Comext Data Script

# Introduction to Script

# Script 1: Prepping Datasets

## Determining the Type of Product Codes Changes

The Eurostat methodology describes the data treatment that is applied to product code changes. The highlight three types of product code changes occurring on a year basis.

1. Merge of several codes into a single code
2. Split of a code into several codes
3. Double: merge + split several codes into several codes

Prior to describing how this occurs, we need to take the data structure that is available to us to determine which type of product codes changes apply to the year of interest. Eurostat provides product code change datasets from the year 2017 until now (currently 2025). However, given the inconsistent structure, we need to apply a relatively universal approach. That is, we need to apply our own logic to identifying the type of changes that occur.

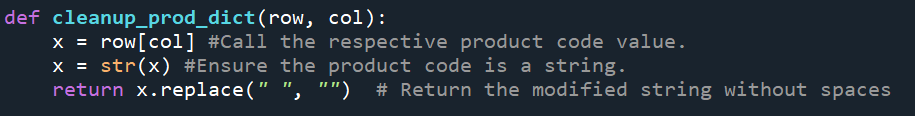
The product cade dictionaries will typically come in the form of (Table x).

|  |  |  |  |
| --- | --- | --- | --- |
| More Detailed Code | Common Nomenclature (CN) Code Year t | Comment | Common Nomenclature (CN) Code Year t-1 |
| - | XXXXXXXX | - | YYYYYYYY |
| - | XXXXXXXX | - | YYYYYYYY |
| - | XXXXXXXX | - | YYYYYYYY |

To determine the relationship for the following steps are taken:

1. **Cleaning the Dataframe**

Given that the codes are not provided the in the equivalent form to those of the actual COMEXT dataframes, we need to clean them up. That is, the *Common Nomenclature Code Year t(-1)* usually contain spaces, whereas the ones in the dataframe do not. Figure X show the function that is applied to both CN code columns in the respective dataframe.

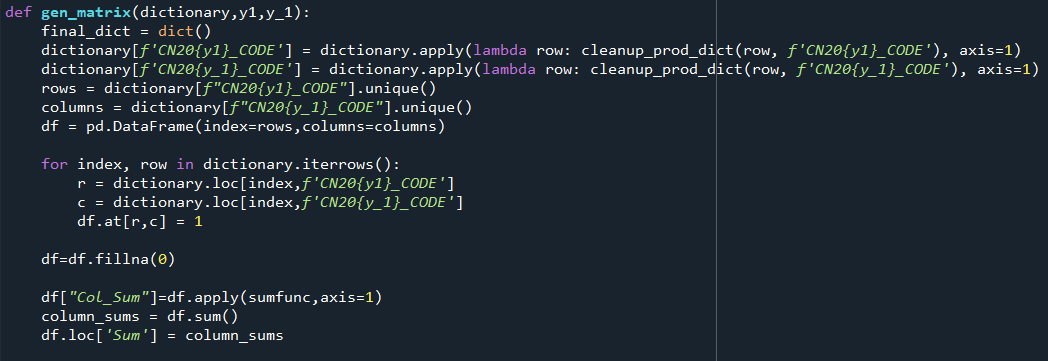


This is simply applied using a *lambda operation* (see Figure x).This lambda operation is part of a larger function that performs the categorization of CN code changes (see Figure x & x). As is evident, the structure allows for dynamic changes, allowing us to automatically process different year combinations through a loop.



1. **Preparing the Relational Matrix**

After doing the essential pre-cleaning, we identify the unique *new* (e.g. year t) and old (e.g. year-t) CN codes. This is achieved through lines 4 -5 in Figure X. These will not be always an identical length, given that splitting CN codes might result in substantially more product codes than before. The list of unique values are then combined into the matrix.



