Input-Output analysis and modelling with MARIO

Hands-on 8 – Parsing, exploring & aggregating

Please, be aware that all the supporting materials required for this hands-on session is available on Zenodo at the following link: https://doi.org/10.5281/zenodo.8308515

Please use the following citation for:

- **MARIO Software**

Mohammad Amin Tahavori, Lorenzo Rinaldi, & Nicolò Golinucci. (2022). SESAM-Polimi/MARIO: MARIO v0.1.0 (v0.1.0). Zenodo. https://doi.org/10.5281/zenodo.5879382

Learning outcomes

By the end of this exercise, you will learn how to:

1) Parse/export tables as .txt (Excel parsing and exporting are shown in Lecture8, 9, & 10)

2) Aggregate a SUT (IOT case is shown in Lecture 8)

3) Navigate indices and matrices for a SUT (IOT case is shown in Lecture 8)

4) Calculate additional matrices

Important requirement

If your PC is not equipped with at least 8 GB RAM (16 GB recommended!), we suggest you do not parse the table in the first step, while you start by parsing the table after the aggregation!

Please make sure you have Microsoft Excel (or an equivalent alternative) installed on your PC.

Step 0: the Zenodo repository

All the supporting files for this and other Hands-ons and Lectures are available in the Zenodo repository associated to this course.

You find the repository at the following link: XXX

Parsing an EEMRSUT table

The table adopted in this hands-on session is an Environmentally-Extended Multi-Regional Supply and Use Table (EEMRSUT).

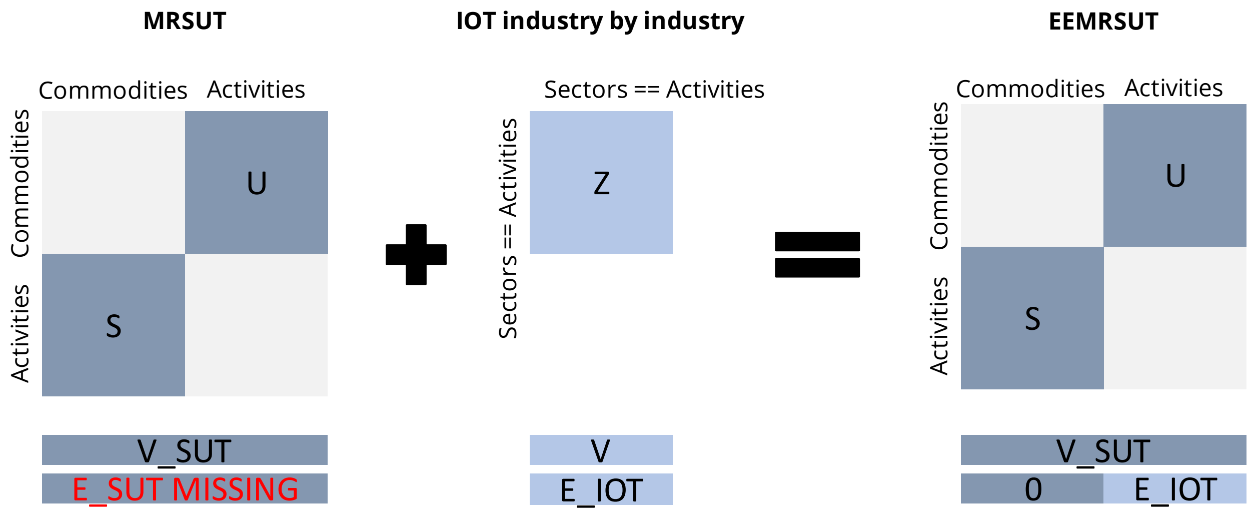
The table is saved into the **Databases\Exiobase - MRSUT_2019_with extensions\flows** of the course Zenodo repository.

In particular it comes from Exiobase 3.8.2 version referred to year 2019 (link to the Exiobase database: https://doi.org/10.5281/zenodo.5589597)

The table is not directly available for download from the Exiobase Zenodo repository, since environmental extensions (also-known-as **satellite accounts**) are available for Exiobase IOT tables and not for SUT tables. However, given the perfect overlapping between Activities in Exiobase MRSUTs and Sectors in Exiobase industry-by-industry IOTs, it is possible to get the satellite accounts (**E matrix**) from the latter and add them to the former, making it an EEMRSUT.

