**Project: Master Database Builder**

**Description:**

This project aims to get alternative data and financial data from a variety of public databases and build an automatically daily updated database (master table) for research use.

Data sources includes:

Federal Reserve Economic Data (FRED); <https://fred.stlouisfed.org/>

World Bank; <https://data.worldbank.org/>

Climate Watch; <https://www.climatewatchdata.org/>

Resource Watch; <https://resourcewatch.org/>

**Files:**

Master\_Database\_Builder.ipynb (Main)

Fred\_grabber.py

* ISSO\_CODE.xlsx (tool file used in Fred\_grabber.py)

WorldBank\_grabber.py

ClimateWatch\_grabber.py

\*ResourceWatch\_grabber.py

\*Builder\_tools.py

Data:

Yearly.csv

Quarterly.csv

**Extensibility:**Additional database that needed to add to the master database should has a file [data].py which contains a method that could be used in Main file to extract data from online databases and format it into desired form as follow:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Country | Year | Var\_1 | Var\_2 | Var\_3 | Var\_4 | … | … | … | Var\_n |
| Country<1> | 1950 | … | … | … | … | … | … | … | … |
| Country<1> | … | … | … | … | … | … | … | … | … |
| Country<1> | 2019 | … | … | … | … | … | … | … | … |
| … |  | … | … | … | … | … | … | … | … |
| Country<2> | 1950 | … | … | … | … | … | … | … | … |
| … |  | … | … | … | … | … | … | … | … |
| Country<m> | 2019 | … | … | … | … | … | … | … | … |

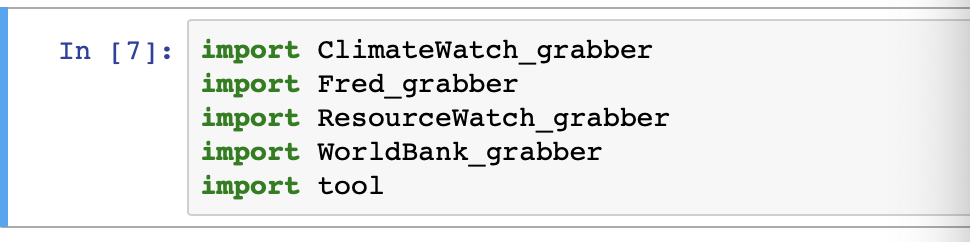
So that in the main file (Master\_Database\_Builder.ipynb), it can be merged into the master table on ‘country’ & ‘year’.

**Prerequisite:**

* **Python 3**
* **Package: Numpy**
* **Package: Pandas**
* **Package: Fred (following instructions here** [**https://github.com/mortada/fredapi**](https://github.com/mortada/fredapi)**)**
* **Package: wbdata (<https://wbdata.readthedocs.io/en/latest/>)**
* **Anaconda Jupiter (for development use only)**