After the generation of the matrix, we populate it based on the relationships provided in the original CN code change dataframe. In the case where a combination between column *Common Nomenclature (CN) Code Year t* and *Common Nomenclature (CN) Code Year t-1* (highlighted in yellow in Table x) applies, the value 1 is provided at their intersection. A visual description of this can be seen below in Table x.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | At-1 | Bt-1 | Ct-1 | Dt-1 | Et-1 |
| At | 1 | 1 |  |  |  |
| Bt | 1 | 1 |  |  |  |
| Ct |  |  | 1 |  |  |
| Dt |  |  | 1 |  |  |
| Et |  |  |  | 1 | 1 |
| Ft |  |  |  |  |  |

The table provides an example of the different combinations that could arise. First, the **orange** colored cells represent a simple **split** of a code. Specifically, Ct-1 is split into Ct and Dt represented by the vertical nature of the values. Given that no other of the t-1 codes have a relation to Ct or Dt , we know that neither the case of merging nor that of merge-split applies in this case. Second, the cells that are shaded **blue** describe the prototypical case of the a simple **merge**. This is recognizable by the fact that the Et is linked to two t-1 codes (Dt-1 and Et-1), but neither seem to be split to other codes (e.g. no vertical 1’s only horizontal). Codes in t-1 that have only a single 1 will exclusively participate in a simple merging operation. Third, the **green** coded value represent the **merge-split** scenario. This is recognizable by the fact that the t-1 codes contain vertically more than 1 relation, while the codes in year t also contain more than 1 relationship. It is at this intersection of if cases where we find **merge-split** relations. It is important to note that while this a simple relation, more complicated relations as in Table x may arise.

The Table x shows a complicated relationship where t-1 codes share varying degrees of overlaps with codes for year t. This produces difficulties, as relatively clear cut splits such as a in a simple split scenario cannot apply. The solution to this is discussed below, however, a simple to the intuition might help. Specifically, the complexity in this case arises from the fact that the codes in year t are weighted combination of the codes in year t-1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | At-1 | Bt-1 | Ct-1 | Dt-1 | Et-1 |
| At | 1 |  | 1 |  |  |
| Bt | 1 | 1 |  |  |  |
| Ct |  | 1 |  |  |  |
| Dt |  |  |  |  |  |
| Et |  |  |  |  |  |
| Ft |  |  |  |  |  |

Therefore, the following combination would likely produce the following equations:

After generation the relational matrix, we sum along rows and columns. The end result would be equivalent to the one provided below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | At-1 | Bt-1 | Ct-1 | Dt-1 | Et-1 | Col Sums |
| At | 1 | 1 |  |  |  | 2 |
| Bt | 1 | 1 |  |  |  | 2 |
| Ct |  |  | 1 |  |  | 1 |
| Dt |  |  | 1 |  |  | 1 |
| Et |  |  |  | 1 | 1 | 2 |
| Ft |  |  |  |  |  |  |
| Sum | 2 | 2 | 2 | 1 | 1 |  |

These are used to categorize the relations, as well as storing the precise CN codes.

1. **Classifying Relations**

To classify the relations we apply the following logics. First we looks at the *Col Sums* column. We know that if the value of *Col Sums* in row x is **greater than** 1, the CN code for year t is part of a merging operation. Then we loop at the *Sum* row. There we know that if the values for Column x of CN code for the year t-1 is greater than 1,then the CN code in year t-1 is part of a splitting operation. These categorizations are stored in a python dictionary. In Table x you can see the logic applied to our example.

**Step 1: Looking at Col Sums**

**At** is classified as **merge.** Then we look at the columns where we the cells are 1. Therefore the dictionary will take the following form:

At = [merge, [At-1,Bt-1]]

The **same** logic is applied to the other rows with *Col Sums* > 1

In the case of **Ct,** the dictionary value will come in the following form:

Ct = [split, [Ct-1]]

**Step 2: Looking at Sums Row**

Here we look at the columns. In the case of **At-1** we see that the Sums Row has the a value greater than 1. Therefore we categorize it as a **split**.

At-1 = [split, [At,Bt]]

Whereas, in the case of Dt-1 we see apply the categorization of **merge**.

Dt-1 = [merge,[Et]]

**Step 3: Generating Combined Categorization**

Once we have generated the two dictionaries, we are able to categorize the different relations. Once again, we refer to the example above. You can see the discussion of the code performing this in Appendix x.