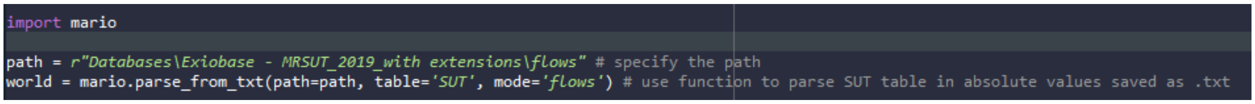
The Figure below represents the process, while this tutorial shows the coding to obtain the table we are adopting from now on by using MARIO.

N.B. satellite accounts are defined only **by activity!** The section of E_SUT "by commodity" **is null**.



Moving to the object of this Hands-on, first of all we start by parsing the table, as follows (please have a look at the **important requirements** section before).

We are using here the parse_from_txt method, which works very similar to the parse_from_excel one: it needs a *path*, in this case directing to a folder **(since MARIO-readable .txt tables are split into multiple files)**, the type of *table* (SUT or IOT) and the *mode* (absolute values, then *flows*, or *coefficients*)

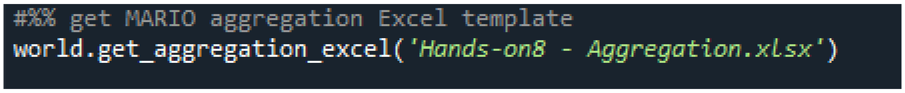


```
import mario

path = r"Databases\Exiobase - MRSUT_2019_with extensions\flows" # specify the path
world = mario.parse_from_txt(path=path, table='SUT', mode='flows') # use function to parse SUT table in absolute values saved as .txt
```

Aggregating the table

In order to better manage this very large table, we proceed performing an aggregation.

First of all, use the 'get_aggregation_excel', to save the MARIO Excel template for aggregation. The path we are passing is "Hands-on8 - Aggregation.xlsx".



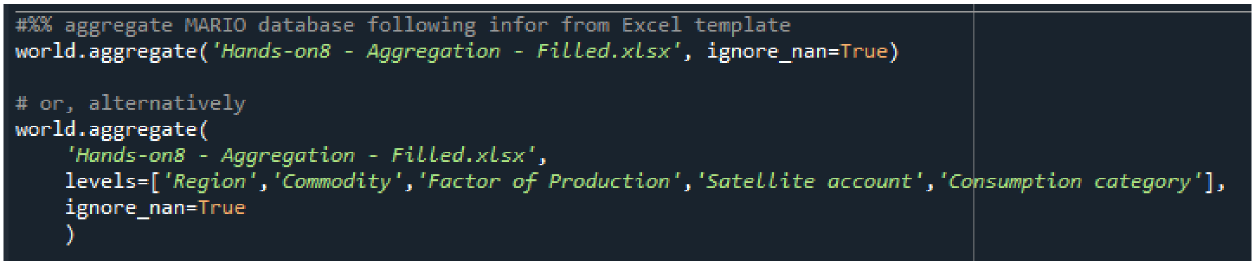
```
#%% get MARIO aggregation Excel template
world.get_aggregation_excel('Hands-on8 - Aggregation.xlsx')
```

Opening the Excel file, we can fill it by aggregating regions, factors of production, consumption categories, satellite accounts and commodities (only "electricity" commodities). We are not aggregating activities.

The "Hands-on8 – Aggregation – Filled.xlsx" file is ready with the desired aggregation, however **we suggest you to do it yourself**.

You can see the aggregation is basically the same we performed in Lecture 8, **however on a SUT instead of on an IOT.**

We can now proceed to aggregate the SUT by reading back the aggregation template once you fill it. You can use **the same method shown in Lecture8** or alternatively, **specify on which level** (or set) you want to perform an aggregation, based on the information you filled in the template.

