README.md

Plus Dental Data Engineering challenge

This is the solution repository for my PlusDental data engineering challenge. Here I have designed a command line application that can be run in two modes using the `run_as` argument - the api extractor tool and the dashboard applicatiion to visualize and explore the data collected.

The solution is also served using the functionality in two modules - `er_extractor` for the extractor applicatoon and `er_dashboard` for the dashboard application.

Instructions for using the solution application

To set up the application on your machine

- I assume that you have docker running on your machine, if you don't have it then please refer this page to install docker.
- Clone this repository on your local and then use the docker-compose to build the target solution image. You can do it via this command:

  ```
  docker-compose build
  ```

  Creating the build for the first time may take upto 2-3 minutes.
- Once the build is successful (Check ss to see what a successful build looks like), we can fire up the application using docker-compose. You can do it via the following command:

  ```
  docker-compose up -d
  ```

  The -d flags runs the containers in daemon mode.

- You can verify that the containers are live using `docker ps` command. This will show you all the live containers.
- Running the containers, runs the application in dashboard mode alongside a mysql db. You can visit the dashboard on your localhost. Right now you might see an empty graph since there would be no data that has been extracted from the API. See th following instructions to run the application in extractor mode.

Using the application in extractor mode

In this step I assume, that you already have the docker containers running in daemon mode and you have access to your exchange rates dashboard.

Before running the application as extractor, let's first see what arguments are available using the -h flag. (We will be running the application inside the docker container)

You can get the list of arguments by using the following command:

```
docker exec -it exchange_rates_extractor_web_1 python main.py -h
```

If everything worked for you, you should see the following help message

```
(env) (base) ➜  exchange_rates_extractor git:(dockerfied) ✗ docker exec -it exchange_rates_extractor_web_1 python main.py -h
usage: main.py [-h] [--run_as {extractor,dashboard}] [--get_data_by {date,date_range,latest,exhaustive}] [--start_date START_DATE] [--end_date END_DATE] [--multithreading]
               [--multithreading_after MULTITHREADING_AFTER] [--num_of_threads NUM_OF_THREADS] [--port PORT_NUM] [--env {dev,prod}] [--initialize_db]

Pipeline to extract the exchange rates data using the exchangerates.io API interface. Use the followring arguments to run the pipeline to get the using the API, and then persist it into the database, or
run a flask application to explore the already recorded data.

optional arguments:
  -h, --help            show this help message and exit
  --run_as {extractor,dashboard}
                        Use this argument to either extract the data using `extractor` argument or initalize the dashboard to explore the data.
  --get_data_by {date,date_range,latest,exhaustive}
                        Use this subargument for extractor to specify the criteria for the data retrieval
  --start_date START_DATE
                        Use this extractor argument to specifiy the start date for the interval for which the data needs to be fetched. The date specified should be greater than equal to 2000-01-01, and
                        must be less than equal to 2020-08-10.
  --end_date END_DATE   Use this extractor argument to specifiy the end date for the interval for which the data needs to be fetched. The date specified should be greater than equal to 2000-01-01, and
                        must be less than equal to 2020-08-10.
  --multithreading      Use this argument to enable multithreading in case of numeral request urls to reduce the network overhead time. If number of request urls excedes 10, then multitrheading is
                        enabled by default. You can use the multithreading_after argument to change this setting.
  --multithreading_after MULTITHREADING_AFTER
                        Use this argument to specifiy after how many requests should multithreading be enabled.
  --num_of_threads NUM_OF_THREADS
                        Use this argument to specify how many threads you want to use for multithreading. It is recomended to use the total number of cores you have on your machine as the number of
                        threads.
  --port PORT_NUM       Use this argument to specifiy the port numebr you wish to use for your desktop application. When running the application in a docker container, only a handful number of ports can
                        be used. It is recomended to stick to the default setting.
  --env {dev,prod}      Use this argument to specify whether the processes are to be run in a development environment or production.
  --initialize_db       Use this argument to initialize the database tables.
```

We can run the application as an extractor using the `run_as` argument. While running the application in extractor mode, you also need to specify the `get_data_by` sub argument. We can fetch data either for a date, or for a date range. We can also do an exhaustive fetch (Takes about 5 mins if multithreading is enabled). Or we can only get the data for the last 7 days (latest data).

