Katapult Visual Investigation: Notebook 1 - Data Gathering, Cleaning, and Writing

Note: It is recommended to enable Data Wrangler before executing this notebook.

Last execution on October 26, 2025

In [1]:
```
from pathlib import Path
import re
import shutil
import pandas as pd
from copy import deepcopy
import requests
import eurostat
from datetime import datetime, timedelta

pd.set_option('display.max_colwidth', None)
```

In [2]:
```
# Create directories if needed
Path('./data/raw/bruegel/down').mkdir(parents=True, exist_ok=True)
Path('./data/raw/bruegel/ren').mkdir(parents=True, exist_ok=True)
Path('./data/processed/').mkdir(parents=True, exist_ok=True)
Path('./figs/english').mkdir(parents=True, exist_ok=True)
Path('./figs/german').mkdir(parents=True, exist_ok=True)

bruegel_raw_down_dir = Path('./data/raw/bruegel/down')
bruegel_raw_ren_dir = Path('./data/raw/bruegel/ren')
processed_data_dir = Path('./data/processed/')
figs_dir_english = Path('./figs/english/')
figs_dir_german = Path('./figs/german/')
```

1. Eurostat

Link to Python package: https://pypi.org/project/eurostat/

Find potential datasets:

In [4]:
```
# Get all datasets from the Eurostat database (each row lists a dataset)
toc_df = eurostat.get_toc_df()

# Search all datasets for keywords: "natural gas"
natural_gas = eurostat.subset_toc_df(toc_df, 'natural gas')
```

In [5]:
```
natural_gas
```

Out[5]:

	title	code	type	last update of data	last table structure change	data start
717	Supply natural gas - short term monthly data	NRG_IND_343M	dataset	2022-02-10T11:00:00+0100	2024-04-25T11:00:00+0200	2008-01
737	Natural gas import dependency by country of origin	NRG_IND_IDOGAS	dataset	2022-09-29T23:00:00+0200	2024-04-25T11:00:00+0200	2015
783	Stock levels for gaseous and liquefied natural gas	NRG_STK_GAS	dataset	2025-06-24T23:00:00+0200	2024-12-13T11:00:00+0100	1990
792	Exports of natural gas by partner country	NRG_TE_GAS	dataset	2025-06-24T23:00:00+0200	2024-12-13T11:00:00+0100	1990
793	Exports of natural gas by partner country - monthly data	NRG_TE_GASM	dataset	2025-10-27T23:00:00+0100	2025-10-07T23:00:00+0200	2008-01
800	Imports of natural gas by partner country	NRG_TI_GAS	dataset	2025-06-24T23:00:00+0200	2024-12-13T11:00:00+0100	1990
801	Imports of natural gas by partner country - monthly data	NRG_TI_GASM	dataset	2025-10-27T23:00:00+0100	2025-10-07T23:00:00+0200	2008-01

⇒ We can use natural gas datasets with codes `NRG_TI_GAS` and `NRG_TI_GASM`, as these datasets deal with the annually (NRG_TI_GAS) monthly (NRG_TI_GASM) imports of gas by partner country.

Querying and inspecting the datasets:

In [6]:
```
nrg_ti_gas = eurostat.get_data_df('NRG_TI_GAS')
nrg_ti_gasm = eurostat.get_data_df('NRG_TI_GASM')
```

In [7]:
```
eurostat_datasets = {
    'nrg_ti_gas': nrg_ti_gas,
```

```
    'nrg_ti_gasm':nrg_ti_gasm,
}
```

In [8]:
```
for dataset_name, dataset in eurostat_datasets.items():
  print(dataset_name)
  display(dataset.head(3))
```

nrg_ti_gas

	freq	siec	partner	unit	geo\TIME_PERIOD	1990	1991	1992	1993	1994	...
0	A	G3000	AD	MIO_M3	AL	0.0	0.0	0.0	0.0	0.0	...
1	A	G3000	AD	MIO_M3	AT	0.0	0.0	0.0	0.0	0.0	...
2	A	G3000	AD	MIO_M3	BA	NaN	NaN	NaN	NaN	NaN	...

