ETL DOCUMENTATION

Created for Fred API

PREPARED FOR

Maria Victoria Salas

Finalis

PREPARED BY

Mariano Tancredi

1. Project Overview

The assignment asked to develop and execute an ETL process using the Federal Reserve API to create a specific dataset. The project contains files for downloading, parsing and uploading the data. Each function can be run individually or as a whole process. Before running any script, the constant file should be updated with the API and Database Data. Note: By default, the flag that enables the script to upload the data to the Database is set to False, so in case Database information is not at hand, the process can be run anyway.

2. Project Structure

The project consists of 4 files (Without counting the constants file). Except for the caller, all scripts are inside the **util** folder, which is divided into a **config** folder and a **files** folder:

- Downloader: Contains every function related to the API usage/content. It uses a mapping file to generate the different URLs and the files
- Parser: Contains every function related to the Data sanitization. This file creates a Dataset like the one asked in the assignment, and slices the data according to the date stated there (**2000-01-01)**.
- Loader: Loads the observations into a Database. Given old observations do not tend to change over time, there is a function which can be activated so only the latest observations are uploaded. In case of a Major update in past observations, the whole dataframe should be inserted again
- Caller: Contains the main workflow where each function is called. It also contains callers for each individual function.

Both downloader and parser contain a flag, which when activated, will write the files into a folder separated by ‘Raw Files’ and ‘Processed Files’. This flag comes activated by default but can be bypassed if sent as a parameter.

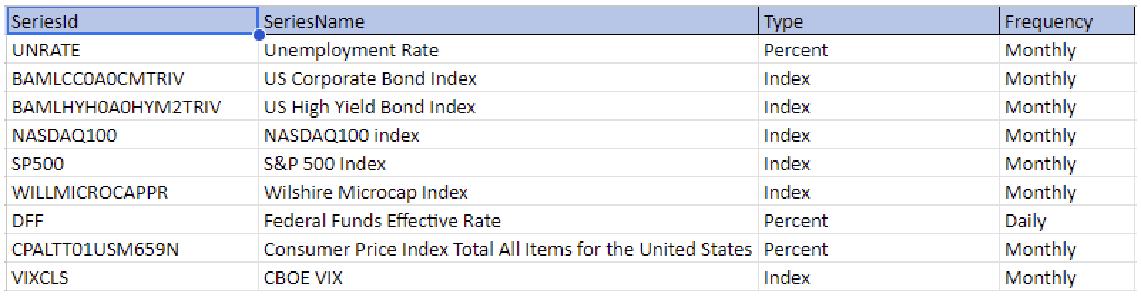
The idea for this workflow, given the amount of observations, is to work with in-memory data through each function. This means passing the data and dataframes as parameters instead of parsing it in each function. Although this could have a high memory usage problem in some cases, this is not one given the amount of observations. In case new series were added and memory would become a problem, all of the functions can be setted to run by parsing local files instead of working with this method.

The process is ready to be triggered by running the caller file, which will run the downloader and parser only. In case SQL configuration is set in the Constants file, the flag to upload to the Database can be set to True instead of False.

