277244_ML_assignment

May 17, 2024

0.1 IMPORTING LIBARIES

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import warnings

# Filter out warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
```

0.2 Files reading

[2]:
```
consumer_price_indicators= '/Users/srijakarmakar/Documents/ML_project/ML␣
↪Coursework Dataset/Consumer prices indicators - FAOSTAT_data_en_2-22-2024.
↪csv'
consumer_price_indicators_df = pd.read_csv(consumer_price_indicators)
#consumer_price_indicators_df.head(10)


crop_production_indicators='/Users/srijakarmakar/Documents/ML_project/ML␣
↪Coursework Dataset/Crops production indicators - FAOSTAT_data_en_2-22-2024.
↪csv'
crop_production_indicators_df=pd.read_csv(crop_production_indicators)
#crop_production_indicators_df.head(10)


emission='/Users/srijakarmakar/Documents/ML_project/ML Coursework Dataset/
↪Emissions - FAOSTAT_data_en_2-27-2024.csv'
emission_df=pd.read_csv(emission)
# emission_df.head(10)
```

```
employment='/Users/srijakarmakar/Documents/ML_project/ML Coursework Dataset/
↪Employment - FAOSTAT_data_en_2-27-2024.csv'
employment_df=pd.read_csv(employment)
# employment_df.head(10)



exchange_rate='/Users/srijakarmakar/Documents/ML_project/ML Coursework Dataset/
↪Exchange rate - FAOSTAT_data_en_2-22-2024.csv'
exchange_rate_df=pd.read_csv(exchange_rate)
# exchange_rate_df.head(10) ##not needed

Fertilizer_use='/Users/srijakarmakar/Documents/ML_project/ML Coursework Dataset/
↪Fertilizers use - FAOSTAT_data_en_2-27-2024.csv'
fertilizer_use_df=pd.read_csv(Fertilizer_use)
#fertilizer_use_df.head(10)

Food_balance_indicator='/Users/srijakarmakar/Documents/ML_project/ML Coursework␣
↪Dataset/Food balances indicators - FAOSTAT_data_en_2-22-2024.csv'
food_balance_df=pd.read_csv(Food_balance_indicator)
#food_balance_df.head(10)

Food_Security_indicators='/Users/srijakarmakar/Documents/ML_project/ML␣
↪Coursework Dataset/Food security indicators  - FAOSTAT_data_en_2-22-2024.csv'
food_security_df=pd.read_csv(Food_Security_indicators)
#food_security_df.head(10)

Food_Trade_indicators='/Users/srijakarmakar/Documents/ML_project/ML Coursework␣
↪Dataset/Food trade indicators - FAOSTAT_data_en_2-22-2024.csv'
food_trade_df=pd.read_csv(Food_Trade_indicators)
#food_trade_df.head(10)


Foreign_direct_investment='/Users/srijakarmakar/Documents/ML_project/ML␣
↪Coursework Dataset/Foreign direct investment - FAOSTAT_data_en_2-27-2024.csv'
foreign_direct_df=pd.read_csv(Foreign_direct_investment)
#foreign_direct_df.head(10)


Land_temperature='/Users/srijakarmakar/Documents/ML_project/ML Coursework␣
↪Dataset/Land temperature change - FAOSTAT_data_en_2-27-2024.csv'
land_temp_df=pd.read_csv(Land_temperature)
#land_temp_df.head(10)

land_use='/Users/srijakarmakar/Documents/ML_project/ML Coursework Dataset/Land␣
↪use - FAOSTAT_data_en_2-22-2024.csv'
```

```
land_use_df=pd.read_csv(land_use)
#land_use_df.head(10)


pesticides_use='/Users/srijakarmakar/Documents/ML_project/ML Coursework Dataset/
↪Pesticides use - FAOSTAT_data_en_2-27-2024.csv'
pesticides_use_df=pd.read_csv(pesticides_use)
#pesticides_use_df.head(10)
```

0.3 —-Data feature selection—-

0.4 Consumer price indicators

[3]:
```
#preparing df_consumer_price_indicator data
selected_columns = ['Area Code (M49)', 'Area', 'Year','Domain', 'Item', 'Value']

# Filtering out rows where Item is 'Food price inflation'
filtered_df =␣
↪consumer_price_indicators_df[selected_columns][consumer_price_indicators_df['Item']␣
↪== 'Food price inflation']

# Resetting index
filtered_df.reset_index(drop=True, inplace=True)

filtered_df = filtered_df.groupby(['Area Code (M49)','Area', 'Year', 'Domain',␣
↪'Item'])['Value'].mean().reset_index()
```

[4]:
```
cpi_df = filtered_df.copy()
cpi_df.head()
```

[4]:
```
   Area Code (M49)         Area  Year                   Domain  \
0                4  Afghanistan  2001  Consumer Price Indices
1                4  Afghanistan  2002  Consumer Price Indices
2                4  Afghanistan  2003  Consumer Price Indices
3                4  Afghanistan  2004  Consumer Price Indices
4                4  Afghanistan  2005  Consumer Price Indices

