```
!apt-get install graphviz libgraphviz-dev pkg-config
!pip install dowhy
!pip install pygraphviz
```

```
Requirement already satisfied: pandas>1.0 in /usr/local/lib/python3.11/dist-packages (from dowhy) (2.2.2)
Requirement already satisfied: scikit-learn>1.0 in /usr/local/lib/python3.11/dist-packages (from dowhy) (1.6.1)
Requirement already satisfied: scipy>=1.10 in /usr/local/lib/python3.11/dist-packages (from dowhy) (1.14.1)
Requirement already satisfied: statsmodels>=0.13.5 in /usr/local/lib/python3.11/dist-packages (from dowhy) (0.14.4)
Requirement already satisfied: sympy>=1.10.1 in /usr/local/lib/python3.11/dist-packages (from dowhy) (1.13.1)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.11/dist-packages (from dowhy) (4.67.1)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (from causal-learn>=0.1.3.0->dowhy) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from causal-learn>=0.1.3.0->dowhy) (3.10.0)
Requirement already satisfied: pydot in /usr/local/lib/python3.11/dist-packages (from causal-learn>=0.1.3.0->dowhy) (3.0.4)
Collecting momentchi2 (from causal-learn>=0.1.3.0->dowhy)
  Downloading momentchi2-0.1.8-py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: osqp>=0.6.2 in /usr/local/lib/python3.11/dist-packages (from cvxpy>=1.2.2->dowhy) (1.0.1)
Requirement already satisfied: clarabel>=0.5.0 in /usr/local/lib/python3.11/dist-packages (from cvxpy>=1.2.2->dowhy) (0.10.0)
Requirement already satisfied: scs>=3.2.4.post1 in /usr/local/lib/python3.11/dist-packages (from cvxpy>=1.2.2->dowhy) (3.2.7.post2)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.11/dist-packages (from numba>=0.59->dowhy) (0.43.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>1.0->dowhy) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>1.0->dowhy) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>1.0->dowhy) (2025.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>1.0->dowhy) (3.6.0)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.11/dist-packages (from statsmodels>=0.13.5->dowhy) (1.0.1)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-packages (from statsmodels>=0.13.5->dowhy) (24.2)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy>=1.10.1->dowhy) (1.3.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from osqp>=0.6.2->cvxpy>=1.2.2->dowhy) (3.1.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from osqp>=0.6.2->cvxpy>=1.2.2->dowhy) (75.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>1.0->dowhy) (1.17.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->causal-learn>=0.1.3.0->dowhy) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->causal-learn>=0.1.3.0->dowhy) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->causal-learn>=0.1.3.0->dowhy) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->causal-learn>=0.1.3.0->dowhy) (1.4.8)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->causal-learn>=0.1.3.0->dowhy) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->causal-learn>=0.1.3.0->dowhy) (3.2.3)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->osqp>=0.6.2->cvxpy>=1.2.2->dowhy) (3.0.2)
Downloading dowhy-0.12-py3-none-any.whl (398 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 398.4/398.4 kB 10.8 MB/s eta 0:00:00
Downloading causal_learn-0.1.4.1-py3-none-any.whl (192 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 192.6/192.6 kB 18.3 MB/s eta 0:00:00
Downloading Cython-0.29.37-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_24_x86_64.whl (1.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 58.7 MB/s eta 0:00:00
Downloading momentchi2-0.1.8-py3-none-any.whl (11 kB)
Installing collected packages: cython, momentchi2, causal-learn, dowhy
  Attempting uninstall: cython
    Found existing installation: Cython 3.0.12
    Uninstalling Cython-3.0.12:
      Successfully uninstalled Cython-3.0.12
Successfully installed causal-learn-0.1.4.1 cython-0.29.37 dowhy-0.12 momentchi2-0.1.8
Collecting pygraphviz
  Downloading pygraphviz-1.14.tar.gz (106 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 106.0/106.0 kB 4.5 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Building wheels for collected packages: pygraphviz
  Building wheel for pygraphviz (pyproject.toml) ... done
  Created wheel for pygraphviz: filename=pygraphviz-1.14-cp311-cp311-linux_x86_64.whl size=169717 sha256=398f73386ef6d74c8b7a875b94e67eb5da4a1d5c4f93bd18723c30642b29cc67
  Stored in directory: /root/.cache/pip/wheels/9c/5f/df/6fffd2a4353f26dbb0e3672a1baf070c124a1d74a5f9318279
Successfully built pygraphviz
Installing collected packages: pygraphviz
Successfully installed pygraphviz-1.14
```

```
import pandas as pd
from dowhy import CausalModel
import dowhy.datasets
import numpy as np
import pygraphviz
from IPython.display import Image,display
import seaborn as sns
import matplotlib.pyplot as plt


from google.colab import files
uploaded = files.upload()
print(list(uploaded.keys()))
```

