Prediction of CO2 emissions from country-specific data

Problem Statement:.

Analysis of country-specific data and development of machine learning models in order to predict CO2 emissions from country parameters. The project uses the publicly available dataset Climate Change Data from the World Bank Group.

In [16]:
```
#Reading the data

import pandas as pd
carbon = pd.read_excel(r'C:\Users\Deepak Singh\Downloads\climate_change_download_0.xls')
```

In [17]:
```
#First 5 rows
carbon.head()
```

Out[17]:

	Country code	Country name	Series code	Series name	SCALE	Decimals	1990	1991	1992	1993	...
0	ABW	Aruba	AG.LND.EL5M.ZS	Land area below 5m (% of land area)	0	1	29.57481	..	..	..	...
1	ADO	Andorra	AG.LND.EL5M.ZS	Land area below 5m (% of land area)	0	1	0	..	..	..	...
2	AFG	Afghanistan	AG.LND.EL5M.ZS	Land area below 5m (% of land area)	0	1	0	..	..	..	...
3	AGO	Angola	AG.LND.EL5M.ZS	Land area below 5m (% of land area)	0	1	0.208235	..	..	..	...
4	ALB	Albania	AG.LND.EL5M.ZS	Land area below 5m (% of land area)	0	1	4.967875	..	..	..	...

5 rows × 28 columns

In [18]:

```
#Last 5 rows
carbon.tail()
```

Out[18]:

	Country code	Country name	Series code	Series name	SCALE	Decimals	1990	19
13507	YEM	Yemen, Rep.	SP.URB.TOTL	Urban population	0	0	2497175.681	2693642.26
13508	ZAF	South Africa	SP.URB.TOTL	Urban population	0	0	18304000	18864881.5760
13509	ZAR	Congo, Dem. Rep.	SP.URB.TOTL	Urban population	0	0	10120930.828	10569454.39
13510	ZMB	Zambia	SP.URB.TOTL	Urban population	0	0	3096860.882	3141668.29
13511	ZWE	Zimbabwe	SP.URB.TOTL	Urban population	0	0	3036068.58	3175022.7

5 rows × 28 columns

In [19]:

```
carbon.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13512 entries, 0 to 13511
Data columns (total 28 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Country code  13512 non-null  object
 1   Country name  13512 non-null  object
 2   Series code   13512 non-null  object
 3   Series name   13512 non-null  object
 4   SCALE         13512 non-null  object
 5   Decimals      13512 non-null  object
 6   1990          10017 non-null  object
 7   1991          10017 non-null  object
 8   1992          10017 non-null  object
 9   1993          10017 non-null  object
 10  1994          10017 non-null  object
 11  1995          10017 non-null  object
 12  1996          10017 non-null  object
 13  1997          10017 non-null  object
 14  1998          10017 non-null  object
 15  1999          10017 non-null  object
 16  2000          10017 non-null  object
 17  2001          10017 non-null  object
 18  2002          10017 non-null  object
 19  2003          10017 non-null  object
 20  2004          10017 non-null  object
 21  2005          10017 non-null  object
 22  2006          10017 non-null  object
 23  2007          10017 non-null  object
 24  2008          10017 non-null  object
 25  2009          10017 non-null  object
 26  2010          10017 non-null  object
 27  2011          12382 non-null  object
dtypes: object(28)
memory usage: 2.9+ MB
```

In [20]:

```
#Descriptive statistics
carbon.describe()
```

Out[20]:

	Country code	Country name	Series code	Series name	SCALE	Decimals	1990	1991	1992	1993
count	13512	13512	13512	13512	13512	13512	10017	10017	10017	10017
unique	233	233	58	58	2	3	4355	3398	3523	3583
top	ABW	Aruba	AG.LND.EL5M.ZS	Land area below 5m (% of land area)	0	1	..	..	..	..
freq	58	58	233	233	10017	5823	5163	6520	6364	6300