                   Item      Value
0  Food price inflation  12.780692
1  Food price inflation  18.254516
2  Food price inflation  14.102244
3  Food price inflation  14.072172
4  Food price inflation  12.606240
```

[5]:
```
columns_to_drop = ['Domain', 'Item']

cpi_df.drop(columns=columns_to_drop, inplace=True)
```

```
cpi_df
cpi_df.rename(columns={'Value' : 'food_price_inflation'}, inplace = True)
cpi_df.head(10)
```

[5]:

	Area Code (M49)	Area	Year	food_price_inflation
0	4	Afghanistan	2001	12.780692
1	4	Afghanistan	2002	18.254516
2	4	Afghanistan	2003	14.102244
3	4	Afghanistan	2004	14.072172
4	4	Afghanistan	2005	12.606240
5	4	Afghanistan	2006	6.305341
6	4	Afghanistan	2007	12.265916
7	4	Afghanistan	2008	41.136456
8	4	Afghanistan	2009	-12.142569
9	4	Afghanistan	2010	0.094617

0.5 Crop production indicators

[6]:

```
##crop_production_indicators_df
selected_columns = ['Area Code (M49)', 'Area', 'Year', 'Value', 'Element']

filtered_df_crpi =␣
↪crop_production_indicators_df[selected_columns][crop_production_indicators_df['Flag␣
↪Description'] == 'Official figure']

filtered_df_crpi.reset_index(drop=True, inplace=True)

filtered_df_crpi = filtered_df_crpi.groupby(['Area Code␣
↪(M49)','Area','Year','Element' ])['Value'].mean().reset_index()
```

[7]:

```
filtered_df_crpi.drop(columns=['Element'], inplace=True)
filtered_df_crpi.rename(columns={'Value' : 'Yield'}, inplace = True)
filtered_df_crpi.head(10)
```

[7]:

	Area Code (M49)	Area	Year	Yield
0	4	Afghanistan	2000	64099.333333
1	4	Afghanistan	2001	64704.666667
2	4	Afghanistan	2002	66451.333333
3	4	Afghanistan	2003	52071.750000
4	4	Afghanistan	2004	91307.000000
5	4	Afghanistan	2005	79515.750000
6	4	Afghanistan	2006	84232.250000
7	4	Afghanistan	2007	84776.500000
8	4	Afghanistan	2008	77277.000000
9	4	Afghanistan	2009	164872.666667

0.6 Emission

[8]:
```python
selected_columns = ['Area Code (M49)', 'Area', 'Year','Element','Value']
filtered_emission = emission_df[selected_columns][emission_df['Element'] ==␣
↪'Emissions (CO2)']
filtered_emission.reset_index(drop=True, inplace=True)
filtered_emission = filtered_emission.groupby(['Area Code (M49)','Area',␣
↪'Year', 'Element'])['Value'].mean().reset_index()
```

[9]:
```python
filtered_emission.rename(columns={'Value' : 'EmissionCO2'}, inplace = True)
filtered_emission.drop(columns=['Element'], inplace=True)
filtered_emission.head(10)
```

[9]:

	Area Code (M49)	Area	Year	EmissionCO2
0	4	Afghanistan	2000	0.0
1	4	Afghanistan	2001	0.0
2	4	Afghanistan	2002	0.0
3	4	Afghanistan	2003	0.0
4	4	Afghanistan	2004	0.0
5	4	Afghanistan	2005	0.0
6	4	Afghanistan	2006	0.0
7	4	Afghanistan	2007	0.0
8	4	Afghanistan	2008	0.0
9	4	Afghanistan	2009	0.0

0.7 Exchange rate

[10]:
```python
selected_columns = ['Area Code (M49)', 'Area', 'Year','Value']
filtered_exchange_rate = exchange_rate_df.groupby(['Area Code (M49)','Area',␣
↪'Year'])['Value'].mean().reset_index()
```

[11]:
```python
filtered_exchange_rate.rename(columns={'Value' : 'Average_exchange_rate'},␣
↪inplace = True)
```

[12]:
```python
filtered_exchange_rate.head(10)
```

[12]:

	Area Code (M49)	Area	Year	Average_exchange_rate
0	4	Afghanistan	1980	44.129167
1	4	Afghanistan	1981	49.479902
2	4	Afghanistan	1982	50.599608
3	4	Afghanistan	1983	50.599608
4	4	Afghanistan	1984	50.599606
5	4	Afghanistan	1985	50.599605
6	4	Afghanistan	1986	50.599605
7	4	Afghanistan	1987	50.599605
8	4	Afghanistan	1988	50.599605
9	4	Afghanistan	1989	50.599605

0.8 Fertilizer Use

[13]:
```
selected_columns = ['Area Code (M49)', 'Area', 'Year', 'Value']

filtered_fertilizer =␣
↪fertilizer_use_df[selected_columns][fertilizer_use_df['Flag Description'] ==␣
↪'Official figure']

filtered_fertilizer.reset_index(drop=True, inplace=True)

