EEA-datasets-handler

Library which handles the air pollution datasets provided by EEA

Author: Enrico Pittini 877345

Installation

```
pip install EEA-datasets-handler
```

Main usage

```python
import EEA_datasets_handler as eea

# Download the datasets
dest_path = "C:\\Datasets"
countries_cities_dict = {"IT":["Milano","Venezia"],
                         "CY":"all",
                         "AT":["Lienz","Wien"],
                         "GB":["London"]}
pollutants = ["PM10", "PM15"]
years = [2015, 2020]
eea.download_datasets(dest_path, countries_cities_dict,
pollutants, years)

# Load the datasets into a raw pandas DataFrame
source_path = "C:\\Datasets\\EEA"
countries_cities_dict = {"IT":["Milano"]}
pollutants = ["PM10"]
years = [2020]
df = eea.load_datasets(source_path, countries_cities_dict,
pollutants, years)

# Preprocess the DataFrame to time series DataFrames
df_mean, df_min, df_max = eea.preprocessing(df, fill=True)
```

References

- EEA. The European Environment Agency (EEA) is an agency of the European Union, whose task is to provide sound, independent information on the environment.
- pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- requests is an elegant and simple HTTP library for Python, built for human beings.

License

MIT

DESCRIPTION

EEA-datasets-handler is a python library which handles the air pollution datasets provided by EEA.

The two main purposes of this library are the following.

- The first is to allow the user to download the EEA datasets, in an intuitive and easy way.
- The second is to allow the user to process the downloaded datasets into a properly cleaned and prepared pandas DataFrame.

Guiding principles

The functionalities of this library are built in order to have an interface similar to the one of the EEA download service.
The aim of that is to have a set of functionalities which interact with the EEA services from python using the same interface exposed by the EEA itself.
For example, in the EEA interface a user can specify a pollutant using either the numeric notation or the textual one: this feature is kept in the library.

Actually, that interface is even improved. It is made richier and more flexible, in order to facilitate and reduce the user work.
The aim of that is to automate the interaction with the EEA services.
For example, if a user wants to download all the datasets about PM10 in Italy, he doesn't have to download the dataset of each Italian city one at a time. The user can simply specify that he is interested in all the Italian cities.

Finally, this library is built in order to appropriately warn the user. The user is informed each time he specifies an inappropriate value. In addition, he is also warned each time the action requested is not performed correctly or completely.

Functionalities

There are three groups of functionalities.

The first group allows the user to know the EEA supported values, by getting them.
The supported values are the values used by the EEA interface. So, in other words, these are the values that the user is allowed to use in order to specify which air pollution datasets are of interest.
There are three kinds of values.

- Years.
- Pollutants.
- Countries and cities. For each country there is an associated list of cities. Actually, there are also countries with no associated cities.

The second group of functionalities allows the user to filter only the EEA supported values. Given a collection of values, only the supported values are kept.

Finally, the third group of functionalities is the most important. It allows the user to actually download the air pollution datasets in his local storage.
Each downloaded dataset is a csv file. Its name has the following structure:

NationCode_CityName_PollutantId_Year_StationId.csv

where the station is the physical place where the air pollution measurements have been made.
This quadruple is an unique identifier of the dataset.
In addition, an appropriate and straightforward structure of directories is built, in order to keep the datasets well organized. If this structure already exists in the local storage, that structure is used (i.e. a new structure is not created) and the datasets with the same quadruple are overwritten.

Inside the third group there are other important functionalities.
These allow the user to retrieve and delete the downloaded datasets.
In addition, they allow the user to load the downloaded datasets into a single pandas DataFrame. This is a raw DataFrame, since it simply contains the air pollution measurements (i.e. the measurements are not grouped by day).
Finally, these functionalities allow the user to process the loaded pandas DataFrame into a properly cleaned and prepared new DataFrame. This returned DataFrame is a time series DataFrame, since the measurements are grouped by day.

It is important to notice that, while the raw DataFrame contains several useless features (i.e. features which are simply EEA codes and indicators), the time series DataFrame contains only the air pollution concentrations. In other words, the returned DataFrame is ready to be used.

Implementation details

In this section, the most significant implementation details are described.

First of all, it is described how the EEA supported values are mainly represented.
It is important to underline that only the python built-in data structures are used.

- The supported years are represented as a list of integers.
- The supported pollutants are mainly represented as a list of integers, which are the pollutants numeric notations.
  In addition, the supported pollutants are also represented as a dictionary, which maps the pollutants numeric notations into the pollutants textual notations (i.e. it is a map from strings to strings).
- The supported countries and the associated lists of cities are represented as a dictionary, which maps the countries code notations into the lists of cities (i.e.it is a map from strings to lists of strings).
  In addition, the supported countries are also represented as a dictionary, which maps the countries code notations into the countries extended notations (i.e. it is a map from strings to strings).
  Other additional data structures are available, such as the list of supported countries and the list of all the supported cities.

Secondly, the main functionality of the library, i.e. the one which is responsible for downloading the datasets, is built on top of the EEA download service.
This means that, under the hood, the same service used by the EEA users is utilized.

The retrieving of the downloaded datasets is performed in a recursive manner. Given the structure of directories where the datasets have been downloaded, the research is firstly carried out on the parent directory and then it is propagated on each sub-folder.