3 rows × 39 columns

nrg_ti_gasm

	freq	siec	partner	unit	geo\TIME_PERIOD	2008-01	2008-02	2008-03	2008-04	2008-05
0	M	G3000	AD	MIO_M3	AL	NaN	NaN	NaN	NaN	NaN
1	M	G3000	AD	MIO_M3	AT	NaN	NaN	NaN	NaN	NaN
2	M	G3000	AD	MIO_M3	BE	NaN	NaN	NaN	NaN	NaN

3 rows × 218 columns

Get an explanation of the columns (same for both datasets, so here just for NRG_TI_GASM):

In [9]:
```
eurostat.get_dic('NRG_TI_GAS')
```

Out[9]:
```
[('freq',
  'Time frequency',
  'This code list contains the periodicity that refers to the frequency.'),
 ('siec',
  'Standard international energy product classification (SIEC)',
  'This code list contains the energy products according to the Standard Intern
ational Energy Product Classification (SIEC) which has been developped as part
of the International Recommendations for Energy Statistics (IRES) adopted by th
e UNSD.'),
 ('partner',
  'Geopolitical entity (partner)',
  'This code list defines the geopolitical entities (partners). Partner countri
es (PARTNER) is based on ISO-3166 (using alpha-2) with minor changes. Partner c
ountry is the last known country of destination /origin.'),
 ('unit', 'Unit of measure', None),
 ('geo',
  'Geopolitical entity (reporting)',
  'This code list defines the reporting geopolitical entities.')]
```

⇒ The column `geo\TIME_PERIOD` is not really one column. `geo` is the reporting country and the index in the original database. `TIME_PERIOD` is the description for the year columns. So overall it's just a parsing mistake, which should be fixed first. If you want to check how an original dataset looks like, you can follow the link down below for the dataset NRG_TI_GASM:

https://ec.europa.eu/eurostat/databrowser/view/nrg_ti_gasm/default/table?lang=en&category=nrg.nrg_quant.nrg_quantm.nrg_t_m.nrg_ti_m

Rename column `geo/TIME_PERIOD` to `import_country`:

In [10]:
```
for key in eurostat_datasets:
    eurostat_datasets[key].rename(columns={r'geo\TIME_PERIOD':'import_country'},
```

Find all occuring SIEC codes in both datasets:

In [11]:
```
for key in eurostat_datasets:
    print("SIEC codes in", key, ":", eurostat_datasets[key].siec.unique())
```

```
SIEC codes in nrg_ti_gas : ['G3000' 'G3200']
SIEC codes in nrg_ti_gasm : ['G3000' 'G3200']
```

⇒ G3000 is the code for natural gas and G3200 is the code for liquified natural gas (LNG). This can be found by following the previous link to the NRG_TI_GASM dataset. Under the header "Selection" you have the option to customize the dataset. On the right side you will see the tab option "Standard international energy product classification (SIEC)". By clicking on this tab you can find the codes with their respective meaning.

As these energy products are needed for this project and no other products occur in these datasets, there is no need to preprocess this column in one the both datasets

Find all occuring units:

In [12]:
```
for key in eurostat_datasets:
    print("Reported Units in", key, ":", eurostat_datasets[key].unit.unique())
```

```
Reported Units in nrg_ti_gas : ['MIO_M3' 'TJ_GCV']
Reported Units in nrg_ti_gasm : ['MIO_M3' 'TJ_GCV']
```

⇒ For this project all entries with the unit `MIO_M3` are kept. In our opinion this unit is easier to understand for the average person. However, using `TJ_GCV` would make sense as well and we encourage to reproduce the notebook with this unit, to compare results.

Filter for entries with unit MIO_M3:

In [13]:
```
for key, df in eurostat_datasets.items():
    unit_mask = df.unit.eq('MIO_M3')
    df.drop(df.index[~unit_mask], inplace=True)
```

As this project deals with the natural gas imports of the Euopean Union (and its member states) from Russia the following preprocessing steps are applied next:

- Just keep entries where the import area/country is the EU or a member state ⇒ country codes: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Glossary:Country_codes
- Just keep entries where the partner country is Russia ⇒ partner definition: https://ec.europa.eu/eurostat/cache/metadata/en/nrg_t_esms.htm#shortunit_measureDi

In [14]:
```
keep_geo = ['EU27_2020', 'BE', 'BG', 'CZ', 'DK', 'DE', 'EE', 'IE', 'EL', 'ES', '
    'CY', 'LV', 'LT', 'LU', 'HU', 'MT', 'NL', 'AT', 'PL', 'PT', 'RO', 'SI', 'SK'
partner = ['RU']

for key, df in eurostat_datasets.items():
    imports_mask = df['import_country'].isin(keep_geo) & df['partner'].isin(part
    df.drop(df.index[~imports_mask], inplace=True)
```