filtered_fertilizer = filtered_fertilizer.groupby(['Area Code␣
↪(M49)','Area','Year'])['Value'].mean().reset_index()
```

[14]:
```
filtered_fertilizer.rename(columns={'Value' : 'Avg_Fertilizer'}, inplace = True)
```

[15]:
```
filtered_fertilizer.head(10)
```

[15]:

	Area Code (M49)	Area	Year	Avg_Fertilizer
0	4	Afghanistan	2018	519122.000000
1	8	Albania	2002	39908.666667
2	8	Albania	2003	39967.666667
3	8	Albania	2004	25846.200000
4	8	Albania	2005	26666.000000
5	8	Albania	2006	21924.400000
6	8	Albania	2007	22313.400000
7	8	Albania	2008	20727.400000
8	8	Albania	2009	24094.400000
9	8	Albania	2010	22947.400000

0.9 Food balance

[16]:
```
food_balance_df['Item'].unique()
```

[16]:
```
array(['Cereals - Excluding Beer', 'Starchy Roots', 'Sugar Crops',
       'Sugar & Sweeteners', 'Pulses', 'Treenuts', 'Oilcrops',
       'Vegetable Oils', 'Vegetables', 'Fruits - Excluding Wine',
       'Stimulants', 'Spices', 'Alcoholic Beverages', 'Meat', 'Eggs',
       'Milk - Excluding Butter', 'Fish, Seafood'], dtype=object)
```

[17]:
```
selected_columns = ['Area Code (M49)', 'Area', 'Year', 'Value','Item']

items_to_remove = ['Meat', 'Eggs', 'Milk - Excluding Butter', 'Fish, Seafood',␣
↪'Alcoholic Beverages']


filtered_food_balance =␣
↪food_balance_df[selected_columns][(food_balance_df['Element'] == 'Export␣
↪Quantity') &
```

```
                                                    (~food_balance_df['Item'].
 ↪isin(items_to_remove))]

filtered_food_balance.reset_index(drop=True, inplace=True)

filtered_food_balance = filtered_food_balance.groupby(['Area Code␣
 ↪(M49)','Area','Year'])['Value'].mean().reset_index()
filtered_food_balance.rename(columns={'Value' : 'food_balance_export'}, inplace␣
 ↪= True)
```

[18]:
```
filtered_food_balance.head(5)
```

[18]:
```
   Area Code (M49)         Area  Year  food_balance_export
0                4  Afghanistan  2010            40.000000
1                4  Afghanistan  2011            30.777778
2                4  Afghanistan  2012            22.000000
3                4  Afghanistan  2013            31.222222
4                4  Afghanistan  2014            37.454545
```

0.10 Food security

[19]:
```
food_security_df['Item Code'].unique()
```

[19]:
```
array([21010, 21013, 21035, 21034, 21033, 21032, 21030, 21031, 21043,
       21049])
```

[20]:
```
food_security_df[food_security_df['Item Code']==21031]
```

[20]:
```
      Domain Code                            Domain  Area Code (M49)  \
140            FS  Suite of Food Security Indicators                4
141            FS  Suite of Food Security Indicators                4
142            FS  Suite of Food Security Indicators                4
143            FS  Suite of Food Security Indicators                4
144            FS  Suite of Food Security Indicators                4
…             …                                 …               …
36466          FS  Suite of Food Security Indicators              716
36467          FS  Suite of Food Security Indicators              716
36468          FS  Suite of Food Security Indicators              716
36469          FS  Suite of Food Security Indicators              716
36470          FS  Suite of Food Security Indicators              716

              Area  Element Code Element  Item Code  \
140    Afghanistan          6128   Value      21031
141    Afghanistan          6128   Value      21031
142    Afghanistan          6128   Value      21031
143    Afghanistan          6128   Value      21031
144    Afghanistan          6128   Value      21031
```

```
…           …           …     …        …
36466     Zimbabwe          6128    Value      21031
36467     Zimbabwe          6128    Value      21031
36468     Zimbabwe          6128    Value      21031
36469     Zimbabwe          6128    Value      21031
36470     Zimbabwe          6128    Value      21031

                                                Item  Year Code  Year  \
140    Per capita food supply variability (kcal/cap/day)       2000  2000
141    Per capita food supply variability (kcal/cap/day)       2001  2001
142    Per capita food supply variability (kcal/cap/day)       2002  2002
143    Per capita food supply variability (kcal/cap/day)       2003  2003
144    Per capita food supply variability (kcal/cap/day)       2004  2004
…                                                   …         …   …
36466  Per capita food supply variability (kcal/cap/day)       2017  2017
36467  Per capita food supply variability (kcal/cap/day)       2018  2018
36468  Per capita food supply variability (kcal/cap/day)       2019  2019
36469  Per capita food supply variability (kcal/cap/day)       2020  2020
36470  Per capita food supply variability (kcal/cap/day)       2021  2021