Finally, the cleaning of the data is mainly performed according to the EEA indications. In fact, the EEA feature “Validity” indicates the reliability of the measurements: a positive value means that the measurement is valid.
In addition, other cleaning operations are performed, in order to guarantee the consistency of the data.

Sources

The supported pollutants have been taken from the official EEA dataset, which contains all the pollutants and the related information.
This dataset has been properly processed, in order to keep only the relevant information and in order to build the data structures mentioned above.

Instead, the supported years have been simply taken from the Web page of the EEA downloading service.

Also the supported countries and the associated lists of supported cities have been simply taken from that page.

Actually, this information is taken from the HTML document and from the associated JavaScript file.
From this information, the data structures mentioned above are built.

DOCUMENTATION

Functions to get the EEA supported values

```
def get_supported_pollutants()
```
Return all the EEA supported pollutants, represented in the numeric notation.

Returns: `list`

```
def get_supported_pollutants_dict()
```
Return the dict that maps the supported-pollutants numeric notations and their associated textual notations.

Returns: `dict`

-[Notes]-
While each pollutant has a different numeric notation, there are pollutants that have the same textual notation. (I.e. duplicate text notation)

```
def get_supported_pollutants_inverse_dict()
```
Return the dict that maps the supported pollutants textual notations and their associated numeric notations.

Returns: `dict`

-[Notes]-
The returned dictionary doesn’t contain all the supported pollutants: the pollutants for which exist at least another pollutant with the same textual-notation are not considered . (I.e. duplicate text notation).

```
def get_supported_years()
```
Return all the EEA supported years.

Returns: `list`

```
def get_supported_countries()
```

Return all the EEA supported countries, represented in the code notation.

Returns: `list`

```
def get_supported_cities()
```

Return all the EEA supported countries.

Returns: `list`

```
def get_supported_countries_dict()
```

Return the dict that maps the supported-countries code notations and their associated extended notations.

Returns: `dict`

-[Notes]-
Each country has both a different code notation and a different extended notation.

```
def get_supported_countries_inverse_dict()
```

Return the dict that maps the supported-countries extended notations and their associated code notations.

Returns: `dict`

```
def get_supported_countries_cities_dict()
```

Return the dict that maps the supported-countries code notations and their associated list of cities.

Returns: `dict`
Dictionary which has as keys the supported countries and as values the associated list of cities.

-[Notes]-
There are countries without supported cities: these countries have simply associated an empty list.

Functions to filter only the EEA supported values

```
def keep_pollutants_supported(pollutants)
```

Keep, from the given pollutants, only the ones supported by EEA.

Parameters:

- `pollutants: list`
  List of pollutants. Each pollutant can be either expressed in the numeric or textual notation.

Returns: `list`
A new list with only the supported pollutants. Each pollutant is expressed in the numeric notation.

Warns: `UserWarning`
When not supported pollutants are given .

-[Notes]-

- `pollutants` can be the string “`all`”: in this case the returned list contains all the supported pollutants.

- In `pollutants`, if a pollutant is expressed with a textual notation that is not unique (i.e. another EEA pollutant has the same textual notation) that pollutant is not considered supported.

```
def keep_years_supported(years)
```

Keep, from the given years, only the ones supported by EEA.

Parameters:

- `years: list`
  List of years.

Returns: `list`
A new list with only the supported years.

Warns: `UserWarning`

When not supported years are given.

```
def keep_countries_cities_supported(countries_cities_dict)
```

Keep, from the dictionary given in input, only the countries and cities supported by EEA.

Parameters:

- `countries_cities_dict: dict`
  Map between countries and list of cities. Each country can be either expressed in the code notation or in the extended notation.

Returns: `dict`
A new dictionary with only the supported countries and cities. Each country is expressed in the code notation.

Warns: `UserWarning`

- When not supported countries are given.
- When not supported cities are given for a certain (supported) country.

-[Notes]-
In `countries_cities_dict` a country can have associated the string "`all`": the returned dictionary will contain all the supported cities for that country. The whole `countries_cities_dict` can be the string "`all`": the returned dictionary will contain all the supported countries and associated cities.

Functions to download and handle the EEA datasets

```
def download_datasets(dest_path, countries_cities_dict,
pollutants, years)
```

Download the selected EEA air pollution datasets in the specified local path.

The EEA datasets are csv files.

Parameters:

- `dest_path : str`
- `countries_cities_dict: dict`
  Map between countries and list of cities. Each country can be either expressed in the code notation or in the extended notation.
- `pollutants : list`
  List of pollutants. Each pollutant can be either expressed in the numeric notation or in

the textual notation.
- `years : list`
  List of years.

Warns: `UserWarning`
- When not supported countries/cities/pollutants/years are given.
- When, for a specified country, no dataset has been downloaded.
- When some problem occurs during an HTTP request. (I.e. a problem during the downloading of certain datasets).

-[Notes]-
In `countries_cities_dict` a country can be associated with the string “`all`”: in this case all the supported cities of that country are taken into account. In addition, the whole `countries_cities_dict` can be the string “`all`”: in this case all the supported countries and associated cities are considered. Also `pollutants` can be “`all`”, which means that all the supported pollutants are taken into account.

```
def retrieve_datasets(source_path, countries_cities_dict,
pollutants, years)
```
Retrieve the selected EEA air pollution datasets from the local storage.

The EEA datasets are csv files.

Parameters:
- `source_path : str`
  Local path in which the selected datasets are searched.

