large queries on powerful computers:

1. You can use `nemseer` to simply download raw data as CSVs or to then cache data in the parquet format. Once you have a cache, you can use tools like polars or dask to process chunks of data in parallel. You may be able to reduce peak memory usage this way. It should be noted that `nemseer` converts a single AEMO CSV into a single parquet file. That is, it does not partition the parquet store.
2. Conversion to `xarray.Dataset` can be memory intensive. As this usually occurs when the data to be converted has a high number of dimensions (as determined by `nemseer`), `nemseer` will print a warning prior to attempting to convert any such data. While xarray integrates with dask, this functionality is contingent on loading data from a netCDF file.

Processed cache

The processed_cache is optional, but may be useful for some users. Specifying a path for this argument will lead to `nemseer` saving queries (i.e. requested data filtered based on user-supplied run times and forecasted times) as parquet (if the `pandas.DataFrame` data structure is specified) or netCDF (if the `xarray.Dataset` data structure is specified).

If subsequent `nemseer` queries include this processed_cache, `nemseer` will check file metadata of the relevant file types to see if a particular table query has already been saved. If it has, `nemseer` will compile data from the processed_cache.

Note

Because `nemseer` looks at metadata stored *in* each file, it does not care about the file name as long as file extensions are preserved (i.e. `*.parquet`, `*.nc`). As such, files in the processed_cache can be renamed from default file names assigned by `nemseer`.

Warning

Saving to netCDF will let you load xarray objects into memory. However, saving these datasets to netCDF files may take up large amounts of hard disk space.

Deprecated tables

If tables have been deprecated, `nemseer` will print a warning when the table is being downloaded. Deprecated tables are documented `here`.

What can I query?

`nemseer` has functionality to tell you what you can query. This includes valid forecast types, months and years for which data is available and requestable tables.

Note

While these functions allow you to explicitly query this information, it's worth noting that functions for compiling data and downloading raw data validate inputs and provide feedback when invalid inputs (such as invalid forecast types or data date ranges) are supplied.

Forecast types

You can access valid forecast types with the command below.

```
>>> import nemseer
>>> nemseer.forecast_types
('P5MIN', 'PREDISPATCH', 'PDPASA', 'STPASA', 'MTPASA')
```

Date range of available data

The years and months available via AEMO's MMSDM Historical Data SQLLoader can be queried as follows.

```
>>> import nemseer
>>> nemseer.get_data_daterange()
{...}
```

Table availability

You can also see which tables are available for a given year, month and forecast type.

Below, we fetch pre-dispatch tables available for January 2022 (i.e. this month would include or be between run_start and run_end):

```
>>> import nemseer
>>> nemseer.get_tables(2022, 1, "PREDISPATCH")
['CASESOLUTION', 'CONSTRAINT', 'CONSTRAINT_D', 'INTERCONNECTORRES', 'INTERCONNECTORRES_D', 'INTERCONNECTR_SENS_D', 'LOAD', 'LOAD_D', 'MNSPBIDTRK', 'OFFERTRK', 'PRICE', 'PRICESENSITIVITIE_D', '
```

AEMO's MMS Data Model reports describe tables and columns that are available via `nemseer`.

`PREDISPATCH` tables

Note

For some pre-dispatch table (`CONSTRAINT`, `LOAD`, `PRICE`, `INTERCONNECTORRES` and `REGIONSUM`), there are two types of tables. Those ending with `_D` only contain the latest forecast for a particular interval, whereas those without `_D` have all relevant forecasts for an interval of interest.

Compiling data

The main use case of `nemseer` is to download raw data (if it is not available in the raw_cache) and then compile it into a data format for further analysis/processing. To do this, `nemseer` has `compile_data`.

This function:

1. Downloads the relevant raw data and converts it into parquet in the raw_cache.
2. If it's supplied, interacts with a processed_cache (see below).
3. Returns a dictionary consisting of compiled `pandas.DataFrame`s or `xarray.Dataset`s (i.e. assembled and filtered based on the supplied run times and forecasted times) mapped to their corresponding table name.

For example, we can compile STPASA forecast data contained in the `CASESOLUTION` and `CONSTRAINTSOLUTION` tables. The query below will filter run times between "2021/02/01 00:00" and "2021/02/28 00:00" and forecasted times between 09:00 on March 1 and 12:00 on March 3. The returned `dict` maps each of the requested tables to their corresponding assembled and filtered datasets. These datasets are `pandas.DataFrame` as `data_format="df"` (this is the default for this argument).

```
>>> import nemseer
>>> data = nemseer.compile_data(
... run_start="2021/02/01 00:00",
... run_end="2021/02/28 00:00",
... forecasted_start="2021/03/01 09:00",
... forecasted_end="2021/03/01 12:00",
... forecast_type="STPASA",
... tables=["CASESOLUTION", "CONSTRAINTSOLUTION"],
... raw_cache="./nemseer_cache/",
... data_format="df",
... )
INFO: Downloading and unzipping CASESOLUTION for 2/2021
INFO: Downloading and unzipping CONSTRAINTSOLUTION for 2/2021
INFO: Converting PUBLIC_DVD_STPASA_CASESOLUTION_202102010000.CSV to parquet
INFO: Converting PUBLIC_DVD_STPASA_CONSTRAINTSOLUTION_202102010000.CSV to parquet
>>> data.keys()
dict_keys(['CASESOLUTION', 'CONSTRAINTSOLUTION'])
```

In the example above we include argument names, but these can be omitted.

You can also just query a single table, such as the query below:

```
>>> import nemseer
>>> data = nemseer.compile_data(
... "2021/02/01 00:00",
... "2021/02/28 00:00",
... "2021/03/01 09:00",
... "2021/03/01 12:00",
... "STPASA",
... "REGIONSOLUTION",
... "./nemseer_cache/",
... )
INFO: Downloading and unzipping REGIONSOLUTION for 2/2021
INFO: Converting PUBLIC_DVD_STPASA_REGIONSOLUTION_202102010000.CSV to parquet
>>> data.keys()
dict_keys(['REGIONSOLUTION'])
```

Note

`nemseer` also accepts datetimes with seconds specified, so long as the seconds are `00`. This is because the datetime fields that are relevant to `nemseer` functionality are specified to the nearest minute.

With datetimes specified down to seconds, you can the same datetimes for `nemseer` as you would for other related tools, such as `NEMOSIS` or `NEMED`.

We can also compile data to an `xarray.Dataset`. To do this, we need to set `data_format="xr"`:

```
>>> import nemseer
>>> data = nemseer.compile_data(
... "2021/02/01 00:00",
... "2021/02/28 00:00",
... "2021/02/28 00:30",
... "2021/02/28 00:55",
... "P5MIN",
... "REGIONSOLUTION",
... "./nemseer_cache/",
... data_format="xr",
... )
INFO: Downloading and unzipping REGIONSOLUTION for 2/2021
INFO: Converting PUBLIC_DVD_P5MIN_REGIONSOLUTION_202102010000.CSV to parquet
INFO: Converting REGIONSOLUTION data to xarray.
>>> data.keys()
dict_keys(['REGIONSOLUTION'])
>>> type(data['REGIONSOLUTION'])
<class 'xarray.core.dataset.Dataset'>
```

Compiling data to a processed cache

As outlined above, compiled data can be saved to the processed_cache as parquet (if `data_format` = “df”) or as netCDF files (if `data_format` = “xr”).

If the same processed_cache is supplied to subsequent queries, `nemseer` will check whether any portion of the subsequent query has already been saved in the processed_cache. If it has, `nemseer` will load data from the processed_cache, thereby bypassing any download/raw data compilation.