            Unit  Value Flag Flag Description Note
140    kcal/pc/d   58.0    E  Estimated value  NaN
141    kcal/pc/d   47.0    E  Estimated value  NaN
142    kcal/pc/d   71.0    E  Estimated value  NaN
143    kcal/pc/d   72.0    E  Estimated value  NaN
144    kcal/pc/d   50.0    E  Estimated value  NaN
…            …    …   …              …  …
36466  kcal/pc/d   60.0    E  Estimated value  NaN
36467  kcal/pc/d   53.0    E  Estimated value  NaN
36468  kcal/pc/d   22.0    E  Estimated value  NaN
36469  kcal/pc/d   20.0    E  Estimated value  NaN
36470  kcal/pc/d   21.0    E  Estimated value  NaN

[3776 rows x 15 columns]
```

[ ]:

[21]:
```python
food_security_df = food_security_df[food_security_df['Item'].isin([
    'Per capita food production variability (constant 2014-2016 thousand int$␣
↪per capita)',
    'Per capita food supply variability (kcal/cap/day']])]
```

[22]:
```python
food_security_df.head(10)
```

[22]:
```
    Domain Code                          Domain  Area Code (M49)  \
120          FS  Suite of Food Security Indicators                4
121          FS  Suite of Food Security Indicators                4
```

```
122          FS   Suite of Food Security Indicators               4
123          FS   Suite of Food Security Indicators               4
124          FS   Suite of Food Security Indicators               4
125          FS   Suite of Food Security Indicators               4
126          FS   Suite of Food Security Indicators               4
127          FS   Suite of Food Security Indicators               4
128          FS   Suite of Food Security Indicators               4
129          FS   Suite of Food Security Indicators               4

            Area  Element Code Element  Item Code  \
120  Afghanistan          6127   Value      21030
121  Afghanistan          6127   Value      21030
122  Afghanistan          6127   Value      21030
123  Afghanistan          6127   Value      21030
124  Afghanistan          6127   Value      21030
125  Afghanistan          6127   Value      21030
126  Afghanistan          6127   Value      21030
127  Afghanistan          6127   Value      21030
128  Afghanistan          6127   Value      21030
129  Afghanistan          6127   Value      21030

                                                   Item  Year Code  Year  \
120  Per capita food production variability (consta…      2001  2001
121  Per capita food production variability (consta…      2002  2002
122  Per capita food production variability (consta…      2003  2003
123  Per capita food production variability (consta…      2004  2004
124  Per capita food production variability (consta…      2005  2005
125  Per capita food production variability (consta…      2006  2006
126  Per capita food production variability (consta…      2007  2007
127  Per capita food production variability (consta…      2008  2008
128  Per capita food production variability (consta…      2009  2009
129  Per capita food production variability (consta…      2010  2010

        Unit  Value Flag Flag Description Note
120  1000 I$   16.3    E  Estimated value  NaN
121  1000 I$   21.0    E  Estimated value  NaN
122  1000 I$   20.8    E  Estimated value  NaN
123  1000 I$   17.3    E  Estimated value  NaN
124  1000 I$   12.4    E  Estimated value  NaN
125  1000 I$   14.4    E  Estimated value  NaN
126  1000 I$    8.0    E  Estimated value  NaN
127  1000 I$    8.5    E  Estimated value  NaN
128  1000 I$    8.9    E  Estimated value  NaN
129  1000 I$   10.9    E  Estimated value  NaN
```

[23]:
```python
food_security_df = food_security_df.groupby(['Area Code (M49)','Area',
 ↪'Year'])['Value'].mean().reset_index()
```

```
[24]: food_security_df = food_security_df.rename(columns={'Value':
       ↪'Total_food_security'})
      food_security_df.head()
```

```
[24]:    Area Code (M49)         Area  Year  Total_food_security
      0                4  Afghanistan  2001                 16.3
      1                4  Afghanistan  2002                 21.0
      2                4  Afghanistan  2003                 20.8
      3                4  Afghanistan  2004                 17.3
      4                4  Afghanistan  2005                 12.4
```

0.11 Food trade indicators

```
[25]: selected_columns = ['Area Code (M49)', 'Area', 'Year', 'Value']

      items_to_remove = ['Meat and Meat Preparations', 'Dairy Products and Eggs',
       ↪'Non-food', 'Other food', 'Alcoholic Beverages']



      # Filtering out rows where Item is 'Food price inflation'
      filtered_food_trade = food_trade_df[selected_columns][(food_trade_df['Element']
       ↪== 'Export Value') &
                                                        (~food_trade_df['Item'].
       ↪isin(items_to_remove))]
      # Resetting index
      filtered_food_trade.reset_index(drop=True, inplace=True)

      filtered_food_trade = filtered_food_trade.groupby(['Area Code
       ↪(M49)','Area','Year'])['Value'].sum().reset_index()
```

```
[26]: filtered_food_trade.rename(columns = {'Value': 'Export_Value'}, inplace = True)
```

```
[27]: filtered_food_trade.head()
```

```
[27]:    Area Code (M49)         Area  Year  Export_Value
      0                4  Afghanistan  1991       51858.0
      1                4  Afghanistan  1992       19062.0
      2                4  Afghanistan  1993       21324.0
      3                4  Afghanistan  1994       26907.0
      4                4  Afghanistan  1995       24240.0
```

0.12 Foreign direct investment