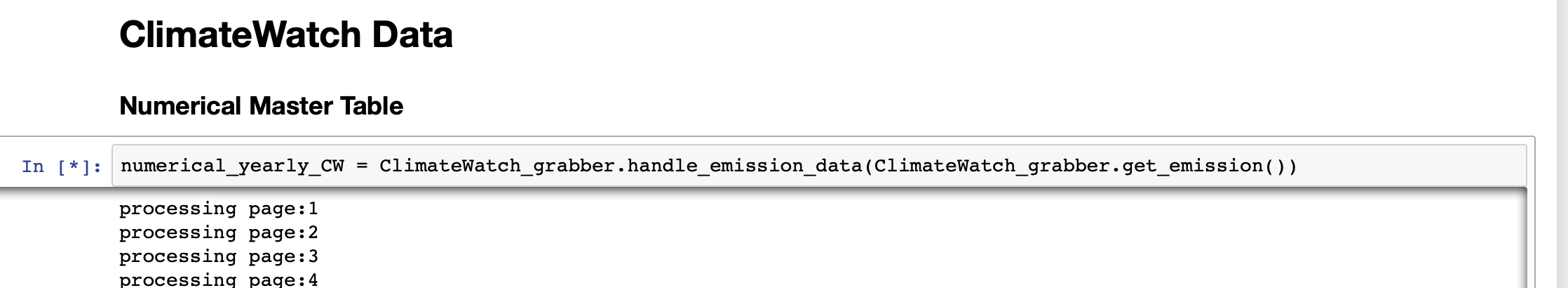
**Use**

**Step1:** import all grabber files and tool file

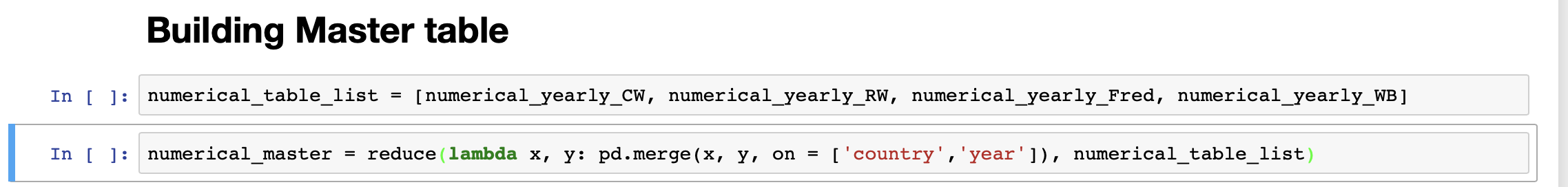
****

**Step2:** get data from different data source using grabber and assign variable name in following fashion:

numerical/categorical\_yearly/quarterly\_<data\_name>



**Step3:** Build 2 master table, by merging all numerical data into one master table on ‘country’ and ‘year’

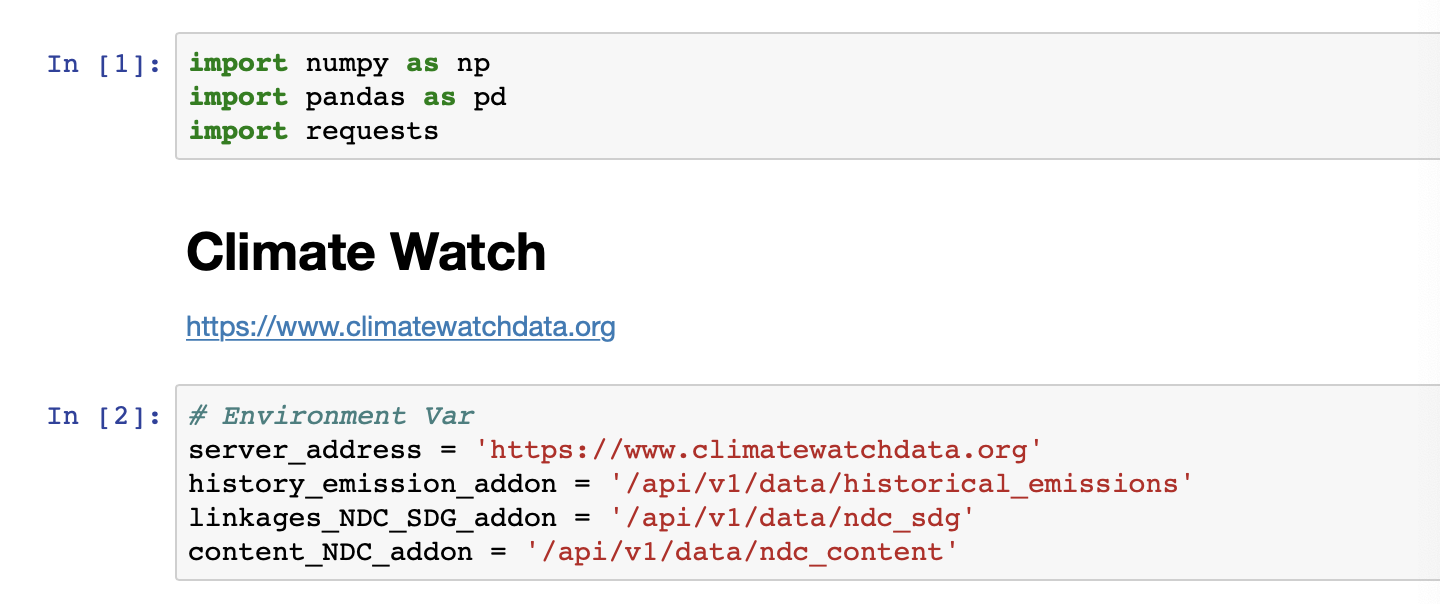


**File documentations:**

(For modification & development, better use .ipynb file which contain more specific documentations)