- `countries_cities_dict : dict`
  Map between countries and list of cities. Each country can be either expressed in the code notation or in the extended notation.
- `pollutants : list`
  List of pollutants. Each pollutant can be either expressed in the numeric notation or in the textual notation.
- `years : list`
  List of years.

Returns: `list`
List of the retrieved datasets. Each dataset is represented as a string (i.e. his local path).

Warns: `UserWarning`
- When not supported countries/pollutants/years are given.
- When, for a specified country, no dataset has been found.

-[Notes]-

In `countries_cities_dict` a country can be associated with the string “`all`”: in this case all the supported cities of that country are taken into account. In addition, the whole `countries_cities_dict` can be the string “`all`”: in this case all the supported countries and associated cities are considered. Also `pollutants` can be “`all`”, which means that all the supported pollutants are taken into account.

```
def remove_datasets(source_path, countries_cities_dict,
pollutants, years)
```

Delete the selected EEA air pollution datasets from the local storage. (The EEA datasets are csv files).

Parameters:

- `source_path : str`
  Local path from which the selected datasets are removed.
- `countries_cities_dict : dict`
  Map between countries and list of cities. Each country can be either expressed in the code notation or in the extended notation.
- `pollutants : list`
  List of pollutants. Each pollutant can be either expressed in the numeric notation or in the textual notation.
- `years : list`
  List of years.

Warns: `UserWarning`

- When not supported countries/pollutants/years are given.
- When, for a specified country, no dataset has been found.

-[Notes]-

In `countries_cities_dict` a country can be associated with the string “`all`”: in this case all the supported cities of that country are taken into account. In addition, the whole `countries_cities_dict` can be the string “`all`”: in this case all the supported countries and associated cities are considered. Also `pollutants` can be “`all`”, which means that all the supported pollutants are taken into account.

```
def load_datasets(source_path, countries_cities_dict, pollutants,
years)
```

Load the selected EEA air pollution datasets, retrieved from the local storage, into a single pandas DataFrame. (The EEA datasets are csv files).

The returned DataFrame is a raw DataFrame. This means two things.

1. The DataFrame simply contains air pollution concentration measurements, which are not properly grouped by their days.

2. The values in the DataFrame have not been cleaned.

Parameters:
- `source_path : str`
  Local path in which the selected datasets are searched.
- `countries_cities_dict : dict`
  Map between countries and list of cities. Each country can be either expressed in the code notation or in the extended notation.
- `pollutants : list`
  List of pollutants. Each pollutant can be either expressed in the numeric notation or in the textual notation.
- `years : list`
  List of years.

Returns: `pd.DataFrame`
DataFrame containing all the selected datasets.

Warns: `UserWarning`
- When not supported countries/pollutants/years are given.
- When, for a specified country, no dataset has been found.
- When no dataset at all has been found.

-[Notes]-
In `countries_cities_dict` a country can be associated with the string “`all`”: in this case all the supported cities of that country are taken into account. In addition, the whole `countries_cities_dict` can be the string “`all`”: in this case all the supported countries and associated cities are considered. Also `pollutants` can be “`all`”, which means that all the supported pollutants are taken into account.

```
def preprocessing(df, fill=True, fill_n_days=10, fill_aggr="mean")
```

Prepare and clean the given raw EEA DataFrame, grouping the air pollution concentration measurements by day.

Return three DataFrames. All of these DataFrames are indexed by days and all of them have only one column. But:
- the first DataFrame contains, for each day, the daily mean concentration;
- the second contains, for each day, the daily min concentration;
- the third contains, for each day, the daily max concentration.

Parameters:
- `df : pd.DataFrame`
  The DataFrame to clean. It’s a raw EEA DataFrame, loaded using the `load_datasets` function.
- `fill : bool`
  If `True`, the missing days (i.e. the days in `df` without any measurement) are filled.

Otherwise, they remain with a missing value. (In all the three returned DataFrame).
- `fill_n_days : int` or `str`
  The number of preceding days, contained in `df`, used to fill a missing day. `fill_n_days` can be either an integer or the string `"all"`: in the latter case all the preceding days in `df` are used to fill the missing days.
- `fill_aggr : str`
  The statistic aggregation used to fill a missing day. It can be either `"mean"` or `"min"` or `"max"`.

Returns:
- `pd.DataFrame`
  The prepared and cleaned DataFrame, containing the daily mean concentrations.
- `pd.DataFrame`
  The prepared and cleaned DataFrame, containing the daily min concentrations.
- `pd.DataFrame`
  The prepared and cleaned DataFrame, containing the daily max concentrations.

Warns: `UserWarning`
  When missing days are contained in `df`.

-[Notes]-
- If `fill` is `True`, the missing days are filled computing the aggregation `fill_aggr` on the `fill_n_days` days in `df` preceding the missing day. (In all the three returned DataFrame).
  Moreover, the missing days for which no preceding day has been found in `df` are deleted from the returned DataFrames. These days are surely the first days in `df`.
- The returned DataFrames are indexed by days. In particular, the pandas built-in types are used: the index type is `pd.DatetimeIndex`.