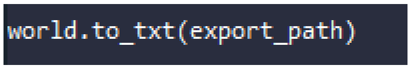


```
#%% aggregate MARIO database following infor from Excel template
world.aggregate('Hands-on8 - Aggregation - Filled.xlsx', ignore_nan=True)

# or, alternatively
world.aggregate(
    'Hands-on8 - Aggregation - Filled.xlsx',
    levels=['Region','Commodity','Factor of Production','Satellite account','Consumption category'],
    ignore_nan=True
    )
```

Exporting a table

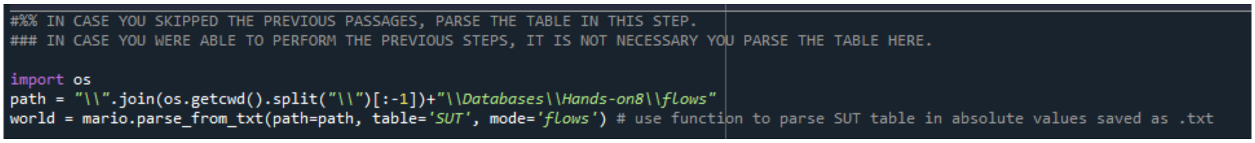
The aggregated table can be now exported.

We decide to export it to a desired path variable we name *export_path* and we are using the '[to_txt](#)' method, to save it in .txt extensions, **since it is much less computationally intensive.**



```
world.to_txt(export_path)
```

In case **you had to skip the previous passages** due to computational reasons, you can start now by parsing the aggregated table as follows (you find the code in the **supporting script** as well):

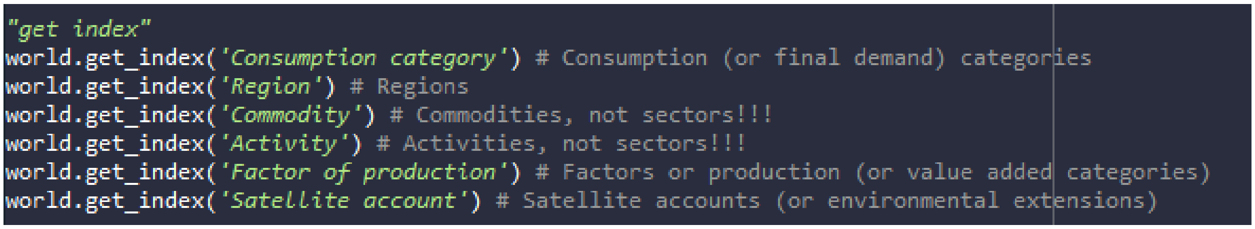


```
#%% IN CASE YOU SKIPPED THE PREVIOUS PASSAGES, PARSE THE TABLE IN THIS STEP.
### IN CASE YOU WERE ABLE TO PERFORM THE PREVIOUS STEPS, IT IS NOT NECESSARY YOU PARSE THE TABLE HERE.

import os
path = "\\".join(os.getcwd().split("\\")[:-1])+"\\Databases\\Hands-on8\\flows"
world = mario.parse_from_txt(path=path, table='SUT', mode='flows') # use function to parse SUT table in absolute values saved as .txt
```

Exploring the table

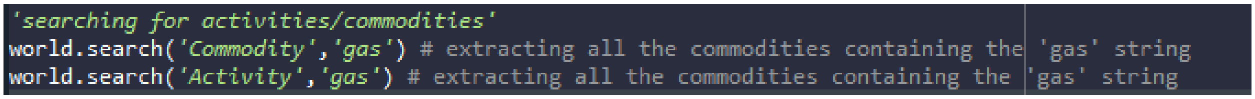
As shown in the Lecture8, you can explore any MARIO Database object by using basic functions such as:

- ‘get_index’, to get a list of any *set* of the table (sectors, regions…)



