1.1_main_script_gas

March 31, 2025

[ ]:

```
import numpy as np
import os
import pymrio
import pandas as pd
from IPython.display import display
import country_converter as coco
from scipy.sparse.linalg import inv
import scipy.sparse as sp
import matplotlib.pyplot as plt
```

[ ]:

```
pd.set_option('display.max_rows', 200)
pd.set_option('display.max_columns', 100)
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', 100)
```

[ ]:

```
# Define EU28 country codes
eu28_countries = [
    "AT", "BE", "BG", "CY", "CZ", "DE", "DK", "EE", "ES", "FI", "FR", "GB",␣
  ↪"GR", "HR", "HU",
    "IE", "IT", "LT", "LU", "LV", "MT", "NL", "PL", "PT", "RO", "SE", "SI", "SK"
]
```

[ ]:

```
# Define the storing folder for Exiobase3 data
exio3_folder = 'C:/Users/danie/Nextcloud/Coding/Masterthesis/exiobase'
download_folder = os.path.join(exio3_folder, 'exio_download')

# Check if the exio_download folder exists, create if not
if not os.path.exists(download_folder):
    os.makedirs(download_folder)
    print(f"Created directory: {download_folder}")
else:
    print(f"Directory already exists: {download_folder}")

# Download Exiobase3 data to the specified folder
exio_downloadlog = pymrio.download_exiobase3(storage_folder=download_folder,␣
  ↪system="ixi", years=[2018, 2019, 2020, 2021, 2022])
print(exio_downloadlog)
```

[ ]:

```
# Parse Exiobase3 (2021) data
exio3 = pymrio.parse_exiobase3(path='C:/Users/danie/Nextcloud/Coding/
 ↪Masterthesis/exiobase/exio_download/IOT_2021_ixi.zip')
```

[ ]:

```
# Assess meta data
print(exio3.meta)
```

[ ]:

```
### Check for geographical sampling differences between FIGARO and EXIOBASE 3␣
 ↪###

# FIGARO countries
figaro_countries = [
    'AR', 'AT', 'AU', 'BE', 'BG', 'BR', 'CA', 'CH', 'CN', 'CY', 'CZ', 'DE',␣
 ↪'DK', 'EE', 'ES', 'FI', 'FIGW1', 'FR', 'GB',
    'GR', 'HR', 'HU', 'ID', 'IE', 'IN', 'IT', 'JP', 'KR', 'LT', 'LU', 'LV',␣
 ↪'MT', 'MX', 'NL', 'NO', 'PL', 'PT', 'RO', 'RU',
    'SA', 'SE', 'SI', 'SK', 'TR', 'US', 'ZA'
]

# Extract country codes from EXIOBASE 3 dataset
exio_countries = exio3.get_regions()

# Compare country codes
common_countries = sorted(set(figaro_countries).intersection(exio_countries))
figaro_only_countries = sorted(set(figaro_countries) - set(exio_countries))
exio_only_countries = sorted(set(exio_countries) - set(figaro_countries))

print("Common countries:", common_countries)
print("Countries only in FIGARO:", figaro_only_countries)
print("Countries only in EXIOBASE 3:", exio_only_countries)
```

[ ]:

```
# Argentina and Saudi Arabia are not in EXIOBASE 3, but in FIGARO
# Taiwan is in EXIOBASE 3, but not in FIGARO
# FIGW1 is ROW in Figaro
# WA (Asia), WE (Europe), WF (Africa), WL (Latin America), WM (Middle East), WP␣
 ↪(Pacific) are ROW regions in Exiobase3

# Collect all RoW regions and Taiwan in one FIGARO region
exio3.rename_regions({'WA': 'FIGW1', 'WE': 'FIGW1', 'WF': 'FIGW1', 'WL':␣
 ↪'FIGW1', 'WM': 'FIGW1', 'WP': 'FIGW1', 'TW': 'FIGW1'})

# Aggregate EXIOBASE 3 data to FIGARO regions
exio3.aggregate_duplicates(inplace=True)
exio3.Z.to_csv('C:/Users/danie/Nextcloud/Coding/Masterthesis/exiobase/
 ↪country_agg_exio3_figaro.csv')
```

[ ]:

```
# Check available classification data that contains possibly useful different␣
 ↪names and aggregation levels
mrio_class = pymrio.get_classification(mrio_name='exio3_ixi')
```

[ ]:

```
# Display the full mrio_class
display(mrio_class)
```

[ ]:

```
# Create a conversion dictionnary from ExioName to ExioLabel and check for␣
 ↪correctness by displaying it
conv_dict = mrio_class.get_sector_dict(mrio_class.sectors.ExioName, mrio_class.
 ↪sectors.ExioLabel)
display(conv_dict)

# Rename sectors in the pymrio object
exio3.rename_sectors(conv_dict)

# Check if the renaming was successful
print(exio3.Z.index)
```

[ ]:

```
### Aggregate Exiobase3 data ###
# Done through renaming, which also helps to adapt it to eurostat data

# Renaming of sectors requires mappping of ExioLabel to NACE classification

rename_dict_exio3_NACE = {
    "A_PARI": "A01",
    "A_WHEA": "A01",
    "A_OCER": "A01",
    "A_FVEG": "A01",
    "A_OILS": "A01",
    "A_SUGB": "A01",
    "A_FIBR": "A01",
    "A_OTCR": "A01",
    "A_CATL": "A01",
    "A_PIGS": "A01",
    "A_PLTR": "A01",
    "A_OMEA": "A01",
    "A_OANP": "A01",
    "A_MILK": "A01",
    "A_WOOL": "A01",
    "A_MANC": "A01",
    "A_MANB": "A01",
    "A_FORE": "A02",
    "A_FISH": "A03",
    "A_GASE": "B_gas",
    "A_OGPL": "B_gas",
    "A_COAL": "B_nongas",
    "A_COIL": "B_nongas",
    "A_ORAN": "B_nongas",
    "A_IRON": "B_nongas",
    "A_COPO": "B_nongas",
    "A_NIKO": "B_nongas",
    "A_ALUO": "B_nongas",
    "A_PREO": "B_nongas",
    "A_LZTO": "B_nongas",
    "A_ONFO": "B_nongas",
    "A_STON": "B_nongas",
    "A_SDCL": "B_nongas",
    "A_CHMF": "B_nongas",
    "A_PCAT": "C10-12",
    "A_PPIG": "C10-12",
    "A_PPLT": "C10-12",
    "A_POME": "C10-12",
    "A_VOIL": "C10-12",
    "A_DAIR": "C10-12",
    "A_RICE": "C10-12",
    "A_SUGR": "C10-12",
    "A_OFOD": "C10-12",
    "A_BEVR": "C10-12",
    "A_FSHP": "C10-12",
    "A_TOBC": "C10-12",
    "A_TEXT": "C13-15",
    "A_GARM": "C13-15",
    "A_LETH": "C13-15",
    "A_WOOD": "C16",
    "A_WOOW": "C16",
    "A_PULP": "C17",
    "A_PAPR": "C17",
    "A_PAPE": "C17",
    "A_MDIA": "C18",
    "A_COKE": "C19",
    "A_REFN": "C19",
    "A_PLAS": "C20_21",
    "A_PLAW": "C20_21",
    "A_NFER": "C20_21",
    "A_PFER": "C20_21",
    "A_CHEM": "C20_21",
    "A_RUBP": "C22",
    "A_GLAS": "C23",
    "A_GLAW": "C23",
    "A_CRMC": "C23",
    "A_BRIK": "C23",
    "A_CMNT": "C23",
    "A_ASHW": "C23",
    "A_ONMM": "C23",
    "A_NUCF": "C24",
    "A_STEL": "C24",
    "A_STEW": "C24",
    "A_PREM": "C24",
    "A_PREW": "C24",
    "A_ALUM": "C24",
    "A_ALUW": "C24",
    "A_LZTP": "C24",
    "A_LZTW": "C24",
    "A_COPP": "C24",
    "A_COPW": "C24",
    "A_ONFM": "C24",
    "A_ONFW": "C24",
    "A_METC": "C24",
    "A_FABM": "C25-33",
    "A_MACH": "C25-33",
    "A_OFMA": "C25-33",
    "A_ELMA": "C25-33",
    "A_RATV": "C25-33",
    "A_MEIN": "C25-33",
    "A_MOTO": "C25-33",
    "A_OTRE": "C25-33",
    "A_FURN": "C25-33",
    "A_POWC": "D35",
    "A_POWG": "D35",
    "A_POWN": "D35",
    "A_POWH": "D35",
    "A_POWW": "D35",
    "A_POWP": "D35",
    "A_POWB": "D35",
    "A_POWS": "D35",
    "A_POWE": "D35",
    "A_POWO": "D35",
    "A_POWM": "D35",
    "A_POWZ": "D35",
    "A_POWT": "D35",
    "A_POWD": "D35",
    "A_GASD": "D35",
    "A_HWAT": "D35",
    "A_WATR": "E36",
    "A_RYMS": "E37-39",
    "A_BOTW": "E37-39",
    "A_INCF": "E37-39",
    "A_INCP": "E37-39",
    "A_INCL": "E37-39",
    "A_INCM": "E37-39",
    "A_INCT": "E37-39",
    "A_INCW": "E37-39",
    "A_INCO": "E37-39",
    "A_BIOF": "E37-39",
    "A_BIOP": "E37-39",
    "A_BIOS": "E37-39",
    "A_COMF": "E37-39",
    "A_COMW": "E37-39",
    "A_WASF": "E37-39",
    "A_WASO": "E37-39",
    "A_LANF": "E37-39",
    "A_LANP": "E37-39",
    "A_LANL": "E37-39",
    "A_LANI": "E37-39",
    "A_LANT": "E37-39",
    "A_LANW": "E37-39",
    "A_CONS": "F",
    "A_CONW": "F",
    "A_TDMO": "G45",
    "A_TDWH": "G46",
    "A_TDFU": "G47",
    "A_TDRT": "G47",
    "A_TRAI": "H49",
    "A_TLND": "H49",
    "A_TPIP": "H49",
    "A_TWAS": "H50",
    "A_TWAI": "H50",
    "A_TAIR": "H51",
    "A_TAUX": "H52",
    "A_PTEL": "H53",
    "A_HORE": "I",
    "A_COMP": "J",
    "A_FINT": "K64",
    "A_FINS": "K65",
    "A_FAUX": "K66",
    "A_REAL": "L68",
    "A_RESD": "M_N",
    "A_OBUS": "M_N",
    "A_MARE": "M_N",
    "A_PADF": "O84",
    "A_EDUC": "P85",
    "A_HEAL": "Q",
    "A_RECR": "R_S",
    "A_ORGA": "R_S",
    "A_OSER": "R_S",
    "A_PRHH": "T",
    "A_EXTO": "U"
```

```
}
```

[ ]:
```
# Apply mapping with the rename_sectors tool of pymrio
exio3.rename_sectors(rename_dict_exio3_NACE)
print(exio3.Z.index)

# Aggregate duplicates
exio3.aggregate_duplicates()
print(exio3.Z)
```

[ ]:
```
exio_Z_df = exio3.Z

display(exio_Z_df)
```

[ ]:
```
# Extract energy inputs for B_gas and B_nongas (rows)
energy_inputs = exio_Z_df.loc[(slice(None), ['B_gas', 'B_nongas']), :]

# Compute total energy input per row (sum of B_gas and B_nongas)
total_energy_inputs_row = energy_inputs.groupby(level=0).sum()
total_energy_inputs_row.replace(0, np.nan, inplace=True)  # Avoid division by␣
 ↪zero

# Compute row-wise shares (only for B_gas and B_nongas rows)
energy_shares_row = energy_inputs.div(total_energy_inputs_row, level=0).
 ↪fillna(0)

# Create a copy of the original DataFrame to preserve structure
exio_Z_with_row_shares = exio_Z_df.copy()

# Add energy share values to the original dataframe (only updating B_gas and␣
 ↪B_nongas rows)
exio_Z_with_row_shares.loc[energy_shares_row.index] = energy_shares_row

# Display the full table with updated B_gas and B_nongas rows
print(exio_Z_with_row_shares)
```

[ ]:
```
# Create a copy of the DataFrame to store both row-wise and column-wise shares
exio_Z_with_column_shares = exio_Z_df.copy()

# Identify energy-related columns (MultiIndex: first level = country, second␣
 ↪level = B_gas/B_nongas)
energy_columns = ['B_gas', 'B_nongas']

# Iterate over each country in the column MultiIndex
for country in exio_Z_df.columns.levels[0]:  # First level = country names
```

```
    # Define MultiIndex column tuples for the current country's B_gas and␣
 ↪B_nongas
    country_energy_cols = [(country, col) for col in energy_columns]

    # Ensure the country has both B_gas and B_nongas columns (avoid KeyErrors)
    existing_energy_cols = [col for col in country_energy_cols if col in␣
 ↪exio_Z_df.columns]

    if len(existing_energy_cols) < 2:  # Skip countries missing B_gas or␣
 ↪B_nongas
        continue

    # Compute total energy input per row (sum of only that country's B_gas and␣
 ↪B_nongas values)
    total_energy_inputs_row = exio_Z_df.loc[:, existing_energy_cols].sum(axis=1)

    # Avoid division by zero
    total_energy_inputs_row.replace(0, np.nan, inplace=True)

    # Compute column-wise shares for that country's B_gas and B_nongas
    energy_shares_col = exio_Z_df.loc[:, existing_energy_cols].
 ↪div(total_energy_inputs_row, axis=0).fillna(0)

    # Store the computed shares into the existing table
    exio_Z_with_column_shares.loc[:, existing_energy_cols] = energy_shares_col

# Display the final DataFrame
print(exio_Z_with_column_shares)
```

[ ]:

```
# A condition has to be added, that if both B_gas and B_nongas are zero,␣
 ↪B_nongas should be 1 and B_gas should be 0
```

[ ]:

```
# Step 1: Create a new DataFrame with the same structure, filled with 1s
exio_Z_final = pd.DataFrame(1, index=exio_Z_df.index, columns=exio_Z_df.columns)

# Step 2: Identify B_gas and B_nongas rows and columns
row_sectors = ["B_gas", "B_nongas"]
column_sectors = ["B_gas", "B_nongas"]

# Step 3: Apply row-wise shares **excluding B quadrants**
exio_Z_final.loc[exio_Z_df.index.get_level_values(1).isin(row_sectors),
                 ~exio_Z_df.columns.get_level_values(1).isin(column_sectors)] =␣
 ↪(
    exio_Z_with_row_shares.loc[exio_Z_df.index.get_level_values(1).
 ↪isin(row_sectors),
                               ~exio_Z_df.columns.get_level_values(1).
 ↪isin(column_sectors)].values
)

# Step 4: Apply column-wise shares **excluding B quadrants**
exio_Z_final.loc[~exio_Z_df.index.get_level_values(1).isin(row_sectors),
                 exio_Z_df.columns.get_level_values(1).isin(column_sectors)] = (
    exio_Z_with_column_shares.loc[~exio_Z_df.index.get_level_values(1).
 ↪isin(row_sectors),
                                  exio_Z_df.columns.get_level_values(1).
 ↪isin(column_sectors)].values
)

# Step 5: Assign B quadrant shares for each (supplier country, consuming␣
 ↪country) pair
for supplier_country in exio_Z_df.index.get_level_values(0).unique():
    for consuming_country in exio_Z_df.columns.get_level_values(0).unique():

        # Identify the specific 2x2 quadrant for this country pair
        supplier_rows = (exio_Z_df.index.get_level_values(0) ==␣
 ↪supplier_country) & (exio_Z_df.index.get_level_values(1).isin(row_sectors))
        consumer_cols = (exio_Z_df.columns.get_level_values(0) ==␣
 ↪consuming_country) & (exio_Z_df.columns.get_level_values(1).
 ↪isin(column_sectors))

        # Extract the total energy input for this quadrant from original exio3.Z
        b_quadrant_values = exio3.Z.loc[supplier_rows, consumer_cols]

        # Sum total energy input for this quadrant
        total_energy_quadrant = b_quadrant_values.sum().sum()

        # Normalize: Compute each cell's share of the total energy input
        if total_energy_quadrant > 0:  # Avoid division by zero
            b_quadrant_shares = b_quadrant_values / total_energy_quadrant
        else:
            b_quadrant_shares = b_quadrant_values * 0  # If no energy input,␣
 ↪set to zero

        # Apply the modification: If both B_gas and B_nongas are 0, set␣
 ↪B_nongas = 1, B_gas = 0
        if (b_quadrant_shares.loc[(supplier_country, "B_gas"),␣
 ↪(consuming_country, "B_gas")] == 0 and
            b_quadrant_shares.loc[(supplier_country, "B_nongas"),␣
 ↪(consuming_country, "B_nongas")] == 0):
```

```
            b_quadrant_shares.loc[(supplier_country, "B_nongas"),␣
 ↪(consuming_country, "B_nongas")] = 1
            b_quadrant_shares.loc[(supplier_country, "B_gas"),␣
 ↪(consuming_country, "B_gas")] = 0

        # Insert the calculated shares back into the final DataFrame
        exio_Z_final.loc[supplier_rows, consumer_cols] = b_quadrant_shares.
 ↪values

# Step 6: Done! The final DataFrame now correctly integrates all three parts
print(exio_Z_final)
```

[ ]:
```
# Save the final DataFrame
exio_Z_final.to_csv('C:/Users/danie/Nextcloud/Coding/Masterthesis/exiobase/
 ↪exio3_ixi_2021_energy_shares.csv')
```

[ ]:
```
### Repeat similar prcedure for the Y matrix (demand) as a prerequesite for cpi␣
 ↪weighing ###
```

[ ]:
```
# Insepct the Y matrix
display(exio3.Y)

exio_Y_df = exio3.Y
```

[ ]:
```
# Extract private household demand for B_gas and B_nongas (rows)
energy_inputs = exio_Y_df.loc[(slice(None), ['B_gas', 'B_nongas']), :]

# Compute demand per row (sum of B_gas and B_nongas)
total_energy_inputs_row = energy_inputs.groupby(level=0).sum()
total_energy_inputs_row.replace(0, np.nan, inplace=True)  # Avoid division by␣
 ↪zero

# Compute row-wise shares (only for B_gas and B_nongas rows)
energy_shares_row = energy_inputs.div(total_energy_inputs_row, level=0).
 ↪fillna(0)

# Create a copy of the original DataFrame to preserve structure
exio_Y_row_shares = exio_Y_df.copy()

# Add energy share values to the original dataframe (only updating B_gas and␣
 ↪B_nongas rows)
exio_Y_row_shares.loc[energy_shares_row.index] = energy_shares_row

# Filter columns to only include 'Final consumption expenditure by households'
exio_Y_row_shares = exio_Y_row_shares.xs('Final consumption expenditure by␣
 ↪households', axis=1, level=1, drop_level=False)
```

```
# Display the filtered table with updated B_gas and B_nongas rows
display(exio_Y_row_shares)
```

[ ]:

```
########################################## FIGARO FIGARO FIGARO FIGARO FIGARO␣
 ↪FIGARO FIGARO FIGARO  #############################################
########################################## FIGARO FIGARO FIGARO FIGARO FIGARO␣
 ↪FIGARO FIGARO FIGARO #############################################
########################################## FIGARO FIGARO FIGARO FIGARO FIGARO␣
 ↪FIGARO FIGARO FIGARO #############################################
########################################## FIGARO FIGARO FIGARO FIGARO FIGARO␣
 ↪FIGARO FIGARO FIGARO #############################################
```

[ ]:

```
### Introducing the Figaro data ###

# Load the Figaro data and prepare for further usage
# Split the index and columns of the Figaro into a MultiIndex

def split_index_to_multiindex(df):
    """
    Split the index and columns of the DataFrame into a MultiIndex.
    The index and columns are expected to have a structure like␣
 ↪'XX_sector_code'.
    """
    def split_index(index):
        return pd.MultiIndex.from_tuples([tuple(i.split('_', 1)) for i in␣
 ↪index], names=['Country', 'Sector'])

    df.index = split_index(df.index)
    df.columns = split_index(df.columns)
    return df

def process_files_and_split_index(input_dir, output_dir):
    """
    Process all CSV files in the input folder, split the index and columns into␣
 ↪a MultiIndex,
    and save the new files in the specified output directory with a␣
 ↪'multiindex_' prefix.
    """
    # Ensure the output directory exists
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for filename in os.listdir(input_dir):
        if filename.endswith(".csv"):
            file_path = os.path.join(input_dir, filename)
            print(f"Processing {filename}...")
            df = pd.read_csv(file_path, index_col=0)
            df = split_index_to_multiindex(df)

            # Save the modified DataFrame to the output directory with␣
 ↪'multiindex_' prefix
            output_file_path = os.path.join(output_dir,␣
 ↪f'multiindex_{filename}')
            df.to_csv(output_file_path)
            print(f"Processed and saved {filename} to {output_file_path}")

# Example usage
input_dir = 'C:/Users/danie/Nextcloud/Coding/Masterthesis/data/raw/
 ↪figaro_tables'
output_dir = 'C:/Users/danie/Nextcloud/Coding/Masterthesis/notebooks/
 ↪NB_exio3_figaro_gas/figaro_multi_index'
process_files_and_split_index(input_dir, output_dir)
```

[ ]:

```
# Define the file path for the 2021 multiindex figaro table
file_path_2021 = os.path.join(output_dir, 'multiindex_2021_figaro_64.csv')

# Load the 2021 multiindex figaro table
df_2021 = pd.read_csv(file_path_2021, index_col=[0, 1], header=[0, 1])

# Display the first few rows of the dataframe to verify
print(df_2021.head())
```

[ ]:

```
### Apply sector mapping to the Figaro data ###
```

[ ]:

```
# ---------------------------------------------------
# 1. Sector Mapping (Renaming + Aggregation)
# ---------------------------------------------------

sector_mapping = {
    "C10T12": "C10-12",
    "C13T15": "C13-15",
    "C20": "C20_21",
    "C21": "C20_21",
    "E37T39": "E37-39",
    "J58": "J", "J59_60": "J", "J61": "J", "J62_63": "J",
    "M69_70": "M_N", "M71": "M_N", "M72": "M_N", "M73": "M_N", "M74_75": "M_N",
    "N77": "M_N", "N78": "M_N", "N79": "M_N", "N80T82": "M_N",
    "Q86": "Q", "Q87_88": "Q",
    "R90T92": "R_S", "R93": "R_S", "S94": "R_S", "S95": "R_S", "S96": "R_S",
    "L": "L68"
}
```

[ ]:

```
# ---------------------------------------------------
# 3. Function to Apply Sector Mapping and Aggregate
# ---------------------------------------------------

def apply_sector_mapping(df, sector_mapping):
    """
    Rename and aggregate sectors in both rows and columns using the provided␣
 ↪mapping.
    """

    #  Step 1: Rename Row Index (Industries)
    new_row_index = [(country, sector_mapping.get(sector, sector)) for country,␣
 ↪sector in df.index]
    df.index = pd.MultiIndex.from_tuples(new_row_index, names=['Country',␣
 ↪'Sector'])

    #  Step 2: Rename Column Index (Industries)
    new_col_index = [(country, sector_mapping.get(sector, sector)) for country,␣
 ↪sector in df.columns]
    df.columns = pd.MultiIndex.from_tuples(new_col_index, names=['Country',␣
 ↪'Sector'])

    #  Step 3: Aggregate Mapped Sectors
    df = df.groupby(level=['Country', 'Sector']).sum()  # Aggregate rows
    df = df.groupby(level=['Country', 'Sector'], axis=1).sum()  # Aggregate␣
 ↪columns

    return df
```

[ ]:

```
# ---------------------------------------------------
# 4. Apply Sector Mapping to FIGARO Data
# ---------------------------------------------------

df_2021_mapped = apply_sector_mapping(df_2021, sector_mapping)

# Display the first few rows to verify the aggregation worked
display(df_2021_mapped)
df_2021_mapped.to_csv('C:/Users/danie/Nextcloud/Coding/Masterthesis/notebooks/
 ↪NB_exio3_figaro_gas/figaro_mapped.csv')
```

[ ]:

```
### Merge countries in the Figaro data to match the Exiobase 3 data ###

def merge_countries(df, countries_to_merge, target='FIGW1'):
    """
    Relabels rows and columns so that any country in countries_to_merge is␣
 ↪replaced by target.
    Then groups by the MultiIndex to sum the duplicated entries.
    Assumes both rows and columns are MultiIndex with levels ['Country',␣
 ↪'Sector'].

    Parameters:
      df: pd.DataFrame with MultiIndex for both rows and columns
      countries_to_merge: list of country codes to merge (e.g., ['AR', 'SA'])
      target: the target country code to absorb the values (default 'FIGW1')

    Returns:
      A DataFrame with the specified countries merged into the target.
    """
    # Save the original index names (should be ['Country', 'Sector'])
    row_index_names = df.index.names
    col_index_names = df.columns.names

    # --- Relabel row index: Replace countries in countries_to_merge with target
    new_row_index = [
        (target if country in countries_to_merge else country, sector)
        for country, sector in df.index
    ]
    df.index = pd.MultiIndex.from_tuples(new_row_index, names=row_index_names)

    # --- Relabel column index: Replace countries in countries_to_merge with␣
 ↪target
    new_col_index = [
        (target if country in countries_to_merge else country, sector)
        for country, sector in df.columns
    ]
    df.columns = pd.MultiIndex.from_tuples(new_col_index, names=col_index_names)

    # --- Group by the MultiIndex levels to aggregate duplicate entries␣
 ↪(summing over duplicates)
    df = df.groupby(level=row_index_names).sum()
    df = df.groupby(axis=1, level=col_index_names).sum()

    return df


merged_df = merge_countries(df_2021_mapped, ['AR', 'SA'])
display(merged_df)
```

[ ]:

```
# Add gross ouput row

def add_gross_output_row(df):
    gross_output = df.sum(axis=0)
    gross_output.name = ('GO', 'GO')
    df = pd.concat([df, pd.DataFrame(gross_output).T])
    return df

df_2021_rdy = add_gross_output_row(merged_df)
display(df_2021_rdy)
```

[ ]:
```
df_2021_rdy.to_csv('C:/Users/danie/Nextcloud/Coding/Masterthesis/notebooks/
 ↪NB_exio3_figaro_gas/figaro_mapped_2021_merged.csv')
```

[ ]:
```
### Calculation of CPI weights ###



import pandas as pd

def prepare_cpi_weight_data(df):
    """
    Prepares the FIGARO final consumption data for CPI weight calculation.

    This involves:
    - Extracting only `P3_S14` columns
    - Removing the `GO` row
    - Duplicating `B` sector rows into `B_gas` and `B_nongas`

    Parameters:
      df : pd.DataFrame
          The FIGARO final demand table with MultiIndex (Country, Sector)

    Returns:
      pd.DataFrame
          Modified DataFrame with `B` sector split into `B_gas` and `B_nongas`,␣
 ↪without `GO`
    """
    final_sector = "P3_S14"

    #  **Step 1: Extract only the relevant final consumption column**
    df_fc = df.xs(final_sector, axis=1, level=1, drop_level=False)
    print(f" Extracted only `{final_sector}` from FIGARO.")

    #  **Step 2: Exclude the `GO` row**
    df_fc = df_fc[df_fc.index.get_level_values(1) != "GO"]
    print("  `GO` row successfully removed.")

    #  **Step 3: Identify `B` sector rows**
    b_rows = df_fc[df_fc.index.get_level_values(1) == "B"].copy()

    #  **Step 4: Duplicate and rename `B` sector rows**
    b_rows_gas = b_rows.copy()
    b_rows_nongas = b_rows.copy()
    b_rows_gas.index = [(c, "B_gas") for c, _ in b_rows.index]
    b_rows_nongas.index = [(c, "B_nongas") for c, _ in b_rows.index]

    #   **Step 5: Remove original `B` row and add new ones**
    df_fc = df_fc[df_fc.index.get_level_values(1) != "B"]
    df_fc = pd.concat([df_fc, b_rows_gas, b_rows_nongas]).sort_index()

    print("  `B` sector successfully split into `B_gas` and `B_nongas`.")
    return df_fc

#   **Run the preparation function**
df_cpi_ready = prepare_cpi_weight_data(df_2021_rdy)
```

[ ]:

```
display(df_cpi_ready)
```

[ ]:

```
print(exio_Y_row_shares.index)
print(exio_Y_row_shares.columns)
print(df_cpi_ready.index)
print(df_cpi_ready.columns)
```

[ ]:

```
import pandas as pd

def apply_energy_shares(df_cpi_ready, energy_shares_df):
    """
    Apply energy shares from `exio_Y_row_shares` to `B_gas` and `B_nongas` in
 ↪`df_cpi_ready`.

    Fixes:
        Ensures correct index naming for `df_cpi_ready`.
        Extracts the energy shares from the correct **column**, not index.

    Parameters:
      df_cpi_ready : pd.DataFrame
          FIGARO final consumption table with `B_gas` and `B_nongas` rows.
      energy_shares_df : pd.DataFrame
          EXIOBASE Y matrix shares for energy sectors.

    Returns:
      pd.DataFrame
          Updated `df_cpi_ready` with adjusted `B_gas` and `B_nongas` values.
    """

    #   Step 1: Ensure `df_cpi_ready` has correct index names
    df_cpi_ready.index.set_names(["Country", "Sector"], inplace=True)

    #   Step 2: Ensure energy shares dataframe has correct index names
    energy_shares_df = energy_shares_df.rename_axis(index={"region": "Country",␣
 ↪"sector": "Sector", "category": "Sector"})

    #  Step 3: Extract the column `"Final consumption expenditure by␣
 ↪households"` from the correct **country-wise** level
    final_sector_exio = "Final consumption expenditure by households"

    if final_sector_exio not in energy_shares_df.columns.get_level_values(1):
        raise KeyError(f"  `{final_sector_exio}` not found in␣
 ↪`exio_Y_row_shares` columns! Available: {energy_shares_df.columns.
 ↪get_level_values(1).unique()}")

    #  **Extract Energy Shares Per Country**
    energy_shares = energy_shares_df.xs(final_sector_exio, axis=1, level=1)
    energy_shares.columns = pd.MultiIndex.from_product([energy_shares.columns,␣
 ↪["P3_S14"]], names=["Country", "Sector"])

    print(energy_shares.index)
    print(energy_shares.columns)


    print(f"  Extracted energy shares for '{final_sector_exio}' from␣
 ↪`exio_Y_row_shares`.")

    #  Step 4: Identify affected rows (B_gas and B_nongas)
    energy_rows = df_cpi_ready.index.get_level_values(1).isin(['B_gas',␣
 ↪'B_nongas'])

    #  Step 5: Apply energy shares **only to B_gas and B_nongas**, matching by␣
 ↪country
    df_cpi_ready.loc[energy_rows] *= energy_shares

    print("  Energy shares successfully applied to `B_gas` and `B_nongas`.")

    return df_cpi_ready


#  Apply the energy shares BEFORE CPI calculation
df_cpi_ready = apply_energy_shares(df_cpi_ready, exio_Y_row_shares)
```

```
[ ]: ### Calculate CPI weights ###

     def calculate_cpi_weights(df_cpi_ready):
         """
         Compute CPI weights by normalizing each column.
```

```
    Steps:
       Divide each cell by the total column sum.
       Preserve MultiIndex structure.
       Handle cases where column sums are zero (avoid division errors).

    Parameters:
      df_cpi_ready : pd.DataFrame
          The adjusted FIGARO final consumption table with energy shares␣
 ↪applied.

    Returns:
      pd.DataFrame
          CPI weights as a normalized version of `df_cpi_ready`.
    """

    #  Step 1: Compute column sums (total final consumption per country)
    column_sums = df_cpi_ready.sum(axis=0)

    #  Step 2: Avoid division by zero (replace zero sums with NaN to prevent␣
 ↪errors)
    column_sums.replace(0, pd.NA, inplace=True)

    #  Step 3: Normalize each column (divide each cell by its column sum)
    cpi_weights = df_cpi_ready.div(column_sums, axis=1)

    print("  CPI weights successfully calculated.")

    return cpi_weights


#  Apply CPI weight calculation
df_cpi_weights = calculate_cpi_weights(df_cpi_ready)

#  Define the output directory and ensure it exists
base_output_dir = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/
 ↪cpi_weights"
final_output_dir = os.path.join(base_output_dir, "exio3_figaro_cpi_weights")
os.makedirs(final_output_dir, exist_ok=True)  # Create the subfolder if it␣
 ↪doesn't exist

#  Define file path and save
output_file = os.path.join(final_output_dir, "cpi_weights_figaro_2021.csv")
df_cpi_weights.to_csv(output_file)

print(f"  CPI weights saved to {output_file}")
```

[ ]:

```
###  Create EU28 household demand column based only on EU28 countries ###

# Step 1: Filter for EU28 countries and 'P3_S14' only
eu28_p3_s14_columns = [(country, 'P3_S14') for country in eu28_countries]
df_p3_s14_eu28 = df_cpi_ready.loc[:, eu28_p3_s14_columns]

# Step 2: Sum across these EU28 household demand columns
df_cpi_ready_eu28 = df_p3_s14_eu28.sum(axis=1).to_frame(name='EU28')

# Step 3: Calculate CPI weights using the existing function
df_cpi_weights_eu28 = calculate_cpi_weights(df_cpi_ready_eu28)

#  Save result
output_file_eu28 = os.path.join(final_output_dir, "cpi_weights_figaro_2021_EU28.
 ↪csv")
df_cpi_weights_eu28.to_csv(output_file_eu28)

print(f" CPI weights for EU28 saved to {output_file_eu28}")
```

[ ]:

```
### Technical Coefficient Matrix Calculation ###

# ----- STEP 0: Extract the gross output row (since it’s unique, with index␣
 ↪("GO","GO"))
go_row = df_2021_rdy.loc[("GO", "GO")]

# ----- STEP 1: Build a dictionary for gross output for each supplier column
gross_output = {}
for col in merged_df.columns:
    country, supplier_sector = col
    if supplier_sector != "GO":  # Only for supplier columns
        try:
            go_value = go_row[col]
            gross_output[col] = go_value
        except KeyError:
            print(f"Warning: Gross output for supplier column {col} not found;␣
 ↪defaulting to 1.")
            gross_output[col] = 1

# ----- STEP 2: Create the interindustry block by excluding rows and columns␣
 ↪where Sector == "GO"
rows_mask = merged_df.index.get_level_values("Sector") != "GO"
cols_mask = merged_df.columns.get_level_values("Sector") != "GO"
interindustry_block = merged_df.loc[rows_mask, cols_mask]

# ----- STEP 3: Compute the technical coefficients
def normalize_column(col):
    supplier_key = col.name  # a tuple (country, supplier_sector)
    denominator = gross_output.get(supplier_key, 1)
    return col / denominator

tech_coeff = interindustry_block.apply(normalize_column, axis=0)

# ----- STEP 4: Duplicate B rows with new sector names "B_gas" and "B_nongas"
b_rows = tech_coeff.loc[tech_coeff.index.get_level_values("Sector") == "B"].
 ↪copy()

# Function to update MultiIndex for row transformations
def update_sector(index, new_label):
    new_tuples = [(country, new_label if sector == "B" else sector) for␣
 ↪country, sector in index]
    return pd.MultiIndex.from_tuples(new_tuples, names=index.names)

# Duplicate rows
b_rows_gas = b_rows.copy()
b_rows_nongas = b_rows.copy()
b_rows_gas.index = update_sector(b_rows.index, "B_gas")
b_rows_nongas.index = update_sector(b_rows.index, "B_nongas")

# Remove original B rows and add new ones
non_b_rows = tech_coeff.loc[tech_coeff.index.get_level_values("Sector") != "B"]
tech_coeff_rows_modified = pd.concat([non_b_rows, b_rows_gas, b_rows_nongas]).
 ↪sort_index()

# ----- STEP 4a (New): Duplicate B columns with new sector names "B_gas" and␣
 ↪"B_nongas"
b_columns = tech_coeff_rows_modified.loc[:, tech_coeff_rows_modified.columns.
 ↪get_level_values("Sector") == "B"].copy()

# Function to update MultiIndex for column transformations
def update_column(index, new_label):
    new_tuples = [(country, new_label if sector == "B" else sector) for␣
 ↪country, sector in index]
    return pd.MultiIndex.from_tuples(new_tuples, names=index.names)

# Duplicate columns
b_columns_gas = b_columns.copy()
b_columns_nongas = b_columns.copy()
b_columns_gas.columns = update_column(b_columns.columns, "B_gas")
b_columns_nongas.columns = update_column(b_columns.columns, "B_nongas")

# Remove original B columns and add new ones
non_b_columns = tech_coeff_rows_modified.loc[:, tech_coeff_rows_modified.
 ↪columns.get_level_values("Sector") != "B"]
```

```
tech_coeff_final = pd.concat([non_b_columns, b_columns_gas, b_columns_nongas],␣
 ↪axis=1).sort_index(axis=1)

# ----- STEP 6: Save the final modified table to CSV
tech_coeff_final.to_csv("C:/Users/danie/Nextcloud/Coding/Masterthesis/notebooks/
 ↪NB_exio3_figaro_gas/technical_coefficients_modified.csv")
print("Modified technical coefficients table saved to␣
 ↪'technical_coefficients_modified.csv'.")

display(tech_coeff_final)
```

[ ]:

```
# Assume figaro_B is the Figaro technical coefficients table (B matrix) with a␣
 ↪MultiIndex
figaro_B = tech_coeff_final
weights_B = exio_Z_final
```

[ ]:

```
# Define aggregated sector mapping
aggregated_sector = "C25-33"
disaggregated_sectors = ['C25', 'C26', 'C27', 'C28', 'C29', 'C30', 'C31_32',␣
 ↪'C33']

# Ensure MultiIndexes are correctly assigned
figaro_B.index.names = ['Country', 'Sector']
figaro_B.columns.names = ['Country', 'Sector']
weights_B.index.names = ['Country', 'Sector']
weights_B.columns.names = ['Country', 'Sector']


display(figaro_B.index)
display(weights_B.index)
```

[ ]:

```
### Step 1: Apply row-wise weightings (B_gas and B_nongas rows)
for country in figaro_B.index.get_level_values(0).unique():
    for row in ['B_gas', 'B_nongas']:
        if (country, row) in figaro_B.index and (country, row) in weights_B.
 ↪index:
            for target_country, sector in figaro_B.columns:
                # Normal case: Use direct weight if sector exists in weights_B
                if (target_country, sector) in weights_B.columns:
                    weight = weights_B.loc[(country, row), (target_country,␣
 ↪sector)]
                    figaro_B.loc[(country, row), (target_country, sector)] *=␣
 ↪weight

                # Special handling for disaggregated C25-33 sectors
                elif sector in disaggregated_sectors and (target_country,␣
 ↪aggregated_sector) in weights_B.columns:
                    weight = weights_B.loc[(country, row), (target_country,␣
 ↪aggregated_sector)]
                    figaro_B.loc[(country, row), (target_country, sector)] *=␣
 ↪weight
                    print(f"Applied ROW weight to disaggregated sector:␣
 ↪{country}, {row}, {target_country}, {sector} -> {figaro_B.loc[(country,␣
 ↪row), (target_country, sector)]}")  # Debugging

### Step 2: Apply column-wise weightings (only to B_gas and B_nongas columns)
for country in figaro_B.columns.get_level_values(0).unique():
    for col in ['B_gas', 'B_nongas']:
        if (country, col) in figaro_B.columns and (country, col) in weights_B.
 ↪columns:
            for target_country, row in figaro_B.index:
                if row not in ['B_gas', 'B_nongas']:  # Skip these rows
                    # Normal case: Use direct weight if sector exists in␣
 ↪weights_B
                    if (target_country, row) in weights_B.index:
                        weight = weights_B.loc[(target_country, row), (country,␣
 ↪col)]
                        figaro_B.loc[(target_country, row), (country, col)] *=␣
 ↪weight

                    # Special handling for disaggregated C25-33 sectors
                    elif row in disaggregated_sectors and (target_country,␣
 ↪aggregated_sector) in weights_B.index:
                        weight = weights_B.loc[(target_country,␣
 ↪aggregated_sector), (country, col)]
                        figaro_B.loc[(target_country, row), (country, col)] *=␣
 ↪weight
                        print(f"Applied COLUMN weight to disaggregated sector:␣
 ↪{target_country}, {row}, {country}, {col} -> {figaro_B.loc[(target_country,␣
 ↪row), (country, col)]}")  # Debugging

# Display results
display(figaro_B)
```

[ ]:

```
# Define the final demand sectors to remove
final_demand_sectors = ['P3_S13', 'P3_S14', 'P3_S15', 'P51G', 'P5M']

# Step 1: Drop all columns where the sector is in final_demand_sectors
figaro_df = figaro_B.drop(columns=[col for col in figaro_B.columns if col[1] in␣
 ↪final_demand_sectors])
```

```
# Step 2: Drop all rows where 'Country' is 'W2'
figaro_df = figaro_df.drop(index='W2', level=0)  # Drops rows where first index␣
 ↪level (Country) is 'W2'

# Step 3: Replace all NaN values with 0
figaro_df = figaro_df.fillna(0)

# Display the cleaned DataFrame
display(figaro_df)
```

[ ]:

```
figaro_df.to_csv("C:/Users/danie/Nextcloud/Coding/Masterthesis/notebooks/
 ↪NB_exio3_figaro_gas/figaro_B_modified.csv")
print("Modified Figaro B matrix saved to 'figaro_B_modified.csv'.")
```

[ ]:

```
### Anaysis ###
```

[ ]:

```
### 500% shock to imported gas into EU28 from outside EU28 ###



# Define output directory
output_dir = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/
 ↪results"
os.makedirs(output_dir, exist_ok=True)  # Ensure directory exists

# Define the shock factor (500% increase)
shock_factor = 6.0  # 500% increase
no_shock_factor = 1.0  # No change for EU countries

# Define energy and non-energy sectors
energy_sectors = ['B_gas']
non_energy_sectors = [s for s in figaro_df.index.get_level_values(1).unique()␣
 ↪if s not in energy_sectors]

# Ensure MultiIndex is correctly set
figaro_df.index = figaro_df.index.set_names(['Country', 'Sector'])
figaro_df.columns = figaro_df.columns.set_names(['Country', 'Sector'])

# Placeholder for results
shock_impacts = []

# **Apply shock for EU28 collectively (imported gas from non-EU28)**
print("Processing imported gas shock into EU28")

# Define block partitioning for energy vs. non-energy (within EU28)
N_indices = [(c, sec) for c, sec in figaro_df.index if c in eu28_countries and␣
 ↪sec in non_energy_sectors]
```

```python
E_indices = [(c, sec) for c, sec in figaro_df.index if sec in energy_sectors] # Include ALL countries

# Partition the IO coefficients (EU28)
A_EE = figaro_df.loc[N_indices, N_indices].values  # endogenous (non-energy) to endogenous (non-energy)
A_XE = figaro_df.loc[E_indices, N_indices].values  # endogenous (non-energy) consuming exogenous (energy)

# Ensure A_XE is correctly shaped: (N × E)
A_XE_T = A_XE.T  # Now (E × N), ensuring proper multiplication

# Identity matrix for energy block
I_EE = np.eye(A_EE.shape[0])

# Compute the Leontief inverse for EU28's energy sectors
try:
    L_EE = np.linalg.inv(I_EE - A_EE.T)  # (I - A'_{EE})^-1
except np.linalg.LinAlgError:
    print("Singular matrix encountered for EU28. Skipping...")
    exit()

# Start with a default price vector (all set to 1)
P_X = np.ones((A_XE.shape[0], 1))  # Each row corresponds to an energy sector in each country

# Iterate over all energy sectors (suppliers)
for i, (supplier_country, sec) in enumerate(E_indices):
    if sec == 'B_gas':  # Only shock B_gas
        for j, (buyer_country, buyer_sec) in enumerate(N_indices):  # Iterate over non-energy buyers
            if buyer_country in eu28_countries and supplier_country not in eu28_countries:
                # If gas is being delivered from a non-EU country to an EU28 country, apply the shock
                P_X[i, 0] = shock_factor  # Set price to 5 for affected gas imports


# Compute price changes for EU28's energy sectors
delta_P_E = L_EE @ (A_XE_T @ P_X)


# Store results in a properly structured DataFrame
```

```
shock_results_extra_df = pd.DataFrame(delta_P_E.flatten(), index=pd.MultiIndex.
 ↪from_tuples(N_indices, names=["Country", "Sector"]), columns=["Price␣
 ↪Change"])

# Save results in three-column format
output_file = os.path.join(output_dir, 'imported_gas_shock_extra_EU28.csv')
shock_results_extra_df.to_csv(output_file, index=True)  #  Keeps MultiIndex

print(f"Shock impacts saved to {output_file}")
```

[ ]:

```
### 500% shock to ALL imported gas ###



# Define output directory
output_dir = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/
 ↪results"
os.makedirs(output_dir, exist_ok=True)  # Ensure directory exists

# Define the shock factor (500% increase)
shock_factor = 6.0  # 500% increase
no_shock_factor = 1.0  # No change for EU countries

# Define energy and non-energy sectors
energy_sectors = ['B_gas']
non_energy_sectors = [s for s in figaro_df.index.get_level_values(1).unique()␣
 ↪if s not in energy_sectors]

# Ensure MultiIndex is correctly set
figaro_df.index = figaro_df.index.set_names(['Country', 'Sector'])
figaro_df.columns = figaro_df.columns.set_names(['Country', 'Sector'])

# Placeholder for results
shock_impacts = []

# **Apply shock for EU28 collectively (imported gas from non-EU28)**
print("Processing imported gas shock into EU28")

# Define block partitioning for energy vs. non-energy (within EU28)
N_indices = [(c, sec) for c, sec in figaro_df.index if c in eu28_countries and␣
 ↪sec in non_energy_sectors]
E_indices = [(c, sec) for c, sec in figaro_df.index if sec in energy_sectors] ␣
 ↪# Include ALL countries

# Partition the IO coefficients (EU28)
A_EE = figaro_df.loc[N_indices, N_indices].values  # endogenous (non-energy) to␣
 ↪endogenous (non-energy)
```

```
A_XE = figaro_df.loc[E_indices, N_indices].values  # endogenous (non-energy)␣
 ↪consuming exogenous (energy)

# Ensure A_XE is correctly shaped: (N × E)
A_XE_T = A_XE.T  # Now (E × N), ensuring proper multiplication

# Identity matrix for energy block
I_EE = np.eye(A_EE.shape[0])

# Compute the Leontief inverse for EU28's energy sectors
try:
    L_EE = np.linalg.inv(I_EE - A_EE.T)  # (I - A'_{EE})^-1
except np.linalg.LinAlgError:
    print("Singular matrix encountered for EU28. Skipping...")
    exit()

# Start with 1s (no shock)
P_X = np.ones((A_XE.shape[0], 1))

# Apply shocks to extra-EU imports (like in Scenario 1)
for i, (supplier_country, sec) in enumerate(E_indices):
    if sec == 'B_gas' and supplier_country not in eu28_countries:
        P_X[i, 0] = shock_factor  #  Shock extra-EU gas imports (5.0)

# Apply shocks to intra-EU imports (Scenario 2 addition)
for i, (buyer_country, sec) in enumerate(E_indices):
    if sec == 'B_gas' and buyer_country in eu28_countries:
        for j, (supplier_country, sec2) in enumerate(E_indices):
            if sec2 == 'B_gas' and supplier_country in eu28_countries and␣
 ↪supplier_country != buyer_country:
                P_X[j, 0] = shock_factor  #  Shock intra-EU gas imports (5.0)

# Compute price changes for EU28's energy sectors
delta_P_E = L_EE @ (A_XE_T @ P_X)


# Store results in a properly structured DataFrame
shock_results_intra_extra_df = pd.DataFrame(delta_P_E.flatten(), index=pd.
 ↪MultiIndex.from_tuples(N_indices, names=["Country", "Sector"]),␣
 ↪columns=["Price Change"])

# Save results in three-column format
output_file = os.path.join(output_dir, 'imported_gas_shock_intra_extra_EU28.
 ↪csv')
shock_results_intra_extra_df.to_csv(output_file, index=True)  #  Keeps␣
 ↪MultiIndex
```

```
print(f"Shock impacts saved to {output_file}")
```

[ ]:

```
print(shock_results_extra_df.index)
print(shock_results_extra_df.columns)
print(shock_results_intra_extra_df.index)
print(shock_results_intra_extra_df.columns)
print(df_cpi_weights.index)
print(df_cpi_weights.columns)
```

[ ]:

```
display(df_cpi_weights)
display(shock_results_extra_df)
```

[ ]:

```
# Define the output directory
output_dir = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/
 ↪results/cpi_weighted_shocks"
os.makedirs(output_dir, exist_ok=True)  # Ensure the directory exists

# Define file paths for the CPI-weighted results
file_cpi_scenario_1 = os.path.join(output_dir,␣
 ↪"cpi_weighted_imported_gas_shock_extra_EU28.csv")
file_cpi_scenario_2 = os.path.join(output_dir,␣
 ↪"cpi_weighted_imported_gas_shock_intra_extra_EU28.csv")

# Define weights and both shock scenarios
weights = df_cpi_weights
shock_scenarios = {
    file_cpi_scenario_1: shock_results_extra_df,
    file_cpi_scenario_2: shock_results_intra_extra_df
}

# Drop the unnecessary second level in the columns of weights (P3_S14)
if isinstance(weights.columns, pd.MultiIndex) and len(weights.columns.levels) >␣
 ↪1:
    weights = weights.droplevel(level=1, axis=1)  # Ensure we only have␣
 ↪'Country' as columns

# Get the list of consuming countries
consuming_countries = weights.columns  # Now only country names remain as␣
 ↪column labels

# Loop over both scenarios
for file_path, shock_results in shock_scenarios.items():

    # Initialize an empty DataFrame with the MultiIndex from shock_results
    cpi_weighted_results = pd.DataFrame(index=shock_results.index)

    # Loop through each consuming country
    for consuming_country in consuming_countries:

        # Select the CPI weights for this consuming country
        country_weights = weights[consuming_country]  # This should now be a Series

        # Broadcast these weights to the full (Country, Sector) index of shock_results
        aligned_weights = country_weights.reindex(shock_results.index, fill_value=0)

        # Multiply price changes by weights
        cpi_weighted_values = shock_results['Price Change'] * aligned_weights

        # Ensure we store a **flat** 1D Series (not a DataFrame)
        cpi_weighted_results[consuming_country] = cpi_weighted_values.squeeze()  # Convert to 1D Series

    # Save results
    cpi_weighted_results.to_csv(file_path)
    print(f"CPI-weighted shocks saved to {file_path}")
```

[ ]:

```
#  Load EU28 CPI weights
cpi_weights_eu28_path = os.path.join(final_output_dir, "cpi_weights_figaro_2021_EU28.csv")
weights_eu28 = pd.read_csv(cpi_weights_eu28_path, index_col=[0, 1])['EU28']

#  Define output file paths
file_cpi_scenario_1_eu28 = os.path.join(output_dir, "cpi_weighted_imported_gas_shock_extra_EU28_eu28weighted.csv")
file_cpi_scenario_2_eu28 = os.path.join(output_dir, "cpi_weighted_imported_gas_shock_intra_extra_EU28_eu28weighted.csv")

#  Define mapping of files and shocks
shock_scenarios_eu28 = {
    file_cpi_scenario_1_eu28: shock_results_extra_df,
    file_cpi_scenario_2_eu28: shock_results_intra_extra_df
}

#  Loop through each scenario and apply EU28 weights
for file_path, shock_results in shock_scenarios_eu28.items():

    # Align the EU28 weights to the index of the shock result
    aligned_weights = weights_eu28.reindex(shock_results.index, fill_value=0)

    # Apply the EU28 weights to the price change column
    cpi_weighted_result_eu28 = (shock_results['Price Change'] *␣
 ↪aligned_weights).to_frame(name='EU28')

    # Save result
    cpi_weighted_result_eu28.to_csv(file_path)
    print(f" EU28 CPI-weighted shock saved to {file_path}")
```

[ ]:

```
### Visualization of CPI-weighted Results in a stacked bar chart ###

# Define file paths for the CPI-weighted results
file_cpi_scenario_1 = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/
 ↪processed/results/cpi_weighted_shocks/
 ↪cpi_weighted_imported_gas_shock_extra_EU28.csv"
file_cpi_scenario_2 = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/
 ↪processed/results/cpi_weighted_shocks/
 ↪cpi_weighted_imported_gas_shock_intra_extra_EU28.csv"

# Define base output directory
base_output_dir = "C:/Users/danie/Nextcloud/Coding/Masterthesis/visualization"
figures_dir_cpi = os.path.join(base_output_dir,␣
 ↪"figures_cpi_weighted_gas_shocks")
os.makedirs(figures_dir_cpi, exist_ok=True)  # Ensure folder exists

# Read the CSV files into DataFrames (CPI-weighted results)
df_cpi_scenario_1 = pd.read_csv(file_cpi_scenario_1, index_col=[0, 1])
df_cpi_scenario_2 = pd.read_csv(file_cpi_scenario_2, index_col=[0, 1])

# Multiply by 100 to convert to percentage values
df_cpi_scenario_1 *= 100
df_cpi_scenario_2 *= 100

# Define EU28 country codes
eu28_countries = [
    "AT", "BE", "BG", "CY", "CZ", "DE", "DK", "EE", "ES", "FI", "FR", "GB",␣
 ↪"GR", "HR", "HU",
    "IE", "IT", "LT", "LU", "LV", "MT", "NL", "PL", "PT", "RO", "SE", "SI", "SK"
]

# Function to determine the y-axis limit based on max impact
def get_dynamic_y_limit(max_shock):
    step_size = 0.05  # Define step size for y-axis scaling
    return np.ceil(max_shock / step_size) * step_size  # Round up to the next␣
 ↪step

# Iterate over each EU28 country to generate CPI-weighted plots
for country in eu28_countries:
    try:
        # Extract the CPI-weighted impact for this country
        country_cpi_scenario_1 = df_cpi_scenario_1[country].
↪sort_values(ascending=False)
        country_cpi_scenario_2 = df_cpi_scenario_2[country].
↪sort_values(ascending=False)

        # Select the top 20 affected sectors
        top_20_country_cpi_2 = country_cpi_scenario_2.head(20)  # Scenario 2␣
↪(full intra + extra shock)

        # Convert MultiIndex (Country, Sector) into readable strings
        top_20_labels = [f"{idx[0]} - {idx[1]}" for idx in top_20_country_cpi_2.
↪index]

        # Calculate `max_shock_2` inside the loop, after extracting top sectors
        max_shock_2 = top_20_country_cpi_2.max()
        y_axis_limit = get_dynamic_y_limit(max_shock_2)

        ### **Bar Chart for Scenario 2 (Intra- & Extra-EU Imports Shock)**
        plt.figure(figsize=(12, 6))
        plt.bar(top_20_labels, top_20_country_cpi_2, color='darkorange')
        plt.title(f"Top 20 CPI-Weighted Sectors in {country} (Intra- & Extra-EU␣
↪Gas Shock)")
        plt.ylabel("CPI-Weighted Price Impact (%)")
        plt.xlabel("Sectors")
        plt.xticks(rotation=45, ha="right")
        plt.ylim(0, y_axis_limit)
        plt.grid(axis="y", linestyle="--", alpha=0.7)
        plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f"{x:
↪.2f}%"))
        plt.savefig(os.path.join(figures_dir_cpi,␣
↪f"Top_20_CPI_Sectors_{country}.png"), bbox_inches="tight")
        plt.close()

        ### **Stacked Bar Chart for Intra vs. Extra EU Impact**
        diff_cpi_scenario = country_cpi_scenario_2.loc[top_20_country_cpi_2.
↪index] - country_cpi_scenario_1.loc[top_20_country_cpi_2.index]

        plt.figure(figsize=(12, 6))
        plt.bar(
            top_20_labels,
            country_cpi_scenario_1.loc[top_20_country_cpi_2.index],
            label="Extra-EU Imports Shock",
            color="royalblue"
        )
        plt.bar(
            top_20_labels,
            diff_cpi_scenario,
            bottom=country_cpi_scenario_1.loc[top_20_country_cpi_2.index],
            label="Intra-EU Impact",
            color="darkorange"
        )

        # Format plot
        plt.title(f"CPI-Weighted Impact in {country}: Gas Shock (Stacked)")
        plt.ylabel("CPI-Weighted Price Impact (%)")
        plt.xlabel("Sectors")
        plt.xticks(rotation=45, ha="right")
        plt.ylim(0, y_axis_limit)  # Set scaling
        plt.grid(axis="y", linestyle="--", alpha=0.7)
        plt.legend()
        plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f"{x:
 ↪.2f}%"))

        # Save figure
        plt.savefig(os.path.join(figures_dir_cpi,␣
 ↪f"Stacked_CPI_Sectors_{country}.png"), bbox_inches="tight")
        plt.close()

        print(f"  Saved CPI-weighted visualizations for {country} in␣
 ↪{figures_dir_cpi}")

    except KeyError:
        print(f"  Skipping {country}: No CPI-weighted data available.")
```

[ ]:

```
# Choose scenario (e.g., intra + extra EU shock)
df_cpi_weighted = df_cpi_scenario_2 *100  # Or df_cpi_scenario_1 for only␣
 ↪Extra-EU

# Step 1: Filter to EU28 countries only (columns)
df_cpi_weighted_eu28 = df_cpi_weighted[eu28_countries]

# Step 2: Compute average CPI-weighted impact for each sector across all EU28␣
 ↪countries
# Group by the 'Sector' level of the index (level=1)
sector_avg_impact = df_cpi_weighted_eu28.groupby(level="Sector").mean().
 ↪mean(axis=1)

# Step 3: Sort and select top sectors
top_20_sectors = sector_avg_impact.sort_values(ascending=False).head(20)

# Step 4: Plot
plt.figure(figsize=(12, 6))
```

```
top_20_sectors.plot(kind="bar", color="darkorange", title="Top 20 Sectors by␣
 ↪Average CPI-Weighted Impact (EU28)")
plt.ylabel("Average CPI-Weighted Price Impact (%)")
plt.xlabel("Sectors")
plt.xticks(rotation=45, ha="right")
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Show the plot
plt.tight_layout()
plt.show()
```

[ ]:

```
# ===  Add EU28 CPI-weighted stacked bar chart ===

# File paths for EU28-weighted results
file_cpi_eu28_extra = os.path.join(output_dir,␣
 ↪"cpi_weighted_imported_gas_shock_extra_EU28_eu28weighted.csv")
file_cpi_eu28_intra_extra = os.path.join(output_dir,␣
 ↪"cpi_weighted_imported_gas_shock_intra_extra_EU28_eu28weighted.csv")

# Load and convert to percentage
df_eu28_extra = pd.read_csv(file_cpi_eu28_extra, index_col=[0, 1]) * 100
df_eu28_intra_extra = pd.read_csv(file_cpi_eu28_intra_extra, index_col=[0, 1])␣
 ↪* 100

# Sort sectors by descending impact (Scenario 2 = intra + extra)
top_20_eu28 = df_eu28_intra_extra['EU28'].sort_values(ascending=False).head(40)

# Generate readable labels
top_20_labels = [f"{idx[0]} - {idx[1]}" for idx in top_20_eu28.index]

# Determine y-axis limit
max_shock_eu28 = top_20_eu28.max()
y_axis_limit = get_dynamic_y_limit(max_shock_eu28)

# Compute difference = intra-EU contribution
diff_eu28 = df_eu28_intra_extra.loc[top_20_eu28.index, 'EU28'] - df_eu28_extra.
 ↪loc[top_20_eu28.index, 'EU28']

# === Stacked Bar Chart for EU28 ===
plt.figure(figsize=(12, 6))
plt.bar(
    top_20_labels,
    df_eu28_extra.loc[top_20_eu28.index, 'EU28'],
    label="Extra-EU Imports Shock",
    color="royalblue"
)
plt.bar(
    top_20_labels,
    diff_eu28,
    bottom=df_eu28_extra.loc[top_20_eu28.index, 'EU28'],
    label="Intra-EU Impact",
    color="darkorange"
)

# Format plot
plt.title("EU28 CPI-Weighted Impact: Gas Shock (Stacked)")
plt.ylabel("CPI-Weighted Price Impact (%)")
plt.xlabel("Sectors")
plt.xticks(rotation=45, ha="right")
plt.ylim(0, y_axis_limit)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.legend()
plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f"{x:.2f}%"))

# Save plot
plt.savefig(os.path.join(figures_dir_cpi, "Stacked_CPI_Sectors_EU28_t40.png"),␣
 ↪bbox_inches="tight")
plt.close()

print("  Saved CPI-weighted stacked bar chart for EU28 in", figures_dir_cpi)
```

[ ]:

```
# Load EU28-weighted CPI shock results
file_cpi_eu28_extra = os.path.join(output_dir,␣
 ↪"cpi_weighted_imported_gas_shock_extra_EU28_eu28weighted.csv")
file_cpi_eu28_intra_extra = os.path.join(output_dir,␣
 ↪"cpi_weighted_imported_gas_shock_intra_extra_EU28_eu28weighted.csv")

df_eu28_extra = pd.read_csv(file_cpi_eu28_extra, index_col=[0, 1]) * 100
df_eu28_intra_extra = pd.read_csv(file_cpi_eu28_intra_extra, index_col=[0, 1])␣
 ↪* 100

# Step 1: Accumulate sectoral impacts by **producing country** (first level of␣
 ↪index)
impact_extra_by_origin = df_eu28_extra.groupby(level=0)['EU28'].sum()
impact_intra_extra_by_origin = df_eu28_intra_extra.groupby(level=0)['EU28'].
 ↪sum()

# Step 2: Compute the **intra-EU contribution** per origin country
impact_intra_only_by_origin = impact_intra_extra_by_origin -␣
 ↪impact_extra_by_origin

# Step 3: Sort by total impact (descending)
```

```
sorted_countries = impact_intra_extra_by_origin.sort_values(ascending=False).
 ↪index

# Step 4: Plot stacked bar chart
plt.figure(figsize=(14, 7))
plt.bar(
    sorted_countries,
    impact_extra_by_origin.loc[sorted_countries],
    label="Extra-EU Shock Impact",
    color="royalblue"
)
plt.bar(
    sorted_countries,
    impact_intra_only_by_origin.loc[sorted_countries],
    bottom=impact_extra_by_origin.loc[sorted_countries],
    label="Intra-EU Impact",
    color="darkorange"
)

# Format plot
plt.title("Origin Countries' Contribution to EU28 CPI-Weighted Inflation␣
 ↪(Stacked)")
plt.ylabel("Total CPI-Weighted Inflation Impact (%)")
plt.xlabel("Origin Country")
plt.xticks(rotation=45, ha="right")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.legend()
plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f"{x:.2f}%"))

# Save figure
output_path = os.path.join(figures_dir_cpi,␣
 ↪"Country_Origin_Impact_on_EU28_Inflation_Stacked.png")
plt.savefig(output_path, bbox_inches="tight")
plt.close()

print("  Saved origin country stacked CPI-weighted impact chart to:",␣
 ↪output_path)
```

[ ]:

```
### Cumulative Inflation Impact by EU28-Country (Stacked) ###



# Define file paths for the CPI-weighted results
file_cpi_scenario_1 = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/
 ↪processed/results/cpi_weighted_shocks/
 ↪cpi_weighted_imported_gas_shock_extra_EU28.csv"
```

```python
file_cpi_scenario_2 = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/results/cpi_weighted_shocks/cpi_weighted_imported_gas_shock_intra_extra_EU28.csv"

# Define output directory for charts
base_output_dir = "C:/Users/danie/Nextcloud/Coding/Masterthesis/visualization"
figures_dir_cpi = os.path.join(base_output_dir, "figures_cpi_weighted_gas_shocks")
os.makedirs(figures_dir_cpi, exist_ok=True)  # Ensure folder exists

# Read CPI-weighted results into DataFrames
df_cpi_scenario_1 = pd.read_csv(file_cpi_scenario_1, index_col=[0, 1])
df_cpi_scenario_2 = pd.read_csv(file_cpi_scenario_2, index_col=[0, 1])

# Convert values to percentage
df_cpi_scenario_1 *= 100
df_cpi_scenario_2 *= 100

# Filter columns to only include EU28 countries
df_cpi_scenario_1 = df_cpi_scenario_1[eu28_countries]
df_cpi_scenario_2 = df_cpi_scenario_2[eu28_countries]

# Sum total inflation impact across all sectors for each EU28 country
cumulative_impact_scenario_1 = df_cpi_scenario_1.sum(axis=0)  # Extra-EU Gas Shock
cumulative_impact_scenario_2 = df_cpi_scenario_2.sum(axis=0)  # Intra+Extra-EU Gas Shock

# Calculate difference (intra-EU contribution)
cumulative_intra_impact = cumulative_impact_scenario_2 - cumulative_impact_scenario_1

# Sort countries by total impact (Scenario 2)
sorted_countries = cumulative_impact_scenario_2.sort_values(ascending=False).index

# Generate stacked bar chart
plt.figure(figsize=(14, 7))
plt.bar(sorted_countries, cumulative_impact_scenario_1.loc[sorted_countries], label="Extra-EU Gas Shock", color="royalblue")
plt.bar(sorted_countries, cumulative_intra_impact.loc[sorted_countries],
        bottom=cumulative_impact_scenario_1.loc[sorted_countries],
        label="Intra-EU Impact", color="darkorange")

# Format plot
plt.title("Cumulative CPI-Weighted Inflation Impact by Country (Stacked)")
plt.ylabel("Total CPI-Weighted Inflation Impact (%)")
plt.xlabel("Country")
plt.xticks(rotation=45, ha="right")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.legend()
plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f"{x:.2f}%"))

# Save figure
output_path = os.path.join(figures_dir_cpi, "Cumulative_CPI_Impact_Stacked.png")
plt.savefig(output_path, bbox_inches="tight")
plt.close()

print(f"  Saved cumulative CPI-weighted stacked bar chart: {output_path}")
```

[ ]:

```
########################################## END OF WORKING AND FINAL SCRIPT␣
  ↪##########################################
```

[ ]:

```
### Cumulative Inflation Impact by Country (Stacked) also outside EU␣
  ↪countries###

# Define file paths for the CPI-weighted results
file_cpi_scenario_1 = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/
  ↪processed/results/cpi_weighted_shocks/
  ↪cpi_weighted_imported_gas_shock_extra_EU28.csv"
file_cpi_scenario_2 = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/
  ↪processed/results/cpi_weighted_shocks/
  ↪cpi_weighted_imported_gas_shock_intra_extra_EU28.csv"

# Define output directory for charts
base_output_dir = "C:/Users/danie/Nextcloud/Coding/Masterthesis/visualization"
figures_dir_cpi = os.path.join(base_output_dir,␣
  ↪"figures_cpi_weighted_gas_shocks")
os.makedirs(figures_dir_cpi, exist_ok=True)  # Ensure folder exists

# Read CPI-weighted results into DataFrames
df_cpi_scenario_1 = pd.read_csv(file_cpi_scenario_1, index_col=[0, 1])
df_cpi_scenario_2 = pd.read_csv(file_cpi_scenario_2, index_col=[0, 1])

# Convert values to percentage
df_cpi_scenario_1 *= 100
df_cpi_scenario_2 *= 100

# Sum total inflation impact across all sectors for each country
cumulative_impact_scenario_1 = df_cpi_scenario_1.sum(axis=0)  # Extra-EU Gas␣
  ↪Shock
cumulative_impact_scenario_2 = df_cpi_scenario_2.sum(axis=0)  # Intra+Extra-EU␣
  ↪Gas Shock
```

```
# Calculate difference (intra-EU contribution)
cumulative_intra_impact = cumulative_impact_scenario_2 -␣
 ↪cumulative_impact_scenario_1

# Sort countries by total impact
sorted_countries = cumulative_impact_scenario_2.sort_values(ascending=False).
 ↪index

# Generate stacked bar chart
plt.figure(figsize=(14, 7))
plt.bar(sorted_countries, cumulative_impact_scenario_1.loc[sorted_countries],␣
 ↪label="Extra-EU Gas Shock", color="royalblue")
plt.bar(sorted_countries, cumulative_intra_impact.loc[sorted_countries],
        bottom=cumulative_impact_scenario_1.loc[sorted_countries],
        label="Intra-EU Impact", color="darkorange")

# Format plot
plt.title("Cumulative CPI-Weighted Inflation Impact by Country (Stacked)")
plt.ylabel("Total CPI-Weighted Inflation Impact (%)")
plt.xlabel("Country")
plt.xticks(rotation=45, ha="right")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.legend()
plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f"{x:.2f}%"))

# Save figure
output_path = os.path.join(figures_dir_cpi, "Cumulative_CPI_Impact_Stacked.png")
plt.savefig(output_path, bbox_inches="tight")
plt.close()

print(f"  Saved cumulative CPI-weighted stacked bar chart: {output_path}")
```

[ ]:

```
### Overall Impact Visualization ###

# Define file path for the CPI-weighted results
file_cpi_scenario_2 = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/
 ↪processed/results/cpi_weighted_shocks/
 ↪cpi_weighted_imported_gas_shock_intra_extra_EU28.csv"

# Define output directory for visualization
base_output_dir = "C:/Users/danie/Nextcloud/Coding/Masterthesis/visualization"
figures_dir_cpi = os.path.join(base_output_dir,␣
 ↪"figures_cpi_weighted_gas_shocks")
os.makedirs(figures_dir_cpi, exist_ok=True)  # Ensure folder exists

# Read the CPI-weighted results into a DataFrame
```

```
df_cpi_scenario_2 = pd.read_csv(file_cpi_scenario_2, index_col=[0, 1])

# Multiply by 10,000 to get percentage values
df_cpi_scenario_2 *= 100

# Compute the total CPI-weighted impact per country (sum of all sectoral␣
 ↪effects)
total_cpi_impact = df_cpi_scenario_2.sum()

# Sort countries by total impact
total_cpi_impact_sorted = total_cpi_impact.sort_values(ascending=False)

# Determine dynamic y-axis limit starting from 0.5, increasing in 0.5 steps
y_axis_limit = np.ceil(total_cpi_impact_sorted.max() / 0.5) * 0.5

# Create bar chart
plt.figure(figsize=(12, 6))
total_cpi_impact_sorted.plot(kind="bar", color="darkblue")
plt.title("Total CPI-Weighted Impact per Country (Intra- & Extra-EU Gas Shock)")
plt.ylabel("Summed CPI-Weighted Price Impact (%)")
plt.xlabel("Country")
plt.xticks(rotation=45, ha="right")
plt.ylim(0, y_axis_limit)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f"{x:.1f}%"))

# Save figure
plot_path = os.path.join(figures_dir_cpi, "Total_CPI_Impact_Per_Country.png")
plt.savefig(plot_path, bbox_inches="tight")
plt.close()

print(f"  Saved CPI-weighted impact plot to: {plot_path}")
```

[ ]:

```
### Visualization ###



### Unweighted shock impacts ###

# Load the computed shock results
output_file = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/
 ↪results/imported_gas_shock_intra_extra_EU28.csv"

# Read the CSV file into a DataFrame with MultiIndex
shock_results_df = pd.read_csv(output_file, index_col=["Country", "Sector"])

# Sort sectors by the absolute price change (largest impacts)
```

```python
shock_results_sorted = shock_results_df.abs().sort_values(by="Price Change", ascending=False)

# Select the top 20 sectors with the highest price impact
top_20_sectors = shock_results_sorted.head(20)

# Plot the results
plt.figure(figsize=(12, 6))
top_20_sectors.plot(kind="bar", legend=False, title="Top 20 Sectors by Price Impact (500% Gas Shock to imports into EU28)")
plt.ylabel("Relative Price Change")
plt.xlabel("Sectors")
plt.xticks(rotation=45, ha="right")
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Show the plot
plt.show()

import pandas as pd
import matplotlib.pyplot as plt

# Load the computed shock results while preserving MultiIndex
output_file = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/results/imported_gas_shock_into_EU28_fixed.csv"

# Read CSV and restore MultiIndex
shock_results_df = pd.read_csv(output_file, index_col=["Country", "Sector"])

# Convert to percentage changes (already stored in % but ensuring consistency)
shock_results_df["Price Change"] *= 100

# Normalize by taking the mean absolute price impact per sector across all countries
sector_impact_normalized = shock_results_df.groupby(level="Sector")["Price Change"].mean().abs()

# Sort sectors by the most affected
top_20_sectors = sector_impact_normalized.sort_values(ascending=False).head(20)

# Plot the results
plt.figure(figsize=(12, 6))
top_20_sectors.plot(kind="bar", legend=False, title="Top 20 Most Affected Sectors (Normalized)")
plt.ylabel("Average Price Impact (%)")  # Normalized
plt.xlabel("Sectors")
plt.xticks(rotation=45, ha="right")
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Show the plot
plt.show()
```

[ ]:

```
### Visualization loop over all countries ###

# Define file paths for the first two scenarios
file_scenario_1 = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/
 ↪results/imported_gas_shock_extra_EU28.csv"
file_scenario_2 = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/
 ↪results/imported_gas_shock_intra_extra_EU28.csv"

# Read the CSV files into DataFrames with MultiIndex
df_scenario_1 = pd.read_csv(file_scenario_1, index_col=[0, 1])
df_scenario_2 = pd.read_csv(file_scenario_2, index_col=[0, 1])

# Define EU28 country codes
eu28_countries = [
    "AT", "BE", "BG", "CY", "CZ", "DE", "DK", "EE", "ES", "FI", "FR", "GB",␣
 ↪"GR", "HR", "HU",
    "IE", "IT", "LT", "LU", "LV", "MT", "NL", "PL", "PT", "RO", "SE", "SI", "SK"
]

# Iterate over each EU28 country to generate plots
for country in eu28_countries:
    try:
        # Filter for the current country
        df_country_scenario_1 = df_scenario_1.loc[country].
 ↪sort_values(by="Price Change", ascending=False)
        df_country_scenario_2 = df_scenario_2.loc[country].
 ↪sort_values(by="Price Change", ascending=False)

        # Select the top 20 affected sectors
        top_20_country_scenario_1 = df_country_scenario_1.head(20)
        top_20_country_scenario_2 = df_country_scenario_2.head(20)

        # Plot Scenario 1 for the country
        plt.figure(figsize=(12, 6))
        top_20_country_scenario_1.plot(kind="bar", legend=False, title=f"Top 20␣
 ↪Sectors in {country} (Scenario 1: Extra-EU Gas Shock)")
        plt.ylabel("Relative Price Change (%)")
        plt.xlabel("Sectors")
        plt.xticks(rotation=45, ha="right")
        plt.grid(axis="y", linestyle="--", alpha=0.7)
        plt.show()
        # Plot Scenario 2 for the country
        plt.figure(figsize=(12, 6))
        top_20_country_scenario_2.plot(kind="bar", legend=False, title=f"Top 20 Sectors in {country} (Scenario 2: Intra- & Extra-EU Gas Shock)")
        plt.ylabel("Relative Price Change (%)")
        plt.xlabel("Sectors")
        plt.xticks(rotation=45, ha="right")
        plt.grid(axis="y", linestyle="--", alpha=0.7)
        plt.show()

    except KeyError:
        print(f"Skipping {country}: No data available.")
```

[ ]:

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Define file paths for the first two scenarios
file_scenario_1 = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/results/imported_gas_shock_extra_EU28.csv"
file_scenario_2 = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/results/imported_gas_shock_intra_extra_EU28.csv"

# Define base output directory
base_output_dir = "C:/Users/danie/Nextcloud/Coding/Masterthesis/visualization"
figures_dir = os.path.join(base_output_dir, "figures_gas_shocks")

# Ensure the output directory exists
os.makedirs(figures_dir, exist_ok=True)

# Read the CSV files into DataFrames with MultiIndex
df_scenario_1 = pd.read_csv(file_scenario_1, index_col=[0, 1])
df_scenario_2 = pd.read_csv(file_scenario_2, index_col=[0, 1])

# Multiply price changes by 100 to get percentages
df_scenario_1["Price Change"] *= 100
df_scenario_2["Price Change"] *= 100

# Define EU28 country codes
eu28_countries = [
    "AT", "BE", "BG", "CY", "CZ", "DE", "DK", "EE", "ES", "FI", "FR", "GB", "GR", "HR", "HU",
    "IE", "IT", "LT", "LU", "LV", "MT", "NL", "PL", "PT", "RO", "SE", "SI", "SK"
]

# Function to determine the y-axis limit based on max shock
```

```python
def get_y_axis_limit(max_shock):
    if max_shock <= 20:
        return 20
    elif max_shock <= 30:
        return 30
    elif max_shock <= 40:
        return 40
    elif max_shock <= 50:
        return 50
    else:
        return np.ceil(max_shock / 10) * 10  # Round up to the next multiple of 10

# Iterate over each EU28 country to generate plots
for country in eu28_countries:
    try:
        # Filter for the current country
        df_country_scenario_1 = df_scenario_1.loc[country].sort_values(by="Price Change", ascending=False)
        df_country_scenario_2 = df_scenario_2.loc[country].sort_values(by="Price Change", ascending=False)

        # Select the top 20 affected sectors
        top_20_country_scenario_1 = df_country_scenario_1.head(20)
        top_20_country_scenario_2 = df_country_scenario_2.head(20)

        # Determine the y-axis limit based on max price change in each country
        max_shock_1 = top_20_country_scenario_1["Price Change"].max()
        max_shock_2 = top_20_country_scenario_2["Price Change"].max()
        y_axis_limit = get_y_axis_limit(max(max_shock_1, max_shock_2))

        # Plot and save Scenario 1 for the country
        plt.figure(figsize=(12, 6))
        top_20_country_scenario_1["Price Change"].plot(kind="bar", legend=False, color='royalblue')
        plt.title(f"Top 20 Sectors in {country} (Scenario 1: Extra-EU Gas Shock)")
        plt.ylabel("Relative Price Change (%)")
        plt.xlabel("Sectors")
        plt.xticks(rotation=45, ha="right")
        plt.ylim(0, y_axis_limit)  # Set country-specific scaling
        plt.grid(axis="y", linestyle="--", alpha=0.7)
        plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f"{x:.0f}%"))  # Format y-axis as percentage
        plt.savefig(os.path.join(figures_dir, f"Top_20_Sectors_{country}_Scenario1.png"), bbox_inches="tight")
        plt.close()

        # Plot and save Scenario 2 for the country
        plt.figure(figsize=(12, 6))
        top_20_country_scenario_2["Price Change"].plot(kind="bar",␣
 ↪legend=False, color='darkorange')
        plt.title(f"Top 20 Sectors in {country} (Scenario 2: Intra- & Extra-EU␣
 ↪Gas Shock)")
        plt.ylabel("Relative Price Change (%)")
        plt.xlabel("Sectors")
        plt.xticks(rotation=45, ha="right")
        plt.ylim(0, y_axis_limit)  # Set country-specific scaling
        plt.grid(axis="y", linestyle="--", alpha=0.7)
        plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f"{x:
 ↪.0f}%"))  # Format y-axis as percentage
        plt.savefig(os.path.join(figures_dir,␣
 ↪f"Top_20_Sectors_{country}_Scenario2.png"), bbox_inches="tight")
        plt.close()

        print(f"Saved figures for {country} in {figures_dir}")

    except KeyError:
        print(f"Skipping {country}: No data available.")
```

```
[ ]: import os
     import pandas as pd
     import matplotlib.pyplot as plt
     import numpy as np

     # Define file paths for the two scenarios
     file_scenario_1 = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/
      ↪results/imported_gas_shock_extra_EU28.csv"
     file_scenario_2 = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/
      ↪results/imported_gas_shock_intra_extra_EU28.csv"

     # Define output directory
     base_output_dir = "C:/Users/danie/Nextcloud/Coding/Masterthesis/visualization"
     figures_dir = os.path.join(base_output_dir, "figures_gas_shocks_combined")
     os.makedirs(figures_dir, exist_ok=True)  # Ensure the folder exists

     # Read the CSV files into DataFrames with MultiIndex
     df_scenario_1 = pd.read_csv(file_scenario_1, index_col=[0, 1])
     df_scenario_2 = pd.read_csv(file_scenario_2, index_col=[0, 1])

     # Convert price changes to percentages
     df_scenario_1["Price Change"] *= 100
     df_scenario_2["Price Change"] *= 100
```

```python
# Define EU28 country codes
eu28_countries = [
    "AT", "BE", "BG", "CY", "CZ", "DE", "DK", "EE", "ES", "FI", "FR", "GB", "GR", "HR", "HU",
    "IE", "IT", "LT", "LU", "LV", "MT", "NL", "PL", "PT", "RO", "SE", "SI", "SK"
]

# Function to determine the y-axis limit based on max shock
def get_y_axis_limit(max_shock):
    if max_shock <= 10:
        return 10
    elif max_shock <= 20:
        return 20
    elif max_shock <= 30:
        return 30
    elif max_shock <= 40:
        return 40
    elif max_shock <= 50:
        return 50
    else:
        return np.ceil(max_shock / 10) * 10  # Round up to the next multiple of 10

# Iterate over each EU28 country to generate stacked bar plots
for country in eu28_countries:
    try:
        # Filter data for the current country
        df_country_scenario_1 = df_scenario_1.loc[country]
        df_country_scenario_2 = df_scenario_2.loc[country]

        # Compute total shock (Scenario 2)
        total_impact = df_country_scenario_2["Price Change"]

        # Sort sectors by total impact in descending order
        sorted_indices = total_impact.sort_values(ascending=False).index

        # Select the top 20 affected sectors
        top_20_country_scenario_1 = df_country_scenario_1.loc[sorted_indices].head(20)
        top_20_country_scenario_2 = df_country_scenario_2.loc[sorted_indices].head(20)

        # Compute the additional impact from Intra-EU imports (Scenario 2 - Scenario 1)
        diff_scenario = top_20_country_scenario_2["Price Change"] - top_20_country_scenario_1["Price Change"]
        # Determine the y-axis limit
        max_shock = top_20_country_scenario_2["Price Change"].max()
        y_axis_limit = get_y_axis_limit(max_shock)

        # Create stacked bar chart
        plt.figure(figsize=(12, 6))
        plt.bar(
            top_20_country_scenario_1.index,
            top_20_country_scenario_1["Price Change"],
            label="Extra-EU Imports Shock",
            color="royalblue"
        )
        plt.bar(
            top_20_country_scenario_1.index,
            diff_scenario,
            bottom=top_20_country_scenario_1["Price Change"],
            label="Intra-EU Impact",
            color="darkorange"
        )

        # Format plot
        plt.title(f"Top 20 Sectors in {country}: Gas Shock (Stacked)")
        plt.ylabel("Relative Price Change (%)")
        plt.xlabel("Sectors")
        plt.xticks(rotation=45, ha="right")
        plt.ylim(0, y_axis_limit)  # Set scaling
        plt.grid(axis="y", linestyle="--", alpha=0.7)
        plt.legend()
        plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f"{x:
 ↪.0f}%"))  # Format y-axis as percentage

        # Save figure
        plt.savefig(os.path.join(figures_dir, f"Stacked_Sectors_{country}.
 ↪png"), bbox_inches="tight")
        plt.close()

        print(f"Saved stacked bar figure for {country}")

    except KeyError:
        print(f"Skipping {country}: No data available.")
```

[ ]:

```
# Visualization

# Load the computed shock results
output_file = "C:/Users/danie/Nextcloud/Coding/Masterthesis/data/processed/
 ↪results/imported_gas_shock_extra_EU28.csv"
```

```
# Read the CSV file into a DataFrame with MultiIndex
shock_results_df = pd.read_csv(output_file, index_col=["Country", "Sector"])

# Sort sectors by the absolute price change (largest impacts)
shock_results_sorted = shock_results_df.abs().sort_values(by="Price Change",␣
 ↪ascending=False)

# Select the top 20 sectors with the highest price impact
top_20_sectors = shock_results_sorted.head(20)

# Plot the results
plt.figure(figsize=(12, 6))
top_20_sectors.plot(kind="bar", legend=False, title="Top 20 Sectors by Price␣
 ↪Impact (500% Gas Shock to imports into EU28)")
plt.ylabel("Relative Price Change")
plt.xlabel("Sectors")
plt.xticks(rotation=45, ha="right")
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Show the plot
plt.show()

# Read CSV and restore MultiIndex
shock_results_df = pd.read_csv(output_file, index_col=["Country", "Sector"])

# Convert to percentage changes (already stored in % but ensuring consistency)
shock_results_df["Price Change"] *= 100

# Normalize by taking the mean absolute price impact per sector across all␣
 ↪countries
sector_impact_normalized = shock_results_df.groupby(level="Sector")["Price␣
 ↪Change"].mean().abs()

# Sort sectors by the most affected
top_20_sectors = sector_impact_normalized.sort_values(ascending=False).head(20)

# Plot the results
plt.figure(figsize=(12, 6))
top_20_sectors.plot(kind="bar", legend=False, title="Top 20 Most Affected␣
 ↪Sectors (Normalized)")
plt.ylabel("Average Price Impact (%)")  # Normalized
plt.xlabel("Sectors")
plt.xticks(rotation=45, ha="right")
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Show the plot
```

```
plt.show()
```