EXAMPLES

```
>>> import EEA_datasets_handler as eea
```

Functions the get the EEA supported values

```
>>> eea.get_supported_pollutants()
[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
36, 37, 38, 39, 40, 41, 42, 45, 46, 47, 48, 49, 50, 51, 52, 53,
…]
```

```
>>> eea.get_supported_pollutants_dict()
{1: 'SO2', 3: 'SA', 4: 'SPM', 5: 'PM10', 6: 'BS', 7: 'O3', 8:
'NO2', 9: 'NOX as NO2', 10: 'CO', 11: 'H2S', 12: 'Pb', 13: 'Hg',
14: 'Cd', 15: 'Ni', 16: 'Cr', 17: 'Mn', 18: 'As', 19: 'CS2', 20:
'C6H6', 21: 'C6H5-CH3', 22: 'C6H5-CH=CH2', 23: 'CH2=CH-CN', 24:
'CH2=CH-CH=CH2', 25: 'HCHO', 26: 'CHCl=CCl2', 27: 'C2Cl4',
…}
```

```
>>> eea.get_supported_pollutants_inverse_dict()
{'SO2': 1, 'CO': 10, 'Pb in PM2.5': 1012, 'Hg in PM2.5': 1013, 'Cd
in PM2.5': 1014, 'Ni in PM2.5': 1015, 'Cr in PM2.5': 1016, 'Mn in
PM2.5': 1017, 'As in PM2.5': 1018, 'NH4+ in PM2.5': 1045, 'NO3- in
PM2.5': 1046, 'SO42- in PM2.5': 1047, 'Se in PM2.5': 1048, 'V in
PM2.5': 1049, 'Zn in PM2.5': 1063, 'Co in PM2.5': 1064, 'Fe in
PM2.5': 1065, 'Cu in PM2.5': 1073, 'H2S': 11,
…}
```

```
>>> eea.get_supported_years()
[2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021]
```

```
>>> eea.get_supported_countries()
['AT', 'BE', 'BG', 'CH', 'CY', 'CZ', 'DE', 'DK', 'EE', 'ES', 'FI',
'FR', 'GB', 'GR', 'HR', 'HU', 'IE', 'IS', 'IT', 'LT', 'LU', 'LV',
'MT', 'NL', 'NO', 'PL', 'PT', 'RO', 'SE', 'SI', 'SK', 'AD', 'AL',
'BA', 'GI', 'ME', 'MK', 'RS', 'TR', 'XK']
```

```
>>> eea.get_supported_cities()
['Graz', 'Innsbruck', 'Klagenfurt', 'Linz', 'Salzburg', 'Wien',
'Antwerpen', 'Brugge', 'Bruxelles / Brussel', 'Charleroi', 'Gent',
'Kortrijk', 'Leuven', 'Liège', 'Mons', 'Namur', 'Blagoevgrad',
'Burgas', 'Dobrich', 'Haskovo', 'Pazardzhik', 'Pernik', 'Pleven',
'Plovdiv', 'Ruse', 'Shumen', 'Sliven', 'Sofia', 'Stara Zagora',
'Varna', 'Veliko Tarnovo', 'Vidin', 'Vratsa', 'Basel', 'Bern',
…]
```

```
>>> eea.get_supported_countries_dict()
{'AD': 'Andorra', 'AL': 'Albania', 'AT': 'Austria', 'BA': 'Bosnia
and Herzegovina', 'BE': 'Belgium', 'BG': 'Bulgaria', 'CH':
'Switzerland', 'CY': 'Cypern', 'CZ': 'Czech Republic', 'DE':
'Germany', 'DK': 'Denmark', 'EE': 'Estonia', 'ES': 'Spain', 'FI':
'Finland', 'FR': 'France', 'GB': 'United Kingdom', 'GI':
'Gibraltar', 'GR': 'Greece', 'HR': 'Croatia', 'HU': 'Hungary',
'IE': 'Ireland', 'IS': 'Island', 'IT': 'Italy', 'LT': 'Lithuania',
'LU': 'Luxembourg', 'LV': 'Latvia', 'ME': 'Montenegro', 'MK':
'former Yogoslav Republic of Macedonia, the', 'MT': 'Malta', 'NL':
'Netherlands', 'NO': 'Norway', 'PL': 'Poland', 'PT': 'Portugal',
'RO': 'Romania', 'RS': 'Serbia', 'SE': 'Sweden', 'SI': 'Slovenia',
'SK': 'Slovakia', 'TR': 'Turkey', 'XK': 'Kosovo'}
```

```
>>> eea.get_supported_countries_inverse_dict()
{'Andorra': 'AD', 'Albania': 'AL', 'Austria': 'AT', 'Bosnia and
Herzegovina': 'BA', 'Belgium': 'BE', 'Bulgaria': 'BG',
'Switzerland': 'CH', 'Cypern': 'CY', 'Czech Republic': 'CZ',
'Germany': 'DE', 'Denmark': 'DK', 'Estonia': 'EE', 'Spain': 'ES',
'Finland': 'FI', 'France': 'FR', 'United Kingdom': 'GB',
'Gibraltar': 'GI', 'Greece': 'GR', 'Croatia': 'HR', 'Hungary':
'HU', 'Ireland': 'IE', 'Island': 'IS', 'Italy': 'IT', 'Lithuania':
'LT', 'Luxembourg': 'LU', 'Latvia': 'LV', 'Montenegro': 'ME',
'former Yogoslav Republic of Macedonia, the': 'MK', 'Malta': 'MT',
'Netherlands': 'NL', 'Norway': 'NO', 'Poland': 'PL', 'Portugal':
'PT', 'Romania': 'RO', 'Serbia': 'RS', 'Sweden': 'SE', 'Slovenia':
'SI', 'Slovakia': 'SK', 'Turkey': 'TR', 'Kosovo': 'XK'}
```

```
>>> eea.get_supported_countries_cities_dict()
{'AT': ['Graz', 'Innsbruck', 'Klagenfurt', 'Linz', 'Salzburg',
'Wien'], 'BE': ['Antwerpen', 'Brugge', 'Bruxelles / Brussel',
'Charleroi', 'Gent', 'Kortrijk', 'Leuven', 'Liège', 'Mons',
'Namur'], 'BG': ['Blagoevgrad', 'Burgas', 'Dobrich', 'Haskovo',
'Pazardzhik', 'Pernik', 'Pleven', 'Plovdiv', 'Ruse', 'Shumen',
'Sliven', 'Sofia', 'Stara Zagora', 'Varna', 'Veliko Tarnovo',
'Vidin', 'Vratsa'], 'CH': ['Basel', 'Bern', 'Genève', 'Lausanne',
'Lugano', 'Luzern', 'St. Gallen', 'Winterthur', 'Zürich'], 'CY':
['Lefkosia', 'Lemesos'], 'CZ': ['Brno', 'Ceské Budejovice',
'Chomutov-Jirkov', 'Havírov', 'Hradec Králové', 'Jihlava',
'Karlovy Vary', 'Karviná', 'Kladno', 'Liberec', 'Most', 'Olomouc',
'Ostrava', 'Pardubice', 'Plzen', 'Praha', 'Ústí nad Labem',
'Zlín'],
…}
```

Functions to filter only the EEA supported values

A warning is given for each not supported value.
The values are also sorted and the duplicates are deleted.

```
>>> pollutants = ["PM10", 5, "SO", "As in PM10", "C6H6", 112, "KK",
5018]  #Either numeric notation or textual notation
>>> eea.keep_pollutants_supported(pollutants)
UserWarning: The pollutants ['SO', 112, 'KK'] are not supported by
EEA
[5, 5018, 20]
```

PM10 has 5 as numeric notation. C6H6 is 5018. And As in PM10 is 20.

`pollutants` can be the string `"all"`: in this case all the pollutants are taken into account.

```
>>> pollutants = "all"
>>> eea.keep_pollutants_supported(pollutants)
[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
36, 37, 38, 39, 40, 41, 42, 45, 46, 47, 48, 49, 50, 51, 52, 53,
…]
```

```
>>> years = [1342, 2345, 2333, 2015, 2019, 2015]
>>> eea.keep_years_supported(years)
UserWarning: The years [1342, 2345, 2333] are not supported by EEA
```

```
[2015, 2019]
```

```
>>> countries_cities_dict = {
"IT": ["Milano", "Bergamo", "FAKE_CITY", "Brescia", "Como",
"Cremona", "Lecco", "Pavia", "Varese"],
"CY": ["FAKE_CITY"],
"AT": ["Lienz","FAKE_CITY","Wien"],
"FAKE_COUNTRY": ["Moscow"]
}
>>> eea.keep_countries_cities_supported(countries_cities_dict)
UserWarning: The countries ['FAKE_COUNTRY'] are not supported by
EEA
UserWarning: The cities ['FAKE_CITY'] are not supported by EEA for
the country IT
UserWarning: The cities ['FAKE_CITY'] are not supported by EEA for
the country CY
UserWarning: The cities ['FAKE_CITY'] are not supported by EEA for
the country AT
{'IT': ['Milano', 'Bergamo', 'Brescia', 'Como', 'Cremona',
'Lecco', 'Pavia', 'Varese'], 'CY': [], 'AT': [‘Linz’, 'Wien']}
```

```
# Either code notation or extended notation
>>> countries_cities_dict = {
"Italy": ["Milano", "Bergamo", "FAKE_CITY", "Brescia", "Como",
"Cremona", "Lecco", "Pavia", "Varese"],
“CY": ["FAKE_CITY"],
"AT": ["Lienz", "FAKE_CITY", "Wien"]
}
>>> eea.keep_countries_cities_supported(countries_cities_dict)
UserWarning: The cities ['FAKE_CITY'] are not supported by EEA for
the country CY
UserWarning: The cities ['FAKE_CITY'] are not supported by EEA for
the country AT
UserWarning: The cities ['FAKE_CITY'] are not supported by EEA for
the country IT
{'CY': [], 'AT': ['Linz', 'Wien'], 'IT': ['Milano', 'Bergamo',
'Brescia', 'Como', 'Cremona', 'Lecco', 'Pavia', 'Varese']}
```

```
# Each country can be associated with the string "all": in this
case all its cities are taken into account.
>>> countries_cities_dict = {
"Italy": ["Bergamo"],
"CY": "all",
"AT": ["Wien"]
}
>>> eea.keep_countries_cities_supported(countries_cities_dict)
{'CY': ['Lefkosia', 'Lemesos'], 'AT': ['Wien'], 'IT': ['Bergamo']}
```

```
# The whole countries_cities_dict can be "all": all the countries
and cities are taken into account.
>>> countries_cities_dict = "all"
>>> eea.keep_countries_cities_supported(countries_cities_dict)
{'AT': ['Graz', 'Innsbruck', 'Klagenfurt', 'Linz', 'Salzburg',
'Wien'], 'BE': ['Antwerpen', 'Brugge', 'Bruxelles / Brussel',
'Charleroi', 'Gent', 'Kortrijk', 'Leuven', 'Liège', 'Mons',
'Namur'], 'BG': ['Blagoevgrad', 'Burgas', 'Dobrich', 'Haskovo',
'Pazardzhik', 'Pernik', 'Pleven', 'Plovdiv', 'Ruse', 'Shumen',
'Sliven', 'Sofia', 'Stara Zagora', 'Varna', 'Veliko Tarnovo',
'Vidin', 'Vratsa'], 'CH': ['Basel', 'Bern', 'Genève', 'Lausanne',
'Lugano', 'Luzern', 'St. Gallen', 'Winterthur', 'Zürich'], 'CY':
['Lefkosia', 'Lemesos'], 'CZ': ['Brno', 'Ceské Budejovice',
'Chomutov-Jirkov', 'Havírov', 'Hradec Králové', 'Jihlava',
'Karlovy Vary', 'Karviná', 'Kladno', 'Liberec', 'Most', 'Olomouc',
'Ostrava', 'Pardubice', 'Plzen', 'Praha', 'Ústí nad Labem',
'Zlín'],
…}
```