Get an overview of the data quality:

In [15]:
```
for key, value in eurostat_datasets.items():
    print("Missing values in", key, ":")
    display(value.isna().sum())
```

```
Missing values in nrg_ti_gas :
```

```
freq              0
siec              0
partner           0
unit              0
import_country    0
1990              0
1991              0
1992              0
1993              0
1994              0
1995              0
1996              0
1997              0
1998              0
1999              0
2000              0
2001              0
2002              0
2003              0
2004              0
2005              0
2006              0
2007              0
2008              0
2009              0
2010              0
2011              0
2012              0
2013              0
2014              0
2015              0
2016              0
2017              0
2018              0
2019              0
2020              0
2021              0
2022              0
2023              0
dtype: int64
Missing values in nrg_ti_gasm :
freq              0
siec              0
partner           0
unit              0
import_country    0
                 ..
2025-05           0
2025-06           0
2025-07           0
2025-08           0
2025-09           5
Length: 218, dtype: int64
```

⇒ It can be seen, that the monthly dataset contains missing values in certain years. Therefore we decided to use the following approach:

As the annual dataset has no missing values after preprocessing, the full dataset is kept for now. The monthly dataset has no missing values after December 2020[1]. Russia

invaded Ukraine in February 2022. Therefore, the monthly data is kept from January 2022 because there is clean data from this month onwards. Furthermore, with this approach we can divide the data in the pre-war time (annually data from 1990-2021), and mid-war time (monthly data from 2022-present).

[1]Note: There is just one month with missing values after December 2020, which is September 2025, because it's the last occuring month, **as of the last notebook execution**, and not all data is already updated until this day. Therefore, September 2025 will be dropped in the next steps.

In the following code cell the `freq` column are dropped for both datasets, as these are not needed and do not contain valuable information.

For nrg_ti_gas no other columns are dropped. Even if the annually data will just be used for the pre-war time until 2021, there is no disadvantage in keeping the other years up to 2023 as well.

For nrg_ti_gasm all columns are dropped, which contain data from 2008-01 to 2021-12.

In [ ]:
```
# Drop 'freq' column for both datasets
for key in eurostat_datasets:
    eurostat_datasets[key].drop(columns=['freq'], inplace=True)

# For nrg_ti_gasm: drop all months from 2008 to 2021, and 2025-09 (not all count
nrg_ti_gasm_drop = [col for col in nrg_ti_gasm.columns
                    if '2008-01' <= col <= '2021-12'
                    or col == '2025-09'
]
nrg_ti_gasm.drop(columns=nrg_ti_gasm_drop, inplace=True)
```

Finally, the preprocessed datasets are written as CSV files to the directory `/data/processed`:

In [17]:
```
for key in eurostat_datasets:
    eurostat_datasets[key].to_csv(f'{processed_data_dir}/eurostat_{key}.csv', in
```

In [ ]:
```

```

2. Bruegel

Links to datasets:

- https://www.bruegel.org/dataset/european-natural-gas-imports
- https://www.bruegel.org/dataset/european-natural-gas-demand-tracker

As there is no API or Python package for any Bruegel data, the ZIP-folder with all files related to natural gas imports are donloaded manually.

IMPORTANT NOTE: Each file in the ZIP-folder contains data for one specific picture of the related Bruegel article. As not all files are important for this project, we just took the

ones necessary for our purposes and stored them in the directory `/data/raw/bruegel/down...`. In the following, the selected files are listed **with the initial file names as they occur in the ZIP-folder**:

- "quarterly graph 2025 Q3.xlsx"

Also, as the files are structured differently (not like the two Eurostat datasets from before), we decided to not use an dictionary this time. Last ZIP-download: October 10, 2025

As the file names contain whitespaces and hyphes, the first step is to replace these by underscores and save the renamed files in a separate directory `/data/raw/bruegel/ren...`:

In [ ]:
```
# Create a lambda function for removing whitespaces and hyphes
remove_spaces = lambda s: re.sub(r"[\s-]+", "_", s.strip())

# Save renamed files to new directory
for path in bruegel_raw_down_dir.glob("*.xls*"):
    new_stem = remove_spaces(path.stem)
    new_name = f'{new_stem}{path.suffix}'
    target = bruegel_raw_ren_dir / new_name

    if target.exists():
        continue

    shutil.copy2(path, target)
```