* **ClimateWatch Grabber**

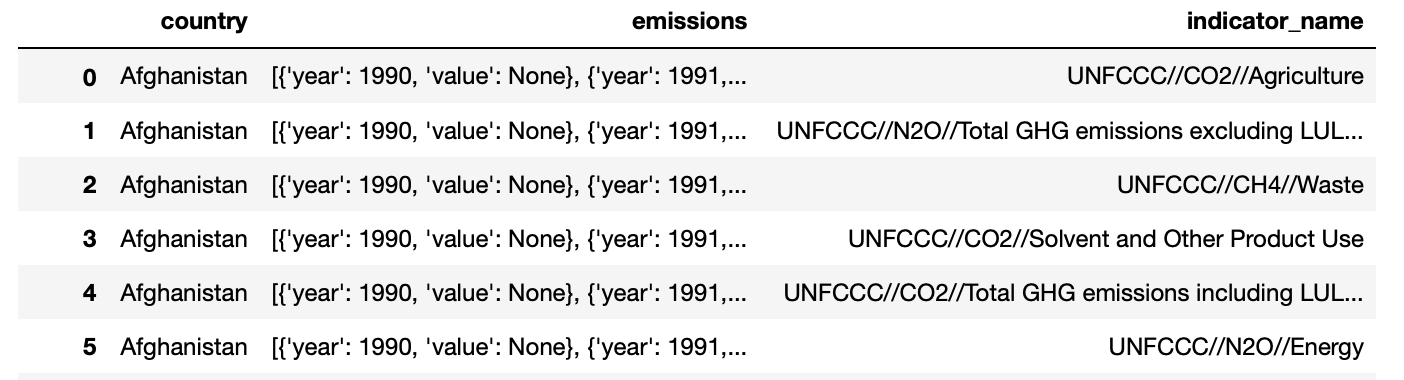
**Step1:** import libraries and setup configuration for ClimateWatch API



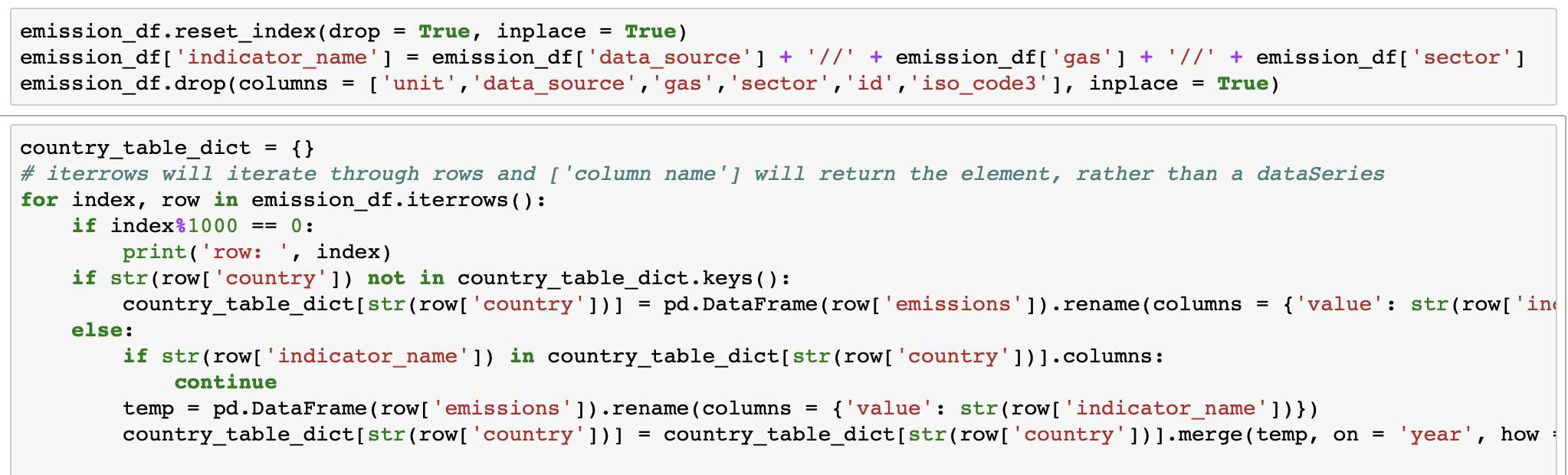
**Step2:** retrieve data through API (cautious: handling pagination), below is an example for emission data function



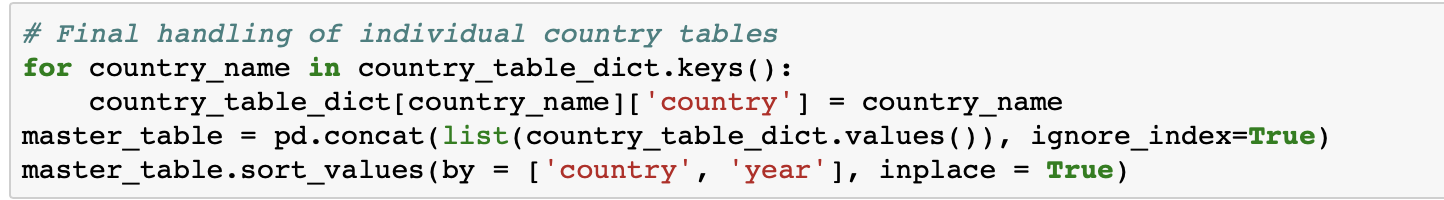
**Step3:**  handling downloaded emission data as it is really ‘messy’, timeseries data are stored into a dictionary which is put into a cell in the DataFrame and each row is an indicator for a country. (this screenshot of table made after deleting uninterested columns)



\*managing columns and turn rows into individual tables and merge in to a table for each country in the country\_table\_dict



\*Final step of handling, concatenating all country tables



And master\_table is what we wanted and will be returned

* **Fred grabber**

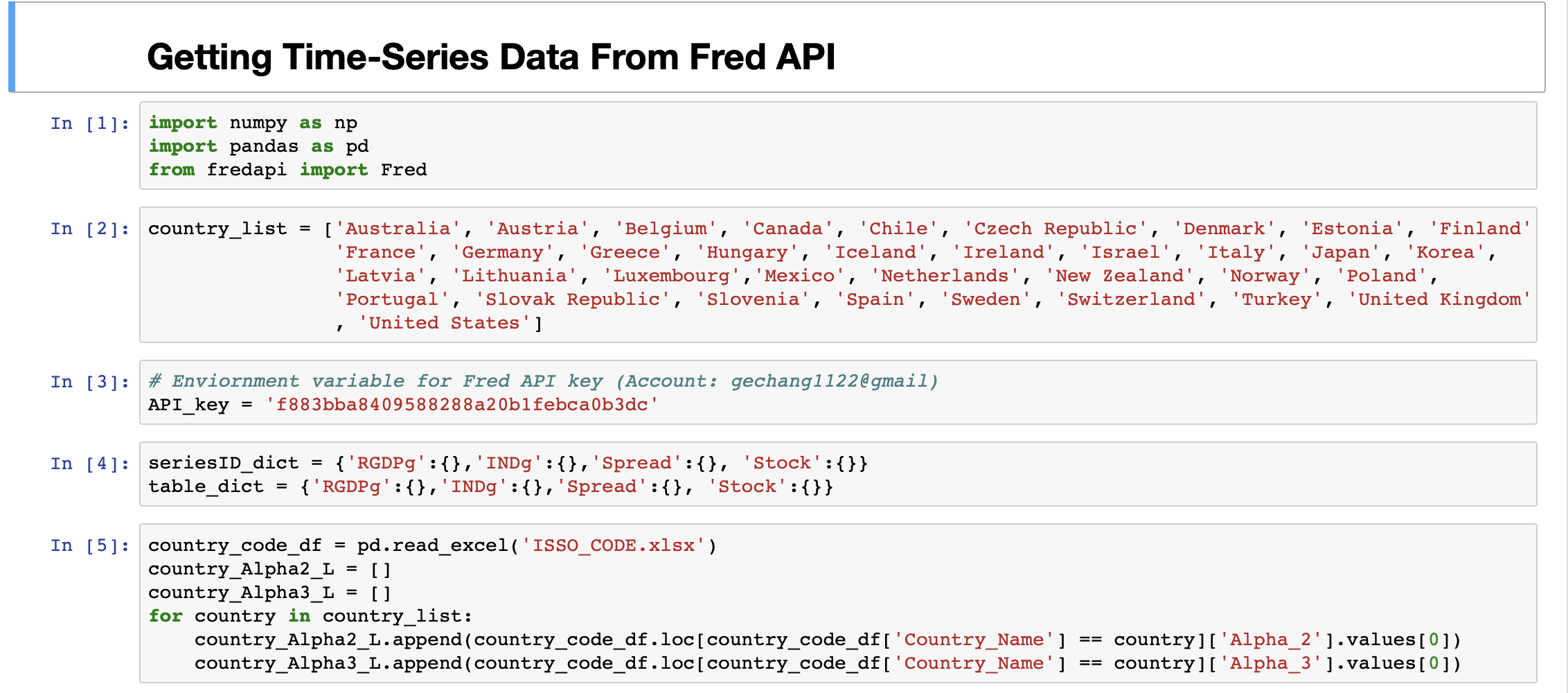
**Prerequisite:** fredapi package from <https://github.com/mortada/fredapi>

**Installation:** type pip install fredapi at terminal/ shell

**Step1:** import libraries and setup environment variables that will be used to help to extract OECD country data from FRED api.

* The api call is different for each data series which is a string that usually consists of ‘series\_name’ + ‘country code’ + ‘frequency’, and we can change the country segment of the string to retrieve the same series for different data

API key can be obtained freely from FRED



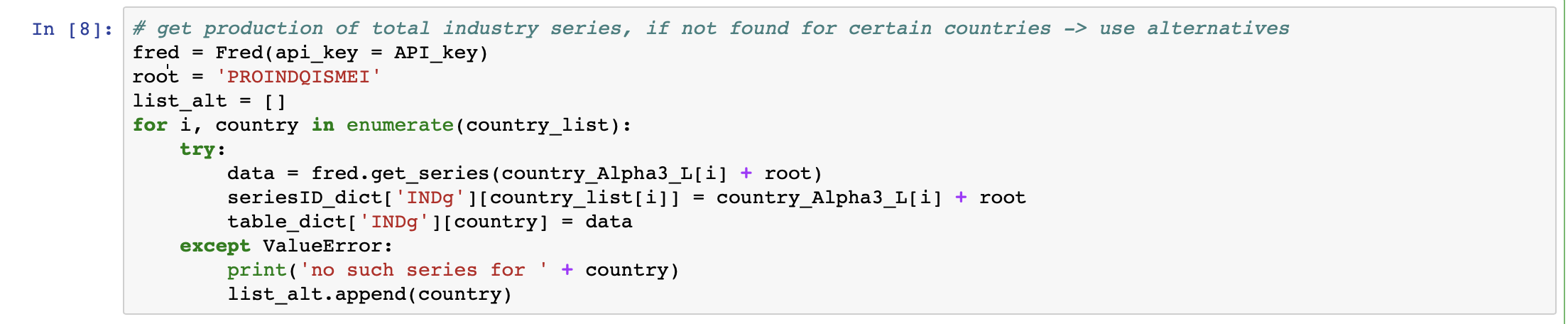
**Step2:** fetch different data series data from Fred API

The first part of the code is to fetch data in consistent fashion, the second part of the code is to hand the country that don’t have the data series we want, and set alternative series number individually for each of them

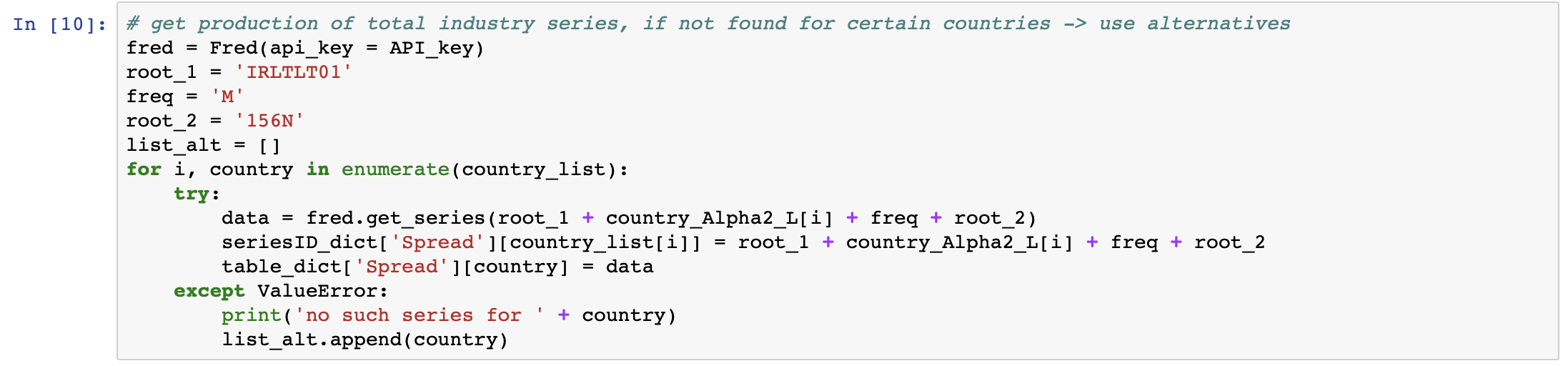
* Real GDP



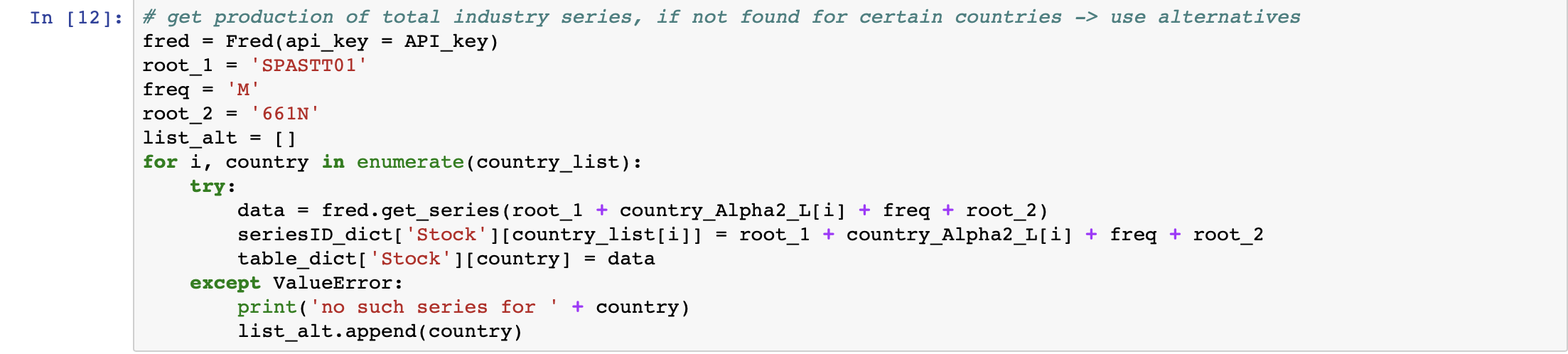
* Production of total industry



* Spread for Long-term Government Bond yields

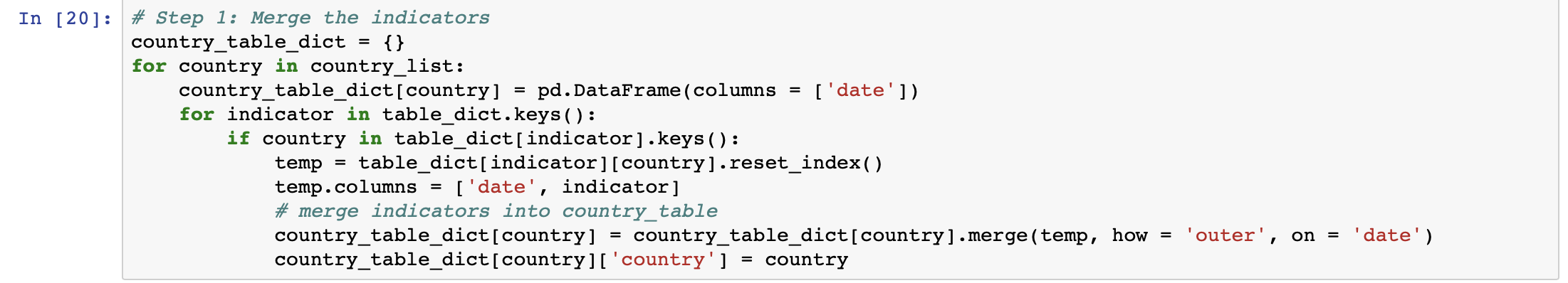


* Total share prices for all shares



**Step3:** Combine all tables in the dictionary (different indicator and different country) in to one single table

1. Merge all indicators into table for each country and save into a new dictionary



1. Concatenate all country tables and create a master table