Download datasets

```
>>> dest_path = "C:\\Datasets"
>>> countries_cities_dict = {"IT":["Milano","Venezia"],
                             "AT":["Linz","Wien"],
                             "GB":["London"]}
>>> pollutants = ["PM10"]
>>> years = [2020]
>>> eea.download_datasets(dest_path, countries_cities_dict,
pollutants, years)
```

The datasets are *csv* files.
A dataset name has the following structure:
NationCode_CityName_PollutantId_Year_StationId.csv.
A hierarchy of directories is created in the specified path.

```
Datasets
+-- EEA
    +-- AT
    |   +-- Linz
    |   |   +-- AT_Linz_5_2020_49647.csv
    |   |   +-- AT_Linz_5_2020_49682.csv
    |   |   +-- AT_Linz_5_2020_49899.csv
    |   |   +-- AT_Linz_5_2020_53269.csv
    |   +-- Wien
    |       +-- AT_Linz_5_2020_49027.csv
    |       +-- AT_Linz_5_2020_49028.csv
    |       +-- AT_Linz_5_2020_49029.csv
    |       +-- AT_Linz_5_2020_49398.csv
    |       +-- AT_Linz_5_2020_49400.csv
    |       +-- AT_Linz_5_2020_49402.csv
    |       +-- AT_Linz_5_2020_49451.csv
    |       +-- AT_Linz_5_2020_49452.csv
    |       +-- AT_Linz_5_2020_49736.csv
    |       +-- AT_Linz_5_2020_49737.csv
    |       +-- AT_Linz_5_2020_49809.csv
    |       +-- AT_Linz_5_2020_49912.csv
    |       +-- AT_Linz_5_2020_63486.csv
    +-- GB
    |   +-- London
    |       +-- GB_London_5_2020_21274.csv
    |       +-- GB_London_5_2020_21877.csv
    |       +-- GB_London_5_2020_22642.csv
    |       +-- GB_London_5_2020_22665.csv
    |       +-- GB_London_5_2020_68646.csv
    +-- IT
        +-- Milano
        |   +-- IT_Milano_5_2020_24044.csv
        |   +-- IT_Milano_5_2020_24089.csv
        |   +-- IT_Milano_5_2020_24293.csv
        |   +-- IT_Milano_5_2020_24321.csv
        |   +-- IT_Milano_5_2020_24779.csv
        |   +-- IT_Milano_5_2020_25228.csv
        |   +-- IT_Milano_5_2020_25606.csv
        |   +-- IT_Milano_5_2020_67851.csv
        |   +-- IT_Milano_5_2020_69646.csv
        +-- Venezia
        |   +-- IT_Venezia_5_2020_25052.csv
        |   +-- IT_Venezia_5_2020_26555.csv
        |   +-- IT_Venezia_5_2020_61894.csv
```

If that structure of directories already exists in the specified path, the structure already present is used (i.e. a new structure is not created).
If a dataset already exists, it is overwritten.

A warning is given for each not supported value, like in the previous functions.

```
>>> dest_path = "C:\\Datasets"
>>> countries_cities_dict = {"IT": ["Milano", "FAKE_CITY"],
                             "FAKE_COUNTRY": ["London"]}
>>> pollutants = ["PM10", "SKK"] # SKK is not supported
>>> years = [2020, 2012, 2019] # 2012 is not supported
>>> eea.download_datasets(dest_path, countries_cities_dict,
pollutants, years)
UserWarning: The pollutants ['SKK'] are not supported by EEA
UserWarning: The years [2012] are not supported by EEA
UserWarning: The countries ['FAKE_COUNTRY'] are not supported by
EEA
UserWarning: The cities ['FAKE_CITY'] are not supported by EEA for
the country IT
```

Like in the previous functions, the string `"all"` *can be used .*

```
>>> dest_path = "C:\\Datasets"
>>> countries_cities_dict = {"IT": "all", # All the italian
supported cities
                             "GB": ["London"]}
>>> pollutants = "all" # All the pollutants
>>> years = [2020]
>>> eea.download_datasets(dest_path, countries_cities_dict,
pollutants, years)
```

```
>>> dest_path = "C:\\Datasets"
>>> countries_cities_dict = "all" # All the countries and cities
>>> pollutants = ["PM10"]
>>> years = [2020]
>>> eea.download_datasets(dest_path, countries_cities_dict,
pollutants, years)
```