All files are read in and again stored in a dictionary. It might be useful for you to know, which abbreviations for the variable names are used and what they mean:

- gas = natural gas
- lng = LNG
- reg = regional
- quart = quarterly
- month = monthly
- imp = imports

In [ ]:
```
gas_reg_quart_imp = pd.read_excel(f'{bruegel_raw_ren_dir}/quarterly_graph_2025_Q
```

Inspection of all Bruegel datasets

1. `gas_reg_quart_imp` : quarterly EU imports of natural gas from Q1 2019 to Q3 2025

First overview:

In [6]:
```
gas_reg_quart_imp.head(4)
```

Out[6]:

	dates	Nord Stream	Yamal (BY,PL)	Ukraine Gas Transit	Turkstream	Russia LNG	Ru
0	2019-03-31	15335.807168	9656.556780	20471.842853	0.0	4081.945029	49546.151
1	2019-06-30	15507.452564	9254.398183	23989.709524	0.0	5463.904113	54215.464
2	2019-09-30	13270.942279	8626.061719	21233.306272	0.0	3465.151200	46595.461
3	2019-12-31	15477.485308	10165.853006	22916.370105	0.0	3943.723846	52503.432

Check for data types and missing values:

In [7]:
```
gas_reg_quart_imp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27 entries, 0 to 26
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype         
---  ------                            --------------  -----         
 0   dates                             27 non-null     datetime64[ns]
 1   Nord Stream                       27 non-null     float64       
 2   Yamal (BY,PL)                     27 non-null     float64       
 3   Ukraine Gas Transit               27 non-null     float64       
 4   Turkstream                        27 non-null     float64       
 5   Russia LNG                        27 non-null     float64       
 6   Russia                            27 non-null     float64       
 7   Norway LNG                        27 non-null     float64       
 8   Norway                            27 non-null     float64       
 9   Norway.1                          27 non-null     float64       
 10  USA LNG                           27 non-null     float64       
 11  Algeria LNG                       27 non-null     float64       
 12  Algeria                           27 non-null     float64       
 13  Algeria.1                         27 non-null     float64       
 14  LNG less RU and USA and NO and AL 27 non-null     float64       
 15  UK                                27 non-null     float64       
 16  Azerbaijan                        27 non-null     float64       
 17  Libya                             27 non-null     float64       
dtypes: datetime64[ns](1), float64(17)
memory usage: 3.9 KB
```

⇒ No need for further preprocessing regarding data types and missing values, so let's proceed.

Both Norway and Algeria have 2 columns for natural gas. For Norway there are `Norway` and `Norway.1`, and for `Algeria` there are `Algeria` and `Algeria.1`. After comparing the values from the table with the ones of the plot on the Bruegel website, it became clear that the columns Norway.1 and Algeria.1 were used. The other two might be outdated. Therefore the unused ones are dropped.

After doing this, many columns are renamed as follows:

- `dates` → `date`
- `Nord Stream` → `Nord_Stream`
- `Yamal (BY,PL)` → `Yamal_BY_PL`
- `Ukraine Gas Transit` → `Ukraine_Gas_Transit`
- `Russia LNG` → `Russia_LNG`
- `Norway LNG` → `Norway_LNG`
- `Norway.1` → `Norway`
- `USA LNG` → `USA_LNG`
- `Algeria LNG` → `Algeria_LNG`
- `Algeria.1` → `Algeria`
- `LNG less RU and USA and NO and AL` → `LNG_less_RU_USA_NO_AL`

In [8]:
```
gas_reg_quart_imp.drop(columns=['Norway', 'Algeria'], inplace=True)
gas_reg_quart_imp.rename(columns={
    'dates': 'date',
    'Nord Stream': 'Nord_Stream',
    'Yamal (BY,PL)': 'Yamal_BY_PL',
    'Ukraine Gas Transit': 'Ukraine_Gas_Transit',
    'Russia LNG': 'Russia_LNG',
    'Norway LNG': 'Norway_LNG',
    'Norway.1': 'Norway',
    'USA LNG': 'USA_LNG',
    'Algeria LNG': 'Algeria_LNG',
    'Algeria.1': 'Algeria',
    'LNG less RU and USA and NO and AL': 'LNG_less_RU_USA_NO_AL',
}, inplace=True)
```