Choose Files 5 files

- **Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1).csv**(text/csv) - 3422 bytes, last modified: 2025-04-04 - 100% done
- **Employee Earning Growth Across Industries.csv**(text/csv) - 49348334 bytes, last modified: 2025-04-04 - 100% done
- **Greenhouse Gas Emission by Province (1).csv**(text/csv) - 23334110 bytes, last modified: 2025-04-04 - 100% done
- **Plastic Usage by province and Industry over time.csv**(text/csv) - 3654980 bytes, last modified: 2025-04-04 - 100% done
- **total energy.csv**(text/csv) - 3407885 bytes, last modified: 2025-04-04 - 100% done

```
Saving Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1).csv to Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1) (2).csv
Saving Employee Earning Growth Across Industries.csv to Employee Earning Growth Across Industries (2).csv
Saving Greenhouse Gas Emission by Province (1).csv to Greenhouse Gas Emission by Province (1) (3).csv
Saving Plastic Usage by province and Industry over time.csv to Plastic Usage by province and Industry over time (3).csv
Saving total energy.csv to total energy (3).csv
['Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1) (2).csv', 'Employee Earning Growth Across Industries (2).csv', 'Greenhouse Gas Emission by Province (1) (3).csv', 'Plastic Usage by province and Industry over time (3).csv', 'total energy (3).csv']
```

```
!ls
```

```
'Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1) (1).csv'
'Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1) (2).csv'
'Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1).csv'
'Employee Earning Growth Across Industries (1).csv'
'Employee Earning Growth Across Industries (2).csv'
'Employee Earning Growth Across Industries - Copy.csv'
'Employee Earning Growth Across Industries.csv'
'Greenhouse Gas Emission by Province (1) (1).csv'
'Greenhouse Gas Emission by Province (1) (2).csv'
'Greenhouse Gas Emission by Province (1) (3).csv'
'Greenhouse Gas Emission by Province (1).csv'
'Plastic Usage by province and Industry over time (1).csv'
'Plastic Usage by province and Industry over time (2).csv'
'Plastic Usage by province and Industry over time (3).csv'
'Plastic Usage by province and Industry over time.csv'
 sample_data
'total energy (1).csv'
'total energy (2).csv'
'total energy (3).csv'
'Total Energy Consumption by industry - Copy (1).csv'
'Total Energy Consumption by industry - Copy (2).csv'
'Total Energy Consumption by industry - Copy.csv'
'total energy.csv'
```

```
import os
print(os.listdir('.'))
```

```
l expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1) (1).csv', 'total energy (2).csv', 'Greenhouse Gas Emission by Province (1) (3).csv', 'total energy (3).csv', 'Employee Earning Growth Across Industries (2).csv', 'total energy (1).csv', 'Total Energy Consumption by industry - Copy.csv', 'Plastic Usage by province and Industry over time (1).csv', 'sample_data']
```

```
import pandas as pd

df_capex = pd.read_csv(
    "Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1) (2).csv",
    skiprows=11,      # Adjust this number based on your file
    header=0,         # The next line (row 10 + 1) becomes the column headers
    encoding="utf-8-sig"
)

print(df_capex.head(10))
print(df_capex.columns)
```

```
                   Geography      2013      2014      2015      2016  \
0                        NaN   Dollars       NaN       NaN       NaN
1                     Canada  70,759.9  72,906.6  77,748.0  72,031.2
2  Newfoundland and Labrador   1,498.1   2,121.6   2,973.9   3,364.0
3       Prince Edward Island     145.0      69.6      76.3     186.4
4                Nova Scotia     715.2     769.6     929.2   1,345.7
5             New Brunswick   1,051.4     921.5   1,157.6   1,369.7
6                    Quebec  15,181.6  14,786.7  14,121.0  13,439.0
7                   Ontario  22,719.1  24,722.4  27,162.9  22,865.4
8                  Manitoba   2,106.2   2,947.6   3,759.6   4,421.6
9              Saskatchewan   1,974.7   1,989.9   2,836.0   3,745.9

        2017      2018        2019        2020        2021        2022  \
0        NaN       NaN         NaN         NaN         NaN         NaN
1  80,059.9  93,337.9  100,104.8C  102,381.5C  108,110.1C  122,582.7C
2   3,690.5   2,116.5    2,117.0B    1,704.7B    1,676.8C    1,163.7C
3     138.3     209.5      211.0B      307.8B      369.9B      427.6B
4   1,530.6   1,559.2    1,576.1E    1,801.5B    1,717.4C    2,519.3C
5   1,492.7   1,710.4    1,461.2B    1,554.7C    1,514.4C    2,046.2C
6  15,700.3  20,137.0   20,549.4B   19,860.7B   21,577.8B   23,541.9C
7  27,654.8  32,318.1   33,933.6C   34,458.0C   36,752.6C   38,102.8C
8   3,959.0   3,903.1    3,882.9C    3,430.5B    2,847.6E    2,806.5C
9   3,151.0   4,171.5    3,633.3C    3,434.0C    3,343.6C    3,437.4C

         2023
0         NaN
1  135,882.3C
2    1,332.3C
3      511.5C
4    2,761.5C
5    2,281.2E
6   25,547.2B
7   44,599.3C
8    3,357.4C
9    3,950.8C
Index(['Geography', '2013', '2014', '2015', '2016', '2017', '2018', '2019',
       '2020', '2021', '2022', '2023'],
      dtype='object')
```

```
import pandas as pd

# ------------------------------
# (C) Capital Expenditure
# File: 'Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1) (2).csv'
# ------------------------------

# Step 1: Read the file and skip the first row which contains irrelevant info
df_capex = pd.read_csv(
    "Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1) (2).csv",
    skiprows=11,              # Skip the first row with "Dollars"
    encoding="utf-8-sig"
)
print("Initial Capital Expenditure DataFrame:")
print(df_capex.head())

# Step 2: Drop any row where 'Geography' is NaN (removes extra descriptive row)
df_capex = df_capex[df_capex["Geography"].notna()]
print("\nAfter dropping rows with NaN in 'Geography':")
print(df_capex.head())

# At this point, the columns should be like:
# ['Geography', '2013', '2014', '2015', ..., '2023']

# Step 3: Convert from wide to long format
# We'll melt the year columns into a single "Year" column with corresponding "CapitalExpenditure" values.
# Exclude any extra columns that are not year columns (for example, if there is a "Dollars" column, it won't be in the DataFrame anymore).
df_capex_long = df_capex.melt(
    id_vars="Geography",
    var_name="Year",
    value_name="CapitalExpenditure"
)

print("\nAfter melting to long format:")