If you plan to fetch rates for a high number of dates it is recommended to enable `multithreading` while running the application. You can also specify the number of threads you wish to create.

To show you an example command, let's say I wanted to fetch the data for the date range 2018-10-01 - 2019-10-31, the command would look like:

```
docker exec -it exchange_rates_extractor_web_1 python main.py --run_as extractor --multithreading --get_data_by date_range --start_date 2018-10-01 --end_date 2019-10-31
```

Or if I wanted to do an exhaustive fetch:

```
docker exec -it exchange_rates_extractor_web_1 python main.py --run_as extractor --multithreading --get_data_by exhaustive
```

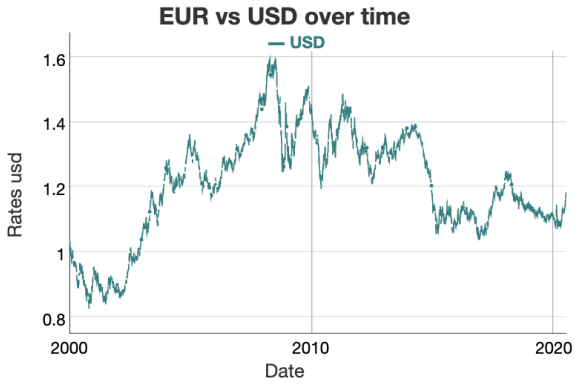
An example run looks like following, although in this run, no data is being fetched right now, this is just to give you an idea of how logging is handled:

```
(base) ➜  exchange_rates_extractor git:(documentation) docker exec -it exchange_rates_extractor_web_1 python main.py --run_as extractor --multithreading --get_data_by exhaustive
2020-08-10 14:47:56,663 - __main__ - INFO - Your namespace - Namespace(end_date=None, env='dev', get_data_by='exhaustive', initdb=False, multithreading=True, multithreading_after=5, num_of_threads=8, port
_num=8080, run_as='extractor', start_date=None)
2020-08-10 14:47:56,663 - __main__ - INFO -
2020-08-10 14:47:56,663 - __main__ - INFO - Running application as extractor process
2020-08-10 14:47:56,663 - __main__ - INFO -
2020-08-10 14:47:56,663 - er_extractor.core - INFO - Collecting dates for request urls
2000-01-01 00:00:00 2020-08-10 00:00:00
2020-08-10 14:47:57,209 - er_extractor.core - INFO - Extracted dates for which requests need to made: 0
2020-08-10 14:47:57,209 - er_extractor.core - INFO - No new dates found
2020-08-10 14:47:57,209 - er_extractor.core - INFO -
2020-08-10 14:47:57,210 - __main__ - INFO - Total time consumed: 0 minutes 0 seconds
2020-08-10 14:47:57,211 - __main__ - INFO -
2020-08-10 14:47:57,211 - __main__ - INFO - -*-*-*-*-*-*-*-*-*-*-*-END-*-*-*-*-*-*-*-*-*-*-*-
2020-08-10 14:47:57,211 - __main__ - INFO -
```

Once you have the data retrieved, your dashboard should have the latest data if you reload it. The updated dashboard looks like following:

Exchange rates

For the bonus tasks mentioned in the data engineering challenge, following is the graph that demonstates the recorded rates.



The chart is *interactive*: you can mouse over to highlight individual values. You can click and drag to zoom. Double-clicking will zoom you back out. Shift-drag will pan. You can change the number and hit enter to adjust the averaging period.

If you don't see any data on the graph then it probably means you don't have any data in your MySql database. Try running the process in extractor mode to get the data. Data collection for the entire 20 years period takes about *5 mins*, if you have multithreading enabled.

Note on extractor:

All the points / feature requests for the extractor proicess can be handled / served using the 4 options for the subargument `get_data_by` . Any further enhancements can be made on top of the existing solution.