4 rows × 28 columns

In [21]:

```
#Shape of the data
carbon.shape
```

Out[21]:

```
(13512, 28)
```

In [22]:

```
#Available columns
carbon.columns
```

Out[22]:

```
Index(['Country code', 'Country name',  'Series code',   'Series name',
                'SCALE',      'Decimals',           1990,            1991,
                   1992,            1993,           1994,            1995,
                   1996,            1997,           1998,            1999,
                   2000,            2001,           2002,            2003,
                   2004,            2005,           2006,            2007,
                   2008,            2009,           2010,            2011],
      dtype='object')
```

In [23]:

```
#Number of unique values
carbon['Series name'].unique()
```

Out[23]:
```
array(['Land area below 5m (% of land area)',
       'Agricultural land under irrigation (% of total ag. land)',
       'Cereal yield (kg per hectare)',
       'Foreign direct investment, net inflows (% of GDP)',
       'Access to electricity (% of total population)',
       'Energy use per units of GDP (kg oil eq./$1,000 of 2005 PPP $)',
       'Energy use per capita (kilograms of oil equivalent)',
       'CO2 emissions, total (KtCO2)',
       'CO2 emissions per capita (metric tons)',
       'CO2 emissions per units of GDP (kg/$1,000 of 2005 PPP $)',
       'Other GHG emissions, total (KtCO2e)',
       'Methane (CH4) emissions, total (KtCO2e)',
       'Nitrous oxide (N2O) emissions, total (KtCO2e)',
       'Annex-I emissions reduction target',
       'Disaster risk reduction progress score (1-5 scale; 5=best)',
       'GHG net emissions/removals by LUCF (MtCO2e)',
       'Hosted Clean Development Mechanism (CDM) projects',
       'Hosted Joint Implementation (JI) projects',
       'Average annual precipitation (1961-1990, mm)',
       'Issued Certified Emission Reductions (CERs) from CDM (thousands)',
       'Issued Emission Reduction Units (ERUs) from JI (thousands)',
       'Droughts, floods, extreme temps (% pop. avg. 1990-2009)',
       'Average daily min/max temperature (1961-1990, Celsius)',
       'NAMA submission', 'NAPA submission',
       'Latest UNFCCC national communication',
       'Projected annual temperature change (2045-2065, Celsius)',
       'Projected change in annual cool days/cold nights',
       'Projected change in annual hot days/warm nights',
       'Projected annual precipitation change (2045-2065, mm)',
       'Renewable energy target', 'Population below 5m (% of total)',
       'Population in urban agglomerations >1million (%)',
       'Annual freshwater withdrawals (% of internal resources)',
       'Nationally terrestrial protected areas (% of total land area)',
       'Ease of doing business (ranking 1-183; 1=best)',
       'Invest. in energy w/ private participation ($)',
       'Invest. in telecoms w/ private participation ($)',
       'Invest. in transport w/ private participation ($)',
       'Invest. in water/sanit. w/ private participation ($)',
       'Public sector mgmt & institutions avg. (1-6 scale; 6=best)',
       'Paved roads (% of total roads)', 'GDP ($)',
       'GNI per capita (Atlas $)',
       'Ratio of girls to boys in primary & secondary school (%)',
       'Primary completion rate, total (% of relevant age group)',
       'Under-five mortality rate (per 1,000)',
       'Access to improved water source (% of total pop.)',
       'Nurses and midwives (per 1,000 people)',
       'Physicians (per 1,000 people)',
       'Malaria incidence rate (per 100,000 people)',
       'Access to improved sanitation (% of total pop.)',
       'Child malnutrition, underweight (% of under age 5)',
       'Population living below $1.25 a day (% of total)',
       'Population growth (annual %)', 'Population',
       'Urban population growth (annual %)', 'Urban population'],
      dtype=object)
```

In [24]:
```
carbon['Series code'].unique()
```

Out[24]:
```
array(['AG.LND.EL5M.ZS', 'AG.LND.IRIG.AG.ZS', 'AG.YLD.CREL.KG',
       'BX.KLT.DINV.WD.GD.ZS', 'EG.ELC.ACCS.ZS', 'EG.USE.COMM.GD.PP.KD',
       'EG.USE.PCAP.KG.OE', 'EN.ATM.CO2E.KT', 'EN.ATM.CO2E.PC',
       'EN.ATM.CO2E.PP.GD.KD', 'EN.ATM.GHGO.KT.CE', 'EN.ATM.METH.KT.CE',
       'EN.ATM.NOXE.KT.CE', 'EN.CLC.AERT', 'EN.CLC.DRSK.XQ',
       'EN.CLC.GHGR.MT.CE', 'EN.CLC.HCDM', 'EN.CLC.HJIP',
       'EN.CLC.HPPT.MM', 'EN.CLC.ICER', 'EN.CLC.IERU', 'EN.CLC.MDAT.ZS',
       'EN.CLC.MMDT.C', 'EN.CLC.NAMA', 'EN.CLC.NAPA', 'EN.CLC.NCOM',
       'EN.CLC.PCAT.C', 'EN.CLC.PCCC', 'EN.CLC.PCHW', 'EN.CLC.PCPT.MM',
       'EN.CLC.RNET', 'EN.POP.EL5M.ZS', 'EN.URB.MCTY.TL.ZS',
       'ER.H2O.FWTL.ZS', 'ER.LND.PTLD.ZS', 'IC.BUS.EASE.XQ',
       'IE.PPI.ENGY.CD', 'IE.PPI.TELE.CD', 'IE.PPI.TRAN.CD',
       'IE.PPI.WATR.CD', 'IQ.CPA.PUBS.XQ', 'IS.ROD.PAVE.ZS',
       'NY.GDP.MKTP.CD', 'NY.GNP.PCAP.CD', 'SE.ENR.PRSC.FM.ZS',
       'SE.PRM.CMPT.ZS', 'SH.DYN.MORT', 'SH.H2O.SAFE.ZS',
       'SH.MED.NUMW.P3', 'SH.MED.PHYS.ZS', 'SH.MLR.INCD', 'SH.STA.ACSN',
       'SH.STA.MALN.ZS', 'SI.POV.DDAY', 'SP.POP.GROW', 'SP.POP.TOTL',
       'SP.URB.GROW', 'SP.URB.TOTL'], dtype=object)
```

In [25]:
```
carbon['SCALE'].unique()
```

Out[25]:
```
array([0, 'Text'], dtype=object)
```

In [26]:
```
carbon['Decimals'].unique()
```

Out[26]:
```
array([1, 0, 'Text'], dtype=object)
```

In [27]:
```
carbon[carbon['SCALE']=='Text']
```

Out[27]:

	Country code	Country name	Series code	Series name	SCALE	Decimals	1990	1991	1992	1993	..
3029	ABW	Aruba	EN.CLC.AERT	Annex-I emissions reduction target	Text	Text	NaN	NaN	NaN	NaN	..
3030	ADO	Andorra	EN.CLC.AERT	Annex-I emissions reduction target	Text	Text	NaN	NaN	NaN	NaN	..
3031	AFG	Afghanistan	EN.CLC.AERT	Annex-I emissions reduction target	Text	Text	NaN	NaN	NaN	NaN	..
3032	AGO	Angola	EN.CLC.AERT	Annex-I emissions reduction target	Text	Text	NaN	NaN	NaN	NaN	..
3033	ALB	Albania	EN.CLC.AERT	Annex-I emissions reduction target	Text	Text	NaN	NaN	NaN	NaN	..
...	...	...	...	...	...	...	...	...	...	...	..
7218	YEM	Yemen, Rep.	EN.CLC.RNET	Renewable energy target	Text	Text	NaN	NaN	NaN	NaN	..
7219	ZAF	South Africa	EN.CLC.RNET	Renewable energy target	Text	Text	NaN	NaN	NaN	NaN	..
7220	ZAR	Congo, Dem. Rep.	EN.CLC.RNET	Renewable energy target	Text	Text	NaN	NaN	NaN	NaN	..
7221	ZMB	Zambia	EN.CLC.RNET	Renewable energy target	Text	Text	NaN	NaN	NaN	NaN	..
7222	ZWE	Zimbabwe	EN.CLC.RNET	Renewable energy target	Text	Text	NaN	NaN	NaN	NaN	..

3495 rows × 28 columns

In [28]:

```
carbon[carbon['Decimals']=='Text']
```

Out[28]:

	Country code	Country name	Series code	Series name	SCALE	Decimals	1990	1991	1992	1993	..
3029	ABW	Aruba	EN.CLC.AERT	Annex-I emissions reduction target	Text	Text	NaN	NaN	NaN	NaN	..
3030	ADO	Andorra	EN.CLC.AERT	Annex-I emissions reduction target	Text	Text	NaN	NaN	NaN	NaN	..
3031	AFG	Afghanistan	EN.CLC.AERT	Annex-I emissions reduction target	Text	Text	NaN	NaN	NaN	NaN	..
3032	AGO	Angola	EN.CLC.AERT	Annex-I emissions reduction target	Text	Text	NaN	NaN	NaN	NaN	..
3033	ALB	Albania	EN.CLC.AERT	Annex-I emissions reduction target	Text	Text	NaN	NaN	NaN	NaN	..
...	...	...	...	...	...	...	...	...	...	...	...
7218	YEM	Yemen, Rep.	EN.CLC.RNET	Renewable energy target	Text	Text	NaN	NaN	NaN	NaN	..
7219	ZAF	South Africa	EN.CLC.RNET	Renewable energy target	Text	Text	NaN	NaN	NaN	NaN	..
7220	ZAR	Congo, Dem. Rep.	EN.CLC.RNET	Renewable energy target	Text	Text	NaN	NaN	NaN	NaN	..
7221	ZMB	Zambia	EN.CLC.RNET	Renewable energy target	Text	Text	NaN	NaN	NaN	NaN	..
7222	ZWE	Zimbabwe	EN.CLC.RNET	Renewable energy target	Text	Text	NaN	NaN	NaN	NaN	..

3495 rows × 28 columns

Findings:

shape: 28 columns, 13512 rows

all columns are of type "object" - neither numeric, nor string/text values

A certain amount of missing values, denoted both as NaN (not a number values) and as the string ".."

The rows marked as 'Text' in the columns 'SCALE' and 'Decimals' do not contain any information, almost completely composed of NaN values

The columns represent key values such as country, but also the corresponding years and the series code/name

The columns 'Country name', 'Series code', 'SCALE' and 'Decimals' do not give any information and are therefore obsolete

The column 'Series name' contains the country-specific features required for the analysis

The names of the features in the column 'Series name' are clear but too long

Data Cleaning

In [29]:
```
# assign the data to a new DataFrame, which will be modified
carbon_clean = carbon

print("Original number of rows:")
print(carbon_clean.shape[0])

# remove rows characterized as "Text" in the SCALE column
carbon_clean = carbon_clean[carbon_clean['SCALE']!='Text']

print("Current number of rows:")
print(carbon_clean.shape[0])
```

```
Original number of rows:
13512
Current number of rows:
10017
```

In [30]:
```
print("Original number of columns:")
print(carbon_clean.shape[1])

carbon_clean = carbon_clean.drop(['Country name', 'Series code', 'SCALE', 'Decimals'], axis

print("Current number of columns:")
print(carbon_clean.shape[1])
```

```
Original number of columns:
28
Current number of columns:
24
```

In [32]:
```
#Transform the ".." strings and emplty cells ("") into NaN values for easier recognission a
import numpy as np
carbon_clean.iloc[:,2:] = carbon_clean.iloc[:,2:].replace({'':np.nan, '..':np.nan})
```

```
C:\Users\Deepak Singh\AppData\Local\Temp\ipykernel_4608\3502545624.py:3: FutureWarning: Down
casting behavior in `replace` is deprecated and will be removed in a future version. To reta
in the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the fu
ture behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  carbon_clean.iloc[:,2:] = carbon_clean.iloc[:,2:].replace({'':np.nan, '..':np.nan})
```

In [33]:
```
#Transform all data columns into a numerical data type
carbon_clean2 = carbon_clean.applymap(lambda x: pd.to_numeric(x, errors='ignore'))

print("Print the column data types after transformation:")
carbon_clean2.dtypes
```

```
C:\Users\Deepak Singh\AppData\Local\Temp\ipykernel_4608\803181690.py:2: FutureWarning: DataF
rame.applymap has been deprecated. Use DataFrame.map instead.
  carbon_clean2 = carbon_clean.applymap(lambda x: pd.to_numeric(x, errors='ignore'))
C:\Users\Deepak Singh\AppData\Local\Temp\ipykernel_4608\803181690.py:2: FutureWarning: error
s='ignore' is deprecated and will raise in a future version. Use to_numeric without passing
`errors` and catch exceptions explicitly instead
  carbon_clean2 = carbon_clean.applymap(lambda x: pd.to_numeric(x, errors='ignore'))
```

```
Print the column data types after transformation:
```

Out[33]:
```
Country code     object
Series name      object
1990            float64
1991            float64
1992            float64
1993            float64
1994            float64
1995            float64
1996            float64
1997            float64
1998            float64
1999            float64
2000            float64
2001            float64
2002            float64
2003            float64
2004            float64
2005            float64
2006            float64
2007            float64
2008            float64
2009            float64
2010            float64
2011            float64
dtype: object
```

In [34]:
```
#Rename the features in column "Series name"
# define shorter names corresponding to most relevant variables in a dictionary
chosen_vars = {'Cereal yield (kg per hectare)': 'cereal_yield',
               'Foreign direct investment, net inflows (% of GDP)': 'fdi_perc_gdp',
               'Access to electricity (% of total population)': 'elec_access_perc',
               'Energy use per units of GDP (kg oil eq./$1,000 of 2005 PPP $)': 'en_per_gdp
               'Energy use per capita (kilograms of oil equivalent)': 'en_per_cap',
               'CO2 emissions, total (KtCO2)': 'co2_ttl',
               'CO2 emissions per capita (metric tons)': 'co2_per_cap',
               'CO2 emissions per units of GDP (kg/$1,000 of 2005 PPP $)': 'co2_per_gdp',
               'Other GHG emissions, total (KtCO2e)': 'other_ghg_ttl',
               'Methane (CH4) emissions, total (KtCO2e)': 'ch4_ttl',
               'Nitrous oxide (N2O) emissions, total (KtCO2e)': 'n2o_ttl',
               'Droughts, floods, extreme temps (% pop. avg. 1990-2009)': 'nat_emerg',
               'Population in urban agglomerations >1million (%)': 'pop_urb_aggl_perc',
               'Nationally terrestrial protected areas (% of total land area)': 'prot_area_
               'GDP ($)': 'gdp',
               'GNI per capita (Atlas $)': 'gni_per_cap',
               'Under-five mortality rate (per 1,000)': 'under_5_mort_rate',
               'Population growth (annual %)': 'pop_growth_perc',
               'Population': 'pop',
               'Urban population growth (annual %)': 'urb_pop_growth_perc',
               'Urban population': 'urb_pop'
                }

# rename all variables in the column "Series name" with comprehensible shorter versions
carbon_clean2['Series name'] = carbon_clean2['Series name'].replace(to_replace=chosen_vars)
```

Data frame transformation

In [35]:
```
carbon_clean2.head()
```

Out[35]:

	Country code	Series name	1990	1991	1992	1993	1994	1995	1996	1997	...	2002	2003	2004
0	ABW	Land area below 5m (% of land area)	29.574810	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
1	ADO	Land area below 5m (% of land area)	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2	AFG	Land area below 5m (% of land area)	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
3	AGO	Land area below 5m (% of land area)	0.208235	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
4	ALB	Land area below 5m (% of land area)	4.967875	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN

5 rows × 24 columns

In [37]:

```
# save the short feature names into a list of strings
chosen_cols = list(chosen_vars.values())

# define an empty list, where sub-dataframes for each feature will be saved
frame_list = []

# iterate over all chosen features
for variable in chosen_cols:

    # pick only rows corresponding to the current feature
    frame = carbon_clean2[carbon_clean2['Series name'] == variable]

    # melt all the values for all years into one column and rename the columns correspondin
    frame = frame.melt(id_vars=['Country code', 'Series name']).rename(columns={'Country co

    # add the melted dataframe for the current feature into the list
    frame_list.append(frame)


# merge all sub-frames into a single dataframe, making an outer binding on the key columns
```

```python
from functools import reduce
all_vars = reduce(lambda left, right: pd.merge(left, right, on=['country','year'], how='out
```

In [38]:
```python
all_vars.head()
```

Out[38]:

	country	year	cereal_yield	fdi_perc_gdp	elec_access_perc	en_per_gdp	en_per_cap	co2_ttl	co
0	ABW	1990	NaN	NaN	NaN	NaN	NaN	1840.834	
1	ABW	1991	NaN	21.185138	NaN	NaN	NaN	1928.842	
2	ABW	1992	NaN	-3.857809	NaN	NaN	NaN	1723.490	
3	ABW	1993	NaN	-1.655492	NaN	NaN	NaN	1771.161	
4	ABW	1994	NaN	-5.874439	NaN	NaN	NaN	1763.827	

5 rows × 23 columns

Removing missing values

In [39]:
```python
print("check the amount of missing values in each column")
all_vars.isnull().sum()
```

```
check the amount of missing values in each column
```

Out[39]:
```
country                 0
year                    0
cereal_yield         1377
fdi_perc_gdp         1111
elec_access_perc     5027
en_per_gdp           2082
en_per_cap           1956
co2_ttl              1143
co2_per_cap          1146
co2_per_gdp          1557
other_ghg_ttl        4542
ch4_ttl              4526
n2o_ttl              4526
nat_emerg            4958
pop_urb_aggl_perc    2582
prot_area_perc        726
gdp                   779
gni_per_cap          1013
under_5_mort_rate     716
pop_growth_perc       278
pop                   252
urb_pop_growth_perc   490
urb_pop               467
dtype: int64
```

In [40]:
```python
#Filtering the years by missing values
all_vars_clean = all_vars

#define an array with the unique year values
years_count_missing = dict.fromkeys(all_vars_clean['year'].unique(), 0)
for ind, row in all_vars_clean.iterrows():
    years_count_missing[row['year']] += row.isnull().sum()

# sort the years by missing values
years_missing_sorted = dict(sorted(years_count_missing.items(), key=lambda item: item[1]))

# print the missing values for each year
print("missing values by year:")
```

```
for key, val in years_missing_sorted.items():
    print(key, ":", val)
```

```
missing values by year:
2005 : 1189
2000 : 1273
1995 : 1317
1990 : 1427
2007 : 1631
2006 : 1633
2004 : 1646
2008 : 1708
2003 : 1714
2002 : 1715
2001 : 1718
1999 : 1729
1998 : 1739
1997 : 1746
1996 : 1756
1994 : 1781
1993 : 1792
1992 : 1810
1991 : 1921
2009 : 2078
2010 : 3038
2011 : 4893
```

Filtering by Year

In [41]:

```
print("number of missing values in the whole dataset before filtering the years:")
print(all_vars_clean.isnull().sum().sum())
print("number of rows before filtering the years:")
print(all_vars_clean.shape[0])

# filter only rows for years between 1991 and 2008 (having less missing values)
all_vars_clean = all_vars_clean[(all_vars_clean['year'] >= 1991) & (all_vars_clean['year']

print("number of missing values in the whole dataset after filtering the years:")
print(all_vars_clean.isnull().sum().sum())
print("number of rows after filtering the years:")
print(all_vars_clean.shape[0])
```

```
number of missing values in the whole dataset before filtering the years:
41254
number of rows before filtering the years:
5126
number of missing values in the whole dataset after filtering the years:
29818
number of rows after filtering the years:
4194
```

In [42]:

```
#Filtering the countries by missing values
# check the amount of missing values by country

# define an array with the unique country values
countries_count_missing = dict.fromkeys(all_vars_clean['country'].unique(), 0)

# iterate through all rows and count the amount of NaN values for each country
for ind, row in all_vars_clean.iterrows():
    countries_count_missing[row['country']] += row.isnull().sum()

# sort the countries by missing values
countries_missing_sorted = dict(sorted(countries_count_missing.items(), key=lambda item: it

# print the missing values for each country
print("missing values by country:")
```

```
for key, val in countries_missing_sorted.items():
    print(key, ":", val)
```

```
missing values by country:
AGO : 81
ARG : 81
AUS : 81
AUT : 81
BGD : 81
BGR : 81
BOL : 81
BRA : 81
CAN : 81
CHE : 81
CHL : 81
CHN : 81
CIV : 81
CMR : 81
COG : 81
COL : 81
CRI : 81
DEU : 81
DNK : 81
DOM : 81
ECU : 81
EGY : 81
EMU : 81
ESP : 81
FIN : 81
FRA : 81
GBR : 81
GHA : 81
GTM : 81
HND : 81
HUN : 81
IDN : 81
IND : 81
IRL : 81
ISR : 81
ITA : 81
JOR : 81
JPN : 81
KEN : 81
KOR : 81
LAC : 81
LMC : 81
LMY : 81
MAR : 81
MEX : 81
MIC : 81
MNA : 81
MOZ : 81
MYS : 81
NGA : 81
NLD : 81
NZL : 81
PAK : 81
PAN : 81
PER : 81
PHL : 81
PRT : 81
PRY : 81
ROM : 81
SAS : 81
SAU : 81
SDN : 81
SEN : 81
SLV : 81
SWE : 81
SYR : 81
```

TGO : 81
THA : 81
TUR : 81
TZA : 81
UMC : 81
URY : 81
USA : 81
VEN : 81
VNM : 81
ZAF : 81
ZMB : 81
GRC : 82
POL : 82
YEM : 82
ZAR : 82
DZA : 84
ETH : 84
LIC : 84
SSA : 84
WLD : 84
ARE : 85
ECA : 85
RUS : 86
UKR : 86
ARM : 87
BLR : 87
UZB : 87
KAZ : 88
CZE : 89
IRN : 89
BEL : 90
AZE : 91
GEO : 92
LBN : 92
HTI : 94
NIC : 96
BEN : 99
BWA : 99
CYP : 99
GAB : 99
HIC : 99
JAM : 99
KHM : 99
LKA : 99
MLT : 99
MNG : 99
NOR : 99
OMN : 99
SGP : 99
TTO : 99
TUN : 99
ALB : 100
EAP : 102
NPL : 103
EST : 104
LVA : 104
NAM : 104
HRV : 105
MDA : 105
SVN : 105
TJK : 105
KGZ : 106
LTU : 106
MKD : 107
SVK : 107
TKM : 107
LBY : 108

LUX : 108
BRN : 109
KWT : 113
BHR : 117
CUB : 117
ISL : 117
ZWE : 117
ERI : 121
IRQ : 122
BIH : 123
HKG : 124
BFA : 126
GIN : 126
MDG : 126
MLI : 126
NER : 126
UGA : 126
QAT : 135
ATG : 136
BHS : 136
BLZ : 136
BRB : 136
COM : 136
CPV : 136
DMA : 136
FJI : 136
GNB : 136
GRD : 136
GUY : 136
MUS : 136
SWZ : 136
VCT : 136
VUT : 136
DJI : 137
SLB : 137
SUR : 137
GMB : 141
BDI : 144
CAF : 144
LAO : 144
MRT : 144
MWI : 144
PNG : 144
RWA : 144
SLE : 144
TCD : 144
BTN : 148
LBR : 149
SID : 152
GNQ : 154
KNA : 154
LCA : 154
SYC : 154
TON : 154
WSM : 154
KIR : 156
SRB : 158
MDV : 164
PLW : 166
AFG : 170
MMR : 171
PRK : 171
FSM : 184
MHL : 184
LSO : 190
MAC : 198
STP : 198

```
TMP : 202
NCL : 204
ADO : 206
SOM : 206
WBG : 207
GRL : 216
ABW : 226
BMU : 226
PRI : 230
MNE : 231
PYF : 232
LIE : 234
MCO : 234
CYM : 250
FRO : 259
GIB : 261
COK : 270
GUM : 270
NIU : 270
SMR : 270
TUV : 272
IMY : 282
VIR : 285
ASM : 288
CHI : 288
MNP : 288
NRU : 288
TCA : 296
MYT : 324
KSV : 325
MAF : 342
CUW : 357
SXM : 357
```

In [43]:
```python
print("number of missing values in the whole dataset before filtering the countries:")
print(all_vars_clean.isnull().sum().sum())
print("number of rows before filtering the countries:")
print(all_vars_clean.shape[0])


# filter only rows for countries with less than 90 missing values
countries_filter = []
for key, val in countries_missing_sorted.items():
    if val<90:
        countries_filter.append(key)

all_vars_clean = all_vars_clean[all_vars_clean['country'].isin(countries_filter)]

print("number of missing values in the whole dataset after filtering the countries:")
print(all_vars_clean.isnull().sum().sum())
print("number of rows after filtering the countries:")
print(all_vars_clean.shape[0])
```

```
number of missing values in the whole dataset before filtering the countries:
29818
number of rows before filtering the countries:
4194
number of missing values in the whole dataset after filtering the countries:
7854
number of rows after filtering the countries:
1728
```

In [44]:
```python
all_vars_clean.isnull().sum()
```

Out[44]:
```
country                 0
year                    0
cereal_yield           10
fdi_perc_gdp           17
elec_access_perc     1728
en_per_gdp              0
en_per_cap              0
co2_ttl                 9
co2_per_cap             9
co2_per_gdp             9
other_ghg_ttl        1446
ch4_ttl              1440
n2o_ttl              1440
nat_emerg            1728
pop_urb_aggl_perc       0
prot_area_perc          0
gdp                     2
gni_per_cap            16
under_5_mort_rate       0
pop_growth_perc         0
pop                     0
urb_pop_growth_perc     0
urb_pop                 0
dtype: int64
```

In [45]:
```python
#Dropping High-NaN Features
# remove features with more than 20 missing values

from itertools import compress

# create a boolean mapping of features with more than 20 missing values
vars_bad = all_vars_clean.isnull().sum()>20

# remove the columns corresponding to the mapping of the features with many missing values
all_vars_clean2 = all_vars_clean.drop(compress(data = all_vars_clean.columns, selectors = v

print("Remaining missing values per column:")
print(all_vars_clean2.isnull().sum())
```

```
Remaining missing values per column:
country                 0
year                    0
cereal_yield           10
fdi_perc_gdp           17
en_per_gdp              0
en_per_cap              0
co2_ttl                 9
co2_per_cap             9
co2_per_gdp             9
pop_urb_aggl_perc       0
prot_area_perc          0
gdp                     2
gni_per_cap            16
under_5_mort_rate       0
pop_growth_perc         0
pop                     0
urb_pop_growth_perc     0
urb_pop                 0
dtype: int64
```

In [46]:
```python
# delete rows with any number of missing values
all_vars_clean3 = all_vars_clean2.dropna(axis='rows', how='any')

print("Remaining missing values per column:")
print(all_vars_clean3.isnull().sum())
```

```python
print("Final shape of the cleaned dataset:")
print(all_vars_clean3.shape)
```

```
Remaining missing values per column:
country                 0
year                    0
cereal_yield            0
fdi_perc_gdp            0
en_per_gdp              0
en_per_cap              0
co2_ttl                 0
co2_per_cap             0
co2_per_gdp             0
pop_urb_aggl_perc       0
prot_area_perc          0
gdp                     0
gni_per_cap             0
under_5_mort_rate       0
pop_growth_perc         0
pop                     0
urb_pop_growth_perc     0
urb_pop                 0
dtype: int64
Final shape of the cleaned dataset:
(1700, 18)
```

In [47]:

```python
# export the clean dataframe to a csv file
all_vars_clean3.to_csv('carbon_cleaned.csv', index=False)
```