**Step 3.1: Simple Split**

In the case of a simple split (from our example) we will see the following dictionary entry:

Ct=[split,[Ct-1]]

Dt=[split,[Ct-1]]

Ct-1=[split]

These values allow us to determine the in what type CN code relations Ct and Dt are.

Specifically we look at Ct-1 and check whether it is present within the CN for year t codes t. If that is the case, we apply the value from Ct-1 to the year t code entries.

Ct=[split,[Ct-1],split]

Dt=[split,[Ct-1],split]

Then we proceed and loop over the combinations. In the cases where there are two “split” values, we take the year t-1 code use it as a reference category and take the year t values and store them on the right. Therefore, the final result will be as follows:

Ct-1 = [split, [Ct,Dt]]

Therefore, on the left we will see the CN code that needs to be split from, whereas on the right we see the codes we need to split into.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | At-1 | Bt-1 | Ct-1 | Dt-1 | Et-1 | Col Sums |
| At |  |  |  |  |  |  |
| Bt |  |  |  |  |  |  |
| Ct |  |  | 1 |  |  | 1 |
| Dt |  |  | 1 |  |  | 1 |
| Et |  |  |  |  |  |  |
| Ft |  |  |  |  |  |  |
| Sum |  |  | 2 |  |  |  |

**Step 3.2: Simple Merge**

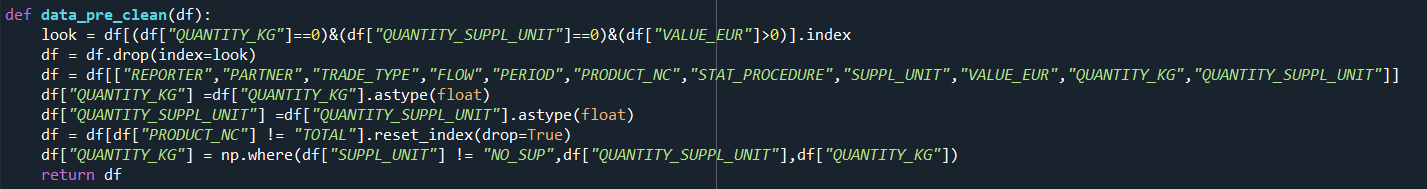
**Step 3.3: Merge-Split**

## Pre-Cleaning

The **pre-cleaning** function (Figure x) allows us to do some data cleaning prior to applying any relevant changes and editing. To begin with, ***line 1***allows us to identify all indices of rows with no quantities. Second, we work only with only certain columns:

1. Reporter: This is the reporter of the (import/export) (e.g. the geographic unit from which/to which the goods moves from).
2. Partner: This the recipient of an export/the performer of an export.
3. Trade Type: This variable shows whether the trade relationship is an import or export relationship.
4. Flow:
5. Period: This is the period within which the trade occurs (structure – YEARMONTH)
6. Product\_NC (CN8 code): This is the comm
7. Stat Procedure:
8. Supplemental Unit (SUPPL\_UNIT):
9. Value in Euros (VALUE\_EUR):
10. Quantities in KG (QUANTITY\_KG):
11. Quantity in Supplemental Units (QUANTITY\_SUPPL\_UNIT):

In ***line 3*** we set the quantity variable in float format to allow us to perform calculations (same thing is applied to supplemental quantity). In ***line 6*** we exclude the rows from the dataset that provide the yearly total, given that the methodology by Eurostat takes a bottom-up approach. Finally, in ***line 7*** we force to the dataset to replace the quantity variable to be the supplemental unit (rather than KG), given that the Eurostat methodology assumes always the use of supplemental units.