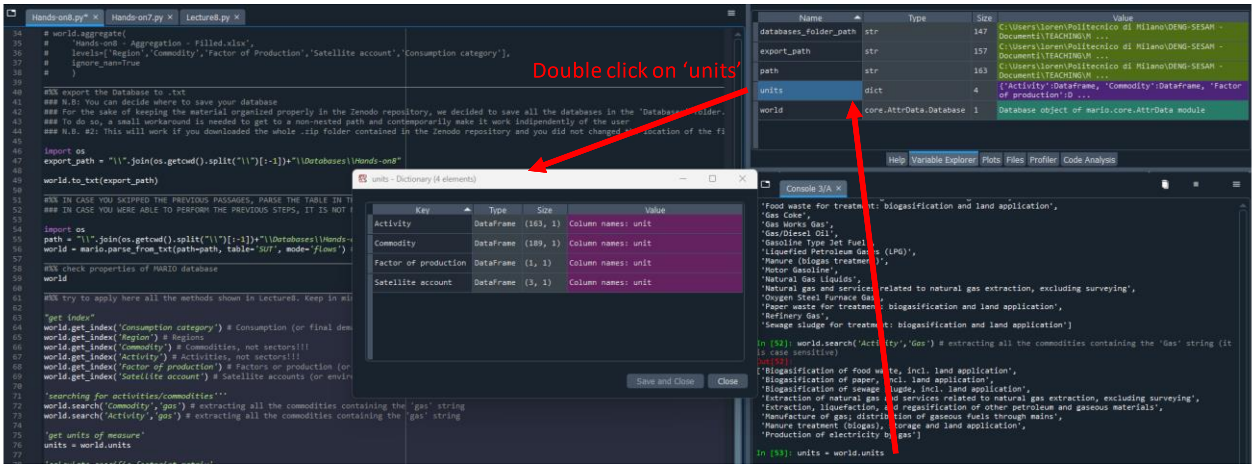
```
"get index"
world.get_index('Consumption category') # Consumption (or final demand) categories
world.get_index('Region') # Regions
world.get_index('Commodity') # Commodities, not sectors!!!
world.get_index('Activity') # Activities, not sectors!!!
world.get_index('Factor of production') # Factors or production (or value added categories)
world.get_index('Satellite account') # Satellite accounts (or environmental extensions)
```

- ‘search’, to search for sets’ labels containing specific strings.



```
'searching for activities/commodities'
world.search('Commodity','gas') # extracting all the commodities containing the 'gas' string
world.search('Activity','gas') # extracting all the commodities containing the 'gas' string
```

- Get accounts’ units of measures by exploring the ‘units’ **dictionary**



Double click on ‘units’

```
# world.aggregate(
#     'Hands-on8 - Aggregation - Filled.xlsx',
#     levels=['Region','Commodity','Factor of Production','Satellite account','Consumption category'],
#     ignore_nan=True
#     )

#%% export the Database to .txt
### N.B: You can decide where to save your database
### For the sake of keeping the material organized properly in the Zenodo repository, we decided to save all the databases in the 'Databases' folder.
### To do so, a small workaround is needed to get to a non-nested path and contemporarily make it work indipendently of the user
### N.B. #2: This will work if you downloaded the whole .zip folder contained in the Zenodo repository and you did not changed the location of the fi

import os
export_path = "\\".join(os.getcwd().split("\\")[:-1])+"\\Databases\\Hands-on8"

world.to_txt(export_path)

#%% IN CASE YOU SKIPPED THE PREVIOUS PASSAGES, PARSE THE TABLE IN TH
### IN CASE YOU WERE ABLE TO PERFORM THE PREVIOUS STEPS, IT IS NOT 

import os
path = "\\".join(os.getcwd().split("\\")[:-1])+"\\Databases\\Hands-
world = mario.parse_from_txt(path=path, table='SUT', mode='flows') 

#%% check properties of MARIO database
world

#%% try to apply here all the methods shown in Lecture8. Keep in mi

"get index"
world.get_index('Consumption category') # Consumption (or final dem
world.get_index('Region') # Regions
world.get_index('Commodity') # Commodities, not sectors!!!
world.get_index('Activity') # Activities, not sectors!!!
world.get_index('Factor of production') # Factors or production (or 
world.get_index('Satellite account') # Satellite accounts (or envir

'searching for activities/commodities'''
world.search('Commodity','gas') # extracting all the commodities containing the 'gas' string
world.search('Activity','gas') # extracting all the commodities containing the 'gas' string

'get units of measure'
units = world.units
```