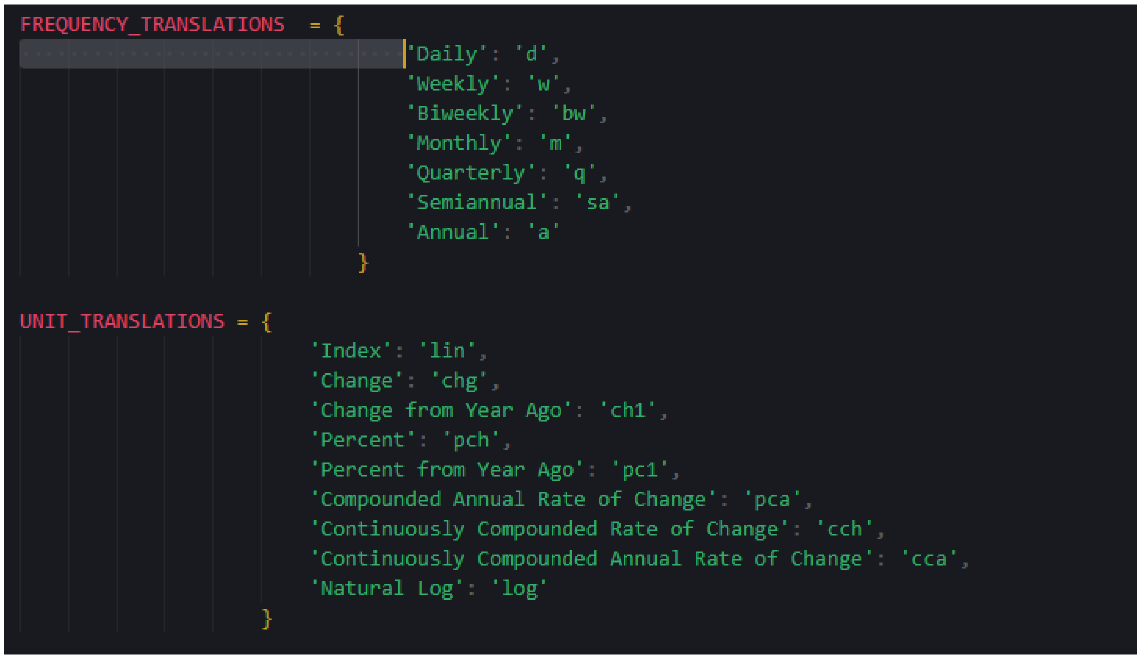
3. Obstacles

For this project, there were only a few obstacles that i faced:

- The information given for each series is written in a more ‘formal’ language and not in the one used by the APIs. This is normal, given that most of the time, the metadata for the series is created by the Analysts and not the programmers. We can't expect the analyst to read the API documentation. To solve this, I created a mapping file in Excel format. This is really helpful because this way, any change or new series that need to be added can be done though the mapping itself and without touching any code. Also, the mapping is written in a more common way and we handle the translations for the request by ourselves in the code. Example:



SeriesId	SeriesName	Type	Frequency
UNRATE	Unemployment Rate	Percent	Monthly
BAMLCC0A0CMTRIV	US Corporate Bond Index	Index	Monthly
BAMLHYH0A0HYM2TRIV	US High Yield Bond Index	Index	Monthly
NASDAQ100	NASDAQ100 index	Index	Monthly
SP500	S&P 500 Index	Index	Monthly
WILLMICROCAPPR	Wilshire Microcap Index	Index	Monthly
DFF	Federal Funds Effective Rate	Percent	Daily
CPALTT01USM659N	Consumer Price Index Total All Items for the United States	Percent	Monthly
VIXCLS	CBOE VIX	Index	Monthly



```
FREQUENCY_TRANSLATIONS  = {
                                'Daily': 'd',
                                'Weekly': 'w',
                                'Biweekly': 'bw',
                                'Monthly': 'm',
                                'Quarterly': 'q',
                                'Semiannual': 'sa',
                                'Annual': 'a'
                            }

UNIT_TRANSLATIONS = {
                        'Index': 'lin',
                        'Change': 'chg',
                        'Change from Year Ago': 'ch1',
                        'Percent': 'pch',
                        'Percent from Year Ago': 'pc1',
                        'Compounded Annual Rate of Change': 'pca',
                        'Continuously Compounded Rate of Change': 'cch',
                        'Continuously Compounded Annual Rate of Change': 'cca',
                        'Natural Log': 'log'
                    }
```

As you can see, the excel contains the data as exactly depicted in the assignment specifications and the dictionaries created in code, translate these names to the one used in the API request. This creates a line between Analyst responsibilities and the programmers one.

4. Constants File:

For this work, instead of using environment variables I opted to save important information on a **constant** file. The file contains:

- API_KEY: Token for the API
- URL_TEMPALTE: The URL used for the request
- FREQUENCY/UNIT_TRANSLATIONS: As stated in the point before, this dictionaries are used to translate the mapping file for the request
- START_DATE: The assignment asked to keep only observations after **2000-01-01**
- TABLE_NAME: Name of the SQL Table
- DATE_COLUMN: Name of the column where the Observation Dates are stored (I used 'Observation Date' as the name for example
- DRIVER_NAME: Driver name for the Database used
- SERVER_NAME: Server name for the Database connection
- DATABASE_NAME: Database Name
- SQL_ENGINE: Given we use a pandas function to upload a dataframe, we need to specify the engine of the Database we are using.