Now, a new column `quarter` is inserted right after the date column. The new column contains the respective quarter of the entry:

In [9]:
```
i = gas_reg_quart_imp.columns.get_loc('date') + 1
gas_reg_quart_imp.insert(
    i, 'quarter',
    gas_reg_quart_imp['date'].dt.to_period('Q')
)
```

Now, **two new dataframes** will be created/derived from gas_reg_quart_imp:

- One with the name `ru_quart_imp`, which is just related to Russia with the following columns: `dates`, `Nord_Stream`, `Yamal_BY_PL`, `Ukraine_Gas_Transit`, `Turkstream`, `Russia_LNG`, `Russia`
- One with the name `lng_reg_quart_imp`, which just contains the columns related to LNG imports with the following columns: `Russia_LNG`, `Norway_LNG`, `USA_LNG`, `Algeria_LNG`, `LNG_less_RU_USA_NO_AL`

In [10]:
```
# 1. Russia related df
ru_quart_imp = deepcopy(gas_reg_quart_imp[[
    'date', 'quarter', 'Nord_Stream', 'Yamal_BY_PL', 'Ukraine_Gas_Transit', 'Tur
]]
)

# 2. LNG related df
lng_reg_quart_imp = deepcopy(gas_reg_quart_imp[[
```

```
    'date', 'quarter', 'Russia_LNG', 'Norway_LNG', 'USA_LNG', 'Algeria_LNG', 'LN
]]
)
```

Let's have a look at the Russia related dataframe:

In [11]:
```
ru_quart_imp.head(4)
```

Out[11]:

	date	quarter	Nord_Stream	Yamal_BY_PL	Ukraine_Gas_Transit	Turkstream	Russia
0	2019-03-31	2019Q1	15335.807168	9656.556780	20471.842853	0.0	4081.94
1	2019-06-30	2019Q2	15507.452564	9254.398183	23989.709524	0.0	5463.90
2	2019-09-30	2019Q3	13270.942279	8626.061719	21233.306272	0.0	3465.15
3	2019-12-31	2019Q4	15477.485308	10165.853006	22916.370105	0.0	3943.72

Let's check if each entry in the last column `Russia` is just the sum of the entries in all previous columns:

In [12]:
```
# Row-wise sum of columns with indices 2 to 6, compare to column 7 'Russia'
prev_ru_cols = gas_reg_quart_imp.iloc[:, 2:7].sum(axis=1)
last_ru_col = gas_reg_quart_imp.iloc[:, 7]
check_equals = prev_ru_cols.eq(last_ru_col)
all_equal = check_equals.all()

if all_equal:
    print('Each entry in the last column "Russia" is the sum of the entries in a
    print('No need for further inspection.')
else:
    print('At least one entry in the last column "Russia" is not the sum of the
    print('Check the mismatches.')
```

```
Each entry in the last column "Russia" is the sum of the entries in all previous
columns.
No need for further inspection.
```

Let's have a look at the LNG related dataframe:

In [13]:
```
lng_reg_quart_imp.head(4)
```

Out[13]:

	date	quarter	Russia_LNG	Norway_LNG	USA_LNG	Algeria_LNG	LNG_less_RU_U
0	2019-03-31	2019Q1	4081.945029	1096.285267	2819.637553	2088.020683	10
1	2019-06-30	2019Q2	5463.904113	1449.456487	3524.870467	2623.734319	11
2	2019-09-30	2019Q3	3465.151200	1231.557514	2429.228526	2053.068944	11
3	2019-12-31	2019Q4	3943.723846	1388.792605	5392.554242	2022.473627	11

⇒ Let's just rename the columns by removing "_LNG" part of all column names. The only exception is the last column which we be completely renamed to "Other":

In [14]:
```
lng_reg_quart_imp.rename(columns={
    'Russia_LNG': 'Russia',
    'Norway_LNG': 'Norway',
    'USA_LNG':'USA',
    'Algeria_LNG':'Algeria',
    'LNG_less_RU_USA_NO_AL': 'Other'
    }, inplace=True
)
```

⇒ Everything looks good now, and the new derived dataframes are clean. Therefore we can drop the LNG related columns and the pipeline related columns from the inital datframe. By doing this, the initial dataframe will be changed in a way that it just contains the (non LNG) quarterly gas imports by regions:

In [15]:
```
gas_reg_quart_imp = gas_reg_quart_imp[['date', 'quarter', 'Russia', 'Norway', 'A
gas_reg_quart_imp.head(4)
```

Out[15]:

	date	quarter	Russia	Norway	Algeria	UK	Azerbaijan	
0	2019-03-31	2019Q1	49546.151830	24026.577320	8834.770599	1234.134675	0.0	1
1	2019-06-30	2019Q2	54215.464384	24577.957396	7528.081767	4120.617418	0.0	1
2	2019-09-30	2019Q3	46595.461470	20215.152798	6460.794922	2589.138787	0.0	1
3	2019-12-31	2019Q4	52503.432264	22925.084803	8703.044040	1582.001807	0.0	1

Now we have the following 3 dataframes, which which were created from the original dataframe `gas_reg_quart_imp`:

1. gas_reg_quart_imp (altered): contains the (non LNG) quarterly gas imports by regions
2. ru_quart_imp: contains the quarterly gas imports from Russia by import way/transit

3. lng_reg_quart_imp: contains the LNG quarterly gas imports by regions

Finally, we can save these dataframes as CSV files in the directory `data/processed/`, where we already stored the processed Eurostat datasets:

In [16]:
```
gas_reg_quart_imp.to_csv(f'{processed_data_dir}/bruegel_gas_reg_quart_imp.csv',
ru_quart_imp.to_csv(f'{processed_data_dir}/bruegel_ru_quart_imp.csv', index=Fals
lng_reg_quart_imp.to_csv(f'{processed_data_dir}/bruegel_lng_reg_quart_imp.csv',
```

3. GIE

Link to Python package: https://pypi.org/project/gie-py/

Link to API documentation: https://www.gie.eu/transparency-platform/GIE_API_documentation_v013.pdf

To use the GIE API for you must sign up and create an account, which does not come with any costs. The sign up can be done via the following link: https://agsi.gie.eu/account. There are two available GIE APIs available, the first one being the `Aggregated Gas Storage Inventory API (AGSI)`, the second one being the `Aggregated LNG Storage Inventory (ALSI)`. For this project, `AGSI` is used. The reason is that when people say "EU gas storage is X% full", they mean underground gas storage (UGS), which are the big storage facilities that balance Europe's gas system across seasons.**AGSI+** is the dataset for underground storage. It gives, per country and day:

- Filling level (% full)
- Gas in storage (energy, e.g. TWh)
- Working gas capacity (total usable capacity)

ALSI is for LNG import terminals, e.g. ship unloading tanks and send-out to the grid. So it provides data for short-term terminal buffers, not the long-term seasonal storage of countries and the EU.

For a reader a storage map should show how full the underground storage is in each country. Therefore AGSI+ is the source in this case.

AGSI+: Latest Gas Filling Levels of Selected EU Countries

Goal: Get data to build a EU storage map using **one common gas day**, so every country's value refers to the same date, and only the countries AGSI actually exposes.

Approach (infos from the API documentation):

1. Get the latest listing by not giving a certain date in the params dictionary. From the response read the value of key `gas_day` to get the info from which date this latest listing is.

2. Try that date via params on the same listing endpoint. If there is no data in the key `data`, **step back 1 day** and retry. Try as long as you get a non-empty list for the key `data`.

3. **Extract country rows** directly from `EU.children` (or the root `data` list), and keep only EU-27 countries with **non-zero working gas volume**, and write one tidy CSV for mapping.

As already mentioned, we only query **underground gas storage (UGS)** (AGSI). LNG tanks (ALSI) are different and not used for the storage map, which will be created from the data. Both platforms cover 100% of EU-27, but some members have **no UGS**.

