IBM zStudent Contest

Source code:

```
#!/usr/bin/env python3
"""
Usage: poetry run test_parser FR production
"""

import pprint
import time
from datetime import datetime
from logging import DEBUG, basicConfig, getLogger
from typing import Any, Callable, Dict, List, Union

import arrow
import click

from electricitymap.contrib.config import ZoneKey
from parsers.lib.parsers import PARSER_KEY_TO_DICT
from parsers.lib.quality import (
    ValidationError,
    validate_consumption,
    validate_exchange,
    validate_production,
)

logger = getLogger(__name__)
basicConfig(level=DEBUG, format="%(asctime)s %(levelname)-8s %(name)-30s
%(message)s")


@click.command()
@click.argument("zone")
@click.argument("data-type", default="production")
@click.option("--target_datetime", default=None, show_default=True)
def test_parser(zone: ZoneKey, data_type, target_datetime):
    """\b
    Parameters
    ----------
    zone: a two letter zone from the map
    data_type: in ['production', 'exchangeForecast', 'production', 'exchange',
      'price', 'consumption', 'generationForecast', 'consumptionForecast']
    target_datetime: string parseable by arrow, such as 2018-05-30 15:00
    \b
    Examples
```

```
-------
>>> poetry run test_parser FR
>>> poetry run test_parser FR production
>>> poetry run test_parser "NO-NO3->SE" exchange
>>> poetry run test_parser GE production --target_datetime="2022-04-10 15:00"

"""
if target_datetime:
    target_datetime = arrow.get(target_datetime).datetime
start = time.time()

parser: Callable[
    ..., Union[List[Dict[str, Any]], Dict[str, Any]]
] = PARSER_KEY_TO_DICT[data_type][zone]
if data_type in ["exchange", "exchangeForecast"]:
    args = zone.split("->")
else:
    args = [zone]
res = parser(*args, target_datetime=target_datetime, logger=getLogger(__name__))

if not res:
    raise ValueError("Error: parser returned nothing ({})".format(res))

elapsed_time = time.time() - start
if isinstance(res, (list, tuple)):
    res_list = list(res)
else:
    res_list = [res]

try:
    dts = [e["datetime"] for e in res_list]
except:
    raise ValueError(
        "Parser output lacks `datetime` key for at least some of the "
        "ouput. Full ouput: \n\n{}\n".format(res)
    )

assert all(
    [type(e["datetime"]) is datetime for e in res_list]
), "Datetimes must be returned as native datetime.datetime objects"

assert (
    any(
        [
            e["datetime"].tzinfo is None
            or e["datetime"].tzinfo.utcoffset(e["datetime"]) is None
```

```
                for e in res_list
            ]
        )
        == False
    ), "Datetimes must be timezone aware"

    last_dt = arrow.get(max(dts)).to("UTC")
    first_dt = arrow.get(min(dts)).to("UTC")
    max_dt_warning = ""
    if not target_datetime:
        max_dt_warning = (
            " :( >2h from now !!!"
            if (arrow.utcnow() - last_dt).total_seconds() > 2 * 3600
            else " -- OK, <2h from now :) (now={} UTC)".format(arrow.utcnow())
        )

    print("Parser result:")
    pp = pprint.PrettyPrinter(width=120)
    pp.pprint(res)
    print(
        "\n".join(
            [
                "---------------------",
                "took {:.2f}s".format(elapsed_time),
                "min returned datetime: {} UTC".format(first_dt),
                "max returned datetime: {} UTC {}".format(last_dt, max_dt_warning),
            ]
        )
    )

    if isinstance(res, dict):
        res = [res]
    for event in res:
        try:
            if data_type == "production":
                validate_production(event, zone)
            elif data_type == "consumption":
                validate_consumption(event, zone)
            elif data_type == "exchange":
                validate_exchange(event, zone)
        except ValidationError as e:
            logger.warning("Validation failed @ {}: {}".format(event["datetime"], e))


if __name__ == "__main__":
    # pylint: disable=no-value-for-parameter
```

```
print(test_parser())
```