```
[28]: #preparing df_consumer_price_indicator data
      selected_columns = ['Area Code (M49)', 'Area', 'Year', 'Item','Value']

      # Filtering out rows where Item is in the list of items
```

```python
filtered_df_fdi = foreign_direct_df[selected_columns][foreign_direct_df['Item'].
 ↪isin(['Total FDI inflows','Total FDI outflows'])]

# Resetting index
filtered_df_fdi.reset_index(drop=True, inplace=True)

filtered_df_fdi = filtered_df_fdi.groupby(['Area Code (M49)','Area','Year',␣
 ↪'Item'])['Value'].mean().reset_index()
```

[29]:
```python
filtered_df_fdi.head(10)
```

[29]:

	Area Code (M49)	Area	Year	Item	Value
0	4	Afghanistan	2000	Total FDI inflows	0.17
1	4	Afghanistan	2001	Total FDI inflows	0.68
2	4	Afghanistan	2002	Total FDI inflows	50.00
3	4	Afghanistan	2003	Total FDI inflows	57.80
4	4	Afghanistan	2003	Total FDI outflows	1.00
5	4	Afghanistan	2004	Total FDI inflows	186.90
6	4	Afghanistan	2004	Total FDI outflows	-0.70
7	4	Afghanistan	2005	Total FDI inflows	271.00
8	4	Afghanistan	2005	Total FDI outflows	1.50
9	4	Afghanistan	2006	Total FDI inflows	238.00

[30]:
```python
import pandas as pd

# Filter the DataFrame based on specified items
filtered_fdi = foreign_direct_df[foreign_direct_df['Item'].isin(['Total FDI␣
 ↪inflows', 'Total FDI outflows'])]

# Pivot the DataFrame
pivot_df = filtered_fdi.pivot_table(
    index=['Area Code (M49)', 'Area', 'Year'],
    columns='Item',
    values='Value',
    aggfunc='mean'
).reset_index()

# Rename the columns
pivot_df.columns.name = None
pivot_df.rename(columns={
    'Total FDI inflows': 'FDI_inflows',
    'Total FDI outflows': 'FDI_outflows'
}, inplace=True)

# Display the pivot table
pivot_df.head(10)
```

[30]:
```
   Area Code (M49)         Area  Year  FDI_inflows  FDI_outflows
0                4  Afghanistan  2000     0.170000           NaN
1                4  Afghanistan  2001     0.680000           NaN
2                4  Afghanistan  2002    50.000000           NaN
3                4  Afghanistan  2003    57.800000      1.000000
4                4  Afghanistan  2004   186.900000     -0.700000
5                4  Afghanistan  2005   271.000000      1.500000
6                4  Afghanistan  2006   238.000000           NaN
7                4  Afghanistan  2007   188.690000           NaN
8                4  Afghanistan  2008    46.033740     -1.918036
9                4  Afghanistan  2009   197.512728      0.334959
```

0.13 Land temperature change

[31]:
```python
selected_columns = ['Area Code (M49)', 'Area', 'Year','Value']

# Filtering out rows where Element is 'Temperature change'
filtered_df_land_temp = land_temp_df[selected_columns][land_temp_df['Element']␣
 ↪== 'Temperature change']
filtered_df_land_temp.reset_index(drop=True, inplace=True)
filtered_ltemp = filtered_df_land_temp.groupby(['Area Code (M49)','Area',␣
 ↪'Year'])['Value'].mean().reset_index()
```

[32]:
```python
filtered_ltemp.head(5)
```

[32]:
```
   Area Code (M49)         Area  Year   Value
0                4  Afghanistan  2000  0.9930
1                4  Afghanistan  2001  1.3110
2                4  Afghanistan  2002  1.3650
3                4  Afghanistan  2003  0.5870
4                4  Afghanistan  2004  1.3732
```

[33]:
```python
filtered_ltemp.rename(columns={'Value' : 'temp_change'}, inplace = True)
```

[34]:
```python
filtered_ltemp.head(3)
```

[34]:
```
   Area Code (M49)         Area  Year  temp_change
0                4  Afghanistan  2000        0.993
1                4  Afghanistan  2001        1.311
2                4  Afghanistan  2002        1.365
```

0.14 Pesticides use

[35]:
```python
selected_columns = ['Area Code (M49)', 'Area', 'Year', 'Value']

# Filtering out rows where Item is 'Pesticides (total)' and Unit is 'kg/ha'
```

```
filtered_pest_use = 
 ↪pesticides_use_df[selected_columns][(pesticides_use_df['Item'] == 
 ↪'Pesticides (total)') &
                                                                    
 ↪(pesticides_use_df['Unit'].isin(['kg/ha']))]
# Resetting index
filtered_pest_use.reset_index(drop=True, inplace=True)

filtered_pest_use = filtered_pest_use.groupby(['Area Code 
 ↪(M49)','Area','Year'])['Value'].mean().reset_index()
```

[36]:
```
filtered_pest_use.rename(columns={'Value' : 'pesticides'}, inplace = True)
```

[37]:
```
filtered_pest_use.head(5)
```

[37]:

	Area Code (M49)	Area	Year	pesticides
0	8	Albania	2000	0.44
1	8	Albania	2001	0.46
2	8	Albania	2002	0.47
3	8	Albania	2003	0.49
4	8	Albania	2004	0.51

0.15 land use

[38]:
```
selected_columns = ['Area Code (M49)', 'Area', 'Year', 'Value']