## Adjusting Common Nomenclature Codes

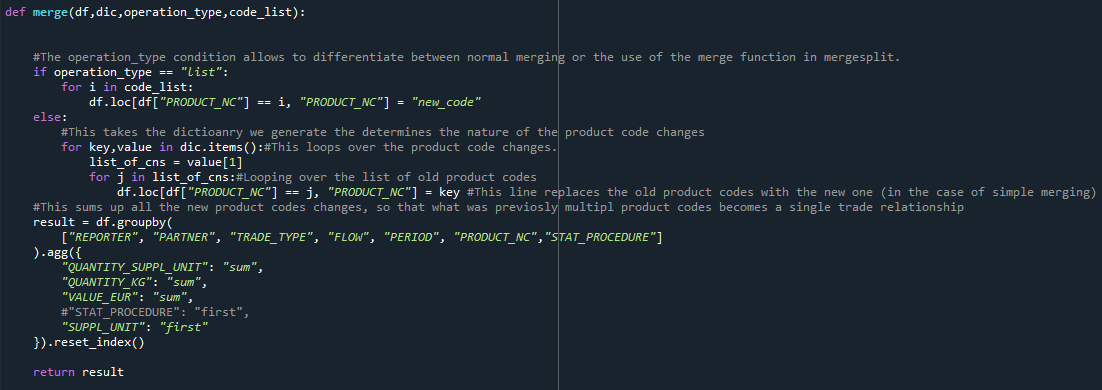
### Merging Codes

Simple merging of codes is relatively the simplest of all processes. This is built on the following structure. After determining precisely which codes are to be merged, and which are not, we call the **merge** function. This function takes the following variables:

1. df: The dataset that is to be modified (y-1)
2. dic: This is the dictionary with only the CN codes relations that are to be merged
3. operation\_type: Only needed in the case of merge-split operations
4. code\_list: Only needed in the case of merge-split operations

As can be seen in Figure x, the function loops over a dictionary of relevant codes changes. It then takes the values, or the CN Codes in y-1 which will need to be replaced by the new (merged code) which in python terms is stored in the dictionary key. It does this for all trade relations available within the dataset.

Following, controlling for trade variables such as the partner, the period, the CN Product, etc. we sum the Quantities and the Value of each monthly trade relationship. This is done, given that the previous multiple codes are transformed into a single code, allowing us to treat it as a single trade relation in year y.



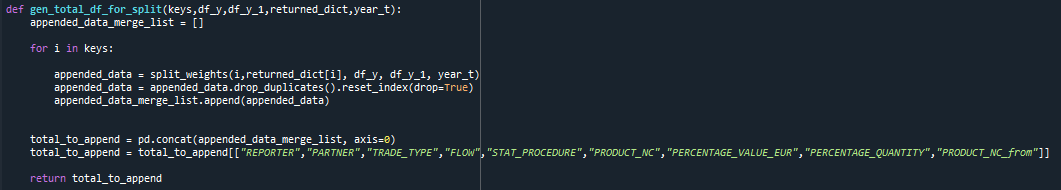
### Splitting Codes

#### Generating New Rows

When performing a simple splitting of codes two functions are applied (Figure x & x).

The outer function is called **gen\_total\_df\_for\_split** (Figure x)**.** The purpose of this function is to generate the new rows that will be appended to the t-1 year dataframe. The function takes the following inputs:

1. **Keys:** These are the CN codes in year t-1 that we need to split.
2. **Df\_y:** This is the dataframe that is for year t (and will be needed to calculate the weights).
3. **Df\_y\_1:** This is the dataframe that contains the trade data for year t-1.
4. **Returned\_dict:** This is the dictionary with the different CN code changes categorized.
5. **Year\_t:** This is the year t that is currently being processed.



The function generates an empty list to store the newly generated CN codes. It achieves this by calling the CN codes (keys) for the year t-1, and loops over them. These are used as an input into the split\_weights function (described below). The appended\_data that is returned by this function provides the weights that are to be ablllied to the year t-1 dataset, as well as the individual weights for each available trade relationship.

The reason we apply this reweighting is that the Eurostat methodological note requires us to apply the yearly mean of Unit Values to all split codes:

However, given that we cannot precisely know the precise share of the individual new CN codes in the year t-1 (given this data is not available publicly) we attempt to ensure consistency by assuming:

This equation means that for all new codes J the share of value/quantity of new CN code j will be the same in year t and t-1 for each trade relation. This allows us to estimate the indices at a later point, while ensuring consistency over the total trade.

Finally, the code cleans up duplicates and returns a final, fully generated dataframe of the new CN codes and their weighting. This will be added later to the function.

#### Generating Weights

Prior to discussing the final stage of splitting, we discuss shortly the way the weights for re-weighting are generated. The reweighting of values and quantities for the newly split product CN codes occurs using the function **split\_weights**. The function takes 5 inputs:

1. **Code\_y\_1:** This is the code from which we need to split from in year t-1.
2. **Value:** This is the list of CN codes we need to split into.
3. **Df\_y:** This is the dataframe for year t.
4. **Df\_y\_1:** This is the dataframe for year t-1.
5. **Year\_t:** This is the year t.

The function, as can be seen in Figure X, is relatively long. To begin with the

codes= value[1][0:]

extracts the CN codes that we will split into. These codes are then used to filter the trade relations we are interested in in the year t dataframe.

filtered\_y = df\_y[df\_y["PRODUCT\_NC"].isin(codes)]

This line of code reduces the processing required to calculate the year t weights, therefore, allowing us to generate totals for each new code j using the python **groupby** **function**:

aggregated\_data = (

filtered\_y

.groupby(group\_cols)

.agg({"VALUE\_EUR": "sum", "q": "sum"})

.reset\_index()

)

We control for relevant variables that form the precise trade relationship. That is, "REPORTER", "PARTNER", "TRADE\_TYPE", "FLOW", "STAT\_PROCEDURE", "PRODUCT\_NC". This ensures relative consistency throughout time. Therefore, this function generates the **nominator.**

To generate the **denominator** of the weights we take the previously generated dataframe, and create their group totals (i.e. the J in eq. X).

total\_values = (

aggregated\_data

.groupby(["REPORTER", "PARTNER", "TRADE\_TYPE", "FLOW", "STAT\_PROCEDURE"])

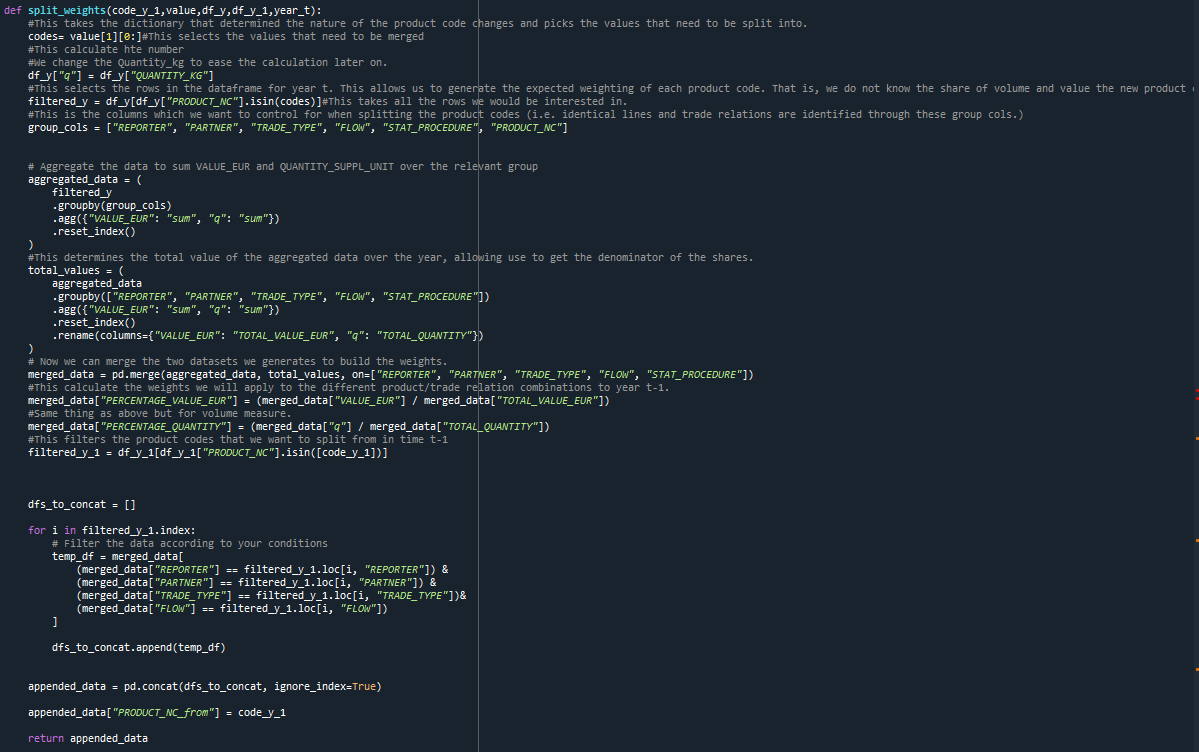
.agg({"VALUE\_EUR": "sum", "q": "sum"})