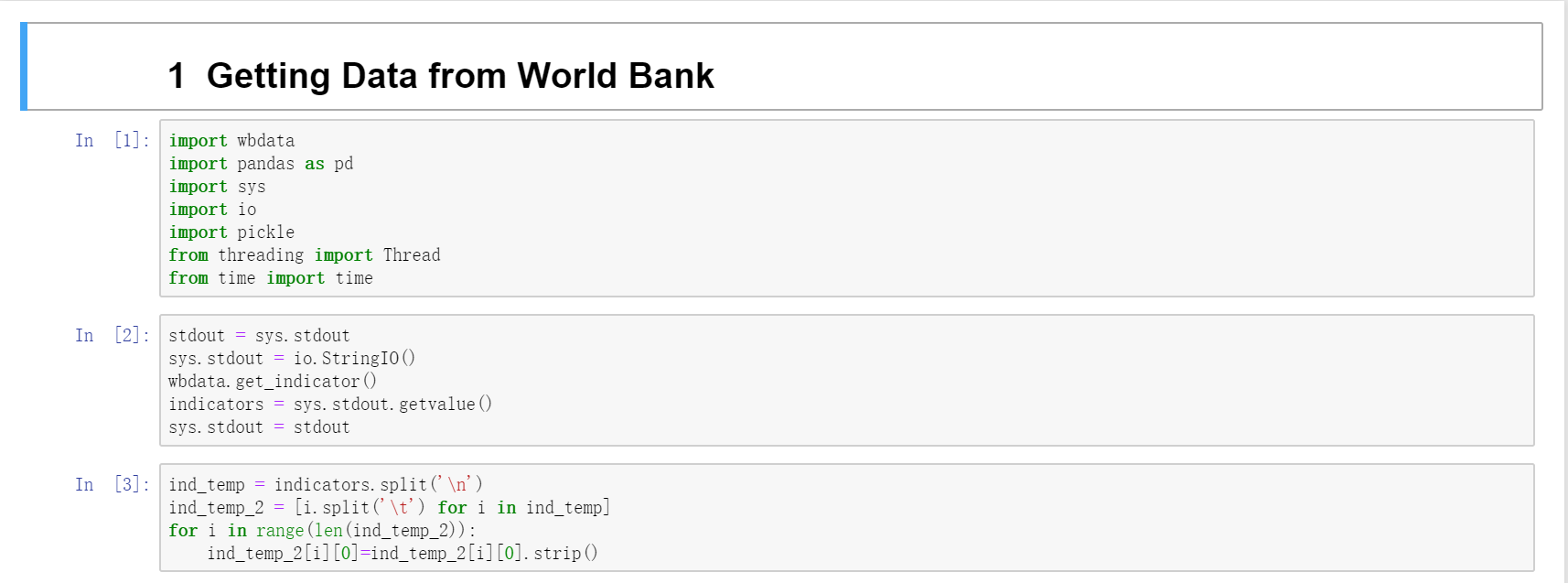
* **WorldBank grabber**

**Prerequisite:** wbdata package from <https://wbdata.readthedocs.io/en/latest/>

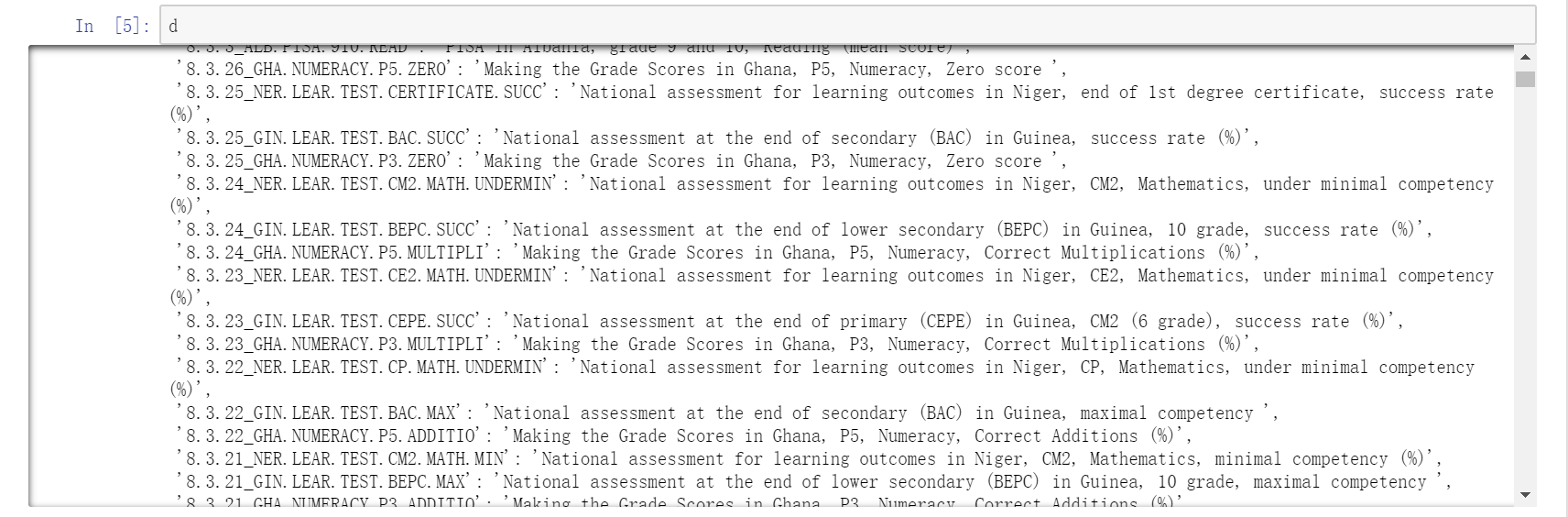
**Installation:** Follow instruction on the website

**Step1:** import libraries and get indicators available in World Bank database

* wbdata get\_indicator() function automatically prints out all available pairs of indicators and their identifiers, the following program stores the content printed out by get\_indicator() and reform it to a dictionary