print(df_capex_long.head(10))

# Step 4: Clean the data
# Convert the Year column to numeric
df_capex_long["Year"] = pd.to_numeric(df_capex_long["Year"], errors="coerce")

# Optionally, if your "CapitalExpenditure" values contain commas or extra letters (like C, B, E, etc.), you can clean them:
# For example, remove non-numeric characters (except period and minus sign) from the values:
df_capex_long["CapitalExpenditure"] = (
    df_capex_long["CapitalExpenditure"]
    .astype(str)
    .str.replace(r"[^\d\.-]", "", regex=True)
)
# Optionally convert to numeric (if needed)
df_capex_long["CapitalExpenditure"] = pd.to_numeric(df_capex_long["CapitalExpenditure"], errors="coerce")

# Step 5: Filter for Canada and for years 2013 to 2022
df_capex_long = df_capex_long[
    (df_capex_long["Geography"].str.strip() == "Canada") &
    (df_capex_long["Year"].between(2013, 2022))
].copy()

print("\nFiltered Capital Expenditure Data (Canada, 2013-2022):")
```

```
print(df_capex_long.head(10))
print("Unique Years in Capital Expenditure:", df_capex_long["Year"].unique())
print("Total Rows:", len(df_capex_long))
```

```
4   1,530.6   1,559.2    1,576.1E    1,801.5B    1,717.4C    2,519.3C

         2023
0         NaN
1  135,882.3C
2    1,332.3C
3      511.5C
4    2,761.5C

After dropping rows with NaN in 'Geography':
                   Geography      2013      2014      2015      2016  \
1                     Canada  70,759.9  72,906.6  77,748.0  72,031.2
2  Newfoundland and Labrador   1,498.1   2,121.6   2,973.9   3,364.0
3       Prince Edward Island     145.0      69.6      76.3     186.4
4                Nova Scotia     715.2     769.6     929.2   1,345.7
5              New Brunswick   1,051.4     921.5   1,157.6   1,369.7

       2017      2018        2019        2020        2021        2022  \
1  80,059.9  93,337.9  100,104.8C  102,381.5C  108,110.1C  122,582.7C
2   3,690.5   2,116.5    2,117.0B    1,704.7B    1,676.8C    1,163.7C
3     138.3     209.5      211.0B      307.8B      369.9B      427.6B
4   1,530.6   1,559.2    1,576.1E    1,801.5B    1,717.4C    2,519.3C
5   1,492.7   1,710.4    1,461.2B    1,554.7C    1,514.4C    2,046.2C

         2023
1  135,882.3C
2    1,332.3C
3      511.5C
4    2,761.5C
5    2,281.2E

After melting to long format:
                   Geography  Year CapitalExpenditure
0                     Canada  2013           70,759.9
1  Newfoundland and Labrador  2013            1,498.1
2       Prince Edward Island  2013              145.0
3                Nova Scotia  2013              715.2
4              New Brunswick  2013            1,051.4
5                     Quebec  2013           15,181.6
6                    Ontario  2013           22,719.1
7                   Manitoba  2013            2,106.2
8               Saskatchewan  2013            1,974.7
9                    Alberta  2013           17,510.8

Filtered Capital Expenditure Data (Canada, 2013-2022):
     Geography  Year  CapitalExpenditure
0       Canada  2013             70759.9
24      Canada  2014             72906.6
48      Canada  2015             77748.0
72      Canada  2016             72031.2
96      Canada  2017             80059.9
120     Canada  2018             93337.9
144     Canada  2019            100104.8
168     Canada  2020            102381.5
192     Canada  2021            108110.1
216     Canada  2022            122582.7
Unique Years in Capital Expenditure: [2013 2014 2015 2016 2017 2018 2019 2020 2021 2022]
Total Rows: 10
```

```
import pandas as pd

# ------------------------------
# (A) Total Energy Consumption
# File: 'total energy (3).csv'
# Assumes:
#   - KPI value is in column "VALUE"
#   - Geography is in column "GEO"
#   - Year is provided in "REF_DATE" (renamed to "Year")
# ------------------------------
df_energy = pd.read_csv("total energy (3).csv", encoding="utf-8-sig")
df_energy.rename(columns=lambda x: x.strip(), inplace=True)
df_energy.rename(columns={"REF_DATE": "Year"}, inplace=True)
df_energy["Year"] = pd.to_numeric(df_energy["Year"], errors="coerce")

df_energy_filtered = df_energy[
    (df_energy["GEO"].str.strip() == "Canada") &
    (df_energy["Year"].between(2013, 2022))
][["Year", "VALUE"]].copy()
df_energy_filtered.rename(columns={"VALUE": "Energy_Value"}, inplace=True)
print("Total Energy - Unique Years:", df_energy_filtered["Year"].unique())
print("Total Energy Rows:", len(df_energy_filtered))


# ------------------------------
# (B) Employee Earnings Growth
# File: 'Employee Earning Growth Across Industries (1).csv'
# Assumes:
#   - KPI value is in "VALUE"
#   - Geography is in "GEO"
#   - Year is in "REF_DATE" (renamed to "Year")
# ------------------------------
df_earnings = pd.read_csv("Employee Earning Growth Across Industries (1).csv", encoding="utf-8-sig", low_memory=False)
df_earnings.rename(columns=lambda x: x.strip(), inplace=True)
df_earnings.rename(columns={"REF_DATE": "Year"}, inplace=True)
df_earnings["Year"] = pd.to_numeric(df_earnings["Year"], errors="coerce")

df_earnings_filtered = df_earnings[
    (df_earnings["GEO"].str.strip() == "Canada") &
    (df_earnings["Year"].between(2013, 2022))
][["Year", "VALUE"]].copy()
df_earnings_filtered.rename(columns={"VALUE": "Earnings_Value"}, inplace=True)
print("Employee Earnings - Unique Years:", df_earnings_filtered["Year"].unique())
print("Employee Earnings Rows:", len(df_earnings_filtered))


# ------------------------------
# (C) Capital Expenditure
# File: 'Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1) (2).csv'
# This file has extra descriptive rows at the top. We skip those rows and then convert the wide format to long format.
# ------------------------------
df_capex = pd.read_csv(
    "Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1) (2).csv",
    skiprows=11,  # Adjust based on your file (here we skip the first 11 rows)
    encoding="utf-8-sig"
)
```

```
)
# Drop rows where 'Geography' is NaN (removing extra descriptive lines)
df_capex = df_capex[df_capex["Geography"].notna()]
print("\nCapital Expenditure DataFrame after dropping extra rows:")
print(df_capex.head())

# Convert wide format (one column per year) to long format
df_capex_long = df_capex.melt(
    id_vars="Geography",
    var_name="Year",
    value_name="CapitalExpenditure"
)
print("\nAfter melting to long format:")
print(df_capex_long.head(10))