# Filtering out rows where Item is 'Cropland' and Flag Description is 'Official 
 ↪figure'
filtered_land_use = land_use_df[selected_columns][(land_use_df['Item'] == 
 ↪'Cropland') &
                                                          (land_use_df['Flag 
 ↪Description'].isin(['Official figure']))]
# Resetting index
filtered_land_use.reset_index(drop=True, inplace=True)

filtered_land_use = filtered_land_use.groupby(['Area Code 
 ↪(M49)','Area','Year'])['Value'].mean().reset_index()
```

[39]:
```
filtered_land_use.rename(columns={'Value' : 'cropland_use'}, inplace = True)
```

[40]:
```
filtered_land_use.head(10)
```

[40]:

	Area Code (M49)	Area	Year	cropland_use
0	4	Afghanistan	1980	8049.0
1	4	Afghanistan	1981	8053.0
2	4	Afghanistan	1982	8054.0
3	4	Afghanistan	1983	8054.0

```
4                 4  Afghanistan  1984       8054.0
5                 4  Afghanistan  1985       8054.0
6                 4  Afghanistan  1986       8054.0
7                 4  Afghanistan  2006       7910.0
8                 4  Afghanistan  2013       7910.0
9                 4  Afghanistan  2014       7910.0
```

1 Merging all the dataframes

[41]:
```
merge_df = pd.merge(filtered_df_crpi,filtered_emission, on=['Area Code␣
↪(M49)','Area','Year'], how='outer')

merge_df = pd.merge(merge_df, filtered_exchange_rate, on=['Area Code␣
↪(M49)','Area','Year'], how='outer')
merge_df = pd.merge(merge_df, filtered_fertilizer, on=['Area Code␣
↪(M49)','Area','Year'], how='outer')
merge_df = pd.merge(merge_df, filtered_food_balance, on=['Area Code␣
↪(M49)','Area','Year'], how='outer')
merge_df = pd.merge(merge_df, filtered_food_trade, on=['Area Code␣
↪(M49)','Area','Year'], how='outer')
merge_df = pd.merge(merge_df, pivot_df, on=['Area Code (M49)','Area','Year'],␣
↪how='outer')
merge_df = pd.merge(merge_df, filtered_ltemp, on=['Area Code␣
↪(M49)','Area','Year'], how='outer')
merge_df = pd.merge(merge_df, filtered_pest_use, on=['Area Code␣
↪(M49)','Area','Year'], how='outer')
merge_df = pd.merge(merge_df, filtered_land_use, on=['Area Code␣
↪(M49)','Area','Year'], how='outer')
```

[42]:
```
merge_df.head(10)
```

[42]:
```
   Area Code (M49)         Area  Year          Yield  EmissionCO2  \
0                4  Afghanistan  2000   64099.333333          0.0
1                4  Afghanistan  2001   64704.666667          0.0
2                4  Afghanistan  2002   66451.333333          0.0
3                4  Afghanistan  2003   52071.750000          0.0
4                4  Afghanistan  2004   91307.000000          0.0
5                4  Afghanistan  2005   79515.750000          0.0
6                4  Afghanistan  2006   84232.250000          0.0
7                4  Afghanistan  2007   84776.500000          0.0
8                4  Afghanistan  2008   77277.000000          0.0
9                4  Afghanistan  2009  164872.666667          0.0

   Average_exchange_rate  Avg_Fertilizer  food_balance_export  Export_Value  \
0           47357.574730             NaN                  NaN       31080.0
1           47500.014520             NaN                  NaN       27110.0
```

```
2        3981.907750        NaN              NaN     31153.0
3          48.762754        NaN              NaN     47612.0
4          47.845312        NaN              NaN     48633.0
5          49.494597        NaN              NaN     61510.0
6          49.925331        NaN              NaN     56335.0
7          49.962018        NaN              NaN    124467.0
8          50.249615        NaN              NaN    161331.0
9          50.325000        NaN              NaN    238861.0