With a supplied processed_cache, we can save the query to parquet (`data_format` = “df”) or to netCDF (`data_format` = “xr”):

```
>>> import nemseer
>>> data = nemseer.compile_data(
... "2021/02/01 00:00",
... "2021/02/28 00:00",
... "2021/03/01 09:00",
... "2021/03/01 12:00",
... "STPASA",
... "REGIONSOLUTION",
... "./nemseer_cache/",
... processed_cache="./processed_cache/",
... )
INFO: Query raw data already downloaded to nemseer_cache
INFO: Writing REGIONSOLUTION to the processed cache as parquet
```

And if this saved query is a portion of another subsequent query, `nemseer` will load data from the processed_cache:

```
>>> import nemseer
>>> data = nemseer.compile_data(
... "2021/02/01 00:00",
... "2021/02/28 00:00",
... "2021/03/01 09:00",
... "2021/03/01 12:00",
... "STPASA",
... ["CASESOLUTION", "REGIONSOLUTION"],
... "./nemseer_cache/",
... processed_cache="./processed_cache/",
... )
INFO: Query raw data already downloaded to nemseer_cache
INFO: Compiling REGIONSOLUTION data from the processed cache
INFO: Writing CASESOLUTION to the processed cache as parquet
```

Validation and feedback

`compile_data` will validate user inputs and provide feedback on valid inputs. Specifically, it validates:

1. Basic datetime chronologies (e.g. run_end not before run_start)
2. Whether the requested forecast type and table type(s) are valid
3. Whether the requested run times and `forecasted times` are valid for the requested forecast type. In other words, forecasts that are run between run_start and run_end only produce data for a certain range of forecasted times. This varies between forecast types. For more information, refer to the forecast-specific datetime `validators`.

Getting valid run times for a set of forecasted times

If you’re interested in forecast data for a particular datetime range (i.e. between forecasted_start and forecasted_end) but not sure what the valid run times for this range are, you can use `generate_runtimes`.

This function returns the first run_start and last run_end between which forecast outputs for the forecasted times are available.

In the example below, we request run times that contain data for the forecasted times used in the compiling data examples:

```
>>> import nemseer
>>> nemseer.generate_runtimes("2021/03/01 09:00", "2021/03/01 12:00", "STPASA")
('2021/02/22 14:00', '2021/02/28 14:00')
```

You can see that in the compiling data examples we had a wider run time range. This is fine since filtering will only retain run times that contain the requested forecasted times. The inverse is not true: `compile_data` will raise errors if the requested forecasted times are not valid/do not have forecast outputs for the requested run times.

Downloading raw data

You can download data to a cache using `download_raw_data()`. This function only downloads data to the raw_cache.

CSVs can be retained by specifying `keep_csv=True`.

Unlike compiling data, only one set of datetimes needs to be provided (though these datetimes are keyword arguments for this function):

1. Provide `forecasted_start` and `forecasted_end` only. `nemseer` will determine the appropriate `run_start` and `run_end` for this forecasted range (via `nemseer.generate_runtimes()`) and download the corresponding raw data.
2. Provide `run_start` and `run_end` only. Dummy forecasted times are used.

```
>>> import nemseer
>>> nemseer.download_raw_data(
... forecast_type="P5MIN",
... tables="REGIONSOLUTION",
... raw_cache="./nemseer_cache/",
... forecasted_start="2020/01/02 00:00",
... forecasted_end="2020/01/02 00:30",
... keep_csv=False
... )
INFO: Downloading and unzipping REGIONSOLUTION for 1/2020
INFO: Converting PUBLIC_DVD_P5MIN_REGIONSOLUTION_202001010000.CSV to parquet
```

Alternatively, provide run times:

```
>>> import nemseer
>>> nemseer.download_raw_data(
... forecast_type="P5MIN",
... tables="REGIONSOLUTION",
... raw_cache="./nemseer_cache/",
... run_start="2021/01/02 00:00",
... run_end="2021/01/02 00:30",
... keep_csv=False
... )
INFO: Downloading and unzipping REGIONSOLUTION for 1/2021
INFO: Converting PUBLIC_DVD_P5MIN_REGIONSOLUTION_202101010000.CSV to parquet
```