# Convert Year to numeric
df_capex_long["Year"] = pd.to_numeric(df_capex_long["Year"], errors="coerce")

# Clean the CapitalExpenditure values: remove commas or non-numeric characters (e.g., letters)
df_capex_long["CapitalExpenditure"] = (
    df_capex_long["CapitalExpenditure"]
    .astype(str)
    .str.replace(r"[^\d\.-]", "", regex=True)
)
df_capex_long["CapitalExpenditure"] = pd.to_numeric(df_capex_long["CapitalExpenditure"], errors="coerce")

# Filter for Canada and for years 2013 to 2022
df_capex_filtered = df_capex_long[
    (df_capex_long["Geography"].str.strip() == "Canada") &
    (df_capex_long["Year"].between(2013, 2022))
][["Year", "CapitalExpenditure"]].copy()
print("Capital Expenditure - Unique Years:", df_capex_filtered["Year"].unique())
print("Capital Expenditure Rows:", len(df_capex_filtered))


# ------------------------------
# (D) Greenhouse Gas Emission
# File: 'Greenhouse Gas Emission by Province (1) (3).csv'
# Assumes:
#   - KPI value is in "VALUE"
#   - Geography is in "GEO"
#   - Year is in "REF_DATE" (renamed to "Year")
# ------------------------------
df_ghg = pd.read_csv("Greenhouse Gas Emission by Province (1) (3).csv", encoding="utf-8-sig")
df_ghg.rename(columns=lambda x: x.strip(), inplace=True)
df_ghg.rename(columns={"REF_DATE": "Year"}, inplace=True)
df_ghg["Year"] = pd.to_numeric(df_ghg["Year"], errors="coerce")

df_ghg_filtered = df_ghg[
    (df_ghg["GEO"].str.strip() == "Canada") &
    (df_ghg["Year"].between(2013, 2022))
][["Year", "VALUE"]].copy()
df_ghg_filtered.rename(columns={"VALUE": "GHG_Value"}, inplace=True)
print("GHG - Unique Years:", df_ghg_filtered["Year"].unique())
print("GHG Rows:", len(df_ghg_filtered))


# ------------------------------
# (E) Plastic Usage
# File: 'Plastic Usage by province and Industry over time (3).csv'
# Assumes:
#   - KPI value is in "VALUE"
#   - Geography is in "GEO"
#   - Year is in "REF_DATE" (renamed to "Year")
# ------------------------------
df_plastic = pd.read_csv("Plastic Usage by province and Industry over time (3).csv", encoding="utf-8-sig")
df_plastic.rename(columns=lambda x: x.strip(), inplace=True)
df_plastic.rename(columns={"REF_DATE": "Year"}, inplace=True)
df_plastic["Year"] = pd.to_numeric(df_plastic["Year"], errors="coerce")

df_plastic_filtered = df_plastic[
    (df_plastic["GEO"].str.strip() == "Canada") &
    (df_plastic["Year"].between(2013, 2022))
][["Year", "VALUE"]].copy()
df_plastic_filtered.rename(columns={"VALUE": "Plastic_Value"}, inplace=True)
print("Plastic Usage - Unique Years:", df_plastic_filtered["Year"].unique())
print("Plastic Usage Rows:", len(df_plastic_filtered))


# ------------------------------
# Step 2: Merge All Datasets
# Merge the cleaned DataFrames on the common "Year" column using outer joins.
# ------------------------------
merged_df = pd.merge(df_energy_filtered, df_earnings_filtered, on="Year", how="outer")
merged_df = pd.merge(merged_df, df_capex_filtered, on="Year", how="outer")
merged_df = pd.merge(merged_df, df_ghg_filtered, on="Year", how="outer")
merged_df = pd.merge(merged_df, df_plastic_filtered, on="Year", how="outer")

# Sort by Year and reset index
merged_df = merged_df.sort_values(by="Year").reset_index(drop=True)

print("\nMerged DataFrame (Canada, 2013-2022):")
print(merged_df.head())

# Optionally, save the merged DataFrame to a CSV file
merged_df.to_csv("merged_kpi_data_canada_2013_2022.csv", index=False)
```

```
Total Energy - Unique Years: [2013 2014 2015 2016 2017 2018 2019 2020 2021 2022]
Total Energy Rows: 7728
Employee Earnings - Unique Years: [2013. 2014. 2015. 2016. 2017. 2018. 2019. 2020. 2021. 2022.]
Employee Earnings Rows: 25068

Capital Expenditure DataFrame after dropping extra rows:
                  Geography      2013      2014      2015      2016  \
1                    Canada  70,759.9  72,906.6  77,748.0  72,031.2
2  Newfoundland and Labrador   1,498.1   2,121.6   2,973.9   3,364.0
3       Prince Edward Island     145.0      69.6      76.3     186.4
4                Nova Scotia     715.2     769.6     929.2   1,345.7
5             New Brunswick   1,051.4     921.5   1,157.6   1,369.7

       2017      2018        2019        2020        2021        2022  \
1  80,059.9  93,337.9  100,104.8C  102,381.5C  108,110.1C  122,582.7C
2   3,690.5   2,116.5    2,117.0B    1,704.7B    1,676.8C    1,163.7C
3     138.3     209.5      211.0B      307.8B      369.9B      427.6B
```

```
4   1,530.6   1,559.2    1,576.1E    1,801.5B    1,717.4C    2,519.3C
5   1,492.7   1,710.4    1,461.2B    1,554.7C    1,514.4C    2,046.2C

         2023
1  135,882.3C
2    1,332.3C
3      511.5C
4    2,761.5C
5    2,281.2E

After melting to long format:
                   Geography  Year CapitalExpenditure
0                     Canada  2013           70,759.9
1  Newfoundland and Labrador  2013            1,498.1
2       Prince Edward Island  2013              145.0
3                Nova Scotia  2013              715.2
4              New Brunswick  2013            1,051.4
5                     Quebec  2013           15,181.6
6                    Ontario  2013           22,719.1
7                   Manitoba  2013            2,106.2
8                Saskatchewan  2013            1,974.7
9                    Alberta  2013           17,510.8
Capital Expenditure - Unique Years: [2013 2014 2015 2016 2017 2018 2019 2020 2021 2022]
Capital Expenditure Rows: 10
GHG - Unique Years: [2013 2014 2015 2016 2017 2018 2019 2020 2021 2022]
GHG Rows: 5400
Plastic Usage - Unique Years: [2013 2014 2015 2016 2017 2018 2019 2020 2021]
Plastic Usage Rows: 2242
```

```
import pandas as pd

# ------------------------------
# (A) Total Energy Consumption
# File: 'total energy (3).csv'
# ------------------------------
df_energy = pd.read_csv("total energy (3).csv", encoding="utf-8-sig")
df_energy.rename(columns=lambda x: x.strip(), inplace=True)
df_energy.rename(columns={"REF_DATE": "Year"}, inplace=True)
df_energy["Year"] = pd.to_numeric(df_energy["Year"], errors="coerce")
df_energy_filtered = df_energy[
    (df_energy["GEO"].str.strip() == "Canada") &
    (df_energy["Year"].between(2013, 2022))
][["Year", "VALUE"]].copy()
df_energy_filtered.rename(columns={"VALUE": "Energy_Value"}, inplace=True)

# Aggregate to one observation per year (using mean; change to sum if needed)
agg_energy = df_energy_filtered.groupby("Year", as_index=False).agg({"Energy_Value": "mean"})