   FDI_inflows  FDI_outflows  temp_change  pesticides  cropland_use
0     0.170000           NaN       0.9930         NaN           NaN
1     0.680000           NaN       1.3110         NaN           NaN
2    50.000000           NaN       1.3650         NaN           NaN
3    57.800000      1.000000       0.5870         NaN           NaN
4   186.900000     -0.700000       1.3732         NaN           NaN
5   271.000000      1.500000       0.4014         NaN           NaN
6   238.000000           NaN       1.7198         NaN        7910.0
7   188.690000           NaN       0.6754         NaN           NaN
8    46.033740     -1.918036       0.7044         NaN           NaN
9   197.512728      0.334959       0.8948         NaN           NaN
```

[43]:
```
merge_df.shape ##checking final shape
```

[43]:
```
(9731, 14)
```

[44]:
```
merge_df['Export_Value'].head(10)
```

[44]:
```
0     31080.0
1     27110.0
2     31153.0
3     47612.0
4     48633.0
5     61510.0
6     56335.0
7    124467.0
8    161331.0
9    238861.0
Name: Export_Value, dtype: float64
```

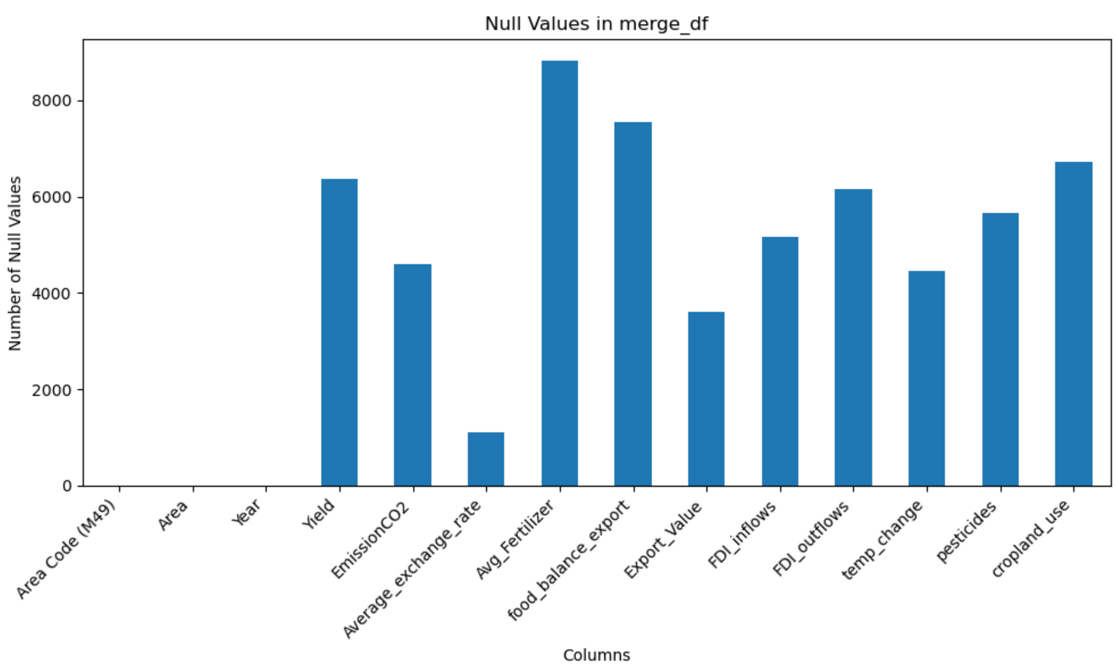
1.1 Data cleaning and exploratory data analysis

[45]:
```
# Checking Null Values
merge_df.isnull().sum()
```

[45]:
```
Area Code (M49)             0
Area                        0
Year                        0
```

```
Yield                    6376
EmissionCO2              4601
Average_exchange_rate    1092
Avg_Fertilizer           8823
food_balance_export      7555
Export_Value             3606
FDI_inflows              5165
FDI_outflows             6152
temp_change              4463
pesticides               5656
cropland_use             6720
dtype: int64
```

[46]:
```python
import matplotlib.pyplot as plt

# Calculate the number of null values in each column
null_counts = merge_df.isnull().sum()

# Create a bar plot
plt.figure(figsize=(10, 6))
null_counts.plot(kind='bar')
plt.title('Null Values in merge_df')
plt.xlabel('Columns')
plt.ylabel('Number of Null Values')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