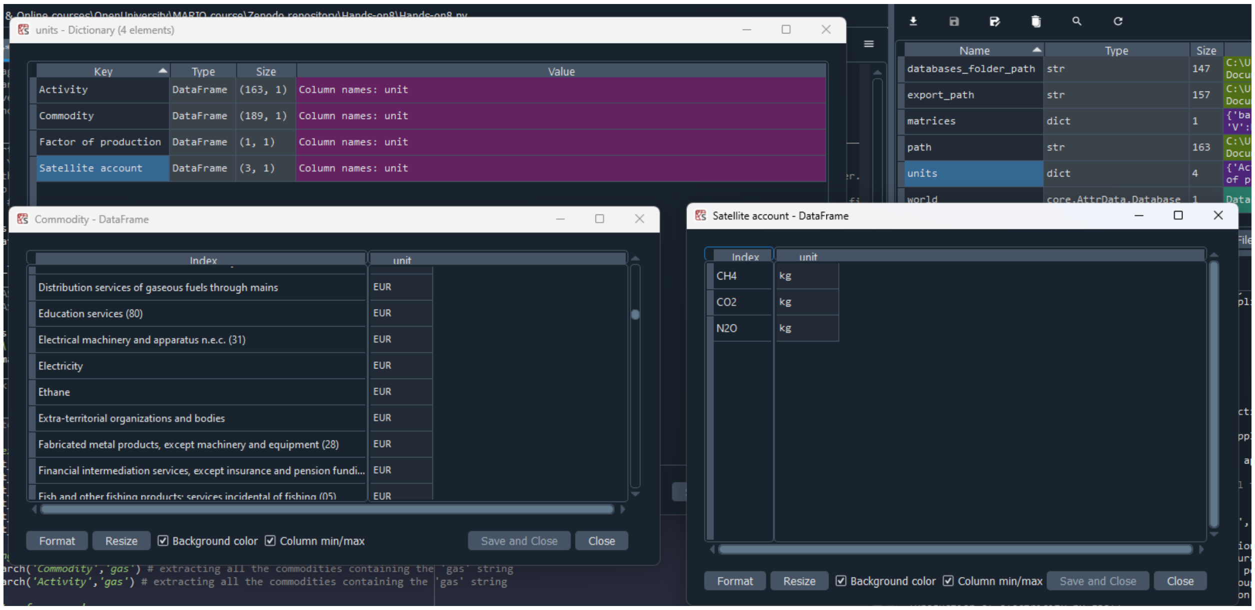
Name	Type	Size	Value
databases_folder_path	str	147	C:\Users\loren\Politecnico di Milano\DENG-SESAM - Documenti\TEACHING\M ...
export_path	str	157	C:\Users\loren\Politecnico di Milano\DENG-SESAM - Documenti\TEACHING\M ...
path	str	163	C:\Users\loren\Politecnico di Milano\DENG-SESAM - Documenti\TEACHING\M ...
units	dict	4	{'Activity':Dataframe, 'Commodity':Dataframe, 'Factor of production':D ...
world	core.AttrData.Database	1	Database object of mario.core.AttrData module

units - Dictionary (4 elements)

Key	Type	Size	Value
Activity	DataFrame	(163, 1)	Column names: unit
Commodity	DataFrame	(189, 1)	Column names: unit
Factor of production	DataFrame	(1, 1)	Column names: unit
Satellite account	DataFrame	(3, 1)	Column names: unit

```
 'Food waste for treatment: biogasification and land application',
 'Gas Coke',
 'Gas Works Gas',
 'Gas/Diesel Oil',
 'Gasoline Type Jet Fuel',
 'Liquefied Petroleum Gases (LPG)',
 'Manure (biogas treatment)',
 'Motor Gasoline',
 'Natural Gas Liquids',
 'Natural gas and services related to natural gas extraction, excluding surveying',
 'Oxygen Steel Furnace Gas',
 'Paper waste for treatment: biogasification and land application',
 'Refinery Gas',
 'Sewage sludge for treatment: biogasification and land application']

In [52]: world.search('Activity','Gas') # extracting all the commodities containing the 'Gas' string (it
is case sensitive)
Out[52]:
['Biogasification of food waste, incl. land application',
 'Biogasification of paper, incl. land application',
 'Biogasification of sewage slugde, incl. land application',
 'Extraction of natural gas and services related to natural gas extraction, excluding surveying',
 'Extraction, liquefaction, and regasification of other petroleum and gaseous materials',
 'Manufacture of gas; distribution of gaseous fuels through mains',
 'Manure treatment (biogas), storage and land application',
 'Production of electricity by gas']

In [53]: units = world.units
```