Like in the previous functions, the code notations and the extended ones can be both used. Both for the countries and for the pollutants.

```
>>> dest_path = "C:\\Datasets"
>>> countries_cities_dict = {"Italy": ["Milano", "Venezia"],
                             "GB": ["London"]}
>>> pollutants = [5, "PM10"]
>>> years = [2020]
>>> eea.download_datasets(dest_path, countries_cities_dict,
pollutants, years)
```

A warning is given also if no dataset has been found for a certain country.

```
>>> dest_path = "C:\\Datasets"
>>> countries_cities_dict = {"Italy":
["Milano","Bergamo","Varese"],
                         "CY": "all",
                         "AT": ["Linz","Wien"]}
>>> pollutants = ["PM10"]
>>> years = [2020]
>>> eea.download_datasets(dest_path, countries_cities_dict,
pollutants, years)
UserWarning: Data have not been found for the country CY for the
pollutants [5] for the years [2020]
```

Retrieve datasets

The input parameters have the same semantics as the previous functions: both code and extended notations can be used (for countries and pollutants); the "all" string can be used. In addition, a warning is given for each not supported value.

```
>>> source_path = "C:\\Datasets\\EEA"
>>> countries_cities_dict = {"IT": ["Milano"],
                             "GB": ["London"]}
>>> pollutants = ["C6H6", "PM10", "SO"] # SO is not supported
>>> years = [2020, 2019]
>>> files = eea.retrieve_datasets(source_path,
countries_cities_dict, pollutants, years)
```

```
>>> print(files)
UserWarning: The pollutants ['SO'] are not supported by EEA
['C:\\Datasets\\EEA\\GB\\London\\GB_London_5_2020_21274.csv',
'C:\\Datasets\\EEA\\GB\\London\\GB_London_5_2020_21887.csv',
'C:\\Datasets\\EEA\\GB\\London\\GB_London_5_2020_22642.csv',
'C:\\Datasets\\EEA\\GB\\London\\GB_London_5_2020_22665.csv',
'C:\\Datasets\\EEA\\GB\\London\\GB_London_5_2020_68646.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2019_24044.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2019_24089.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2019_24293.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2019_24321.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2019_24779.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2019_25228.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2019_25606.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2019_67851.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2019_69646.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2020_24044.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2020_24089.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2020_24293.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2020_24321.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2020_24779.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2020_25228.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2020_25606.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2020_67851.csv',
'C:\\Datasets\\EEA\\IT\\Milano\\IT_Milano_5_2020_69646.csv']
```

A warning is also given if, for a certain country, no dataset has been found.

Remove datasets

The input parameters have the same semantics as the previous functions: both code and extended notations can be used (for countries and pollutants); the "all" string can be used. In addition, a warning is given for each not supported value. Moreover, a warning is given for each country for which no dataset has been found.

```
>>> source_path = "C:\\Datasets\\EEA"
>>> countries_cities_dict = {"Italy": ["Venezia"]}
>>> pollutants = "all"
>>> years = [2020]
>>> eea.remove_datasets(source_path, countries_cities_dict,
pollutants, years)
```