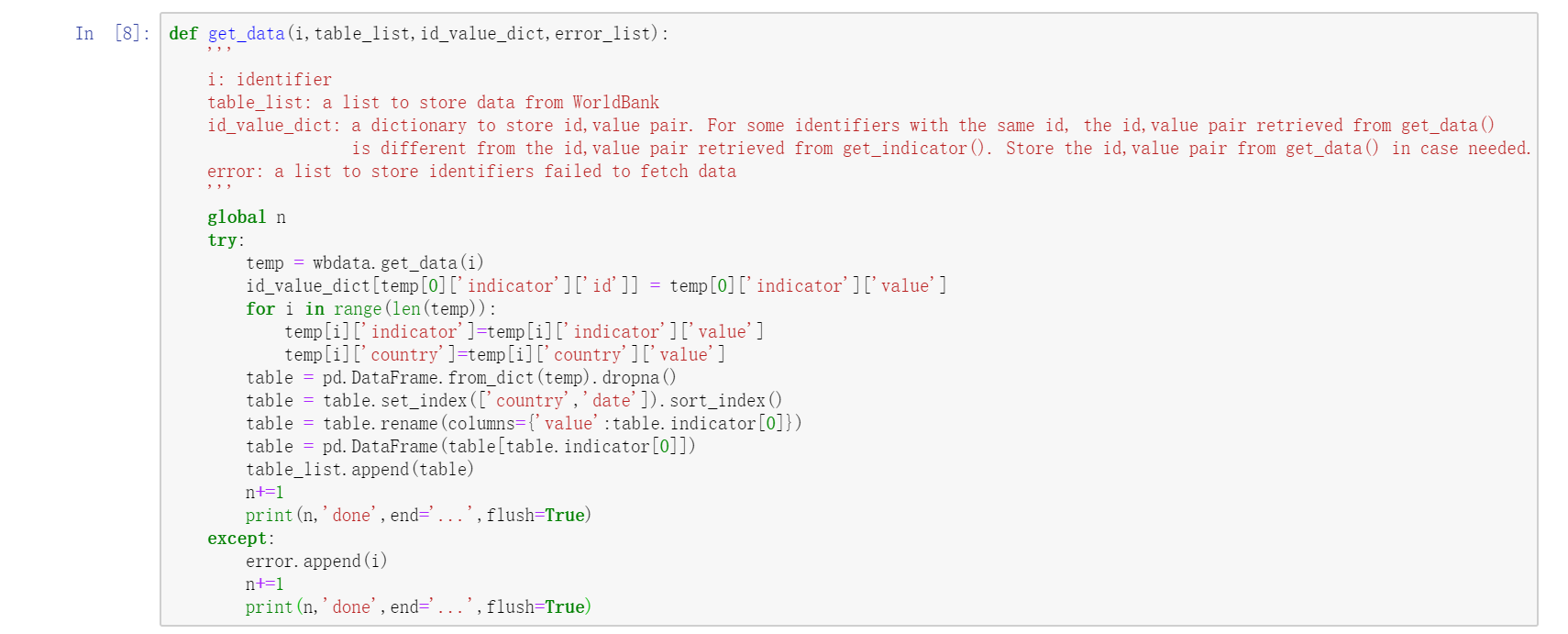


**Step2:** fetch data from World Bank

The following program includes a get\_data function to get data from World Bank and several steps to implement multithreading to call get\_data function. There are roughly 16000 indicators can be got from get\_indicator() function. One thing that needs attention is that the

* get\_data function

This function is written for implementing multithreading. If you want to get data for a single identifier, simply use wbdata.get\_data(). If you don’t want to use multithreading, just add “return table” at the end of try block, and eliminate variables you don’t need.



* Implement multithreading to get data

By the time documenting the file, get\_indicator() retruns 17234 indicators.

One needs attention is that wbdata library sometimes will goes crash if you are continuously fetching data, the program will get killed and you are not able to import this library after that.

One way to solve it is deleting a cache file relating to it. If you are using Google Cloud, there will be a “.cache” file in the directory, type “rm -rf .cache” at the terminal to delete it. Then you can run it again. (you can type “ls -a” or “ls” to check if there is a “.cache” file in your directory)

Due to this reason, the program is fetching data 100 a time. After 100 done, store the data to local driver and begin next round. If program goes crash, just start from where it ended

Recommend to make it as an executable file and run it on Google Cloud whose environment is easier to deal with. To make python script executable on ubuntu system, get all of your code into a text file, add “#!/usr/bin/env python3” at the beginning of your file, this line will let the system to use python3 to run the code in your file. Then type “chmod +x filename” in the terminal to make your file executable. Type “./filename” to run your file.



**Step3:** Merge all indicators into one table



The following two cell will take a long time to run, if you have thousands of tables to merge, especially the second cell.