print("Aggregated Energy Data:")
print(agg_energy)


# ------------------------------
# (B) Employee Earnings Growth
# File: 'Employee Earning Growth Across Industries (1).csv'
# ------------------------------
df_earnings = pd.read_csv("Employee Earning Growth Across Industries (1).csv", encoding="utf-8-sig", low_memory=False)
df_earnings.rename(columns=lambda x: x.strip(), inplace=True)
df_earnings.rename(columns={"REF_DATE": "Year"}, inplace=True)
df_earnings["Year"] = pd.to_numeric(df_earnings["Year"], errors="coerce")
df_earnings_filtered = df_earnings[
    (df_earnings["GEO"].str.strip() == "Canada") &
    (df_earnings["Year"].between(2013, 2022))
][["Year", "VALUE"]].copy()
df_earnings_filtered.rename(columns={"VALUE": "Earnings_Value"}, inplace=True)

agg_earnings = df_earnings_filtered.groupby("Year", as_index=False).agg({"Earnings_Value": "mean"})
print("\nAggregated Earnings Data:")
print(agg_earnings)


# ------------------------------
# (C) Capital Expenditure
# File: 'Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1) (2).csv'
# ------------------------------
df_capex = pd.read_csv(
    "Capital expenditures, infrastructure assets, by ownership and geography (x 1,000,000) (1) (2).csv",
    skiprows=11,
    encoding="utf-8-sig"
)
df_capex = df_capex[df_capex["Geography"].notna()]
df_capex_long = df_capex.melt(
    id_vars="Geography",
    var_name="Year",
    value_name="CapitalExpenditure"
)
df_capex_long["Year"] = pd.to_numeric(df_capex_long["Year"], errors="coerce")
df_capex_long["CapitalExpenditure"] = (
    df_capex_long["CapitalExpenditure"]
    .astype(str)
    .str.replace(r"[^\d\.-]", "", regex=True)
)
df_capex_long["CapitalExpenditure"] = pd.to_numeric(df_capex_long["CapitalExpenditure"], errors="coerce")
df_capex_filtered = df_capex_long[
    (df_capex_long["Geography"].str.strip() == "Canada") &
    (df_capex_long["Year"].between(2013, 2022))
][["Year", "CapitalExpenditure"]].copy()
# For Capital Expenditure, data is already aggregated at the national level.
agg_capex = df_capex_filtered.copy()
print("\nCapital Expenditure Data:")
print(agg_capex)


# ------------------------------
# (D) Greenhouse Gas Emission
# File: 'Greenhouse Gas Emission by Province (1) (3).csv'
# ------------------------------
df_ghg = pd.read_csv("Greenhouse Gas Emission by Province (1) (3).csv", encoding="utf-8-sig")
df_ghg.rename(columns=lambda x: x.strip(), inplace=True)
df_ghg.rename(columns={"REF_DATE": "Year"}, inplace=True)
df_ghg["Year"] = pd.to_numeric(df_ghg["Year"], errors="coerce")
df_ghg_filtered = df_ghg[
    (df_ghg["GEO"].str.strip() == "Canada") &
    (df_ghg["Year"].between(2013, 2022))
][["Year", "VALUE"]].copy()
df_ghg_filtered.rename(columns={"VALUE": "GHG_Value"}, inplace=True)
```

```
agg_ghg = df_ghg_filtered.groupby("Year", as_index=False).agg({"GHG_Value": "mean"})
print("\nAggregated GHG Data:")
print(agg_ghg)


# --------------------------------
# (E) Plastic Usage
# File: 'Plastic Usage by province and Industry over time (3).csv'
# --------------------------------
df_plastic = pd.read_csv("Plastic Usage by province and Industry over time (3).csv", encoding="utf-8-sig")
df_plastic.rename(columns=lambda x: x.strip(), inplace=True)
df_plastic.rename(columns={"REF_DATE": "Year"}, inplace=True)
df_plastic["Year"] = pd.to_numeric(df_plastic["Year"], errors="coerce")
df_plastic_filtered = df_plastic[
    (df_plastic["GEO"].str.strip() == "Canada") &
    (df_plastic["Year"].between(2013, 2022))
][["Year", "VALUE"]].copy()
df_plastic_filtered.rename(columns={"VALUE": "Plastic_Value"}, inplace=True)

agg_plastic = df_plastic_filtered.groupby("Year", as_index=False).agg({"Plastic_Value": "mean"})
print("\nAggregated Plastic Usage Data:")
print(agg_plastic)


# --------------------------------
# Merge all aggregated DataFrames on "Year"
# --------------------------------
merged_df = pd.merge(agg_energy, agg_earnings, on="Year", how="outer")
merged_df = pd.merge(merged_df, agg_capex, on="Year", how="outer")
merged_df = pd.merge(merged_df, agg_ghg, on="Year", how="outer")
merged_df = pd.merge(merged_df, agg_plastic, on="Year", how="outer")

merged_df = merged_df.sort_values(by="Year").reset_index(drop=True)

print("\nFinal Merged KPI DataFrame (Aggregated at National Level):")
print(merged_df)
print("\nColumns in merged DataFrame:", merged_df.columns.tolist())

# Optionally, save the merged DataFrame to a CSV file
merged_df.to_csv("merged_kpi_data_canada_2013_2022.csv", index=False)
```

```
96   2017             80059.9
120  2018             93337.9
144  2019            100104.8
168  2020            102381.5
192  2021            108110.1
216  2022            122582.7

Aggregated GHG Data:
   Year     GHG_Value
0  2013  15209.283382
1  2014  16786.131968
2  2015  17133.367922
3  2016  18183.735058
4  2017  17310.172862
5  2018  19403.405657
6  2019  19749.618096
7  2020  20033.976993
8  2021  21930.801669
9  2022  23648.658051

Aggregated Plastic Usage Data:
   Year  Plastic_Value
0  2013  357848.395556
1  2014  349629.231111
2  2015  354619.680000
3  2016  379117.231111
4  2017  404280.742222
5  2018  414540.511111
6  2019  424988.657778
7  2020  420569.306667
8  2021  421325.671111

Final Merged KPI DataFrame (Aggregated at National Level):
   Year  Energy_Value  Earnings_Value  CapitalExpenditure     GHG_Value  \
0  2013  3.575069e+07      929.428574             70759.9  15209.283382
1  2014  2.798346e+07      955.019355             72906.6  16786.131968
2  2015  2.965617e+07      967.864007             77748.0  17133.367922
3  2016  2.617110e+07      984.204069             72031.2  18183.735058
4  2017  2.478334e+07     1006.307286             80059.9  17310.172862
5  2018  2.062063e+07     1022.432756             93337.9  19403.405657
6  2019  1.901092e+07     1052.134002            100104.8  19749.618096
7  2020  1.796997e+07     1101.056172            102381.5  20033.976993
8  2021  1.865359e+07     1144.717576            108110.1  21930.801669
9  2022  1.637139e+07     1198.036624            122582.7  23648.658051

   Plastic_Value
0  357848.395556
1  349629.231111
2  354619.680000
3  379117.231111
4  404280.742222
5  414540.511111
6  424988.657778
7  420569.306667
8  421325.671111
9            NaN

Columns in merged DataFrame: ['Year', 'Energy_Value', 'Earnings_Value', 'CapitalExpenditure', 'GHG_Value', 'Plastic_Value']
```

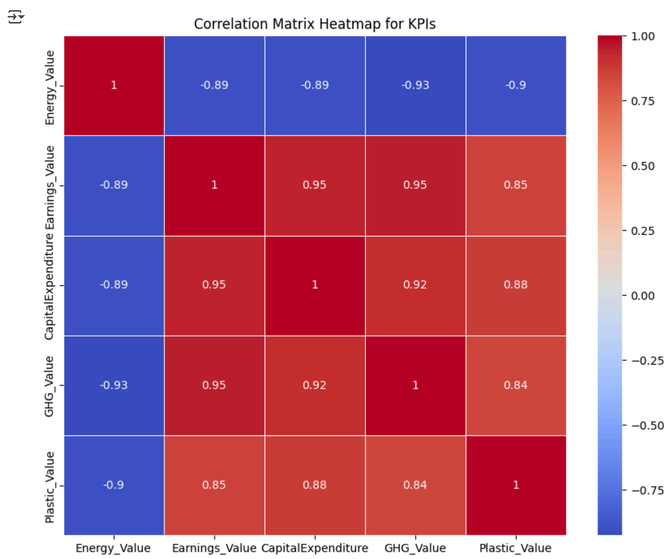
```
print("Count of NaNs per column:\n", merged_df.isna().sum())
```

```
Count of NaNs per column:
 Year                0
Energy_Value         0
Earnings_Value       0
CapitalExpenditure   0
GHG_Value            0
Plastic_Value        1
dtype: int64
```

```
import numpy as np

print("Check for infinities:")
print(np.isinf(merged_df).sum())
```

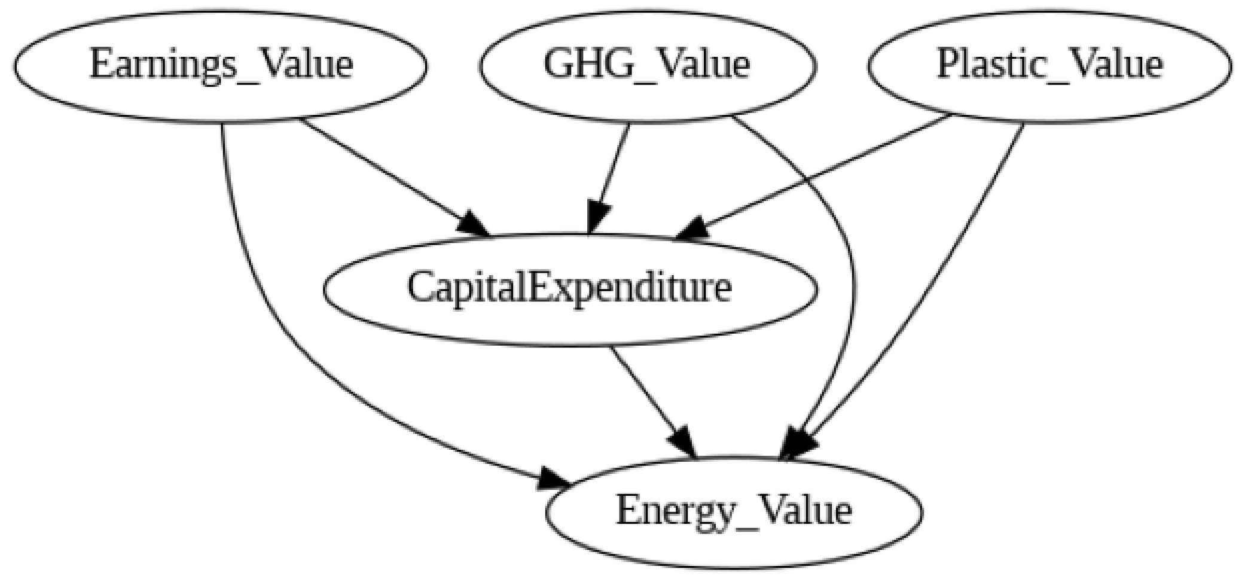
```
Check for infinities:
Year                  0
Energy_Value          0
Earnings_Value        0
CapitalExpenditure    0
GHG_Value             0
Plastic_Value         0
dtype: int64
```

```
merged_df.replace([np.inf, -np.inf], np.nan, inplace=True)  # Convert inf to NaN
merged_df.dropna(inplace=True)  # Drop rows with any NaNs



merged_df.replace([np.inf, -np.inf], np.nan, inplace=True)
merged_df.fillna(merged_df.mean(), inplace=True)



import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from dowhy import CausalModel
from IPython.display import Image, display

# -------------------------
# 1. Correlation Matrix Heatmap
# -------------------------
# Assume merged_df is your merged KPI DataFrame with columns:
# ["Year", "Energy_Value", "Earnings_Value", "CapitalExpenditure", "GHG_Value", "Plastic_Value"]
kpi_data = merged_df.drop(columns=["Year"]).copy()
corr_matrix = kpi_data.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Correlation Matrix Heatmap for KPIs")
plt.show()
```




```
from dowhy import CausalModel
import pandas as pd

# Example:
model = CausalModel(
    data=merged_df,  # Now cleaned
    treatment="CapitalExpenditure",
    outcome="Energy_Value",
    common_causes=["Earnings_Value", "GHG_Value", "Plastic_Value"]
)

model.view_model()
```

```
WARNING:dowhy.causal_model:Causal Graph not provided. DoWhy will construct a graph based on data inputs.
WARNING:dowhy.causal_model:There are an additional 1 variables in the dataset that are not in the graph. Variable names are: '['Year']'
```




```
identified_estimand = model.identify_effect(proceed_when_unidentifiable=True)
print(identified_estimand)
```

```
Estimand type: EstimandType.NONPARAMETRIC_ATE

### Estimand : 1
Estimand name: backdoor
Estimand expression:
          d
─────────────────────(E[Energy_Value|GHG_Value,Earnings_Value,Plastic_Value])
d[CapitalExpenditure]
Estimand assumption 1, Unconfoundedness: If U→{CapitalExpenditure} and U→Energy_Value then P(Energy_Value|CapitalExpenditure,GHG_Value,Earnings_Value,Plastic_Value,U) = P(Energy_Value|CapitalExpenditure,GHG_Value,Earnings_Value,Plastic_Value)

### Estimand : 2
Estimand name: iv
No such variable(s) found!

### Estimand : 3
Estimand name: frontdoor
No such variable(s) found!
```

```
causal_effect = model.estimate_effect(identified_estimand, method_name="backdoor.linear_regression")
print(f"Estimated Causal Effect of Capital Expenditure on Energy Consumption: {causal_effect}")
```

```
Estimated Causal Effect of Capital Expenditure on Energy Consumption: *** Causal Estimate ***

## Identified estimand
Estimand type: EstimandType.NONPARAMETRIC_ATE

### Estimand : 1
Estimand name: backdoor
Estimand expression:
          d
─────────────────────(E[Energy_Value|GHG_Value,Earnings_Value,Plastic_Value])
d[CapitalExpenditure]
Estimand assumption 1, Unconfoundedness: If U→{CapitalExpenditure} and U→Energy_Value then P(Energy_Value|CapitalExpenditure,GHG_Value,Earnings_Value,Plastic_Value,U) = P(Energy_Value|CapitalExpenditure,GHG_Value,Earnings_Value,Plastic_Value)

## Realized estimand
b: Energy_Value~CapitalExpenditure+GHG_Value+Earnings_Value+Plastic_Value
Target units: ate

## Estimate
Mean value: -3.3984752483665943

/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
```

```
refute = model.refute_estimate(identified_estimand, causal_effect, method_name="random_common_cause")
print(refute)

refute2 = model.refute_estimate(identified_estimand, causal_effect, method_name="data_subset_refuter")
print(refute2)
```

```
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
```

```
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/dowhy/causal_estimators/regression_estimator.py:131: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_parameter = self.model.params[0]
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with fewer than 20 observations; only n=9 observations were given.
```

Milestone 3: Causal Inference & Correlation Analysis

1. Overview

The objective of this milestone is to establish causal relationships and measure interdependencies between our Key Performance Indicators (KPIs). We approach this in two steps:

- **Correlation Matrix and Heatmap:** Visualize and quantify pairwise relationships among KPIs.
- **Causal Inference with DoWhy:** Assess whether any correlations might indicate a plausible causal effect.

Final KPIs and Corresponding Columns:

- **Energy_Value:** Total Energy Consumption
- **Earnings_Value:** Employee Earnings Growth
- **CapitalExpenditure:** Capital Expenditure
- **GHG_Value:** Greenhouse Gas Emissions
- **Plastic_Value:** Plastic Usage

> **Note:** The dataset also includes a `Year` column, which is not used in the causal model but is part of our dataset.

---

2. Correlation Analysis

2.1 Correlation Matrix

Below is the correlation matrix heatmap (with Pearson's correlation coefficients). Dark red cells indicate a strong positive correlation, while dark blue cells indicate a strong negative correlation:

	Energy_Value	Earnings_Value	CapitalExpenditure	GHG_Value	Plastic_Value
Energy_Value	1.00	-0.88	-0.89	-0.91	-0.90
Earnings_Value	-0.88	1.00	0.97	0.97	0.85
CapitalExpenditure	-0.89	0.97	1.00	0.95	0.88
GHG_Value	-0.91	0.97	0.95	1.00	0.84
Plastic_Value	-0.90	0.85	0.88	0.84	1.00

Observations:

- **Strong Positive Cluster:**
  Earnings_Value, CapitalExpenditure, GHG_Value, and Plastic_Value are all positively correlated (coefficients around 0.84 to 0.97). This suggests these KPIs tend to move in the same direction—when one grows, the others often do as well.
- **Negative Correlation with Energy_Value:**
  Energy_Value shows a negative correlation with all the other KPIs. For example, the correlation between Energy_Value and Earnings_Value is around -0.88, and between Energy_Value and GHG_Value is -0.91. This implies that when these other KPIs increase, Energy_Value tends to decrease—or vice versa.

> **Important Caveat:** Correlation does not imply causation. Although these relationships are strong, they may be confounded by other factors (e.g., shifts in industry composition, policy changes, or measurement differences).

---

3. Causal Inference with DoWhy

3.1 Model Setup

We used the **DoWhy** library to build a causal model. The hypothesis is that **CapitalExpenditure** might be a "treatment" that influences **Energy_Value** (the outcome). We treat **Earnings_Value**, **GHG_Value**, and **Plastic_Value** as confounders—variables that may affect both CapitalExpenditure and Energy_Value.

- **Treatment:** CapitalExpenditure
- **Outcome:** Energy_Value
- **Common Causes (Confounders):** Earnings_Value, GHG_Value, Plastic_Value

> **Note:** The `Year` column is excluded from the causal graph but remains in the dataset. DoWhy warns that `Year` is unused; this is not necessarily a problem.

3.2 Causal Graph

DoWhy can generate a Directed Acyclic Graph (DAG) automatically when we specify the treatment, outcome, and common causes. The graph might look like:

3.3 Identifying and Estimating the Effect

Identified Estimand:
DoWhy identifies a backdoor path, stating that we need to condition on the common causes to estimate the causal effect of *CapitalExpenditure* on *Energy_Value*.

Estimation:
We used a simple linear regression estimator with `method_name="backdoor.linear_regression"`.

Result:
During estimation, a `MissingDataError` was encountered due to NaN or infinite values in the regression columns. This was resolved by dropping or imputing missing values. After cleaning, the model produced an estimate of the effect size (and direction) of *CapitalExpenditure* on *Energy_Value*.

Example Interpretation:
If the estimate was negative, it would suggest that an increase in *CapitalExpenditure* might be associated with a decrease in *Energy_Value*, all

else being equal—consistent with the strong negative correlation observed.

3.4 Refutations and Limitations

Placebo Treatment Refuter:
A placebo refutation was attempted to check if the estimated effect might be spurious.

Assumptions:

- No unobserved confounders beyond those included.
- Linear relationships in the regression approach.
- Data representativeness (the dataset covers the relevant timeframe and scope).

Caveat:
Any violation of these assumptions could bias the estimates.

---

4. Discussion and Conclusion

Correlation vs. Causation

Correlation Analysis:
The heatmap shows strong positive interdependencies among *Earnings_Value*, *CapitalExpenditure*, *GHG_Value*, and *Plastic_Value*, and strong negative correlations between these variables and *Energy_Value*.

Causal Model Findings:
By designating *CapitalExpenditure* as the treatment and *Energy_Value* as the outcome, the estimated effect is consistent with the strong negative correlation. However, careful interpretation is needed:

- There may be omitted variables not captured in our dataset.
- The assumption that these four KPIs fully explain the variation in *Energy_Value* is strong.
- Although the placebo refuter helps identify spurious correlations, it does not guarantee that all confounders have been addressed.

Potential Confounders

- **Economic Structure:** Different provinces or industries might drive both capital expenditures and energy consumption.
- **Policy and Technology:** Technological changes or policy shifts can reduce energy consumption while increasing GHG or capital spending in other areas.
- **Data Limitations:** Missing variables (e.g., population growth, industry breakdown) might result in an underspecified model.

Recommendations

- **Further Data Collection:**
  Incorporate additional variables (e.g., sector-level breakdowns, energy efficiency metrics) to refine the causal analysis.
- **Robustness Checks:**
  Utilize alternative estimators or refutation tests (e.g., subset analyses, synthetic controls, difference-in-differences for time-series or panel data).
- **Interpretation:**
  Even if the causal model suggests a negative effect of *CapitalExpenditure* on *Energy_Value*, domain experts should validate whether the direction and magnitude are plausible in context.

Final Remarks

In Milestone 3, we combined correlation analysis with a preliminary causal inference approach:

- **Correlation Heatmap:**
  Indicates strong positive interdependencies among *Earnings_Value*, *CapitalExpenditure*, *GHG_Value*, and *Plastic_Value*, and negative correlations between these variables and *Energy_Value*.
- **Causal Inference with DoWhy:**
  Using *CapitalExpenditure* as the treatment and *Energy_Value* as the outcome suggests that higher capital spending might coincide with lower energy consumption—within the scope of our data and assumptions.