Load datasets

The input parameters have the same semantics as the previous functions: both code and extended notations can be used (for countries and pollutants); the "all" string can be used. In addition, a warning is given for each not supported value. Moreover, a warning is given for each country for which no dataset has been found.
This function loads a 'raw' DataFrame pandas, i.e. the EEA datasets are not processed at all.

```
>>> source_path = "C:\\Datasets\\EEA"
>>> countries_cities_dict = {"IT":["Milano"]}
>>> pollutants = ["PM10"]
>>> years = [2020]
>>> df = eea.load_datasets(source_path, countries_cities_dict,
pollutants, years)

>>> df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2773 entries, 0 to 2772
Data columns (total 17 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Countrycode                2773 non-null   object
 1   Namespace                  2773 non-null   object
 2   AirQualityNetwork          2773 non-null   object
 3   AirQualityStation          2773 non-null   object
 4   AirQualityStationEoICode   2773 non-null   object
 5   SamplingPoint              2773 non-null   object
 6   SamplingProcess            2773 non-null   object
 7   Sample                     2698 non-null   object
 8   AirPollutant               2773 non-null   object
 9   AirPollutantCode           2773 non-null   object
 10  AveragingTime              2773 non-null   object
 11  Concentration              2727 non-null   float64
 12  UnitOfMeasurement          2773 non-null   object
 13  DatetimeBegin              2773 non-null   object
 14  DatetimeEnd                2773 non-null   object
 15  Validity                   2773 non-null   int64
 16  Verification               2773 non-null   int64
dtypes: float64(1), int64(2), object(14)
memory usage: 368.4+ KB
>>> df
    Countrycode     Namespace AirQualityNetwork  AirQualityStation
```

```
0              IT  IT.ISPRA.AQD        NET.IT082A       STA.IT1743A
1              IT  IT.ISPRA.AQD        NET.IT082A       STA.IT1743A
2              IT  IT.ISPRA.AQD        NET.IT082A       STA.IT1743A
3              IT  IT.ISPRA.AQD        NET.IT082A       STA.IT1743A
4              IT  IT.ISPRA.AQD        NET.IT082A       STA.IT1743A
...           ...           ...               ...               ...
2768           IT  IT.ISPRA.AQD        NET.IT082A       STA.IT0477A
2769           IT  IT.ISPRA.AQD        NET.IT082A       STA.IT0477A
2770           IT  IT.ISPRA.AQD        NET.IT082A       STA.IT0477A
2771           IT  IT.ISPRA.AQD        NET.IT082A       STA.IT0477A
2772           IT  IT.ISPRA.AQD        NET.IT082A       STA.IT0477A
...
      Concentration UnitOfMeasurement              DatetimeBegin
0         79.600000             µg/m3  2020-01-09 00:00:00 +01:00
1         64.700000             µg/m3  2020-01-10 00:00:00 +01:00
2         81.100000             µg/m3  2020-01-11 00:00:00 +01:00
3         53.400000             µg/m3  2020-01-12 00:00:00 +01:00
4         69.000000             µg/m3  2020-01-13 00:00:00 +01:00
...             ...               ...                        ...
2768       8.449721             µg/m3  2020-12-26 00:00:00 +01:00
2769      29.734741             µg/m3  2020-12-27 00:00:00 +01:00
2770      20.608600             µg/m3  2020-12-28 00:00:00 +01:00
2771      36.742424             µg/m3  2020-12-29 00:00:00 +01:00
2772      48.315550             µg/m3  2020-12-30 00:00:00 +01:00
                    DatetimeEnd  Validity  Verification
0     2020-01-10 00:00:00 +01:00         1             3
1     2020-01-11 00:00:00 +01:00         1             3
2     2020-01-12 00:00:00 +01:00         1             3
3     2020-01-13 00:00:00 +01:00         1             3
4     2020-01-14 00:00:00 +01:00         1             3
...                         ...       ...           ...
2768  2020-12-27 00:00:00 +01:00         1             3
2769  2020-12-28 00:00:00 +01:00         1             3
2770  2020-12-29 00:00:00 +01:00         1             3
2771  2020-12-30 00:00:00 +01:00         1             3
2772  2020-12-31 00:00:00 +01:00         1             3

...

[2773 rows x 17 columns]
```

There are a lot of columns, but only three of them are of interest: `"Concentration"`, `"DatetimeBegin"`, `"Validity"`.

Preprocessing

It processes the raw DataFrame.

```
>>> source_path = "C:\\Datasets\\EEA"
>>> countries_cities_dict = {"IT":["Milano"]}
>>> pollutants = ["PM10"]
>>> years = [2020]
>>> df = eea.load_datasets(source_path, countries_cities_dict,
pollutants, years)

>>> df_mean, df_min, df_max = eea.preprocessing(df, fill=False)
UserWarning: Missing days: ['2020-01-30', '2020-01-31',
'2020-02-01', '2020-02-02', '2020-02-03', '2020-02-04',
'2020-02-05', '2020-02-06', '2020-02-07', '2020-02-08',
'2020-02-09', '2020-02-10', '2020-02-11']

>>> df_mean.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 365 entries, 2020-01-01 to 2020-12-30
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   mean    352 non-null    float64
dtypes: float64(1)
memory usage: 5.7 KB

>>> df_mean
                  mean
Datetime
2020-01-01  134.085714
2020-01-02   63.385286
2020-01-03   74.135229
2020-01-04   49.752786
2020-01-05   41.842857
...                ...
2020-12-26   10.393387
```

```
2020-12-27   26.273187
2020-12-28   18.873236
2020-12-29   38.374420
2020-12-30   56.038968

[366 rows x 1 columns]
```
A warning is given for each missing day.

With `fill=False` the missing days have associated Nan.

```
>>> df_mean["mean"].loc['2020-01-30']
nan
```

`fill=True`

The missing days are filled with the mean of the specified preceding days.

```
>>> source_path = "C:\\Datasets\\EEA"
>>> countries_cities_dict = {"IT": ["Milano"]}
>>> pollutants = ["PM10"]
>>> years = [2020]
>>> df = eea.load_datasets(source_path, countries_cities_dict,
pollutants, years)

>>> df_mean,df_min,df_max = eea.preprocessing(df, fill=True,
fill_n_days=10, fill_aggr="mean")
UserWarning: Missing days: ['2020-01-30', '2020-01-31',
'2020-02-01', '2020-02-02', '2020-02-03', '2020-02-04',
'2020-02-05', '2020-02-06', '2020-02-07', '2020-02-08',
'2020-02-09', '2020-02-10', '2020-02-11']

>>> df_mean
                  mean
Datetime
2020-01-01  134.085714
2020-01-02   63.385286
2020-01-03   74.135229
2020-01-04   49.752786
2020-01-05   41.842857
...                ...
2020-12-26   10.393387
```

```
2020-12-27   26.273187
2020-12-28   18.873236
2020-12-29   38.374420
2020-12-30   56.038968

[366 rows x 1 columns]

>>> df_mean.loc['2020-01-31']
mean    67.493241
Name: 2020-01-31 00:00:00, dtype: float64
>>> df_mean.loc[pd.to_datetime(['2020-01-21', '2020-01-22',
'2020-01-23', '2020-01-24', '2020-01-25', '2020-01-26',
'2020-01-27', '2020-01-28', '2020-01-29', '2020-01-30'])].mean()
mean    67.493241
dtype: float64
```