We can see, for instance, that commodities are measured in EUR, while CO2 in kg, by opening the *Commodity* and *Satellite account* **Pandas DataFrames** respectively.



units - Dictionary (4 elements)

Key	Type	Size	Value
Activity	DataFrame	(163, 1)	Column names: unit
Commodity	DataFrame	(189, 1)	Column names: unit
Factor of production	DataFrame	(1, 1)	Column names: unit
Satellite account	DataFrame	(3, 1)	Column names: unit

Name	Type	Size
databases_folder_path	str	147
export_path	str	157
matrices	dict	1
path	str	163
units	dict	4
world	core.AttrData.Database	1

Commodity - DataFrame

Index	unit
Distribution services of gaseous fuels through mains	EUR
Education services (80)	EUR
Electrical machinery and apparatus n.e.c. (31)	EUR
Electricity	EUR
Ethane	EUR
Extra-territorial organizations and bodies	EUR
Fabricated metal products, except machinery and equipment (28)	EUR
Financial intermediation services, except insurance and pension fundi...	EUR
Fish and other fishing products; services incidental of fishing (05)	EUR

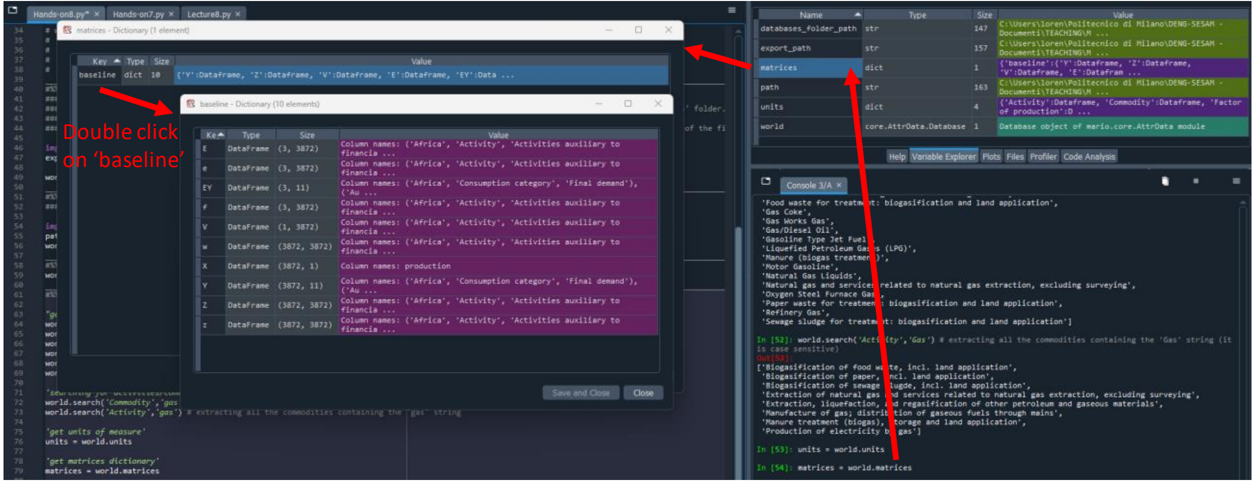
Satellite account - DataFrame

Index	unit
CH4	kg
CO2	kg
N2O	kg

```
arch('Commodity','gas') # extracting all the commodities containing the 'gas' string
arch('Activity','gas') # extracting all the commodities containing the 'gas' string
```

- Explore matrices by getting the 'matrices' **dictionary**

Double click on 'matrices'



Double click on 'baseline'

matrices - Dictionary (1 element)

Key	Type	Size	Value
baseline	dict	10	{'Y':Dataframe, 'Z':Dataframe, 'V':Dataframe, 'E':Dataframe, 'EY':Data ...

baseline - Dictionary (10 elements)

Key	Type	Size	Value
E	DataFrame	(3, 3872)	Column names: ('Africa', 'Activity', 'Activities auxiliary to financia ...
e	DataFrame	(3, 3872)	Column names: ('Africa', 'Activity', 'Activities auxiliary to financia ...
EY	DataFrame	(3, 11)	Column names: ('Africa', 'Consumption category', 'Final demand'), ('Au ...
f	DataFrame	(3, 3872)	Column names: ('Africa', 'Activity', 'Activities auxiliary to financia ...
V	DataFrame	(1, 3872)	Column names: ('Africa', 'Activity', 'Activities auxiliary to financia ...
w	DataFrame	(3872, 3872)	Column names: ('Africa', 'Activity', 'Activities auxiliary to financia ...
X	DataFrame	(3872, 1)	Column names: production
Y	DataFrame	(3872, 11)	Column names: ('Africa', 'Consumption category', 'Final demand'), ('Au ...
Z	DataFrame	(3872, 3872)	Column names: ('Africa', 'Activity', 'Activities auxiliary to financia ...
z	DataFrame	(3872, 3872)	Column names: ('Africa', 'Activity', 'Activities auxiliary to financia ...

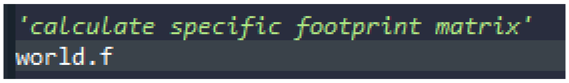
```
world.search('Commodity','gas
world.search('Activity','gas') # extracting all the commodities containing the 'gas' string

'get units of measure'
units = world.units

'get matrices dictionary'
matrices = world.matrices
```

Name	Type	Size	Value
databases_folder_path	str	147	C:\Users\loren\Politecnico di Milano\DENG-SESAM - Documenti\TEACHING\M ...
export_path	str	157	C:\Users\loren\Politecnico di Milano\DENG-SESAM - Documenti\TEACHING\M ...
matrices	dict	1	{'baseline':{'Y':Dataframe, 'Z':Dataframe, 'V':Dataframe, 'E':Datafram ...
path	str	163	C:\Users\loren\Politecnico di Milano\DENG-SESAM - Documenti\TEACHING\M ...
units	dict	4	{'Activity':Dataframe, 'Commodity':Dataframe, 'Factor of production':D ...
world	core.AttrData.Database	1	Database object of mario.core.AttrData module

```
 'Food waste for treatment: biogasification and land application',
 'Gas Coke',
 'Gas Works Gas',
 'Gas/Diesel Oil',
 'Gasoline Type Jet Fuel',
 'Liquefied Petroleum Gases (LPG)',
 'Manure (biogas treatment)',
 'Motor Gasoline',
 'Natural Gas Liquids',
 'Natural gas and services related to natural gas extraction, excluding surveying',
 'Oxygen Steel Furnace Gas',
 'Paper waste for treatment: biogasification and land application',
 'Refinery Gas',
 'Sewage sludge for treatment: biogasification and land application']

In [52]: world.search('Activity','Gas') # extracting all the commodities containing the 'Gas' string (it
is case sensitive)
Out[52]:
['Biogasification of food waste, incl. land application',
 'Biogasification of paper, incl. land application',
 'Biogasification of sewage slugde, incl. land application',
 'Extraction of natural gas and services related to natural gas extraction, excluding surveying',
 'Extraction, liquefaction, and regasification of other petroleum and gaseous materials',
 'Manufacture of gas; distribution of gaseous fuels through mains',
 'Manure treatment (biogas), storage and land application',
 'Production of electricity by gas']

In [53]: units = world.units

In [54]: matrices = world.matrices
```

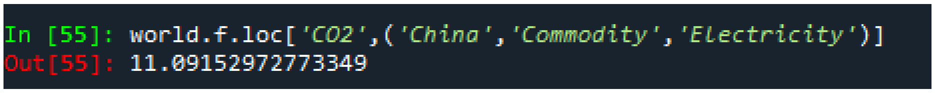
Calculate matrices

As for the IOT case in Lecture 8, it is possible to calculate new matrices, such as the **specific footprint matrix f** by running **world.f**



```
'calculate specific footprint matrix'
world.f
```

It would be interesting to get a footprint value. For example, the specific CO2 footprint of electricity in China (expressed in **kg/EUR**, given the units explored before) is given by the following:



```
In [55]: world.f.loc['CO2',('China','Commodity','Electricity')]
Out[55]: 11.09152972773349
```

To get familiar with this kind of dataframes parsing and slicing, become familiar with Pandas Dataframe.loc function, especially in the case of MultiIndexed Dataframes.

Here the link to a useful guide: https://pandas.pydata.org/docs/user_guide/advanced.html