Code_Sample_Python

July 6, 2026

This notebook prepares the bilateral trade data from OECD for the project "Product Reproducibility and Trade Patterns: Evidence from Cultural Goods".

OECD BIMTS 4 digit HS code level bulk data comes in two files spliting on the time period. Here I am considering 1995 to 2024. Cleaning and preparing of the data take the following steps:
> - a. Renaming the exporter and importer columns

- b. Removing self loops and duplicate rows and keeping the first instance only
- c. Counting hysteresis for all three product categories
- d. Sorting values by country pair and time period and reseting index
- Aggregating the unique and reproducible cultural goods data on country pair and time, while tracking the HS codes and number of products traded among each pair in a particular year in the column 'PRODUCT_HS' and 'product_count'

So this notebook will generate three different csv files in the 'cleaned' subfolder of the folder 'data', for unique and reproducible cultural goods, and non-cultural goods trade.

Step 1: Importing data

[ ]:
```
import pandas as pd
```

[ ]:
```
# Started: 12:13AM, Ended: 12:18AM
oecd = []
dtype_dict = {"REF_AREA": "category","COUNTERPART_AREA":␣
 ↪"category","PRODUCT_HS": "category","ADJUSTMENT": "category", "TIME_PERIOD":␣
 ↪"int16","OBS_VALUE": "float32"}
cols = ["REF_AREA","COUNTERPART_AREA","PRODUCT_HS","TIME_PERIOD","OBS_VALUE",␣
 ↪"ADJUSTMENT"]

for year in range(1995, 2025):
    file = f'../data/raw/HS17_4D_DE_Bulk_{year}.csv'
    df = pd.read_csv(file, sep='|', usecols = cols, dtype = dtype_dict)
    df["year_file"] = year
    oecd.append(df)
t = pd.concat(oecd, ignore_index=True)
```

[ ]:
```
t.head(2)
```

[ ]:
```
t = t[t['ADJUSTMENT'] == 'B_ADJ_RX']
t = t.rename(columns = {'REF_AREA': 'iso_o'})
t = t.rename(columns = {'COUNTERPART_AREA': 'iso_d'})
t = t[t['iso_o'] != t['iso_d']]
```

[ ]:
```
t.shape
```

[ ]:
```
print(t['iso_o'].nunique())
print(t['iso_d'].nunique())
print(t['TIME_PERIOD'].nunique())
print(t['PRODUCT_HS'].nunique())
print(t['TIME_PERIOD'].unique())
print(t.groupby(['iso_o', 'iso_d']).ngroups)
```

Step 2: Data cleaning

[ ]:
```
t = t[t['iso_o'] != 'W']
t = t[t['iso_d'] != 'W']
```

[ ]:
```
print(t.groupby(['iso_o', 'iso_d']).ngroups)
```

[ ]:
```
t_dupli = t.duplicated(keep = 'last')
t_unique = t[~t_dupli ]
```

[ ]:
```
t_unique.shape
```

Step 3: Aggregation

[ ]:
```
unique_codes = ['HS17_9701','HS17_9702','HS17_9703','HS17_9706']
reproducible_codes = ['HS17_4901','HS17_4902','HS17_4904','HS17_3706',␣
 ↪'HS17_8523']
```

[ ]:
```
unique = t_unique[t_unique['PRODUCT_HS'].isin(unique_codes)]
reproducible = t_unique[t_unique['PRODUCT_HS'].isin(reproducible_codes)]
total = t_unique[t_unique['PRODUCT_HS'] == '_T']
```

[ ]:
```
alt_total = t_unique[~t_unique['PRODUCT_HS'].isin(unique_codes)]
alt_total = alt_total[~alt_total['PRODUCT_HS'].isin(reproducible_codes)]
alt_total = alt_total[alt_total['PRODUCT_HS'] != '_T']
```

[ ]:
```
len(set(t_unique['PRODUCT_HS'].unique()) - set(alt_total['PRODUCT_HS'].
 ↪unique()))
```

[ ]:
```
alt_total = (alt_total.groupby(["iso_o", "iso_d", "TIME_PERIOD"]).
 ↪agg({"OBS_VALUE": "sum","PRODUCT_HS": lambda x: tuple(sorted(set(x)))}).
 ↪reset_index())
```

[ ]:
```
alt_total['product_count'] = alt_total['PRODUCT_HS'].apply(len)
```

```
[ ]: alt_total.head(2)
```

```
[ ]: unique = unique.groupby(['iso_o', 'iso_d', 'TIME_PERIOD']).agg({'OBS_VALUE':␣
     ↪'sum', 'PRODUCT_HS': lambda x :tuple(sorted(set(x)))}).reset_index()
```

```
[ ]: unique['product_count'] = unique['PRODUCT_HS'].apply(len)
```

```
[ ]: reproducible = reproducible.groupby(['iso_o', 'iso_d', 'TIME_PERIOD']).
     ↪agg({'OBS_VALUE': 'sum', 'PRODUCT_HS': lambda x :tuple(sorted(set(x)))}).
     ↪reset_index()
```

```
[ ]: reproducible['product_count'] = reproducible['PRODUCT_HS'].apply(len)
```

```
[ ]: total = total.rename(columns = {'OBS_VALUE': 'total'})
     reproducible = reproducible.rename(columns = {'OBS_VALUE': 'reproducible'})
     unique = unique.rename(columns = {'OBS_VALUE': 'unique'})
     alt_total = alt_total.rename(columns = {'OBS_VALUE': 'total'})
```

```
[ ]: t = total.merge(unique, on = ['iso_o', 'iso_d', 'TIME_PERIOD'], how='left').
     ↪merge(reproducible, on = ['iso_o', 'iso_d', 'TIME_PERIOD'], how='left')
```

```
[ ]: t[['unique', 'reproducible', 'total']] = (t[['unique', 'reproducible',␣
     ↪'total']].fillna(0))
```

```
[ ]: t['noncultural_trade'] = (t['total'] - t['unique']- t['reproducible'])
```

```
[ ]: #alt_total[(alt_total['iso_o'] == 'SACU') & (alt_total['iso_d'] == 'DOM')]
     #t[(t['iso_o'] == 'SACU') & (t['iso_d'] == 'DOM')]
```

Both approaches produce identical values.

We therefore retain alt_total for subsequent analyses.

```
[ ]: alt_total = alt_total.rename(columns = {'total':'noncultural_trade'})
```

```
[ ]: alt_total = alt_total.reset_index(drop = True)
     unique = unique.reset_index(drop = True)
     reproducible = reproducible.reset_index(drop = True)
```

Step 4: Calculating hysteresis

```
[ ]: memory = 0.3
     def compute_hysteresis(series):
         """ This function computes the hysteresis measure for a bilateral trade␣
      ↪series.
         Parameters: series : Annual bilateral trade values ordered by year.
         Returns: list: Hysteresis values for each observation.
         """
         hyst = []
```

```
    prev_cum = 0
    prev_val = 0
    for val in series:
        hyst_val = memory * prev_cum + prev_val
        hyst.append(hyst_val)
        prev_cum = hyst_val
        prev_val = val
    return hyst
```

[ ]:
```
unique['hysteresis_unique'] = (unique.groupby(['iso_o', 'iso_d'])['unique'].
 ↪transform(compute_hysteresis))
```

[ ]:
```
reproducible['hysteresis_reproducible'] = (reproducible.groupby(['iso_o',␣
 ↪'iso_d'])['reproducible'].transform(compute_hysteresis))
```

[ ]:
```
alt_total['hysteresis_noncultural_trade'] = (alt_total.groupby(['iso_o',␣
 ↪'iso_d'])['noncultural_trade'].transform(compute_hysteresis))
```

[ ]:
```
alt_total = alt_total.drop(columns = 'PRODUCT_HS')
unique = unique.drop(columns = 'PRODUCT_HS')
reproducible = reproducible.drop(columns = 'PRODUCT_HS')
```

[ ]:
```
assert alt_total['iso_o'].nunique() == alt_total['iso_d'].nunique()
```

[ ]:
```
assert unique['iso_o'].nunique() == unique['iso_d'].nunique()
```

[ ]:
```
assert reproducible['iso_o'].nunique() == reproducible['iso_d'].nunique()
```

[ ]:
```
unique = unique.sort_values(['iso_o', 'iso_d', 'TIME_PERIOD'])
reproducible = reproducible.sort_values(['iso_o', 'iso_d', 'TIME_PERIOD'])
alt_total = alt_total.sort_values(['iso_o', 'iso_d', 'TIME_PERIOD'])
```

[ ]:
```
alt_total.head(2)
```

[ ]:
```
assert unique['iso_o'].nunique() == reproducible['iso_o'].nunique() ==␣
 ↪alt_total['iso_o'].nunique()
```

[ ]:
```
assert unique['iso_d'].nunique() == reproducible['iso_d'].nunique() ==␣
 ↪alt_total['iso_d'].nunique()
```

Step 5: Exporting data

[ ]:
```
unique.to_csv("../data/cleaned/unique.csv", encoding = 'utf-8', index = False)
reproducible.to_csv("../data/cleaned/reproducible.csv", encoding = 'utf-8',␣
 ↪index = False)
alt_total.to_csv("../data/cleaned/noncultural.csv", encoding = 'utf-8', index =␣
 ↪False)
```