.reset\_index()

.rename(columns={"VALUE\_EUR": "TOTAL\_VALUE\_EUR", "q": "TOTAL\_QUANTITY"})

)

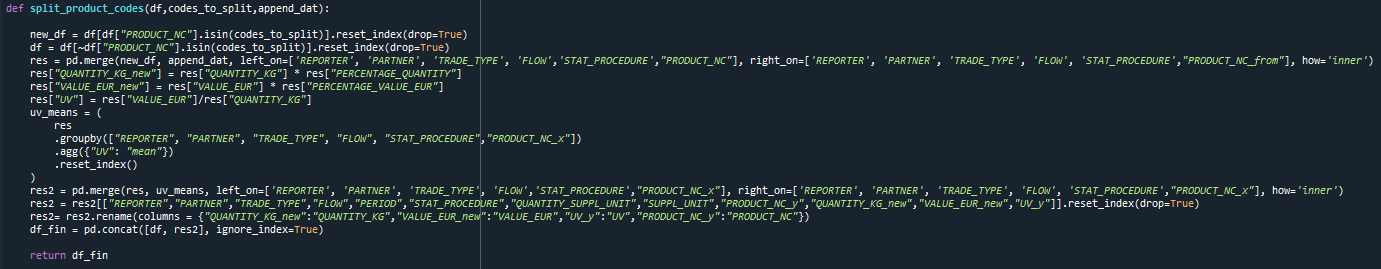
After generating these two datasets, we merge them together, generate the weights both for value and quantity, and combine them, allowing us to replace the trade relations with the CN code of year t-1. To ease later on the process of replacing the CN code of year t-1, we generate a variable with the code we are splitting from.



#### Splitting Product Codes

This stage of the splitting is achieved using the function **split\_product\_codes**. The function takes the following inputs:

1. **df:** Is the dataframe for year t-1.
2. **codes\_to\_split:** This is codes from which we will split.
3. **append\_dat:** This is the data we generated using **gen\_total\_df\_for\_split.**



In the first line we generate a dataframe that includes only the data with the CN code t-1 we will split from. The second line excludes precisely those lines.

We combine the dataframe with exclusive the CN code that need to be split in t-1, as well as the weights we generated previously.

res = pd.merge(new\_df, append\_dat, left\_on=['REPORTER', 'PARTNER', 'TRADE\_TYPE', 'FLOW','STAT\_PROCEDURE',"PRODUCT\_NC"], right\_on=['REPORTER', 'PARTNER', 'TRADE\_TYPE', 'FLOW', 'STAT\_PROCEDURE',"PRODUCT\_NC\_from"], how='inner')

After this, we reweight the quantities and values for each CN code, and generate the Unit Value using the unweighted quantities and values. The mean of the Unit Value for the whole year t-1 is the generated, and applied unanimously on all split codes. Finally, we merge the Unit Value Means we generated, the weighted quantities and values, and select only those as relevant information for our next steps of the COMEXT script.

### Merge-Split

## Running the Script

# Script 2: Determining Outliers

# Script 3: Generating Unit Value Index and Volume Index