Preparation

First, the API key is read from a local file and all variables for the request are set up:

In [31]:
```
with open('GIE_API.txt', 'r') as f:
    GIE_KEY = f.read().strip()
API_URL = 'https://agsi.gie.eu/api'
HEADERS = {'x-key': GIE_KEY}

EU27 = {'BE', 'BG', 'CZ', 'DK', 'DE', 'EE', 'IE', 'GR', 'ES', 'FR', 'HR', 'IT','
        'LT', 'LU', 'HU', 'MT', 'NL', 'AT', 'PL', 'PT', 'RO', 'SI', 'SK', 'FI',
```

Read latest gas_day from listing (without given date in params), and find storage countries

Let's try a first API call without any specified paramters to find out the latest reported gas day:

In [32]:
```
# Perform basic API call to get latest listing and print latest reported gas day
latest_listing = requests.get(API_URL, headers=HEADERS, timeout=10).json()
base_gas_day = latest_listing.get('gas_day', [])
print('Latest reported gas day :', base_gas_day)
```

```
Latest reported gas day : 2025-10-25
```

The latest reported gas day does not guarantee that there is indeed data for this day. The key `data` holds a list as value, which could be empty. This is sometimes the case when the latest gas day is already added to the system, but the actual data will be updated a few hours later. Therefore we want to find the most recent day for which data of EU member states is actually available:

In [33]:
```
got_data = False
days_back = 0
MAX_LOOKBACK = 30

# If there is no data (= empty list vor 'data' key in response), perform loop fo
while not got_data:
    # Calculate new gas day and use for new API call
    latest_gas_day = (datetime.fromisoformat(base_gas_day) - timedelta(days=days
    params = {'date': latest_gas_day}
    new_latest_listing = requests.get(API_URL, headers=HEADERS, params=params, t
```

```
    data = new_latest_listing.get('data', [])

    # If there is still no data, go back another day
    if not data:
        days_back += 1
        continue

    # If there is data, find out if EU has empty children (no reported countries
    eu_node = next((node for node in data if str(node.get('code', '')).lower() =
    eu_children = eu_node.get('children', []) if eu_node else []
    is_empty = (len(data) == 0) or (eu_node is not None and len(eu_children) ==

    if not is_empty:
        chosen_gas_day = latest_gas_day
        print('Chosen gas day:', latest_gas_day)
        got_data = True
    else:
        days_back += 1

if not got_data:
    raise RuntimeError(f'No data found within the last {MAX_LOOKBACK} days start
```

```
Chosen gas day: 2025-10-25
```

As the last execution of this notebook was on October 26, 2025, the day before this date is our chosen gas day and will later be used for the respective visualization. To ensure reproducibility for the visualization of this project, the variable `chosen_gas_day` is again set manually in the following cell. If you want to have data for the actual latest gas day because you want the visualization be based on the most recent data (depending when you are reading this) just comment out or delete the code in the following cell:

In [34]:
```
chosen_gas_day = '2025-10-25'
```

Let's perform a fresh API call with the final `chosen_gas_day` (if there is one with data reported). We could just use the last `new_latest_listing`, but for better understanding a fresh request will be done:

In [35]:
```
# Get the payload for the chosen date
resp = requests.get(API_URL, headers=HEADERS, params={'date': chosen_gas_day}, t
data = resp.get('data', []) or []

# Use eu.children if present, otherwise use root list
eu_node = next((node for node in data if str(node.get('code', '')).lower() == 'e
nodes = eu_node.get('children', []) if (eu_node and eu_node.get('children')) els
```

Finally, let's build a dataframe that shows the relevant data for all countries found for the chosen gas day. In the end the result will also be saved in `data/processed/`, where the processed Eurostat and Bruegel datasets are already stored:

In [36]:
```
rows = []
for node in nodes:
    code = node.get('code')
    # Keep only countries from EU27 set
    if not (isinstance(code, str) and len(code) == 2 and code in EU27):
        continue
```

```
    # Keep only countries with underground storage capacity > 0
    work_gas_vol = node.get('workingGasVolume')
    try:
        work_gas_vol_val = float(work_gas_vol)
    except (TypeError, ValueError):
        work_gas_vol_val = 0.0
    if work_gas_vol_val <= 0:
        continue

    rows.append({
        'country': node.get('name'),
        'country_code': code,
        'gas_day': chosen_gas_day,
        'filling_%': pd.to_numeric(node.get('full'), errors='coerce'),
        'gas_TWh': pd.to_numeric(node.get('gasInStorage'), errors='coerce'),
        'capacity_TWh': pd.to_numeric(work_gas_vol, errors='coerce')
    })

gas_storage_eu = pd.DataFrame(rows).sort_values('country_code').reset_index(drop
```

At the moment, `gas_TWh` (gas in storage) and `capacity_TWh` (working gas volume) are shown in TWh. To allign it with the previous visualization and the units shown on these, TWh will be coverted to million cubic metres (Mm^3) with the following assumption and calculation:

- Assumption for heating value: mid-range **HHV = 37 MJ/m³** at standard conditions (HV of natural gas usually ranges between 35–39 MJ/m^3)
- Formula: $1\,\mathrm{TWh} = 3.6 \times 10^9\,\mathrm{MJ}$
  Volume in m^3: $V_{m^3} = \dfrac{E_{\mathrm{MJ}}}{\mathrm{HV}\ (\mathrm{MJ/m^3})}$ $\Rightarrow$ Convert m^3 to million m^3 by dividing by 10^6
- Result: $\text{million m}^3 = \mathrm{TWh} \times \dfrac{3600}{\mathrm{HV}\ (\mathrm{MJ/m^3})}$
- With $\mathrm{HV} = 37\,\mathrm{MJ/m^3}$:
  $1\,\mathrm{TWh} \approx \dfrac{3600}{37} = \mathbf{97.297}\ \text{million m}^3$

$\Rightarrow$ So for the dataframe a constant heating value `HV_MJ_PER_M3` is defined and then used to convert the columns **gas_TWh to gas_mcm**, and **capacity_TWh to capacity_mcm**:

In [37]:
```
HV_MJ_PER_M3 = 37.0 # Defined heating value
gas_storage_eu['gas_mcm'] = gas_storage_eu['gas_TWh'] * 3600.0/HV_MJ_PER_M3
gas_storage_eu['capacity_mcm'] = gas_storage_eu['capacity_TWh'] * 3600.0/HV_MJ_P
```

Before saving the data to a CSV file there is an important aspect to mention: the column `filling_%` (AGSI key 'full') is defined as `gas_TWh` (AGSI key gasInStorage) divided by `capacity_TWh` (AGSI workingGasVolume). Because working gas volume is a nominal technical capacity and energy content can vary, some entries can exceed a gas filling level of 100%. For clarity, these cases are set to 100% in the dataframe:

In [38]:
```
gas_storage_eu['filling_%'] = gas_storage_eu['filling_%'].clip(upper=100.0)
```

Let's have a look on the result:

In [39]:
```
gas_storage_eu
```

Out[39]:

	country	country_code	gas_day	filling_%	gas_TWh	capacity_TWh	gas_mcr
0	Austria	AT	2025-10-25	82.88	84.7862	102.2986	8249.46810
1	Belgium	BE	2025-10-25	91.13	8.1834	8.9800	796.22270
2	Bulgaria	BG	2025-10-25	83.93	5.8787	7.0044	571.98162
3	Czech Republic	CZ	2025-10-25	91.08	42.8566	47.0535	4169.83135
4	Germany	DE	2025-10-25	75.34	189.0047	250.8801	18389.64648
5	Denmark	DK	2025-10-25	60.71	5.9434	9.7900	578.27675
6	Spain	ES	2025-10-25	86.70	31.0671	35.8318	3022.74486
7	France	FR	2025-10-25	92.78	116.6474	125.7231	11349.47675
8	Croatia	HR	2025-10-25	57.79	2.7579	4.7725	268.33621
9	Hungary	HU	2025-10-25	70.18	47.7144	67.9909	4642.48216
10	Italy	IT	2025-10-25	94.48	191.7512	202.9481	18656.87351
11	Latvia	LV	2025-10-25	58.32	14.5018	24.8653	1410.98594
12	Netherlands	NL	2025-10-25	72.45	104.5417	144.2965	10171.62486
13	Poland	PL	2025-10-25	100.00	36.5669	36.3098	3557.86054
14	Portugal	PT	2025-10-25	96.20	3.4344	3.5700	334.15783
15	Romania	RO	2025-10-25	98.12	33.2281	33.8636	3233.00432
16	Sweden	SE	2025-10-25	100.00	0.0893	0.0875	8.68864
17	Slovakia	SK	2025-10-25	74.37	27.1907	36.5630	2645.58162

Finally, the result is saved in `data/processed/`, where the processed Eurostat and Bruegel datasets are already stored:

In [40]:
```
gas_storage_eu.to_csv(f'{processed_data_dir}/GIE_agsi_gas_storage_eu.csv', index
```

Now, all data needed for the project and visualizations is processed, saved in the processed data folder, and ready for use in the Juypter Notebooks for the visualizations.

In [ ]: