Step 1: Import Required Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import joblib
```

Step 2: Load Dataset

In [2]:
```
excel_file = 'SupplyChainEmissionFactorsforUSIndustriesCommodities.xlsx'
years = range(2010, 2017)
```

In [3]:
```
years[2]
```

Out[3]:
```
2012
```

In [4]:
```
df_1 = pd.read_excel(excel_file, sheet_name=f'{years[0]}_Detail_Commodity')
df_1.head()
```

Out[4]:

	Commodity Code	Commodity Name	Substance	Unit	Supply Chain Emission Factors without Margins	Margins of Supply Chain Emission Factors	Supply Chain Emission Factors with Margins	Unnamed: 7	DQ ReliabilityScore of Factors without Margins	Tempor of Fac
0	1111A0	Fresh soybeans, canola, flaxseeds, and other o...	carbon dioxide	kg/2018 USD, purchaser price	0.398	0.073	0.470	NaN	4	
1	1111A0	Fresh soybeans, canola, flaxseeds, and other o...	methane	kg/2018 USD, purchaser price	0.001	0.001	0.002	NaN	4	
2	1111A0	Fresh soybeans, canola, flaxseeds, and other o...	nitrous oxide	kg/2018 USD, purchaser price	0.002	0.000	0.002	NaN	4	
3	1111A0	Fresh soybeans, canola, flaxseeds, and other o...	other GHGs	kg CO2e/2018 USD, purchaser price	0.002	0.000	0.002	NaN	3	
4	1111B0	Fresh wheat, corn, rice, and other grains	carbon dioxide	kg/2018 USD, purchaser price	0.659	0.081	0.740	NaN	4	

In [5]:
```python
df_2 = pd.read_excel(excel_file, sheet_name=f'{years[0]}_Detail_Industry')
df_2.head()
```

Out[5]:

	Industry Code	Industry Name	Substance	Unit	Supply Chain Emission Factors without Margins	Margins of Supply Chain Emission Factors	Supply Chain Emission Factors with Margins	Unnamed: 7	DQ ReliabilityScore of Factors without Margins	TemporalCorre of Factors w M
0	1111A0	Oilseed farming	carbon dioxide	kg/2018 USD, purchaser price	0.414	0.073	0.487	NaN	4	
1	1111A0	Oilseed farming	methane	kg/2018 USD, purchaser price	0.001	0.001	0.002	NaN	4	
2	1111A0	Oilseed farming	nitrous oxide	kg/2018 USD, purchaser price	0.002	0.000	0.002	NaN	4	
3	1111A0	Oilseed farming	other GHGs	kg CO2e/2018 USD, purchaser price	0.002	0.000	0.002	NaN	3	
4	1111B0	Grain farming	carbon dioxide	kg/2018 USD, purchaser price	0.680	0.082	0.762	NaN	4	

In [6]:
```python
all_data = []
for year in years:
    try:
        df_com = pd.read_excel(excel_file, sheet_name=f'{year}_Detail_Commodity')
        df_ind = pd.read_excel(excel_file, sheet_name=f'{year}_Detail_Industry')

        df_com['Source'] = 'Commodity'
        df_ind['Source'] = 'Industry'
        df_com['Year'] = df_ind['Year'] = year

        df_com.columns = df_com.columns.str.strip()
        df_ind.columns = df_ind.columns.str.strip()

        df_com.rename(columns={
            'Commodity Code': 'Code',
            'Commodity Name': 'Name'
        }, inplace=True)

        df_ind.rename(columns={
            'Industry Code': 'Code',
            'Industry Name': 'Name'
        }, inplace=True)

        all_data.append(pd.concat([df_com, df_ind], ignore_index=True))

    except Exception as e:
        print(f"Error processing year {year}: {e}")
```

In [7]:
```
all_data[3]
```

Out[7]:

	Code	Name	Substance	Unit	Supply Chain Emission Factors without Margins	Margins of Supply Chain Emission Factors	Supply Chain Emission Factors with Margins	Unnamed: 7	DQ ReliabilityScore of Factors without Margins	Tempora of Fac
0	1111A0	Fresh soybeans, canola, flaxseeds, and other o...	carbon dioxide	kg/2018 USD, purchaser price	0.373	0.072	0.444	NaN	4	
1	1111A0	Fresh soybeans, canola, flaxseeds, and other o...	methane	kg/2018 USD, purchaser price	0.001	0.001	0.002	NaN	4	
2	1111A0	Fresh soybeans, canola, flaxseeds, and other o...	nitrous oxide	kg/2018 USD, purchaser price	0.002	0.000	0.002	NaN	4	
3	1111A0	Fresh soybeans, canola, flaxseeds, and other o...	other GHGs	kg CO2e/2018 USD, purchaser price	0.002	0.000	0.002	NaN	3	
4	1111B0	Fresh wheat, corn, rice, and other grains	carbon dioxide	kg/2018 USD, purchaser price	0.722	0.079	0.801	NaN	4	
...	...	...	...	...	...	...	...	...	...	
3151	813B00	Civic, social, professional, and similar organ...	other GHGs	kg CO2e/2018 USD, purchaser price	0.008	0.000	0.008	NaN	4	
3152	814000	Private households	carbon dioxide	kg/2018 USD, purchaser price	0.000	0.000	0.000	NaN	4	
3153	814000	Private households	methane	kg/2018 USD, purchaser price	0.000	0.000	0.000	NaN	4	
3154	814000	Private households	nitrous oxide	kg/2018 USD, purchaser price	0.000	0.000	0.000	NaN	4	
3155	814000	Private households	other GHGs	kg CO2e/2018 USD, purchaser price	0.000	0.000	0.000	NaN	4	

3156 rows × 15 columns

In [8]:
```
len(all_data)
```

Out[8]: 7

In [9]:
```python
df = pd.concat(all_data, ignore_index=True)
df.head(10)
```

Out[9]:

	Code	Name	Substance	Unit	Supply Chain Emission Factors without Margins	Margins of Supply Chain Emission Factors	Supply Chain Emission Factors with Margins	Unnamed: 7	DQ ReliabilityScore of Factors without Margins	TemporalCor of Factors M
0	1111A0	Fresh soybeans, canola, flaxseeds, and other o...	carbon dioxide	kg/2018 USD, purchaser price	0.398	0.073	0.470	NaN	4	
1	1111A0	Fresh soybeans, canola, flaxseeds, and other o...	methane	kg/2018 USD, purchaser price	0.001	0.001	0.002	NaN	4	
2	1111A0	Fresh soybeans, canola, flaxseeds, and other o...	nitrous oxide	kg/2018 USD, purchaser price	0.002	0.000	0.002	NaN	4	
3	1111A0	Fresh soybeans, canola, flaxseeds, and other o...	other GHGs	kg CO2e/2018 USD, purchaser price	0.002	0.000	0.002	NaN	3	
4	1111B0	Fresh wheat, corn, rice, and other grains	carbon dioxide	kg/2018 USD, purchaser price	0.659	0.081	0.740	NaN	4	
5	1111B0	Fresh wheat, corn, rice, and other grains	methane	kg/2018 USD, purchaser price	0.008	0.001	0.009	NaN	2	
6	1111B0	Fresh wheat, corn, rice, and other grains	nitrous oxide	kg/2018 USD, purchaser price	0.004	0.000	0.004	NaN	4	
7	1111B0	Fresh wheat, corn, rice, and other grains	other GHGs	kg CO2e/2018 USD, purchaser price	0.004	0.000	0.004	NaN	3	
8	111200	Fresh vegetables, melons, and potatoes	carbon dioxide	kg/2018 USD, purchaser price	0.183	0.132	0.315	NaN	3	
9	111200	Fresh vegetables, melons, and potatoes	methane	kg/2018 USD, purchaser price	0.001	0.001	0.002	NaN	4	

In [10]:
```python
len(df)
```

Out[10]: 22092

Step 3: Data Preprocessing

In [11]:
```
df.columns
```

Out[11]:
```
Index(['Code', 'Name', 'Substance', 'Unit',
       'Supply Chain Emission Factors without Margins',
       'Margins of Supply Chain Emission Factors',
       'Supply Chain Emission Factors with Margins', 'Unnamed: 7',
       'DQ ReliabilityScore of Factors without Margins',
       'DQ TemporalCorrelation of Factors without Margins',
       'DQ GeographicalCorrelation of Factors without Margins',
       'DQ TechnologicalCorrelation of Factors without Margins',
       'DQ DataCollection of Factors without Margins', 'Source', 'Year'],
      dtype='object')
```

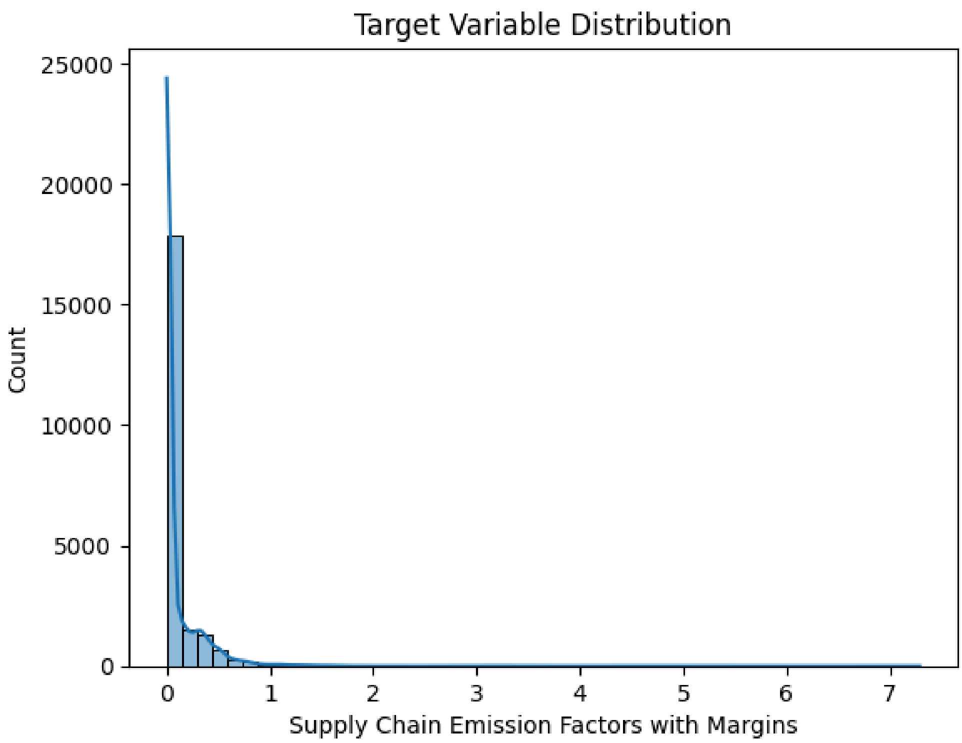
In [12]:
```
df.isnull().sum()
```

Out[12]:
```
Code                                                          0
Name                                                          0
Substance                                                     0
Unit                                                          0
Supply Chain Emission Factors without Margins                 0
Margins of Supply Chain Emission Factors                      0
Supply Chain Emission Factors with Margins                    0
Unnamed: 7                                                22092
DQ ReliabilityScore of Factors without Margins                0
DQ TemporalCorrelation of Factors without Margins             0
DQ GeographicalCorrelation of Factors without Margins         0
DQ TechnologicalCorrelation of Factors without Margins        0
DQ DataCollection of Factors without Margins                  0
Source                                                        0
Year                                                          0
dtype: int64
```

FOR WEEK2

In [13]:
```python
sns.histplot(df['Supply Chain Emission Factors with Margins'], bins=50, kde=True)
plt.title('Target Variable Distribution')
plt.show()
```

```
C:\Users\srine\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_
na option is deprecated and will be removed in a future version. Convert inf values to NaN befo
re operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [14]:
```python
print(df['Substance'].value_counts())
```

```
Substance
carbon dioxide    5523
methane           5523
nitrous oxide     5523
other GHGs        5523
Name: count, dtype: int64
```

In [15]:
```python
print(df['Unit'].value_counts())
```

```
Unit
kg/2018 USD, purchaser price         16569
kg CO2e/2018 USD, purchaser price     5523
Name: count, dtype: int64
```

In [16]:
```python
print(df['Unit'].unique())
```

```
['kg/2018 USD, purchaser price' 'kg CO2e/2018 USD, purchaser price']
```

In [17]:
```python
print(df['Source'].value_counts())
```

```
Source
Industry     11060
Commodity    11032
Name: count, dtype: int64
```

In [18]:
```python
df['Substance'].unique()
```

Out[18]:
```
array(['carbon dioxide', 'methane', 'nitrous oxide', 'other GHGs'],
      dtype=object)
```

In [19]:
```
substance_map={'carbon dioxide':0, 'methane':1, 'nitrous oxide':2, 'other GHGs':3}
```

In [20]:
```
df['Substance']=df['Substance'].map(substance_map)
```

In [21]:
```
df['Substance'].unique()
```

Out[21]:
```
array([0, 1, 2, 3])
```

In [22]:
```
print(df['Unit'].unique())
```

```
['kg/2018 USD, purchaser price' 'kg CO2e/2018 USD, purchaser price']
```

In [23]:
```
unit_map={'kg/2018 USD, purchaser price':0, 'kg CO2e/2018 USD, purchaser price':1}
```

In [24]:
```
df['Unit']=df['Unit'].map(unit_map)
```

In [25]:
```
print(df['Unit'].unique())
```

```
[0 1]
```

In [27]:
```
print(df['Source'].unique())
```

```
['Commodity' 'Industry']
```

In [28]:
```
source_map={'Commodity':0, 'Industry':1}
```

In [29]:
```
df['Source']=df['Source'].map(source_map)
```

In [30]:
```
print(df['Source'].unique())
```

```
[0 1]
```

In [31]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22092 entries, 0 to 22091
Data columns (total 15 columns):
 #   Column                                          Non-Null Count  Dtype  
---  ------                                          --------------  -----  
 0   Code                                            22092 non-null  object 
 1   Name                                            22092 non-null  object 
 2   Substance                                       22092 non-null  int64  
 3   Unit                                            22092 non-null  int64  
 4   Supply Chain Emission Factors without Margins   22092 non-null  float64
 5   Margins of Supply Chain Emission Factors        22092 non-null  float64
 6   Supply Chain Emission Factors with Margins      22092 non-null  float64
 7   Unnamed: 7                                      0 non-null      float64
 8   DQ ReliabilityScore of Factors without Margins  22092 non-null  int64  
 9   DQ TemporalCorrelation of Factors without Margins  22092 non-null  int64  
 10  DQ GeographicalCorrelation of Factors without Margins  22092 non-null  int64  
 11  DQ TechnologicalCorrelation of Factors without Margins  22092 non-null  int64  
 12  DQ DataCollection of Factors without Margins    22092 non-null  int64  
 13  Source                                          22092 non-null  int64  
 14  Year                                            22092 non-null  int64  
dtypes: float64(4), int64(9), object(2)
memory usage: 2.5+ MB
```

In [32]:
```
df.Code.unique()
```

Out[32]:
```
array(['1111A0', '1111B0', '111200', '111300', '111400', '111900',
       '112120', '1121A0', '112300', '112A00', '113000', '114000',
       '115000', '211000', '212100', '212230', '2122A0', '212310',
       '2123A0', '213111', '21311A', '221100', '221200', '221300',
       '230301', '230302', '233210', '233230', '233240', '233262',
       '2332A0', '2332C0', '2332D0', '233411', '233412', '2334A0',
       '311111', '311119', '311210', '311221', '311224', '311225',
       '311230', '311300', '311410', '311420', '311513', '311514',
       '31151A', '311520', '311615', '31161A', '311700', '311810',
       '3118A0', '311910', '311920', '311930', '311940', '311990',
       '312110', '312120', '312130', '312140', '312200', '313100',
       '313200', '313300', '314110', '314120', '314900', '315000',
       '316000', '321100', '321200', '321910', '3219A0', '322110',
       '322120', '322130', '322210', '322220', '322230', '322291',
       '322299', '323110', '323120', '324110', '324121', '324122',
       '324190', '325110', '325120', '325130', '325180', '325190',
       '325211', '3252A0', '325310', '325320', '325411', '325412',
       '325413', '325414', '325510', '325520', '325610', '325620',
       '325910', '3259A0', '326110', '326120', '326130', '326140',
       '326150', '326160', '326190', '326210', '326220', '326290',
       '327100', '327200', '327310', '327320', '327330', '327390',
       '327400', '327910', '327991', '327992', '327993', '327999',
       '331110', '331200', '331313', '33131B', '331410', '331420',
       '331490', '331510', '331520', '332114', '332119', '33211A',
       '332200', '332310', '332320', '332410', '332420', '332430',
       '332500', '332600', '332710', '332720', '332800', '332913',
       '33291A', '332991', '332996', '332999', '33299A', '333111',
       '333112', '333120', '333130', '333242', '33329A', '333314',
       '333316', '333318', '333413', '333414', '333415', '333511',
       '333514', '333517', '33351B', '333611', '333612', '333613',
       '333618', '333912', '33391A', '333920', '333991', '333993',
       '333994', '33399A', '33399B', '334111', '334112', '334118',
       '334210', '334220', '334290', '334300', '334413', '334418',
       '33441A', '334510', '334511', '334512', '334513', '334514',
       '334515', '334516', '334517', '33451A', '334610', '335110',
       '335120', '335210', '335221', '335222', '335224', '335228',
       '335311', '335312', '335313', '335314', '335911', '335912',
       '335920', '335930', '335991', '335999', '336111', '336112',
       '336120', '336211', '336212', '336213', '336214', '336310',
       '336320', '336350', '336360', '336370', '336390', '3363A0',
       '336411', '336412', '336413', '336414', '33641A', '336500',
       '336611', '336612', '336991', '336992', '336999', '337110',
       '337121', '337122', '337127', '33712N', '337215', '33721A',
       '337900', '339112', '339113', '339114', '339115', '339116',
       '339910', '339920', '339930', '339940', '339950', '339990',
       '4200ID', '423100', '423400', '423600', '423800', '423A00',
       '424200', '424400', '424700', '424A00', '425000', '441000',
       '444000', '445000', '446000', '447000', '448000', '452000',
       '454000', '481000', '482000', '483000', '484000', '485000',
       '486000', '48A000', '491000', '492000', '493000', '4B0000',
       '511110', '511120', '511130', '5111A0', '511200', '512100',
       '512200', '515100', '515200', '517110', '517210', '517A00',
       '518200', '519130', '5191A0', '522A00', '523900', '523A00',
       '524113', '5241XX', '524200', '525000', '52A000', '531HSO',
       '531HST', '531ORE', '532100', '532400', '532A00', '533000',
       '541100', '541200', '541300', '541400', '541511', '541512',
       '54151A', '541610', '5416A0', '541700', '541800', '541920',
       '541940', '5419A0', '550000', '561100', '561200', '561300',
       '561400', '561500', '561600', '561700', '561900', '562000',
       '611100', '611A00', '611B00', '621100', '621200', '621300',
       '621400', '621500', '621600', '621900', '622000', '623A00',
       '623B00', '624100', '624400', '624A00', '711100', '711200',
       '711500', '711A00', '712000', '713100', '713200', '713900',
       '721000', '722110', '722211', '722A00', '811100', '811200',
       '811300', '811400', '812100', '812200', '812300', '812900',
       '813100', '813A00', '813B00', '814000', '331314'], dtype=object)
```

In [33]:
```
df.Name.unique()
```

Out[33]:
```
array(['Fresh soybeans, canola, flaxseeds, and other oilseeds',
       'Fresh wheat, corn, rice, and other grains',
       'Fresh vegetables, melons, and potatoes',
       'Fresh fruits and tree nuts',
       'Greenhouse crops, mushrooms, nurseries, and flowers',
       'Tobacco, cotton, sugarcane, peanuts, sugar beets, herbs and spices, and other crop
s',
       'Dairies', 'Cattle ranches and feedlots', 'Poultry farms',
       'Animal farms and aquaculture ponds (except cattle and poultry)',
       'Timber and raw forest products', 'Wild-caught fish and game',
       'Agriculture and forestry support', 'Unrefined oil and gas',
       'Coal', 'Copper, nickel, lead, and zinc',
       'Iron, gold, silver, and other metal ores', 'Dimensional stone',
       'Sand, gravel, clay, phosphate, other nonmetallic minerals',
       'Well drilling', 'Other support activities for mining',
       'Electricity', 'Natural gas',
       'Drinking water and wastewater treatment',
       'Nonresidential maintenance and repair',
       'Residential maintenance and repair', 'Health care structures',
```

In [34]:
```
len(df.Name.unique())
```

Out[34]: 713

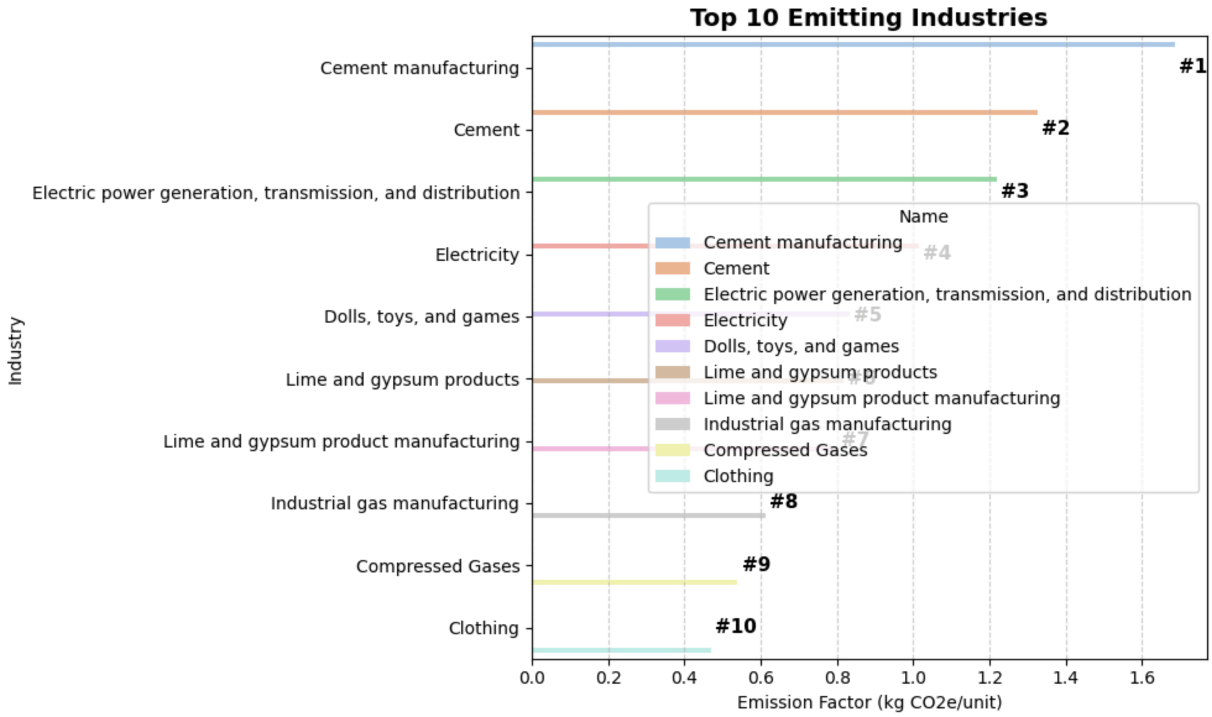
In [35]:
```
top_emitters = df[['Name', 'Supply Chain Emission Factors with Margins']].groupby('Name').mean()
    'Supply Chain Emission Factors with Margins', ascending=False).head(10)
top_emitters = top_emitters.reset_index()
```

In [36]:
```
top_emitters
```

Out[36]:

	Name	Supply Chain Emission Factors with Margins
0	Cement manufacturing	1.686179
1	Cement	1.324964
2	Electric power generation, transmission, and d...	1.220357
3	Electricity	1.016143
4	Dolls, toys, and games	0.832179
5	Lime and gypsum products	0.816536
6	Lime and gypsum product manufacturing	0.799679
7	Industrial gas manufacturing	0.612929
8	Compressed Gases	0.539679
9	Clothing	0.468714

In [37]:
```
plt.figure(figsize=(10,6))
sns.barplot(
    x='Supply Chain Emission Factors with Margins',
    y='Name',
    data=top_emitters,
    hue='Name',
    palette='pastel'
)
for i, (value, name) in enumerate(zip(top_emitters['Supply Chain Emission Factors with Margins']
    plt.text(value + 0.01, i - 1, f'#{i}', va='center', fontsize=11, fontweight='bold', color='b
plt.title('Top 10 Emitting Industries', fontsize=14, fontweight='bold')
plt.xlabel('Emission Factor (kg CO2e/unit)')
plt.ylabel('Industry')
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



In [38]:
```
df.drop(columns=['Name','Code','Year'], inplace=True)
```

In [39]:
```
df.head(1)
```

Out[39]:

	Substance	Unit	Supply Chain Emission Factors without Margins	Margins of Supply Chain Emission Factors	Supply Chain Emission Factors with Margins	Unnamed: 7	DQ ReliabilityScore of Factors without Margins	DQ TemporalCorrelation of Factors without Margins	GeographicalCorre of Factors wi Ma
0	0	0	0.398	0.073	0.47	NaN	4	3	

In [40]:
```
df.shape
```

Out[40]: (22092, 12)

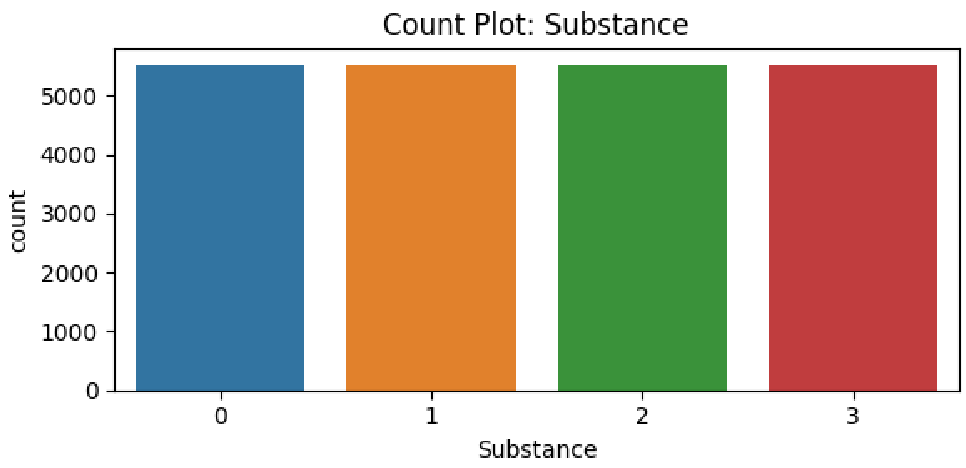
In [41]:
```
X = df.drop(columns=['Supply Chain Emission Factors with Margins'])
y = df['Supply Chain Emission Factors with Margins']
```

In [42]:
```
X.head()
```

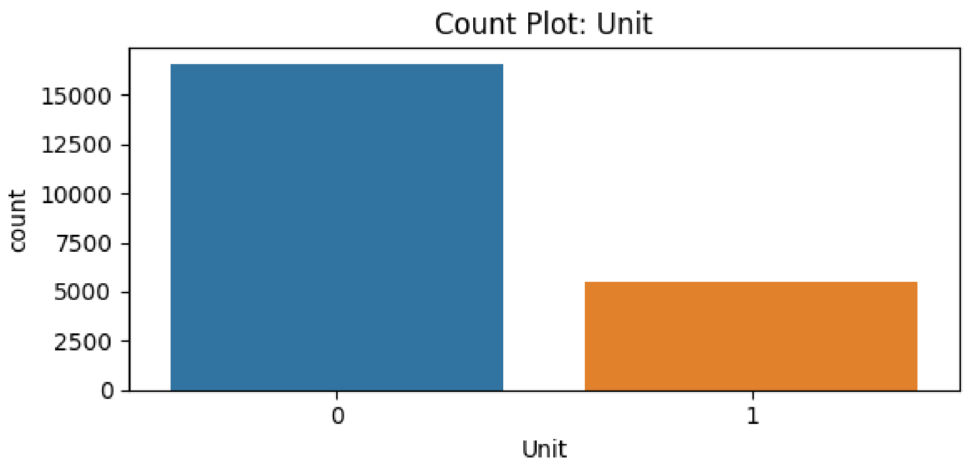
Out[42]:

	Substance	Unit	Supply Chain Emission Factors without Margins	Margins of Supply Chain Emission Factors	Unnamed: 7	DQ ReliabilityScore of Factors without Margins	DQ TemporalCorrelation of Factors without Margins	DQ GeographicalCorrelation of Factors without Margins	Tecl
0	0	0	0.398	0.073	NaN	4	3	1	
1	1	0	0.001	0.001	NaN	4	3	1	
2	2	0	0.002	0.000	NaN	4	3	1	
3	3	1	0.002	0.000	NaN	3	3	1	
4	0	0	0.659	0.081	NaN	4	3	1	

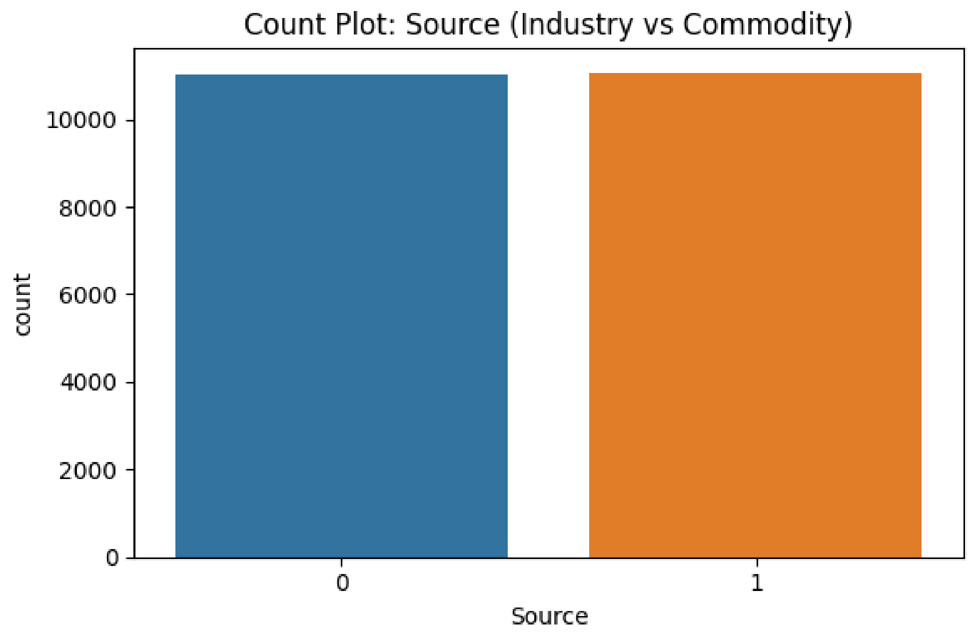
In [43]:
```
plt.figure(figsize=(6, 3))
sns.countplot(x=df["Substance"])
plt.title("Count Plot: Substance")
plt.xticks()
plt.tight_layout()
plt.show()
```



In [44]:
```
plt.figure(figsize=(6, 3))
sns.countplot(x=df["Unit"])
plt.title("Count Plot: Unit")
plt.tight_layout()
plt.show()
```



In [45]:
```
plt.figure(figsize=(6, 4))
sns.countplot(x=df["Source"])
plt.title("Count Plot: Source (Industry vs Commodity)")
plt.tight_layout()
plt.show()
```



In [46]:
```
df.columns
```

Out[46]:
```
Index(['Substance', 'Unit', 'Supply Chain Emission Factors without Margins',
       'Margins of Supply Chain Emission Factors',
       'Supply Chain Emission Factors with Margins', 'Unnamed: 7',
       'DQ ReliabilityScore of Factors without Margins',
       'DQ TemporalCorrelation of Factors without Margins',
       'DQ GeographicalCorrelation of Factors without Margins',
       'DQ TechnologicalCorrelation of Factors without Margins',
       'DQ DataCollection of Factors without Margins', 'Source'],
      dtype='object')
```

In [47]:
```
df.select_dtypes(include=np.number).corr()
```

Out[47]:

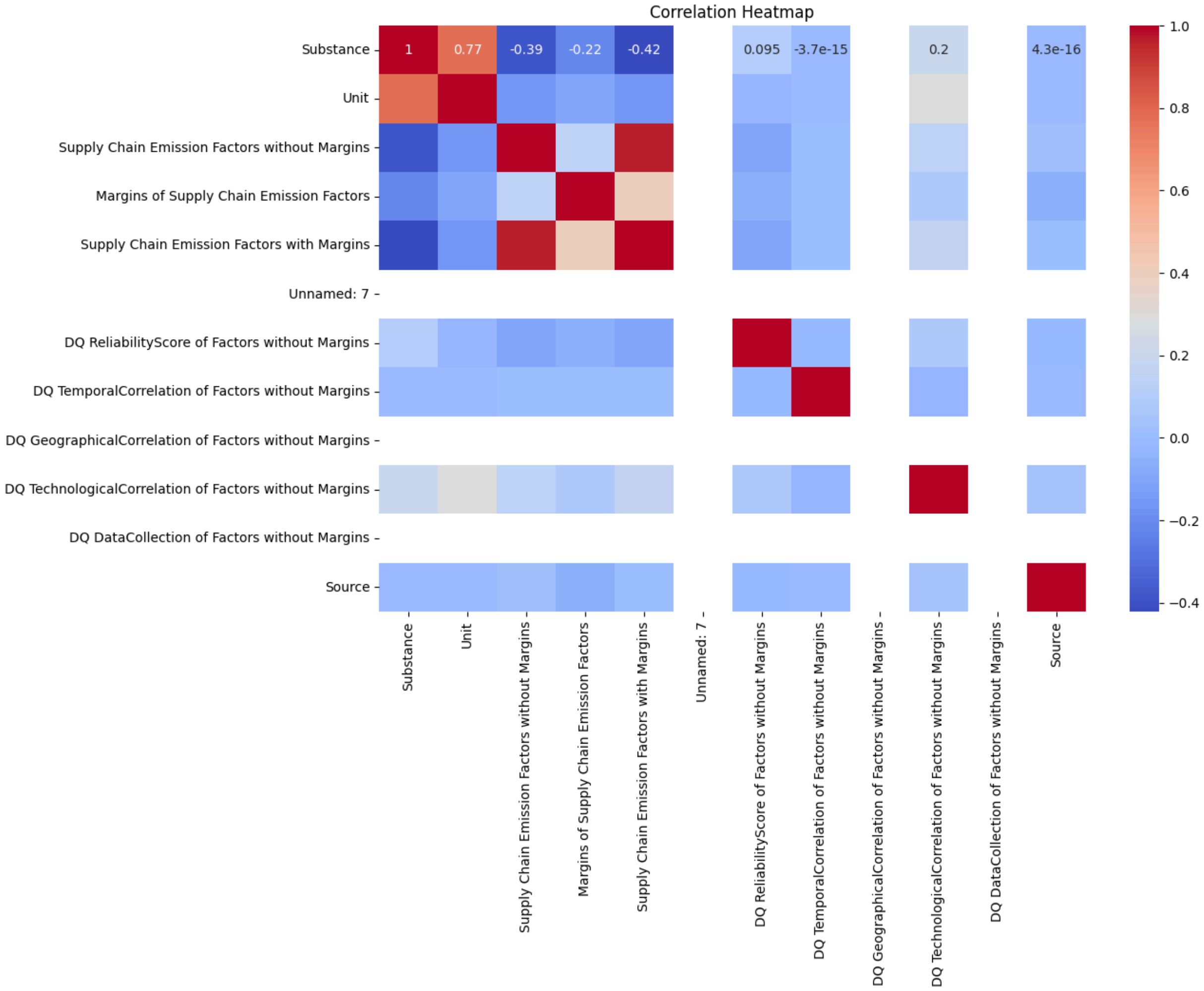
	Substance	Unit	Supply Chain Emission Factors without Margins	Margins of Supply Chain Emission Factors	Supply Chain Emission Factors with Margins	Unnamed: 7	DQ ReliabilityScore of Factors without Margins	Temp of
Substance	1.000000e+00	7.745967e-01	-0.391851	-0.218400	-0.421603	NaN	0.095092	
Unit	7.745967e-01	1.000000e+00	-0.155859	-0.094300	-0.169741	NaN	-0.025159	
Supply Chain Emission Factors without Margins	-3.918505e-01	-1.558594e-01	1.000000	0.143005	0.962971	NaN	-0.098000	
Margins of Supply Chain Emission Factors	-2.184002e-01	-9.429989e-02	0.143005	1.000000	0.404541	NaN	-0.069598	
Supply Chain Emission Factors with Margins	-4.216032e-01	-1.697410e-01	0.962971	0.404541	1.000000	NaN	-0.109494	
Unnamed: 7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
DQ ReliabilityScore of Factors without Margins	9.509190e-02	-2.515938e-02	-0.098000	-0.069598	-0.109494	NaN	1.000000	
DQ TemporalCorrelation of Factors without Margins	-3.667637e-15	-3.173071e-17	0.009284	0.007953	0.010748	NaN	-0.021707	
DQ GeographicalCorrelation of Factors without Margins	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
DQ TechnologicalCorrelation of Factors without Margins	1.984154e-01	2.869901e-01	0.148410	0.086335	0.160574	NaN	0.073583	
DQ DataCollection of Factors without Margins	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
Source	4.273306e-16	-1.545892e-17	0.027131	-0.067504	0.006688	NaN	-0.012287	

In [48]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22092 entries, 0 to 22091
Data columns (total 12 columns):
 #   Column                                                Non-Null Count  Dtype  
---  ------                                                --------------  -----  
 0   Substance                                             22092 non-null  int64  
 1   Unit                                                  22092 non-null  int64  
 2   Supply Chain Emission Factors without Margins         22092 non-null  float64
 3   Margins of Supply Chain Emission Factors              22092 non-null  float64
 4   Supply Chain Emission Factors with Margins            22092 non-null  float64
 5   Unnamed: 7                                            0 non-null      float64
 6   DQ ReliabilityScore of Factors without Margins        22092 non-null  int64  
 7   DQ TemporalCorrelation of Factors without Margins     22092 non-null  int64  
 8   DQ GeographicalCorrelation of Factors without Margins 22092 non-null  int64  
 9   DQ TechnologicalCorrelation of Factors without Margins 22092 non-null  int64  
 10  DQ DataCollection of Factors without Margins          22092 non-null  int64  
 11  Source                                                22092 non-null  int64  
dtypes: float64(4), int64(8)
memory usage: 2.0 MB
```

In [49]:
```python
plt.figure(figsize=(12, 8))
sns.heatmap(df.select_dtypes(include=np.number).corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```

```
C:\Users\srine\anaconda3\Lib\site-packages\seaborn\matrix.py:260: FutureWarning: Format strings passed to MaskedConstant are ignored, but in future may error or produce different behavior
  annotation = ("{:" + self.fmt + "}").format(val)
```



In [50]:
```python
X.describe().T
```

Out[50]:

	count	mean	std	min	25%	50%	75%	max
Substance	22092.0	1.500000	1.118059	0.0	0.75	1.500	2.250	3.000
Unit	22092.0	0.250000	0.433023	0.0	0.00	0.000	0.250	1.000
Supply Chain Emission Factors without Margins	22092.0	0.084807	0.267039	0.0	0.00	0.002	0.044	7.228
Margins of Supply Chain Emission Factors	22092.0	0.012857	0.078720	0.0	0.00	0.000	0.000	3.349
Unnamed: 7	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DQ ReliabilityScore of Factors without Margins	22092.0	3.308030	0.499643	2.0	3.00	3.000	4.000	4.000
DQ TemporalCorrelation of Factors without Margins	22092.0	2.571429	0.494883	2.0	2.00	3.000	3.000	3.000
DQ GeographicalCorrelation of Factors without Margins	22092.0	1.000000	0.000000	1.0	1.00	1.000	1.000	1.000
DQ TechnologicalCorrelation of Factors without Margins	22092.0	2.632129	1.135661	1.0	1.00	3.000	3.000	5.000
DQ DataCollection of Factors without Margins	22092.0	1.000000	0.000000	1.0	1.00	1.000	1.000	1.000
Source	22092.0	0.500634	0.500011	0.0	0.00	1.000	1.000	1.000

In [51]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
C:\Users\srine\anaconda3\Lib\site-packages\sklearn\utils\extmath.py:1101: RuntimeWarning: inval
id value encountered in divide
  updated_mean = (last_sum + new_sum) / updated_sample_count
C:\Users\srine\anaconda3\Lib\site-packages\sklearn\utils\extmath.py:1106: RuntimeWarning: inval
id value encountered in divide
  T = new_sum / new_sample_count
C:\Users\srine\anaconda3\Lib\site-packages\sklearn\utils\extmath.py:1126: RuntimeWarning: inval
id value encountered in divide
  new_unnormalized_variance -= correction**2 / new_sample_count
```

In [52]:
```
X_scaled[0].min(),X_scaled[0].max()
```

Out[52]:
```
(np.float64(nan), np.float64(nan))
```

In [53]:
```
np.round(X_scaled.mean()),np.round(X_scaled.std())
```

Out[53]:
```
(np.float64(nan), np.float64(nan))
```

In [54]:
```
X.shape
```

Out[54]:
```
(22092, 11)
```

In [55]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

In [56]:
```
X_train.shape
```

Out[56]:
```
(17673, 11)
```

In [57]:
```
X_test.shape
```

Out[57]:
```
(4419, 11)
```

In [58]:
```
RF_model = RandomForestRegressor(random_state=42)
```

Step 4: Training

In [59]:
```
RF_model.fit(X_train, y_train)
```

Out[59]:
```
▾     RandomForestRegressor        i ?
                                    (https://scikit-
                                    learn.org/1.6/modules/generated/sklearn.ensemble.RandomForestRegressor.h
RandomForestRegressor(random_state=42)
```

Step 5 Prediction and Evaluation

In [60]:
```
RF_y_pred = RF_model.predict(X_test)
```

In [61]:
```
RF_y_pred[:20]
```

Out[61]:
```
array([2.92930000e-01, 1.00000000e-03, 1.21122793e-03, 1.16130018e-03,
       0.00000000e+00, 4.00000000e-03, 1.24555977e-04, 2.20009044e-03,
       2.00000000e-03, 3.93980000e-01, 0.00000000e+00, 1.40000000e-02,
       4.08395607e-03, 7.00000000e-03, 2.15970231e-03, 2.89160331e-04,
       1.02821706e-03, 3.15430000e-01, 9.00000000e-03, 0.00000000e+00])
```

In [62]:
```python
RF_mse = mean_squared_error(y_test, RF_y_pred)
RF_rmse = np.sqrt(RF_mse)
RF_r2 = r2_score(y_test, RF_y_pred)
print(f'RMSE: {RF_rmse}')
print(f'R² Score: {RF_r2}')
```

```
RMSE: 0.005811885183688606
R² Score: 0.9993986529677515
```

Step 6: Hyperparameter Tuning

In [ ]:
```python
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

# Perform grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(RandomForestRegressor(random_state=42), param_grid, cv=3, n_jobs=-1)

# Fit the grid search model on the training data
grid_search.fit(X_train, y_train)

# Best model from grid search
best_model = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)
```

In [ ]:
```python
y_pred_best = best_model.predict(X_test)


HP_mse = mean_squared_error(y_test, y_pred_best)
HP_rmse = np.sqrt(HP_mse)
HP_r2 = r2_score(y_test, y_pred_best)

print(f'RMSE: {HP_rmse}')
print(f'R² Score: {HP_r2}')
```

Step 7: Comapartive Study and Slecting the Best model

In [ ]:
```python
results = {
    'Model': ['Random Forest (Default)', 'Linear Regression', 'Random Forest (Tuned)'],
    'MSE': [RF_mse, LR_mse, HP_mse],
    'RMSE': [RF_rmse, LR_rmse, HP_rmse],
    'R2': [RF_r2, LR_r2, HP_r2]
}

# Create a DataFrame to compare the results of different models
comparison_df = pd.DataFrame(results)
print(comparison_df)
```

In [70]:
```python
!mkdir models
```

In [75]:
```python
import joblib
joblib.dump(LR_model, 'models/LR_model.pkl')
```

Out[75]:
```
['models/LR_model.pkl']
```

In [ ]:
```python

```