[47]:
```
df_filtered1 = merge_df.copy()
```

[48]:
```
df_filtered1.drop(columns='Area', inplace = True)
```

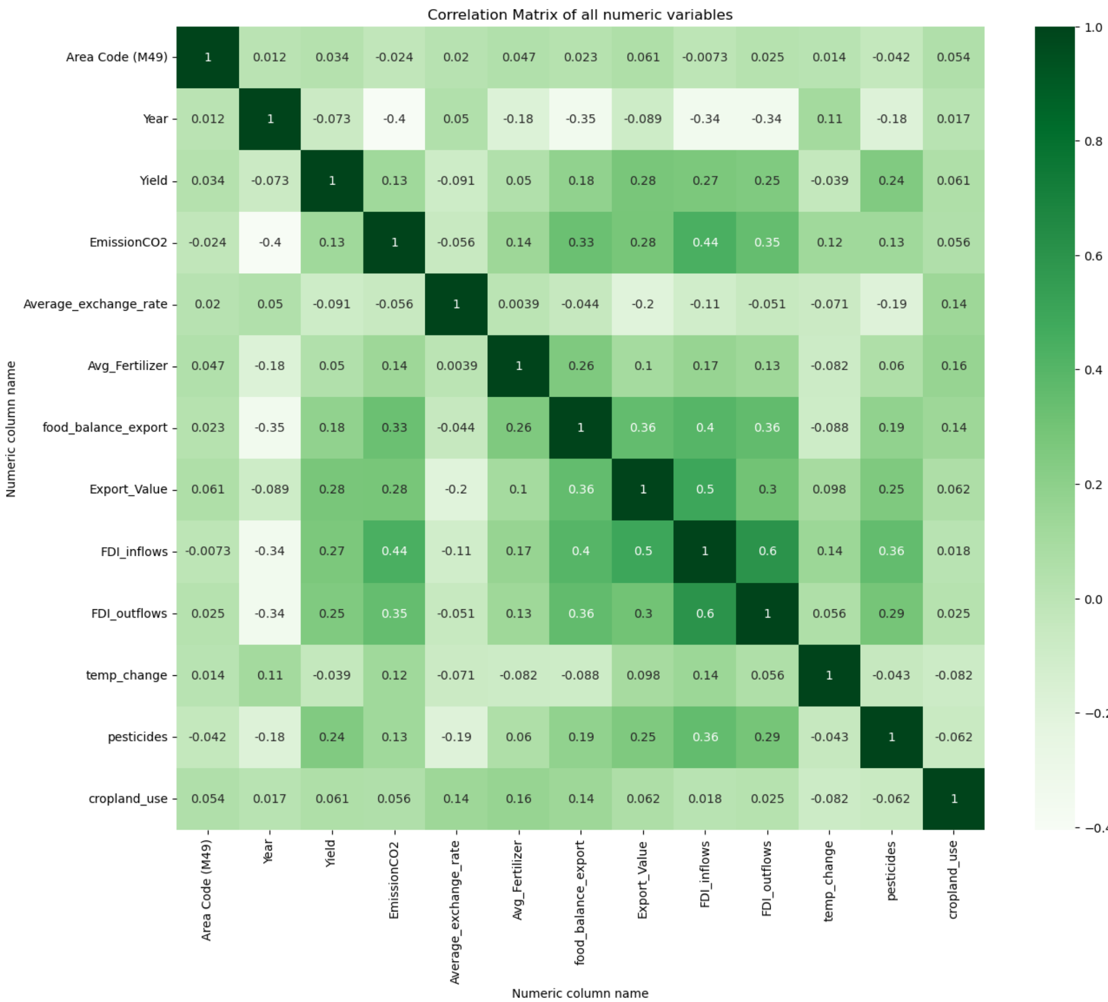
[49]:
```
# Replacing NaN values with the average value of each column
df_filtered_nan_remove = df_filtered1.fillna(df_filtered1.mean())
```

[50]:
```
df_filtered_nan_remove.isnull().sum()
```

[50]:
```
Area Code (M49)          0
Year                     0
Yield                    0
EmissionCO2              0
Average_exchange_rate    0
Avg_Fertilizer           0
food_balance_export      0
Export_Value             0
FDI_inflows              0
FDI_outflows             0
temp_change              0
pesticides               0
cropland_use             0
dtype: int64
```

[51]:
```
correlationMatrix = df_filtered_nan_remove.corr(method='spearman')
plt.figure(figsize=(15,12))
plt.title('Correlation Matrix of all numeric variables')
sns.heatmap(correlationMatrix, cmap="Greens",annot=True)
plt.xlabel('Numeric column name')
plt.ylabel('Numeric column name')
plt.plot()
```

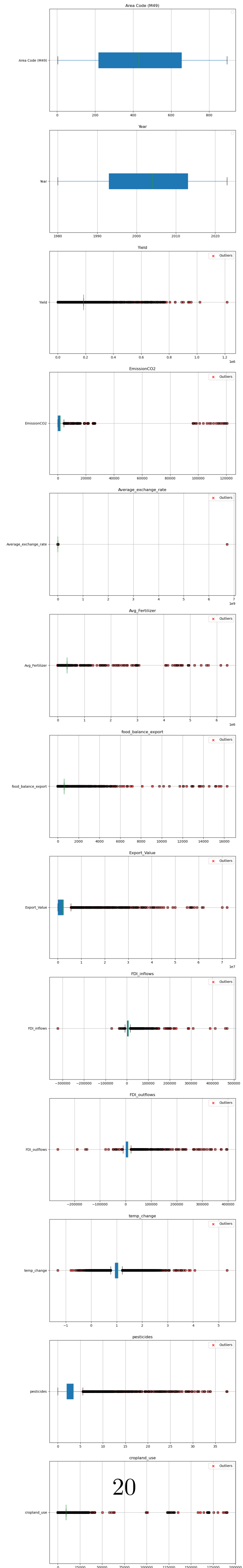
[51]:
```
[]
```



[52]:
```
import matplotlib.pyplot as plt

def outliers(df):
    # Convert columns to numeric if possible
    df = df.apply(pd.to_numeric, errors='ignore')

    # Define a figure and axes for plotting
    fig, axes = plt.subplots(nrows=len(df.columns), ncols=1, figsize=(10,␣
 ↪5*len(df.columns)))

    # Iterate over each column in the DataFrame
    for i, col in enumerate(df.columns):
        # Convert the column to numeric
        df[col] = pd.to_numeric(df[col], errors='coerce')

        # Create a box plot for the column
```

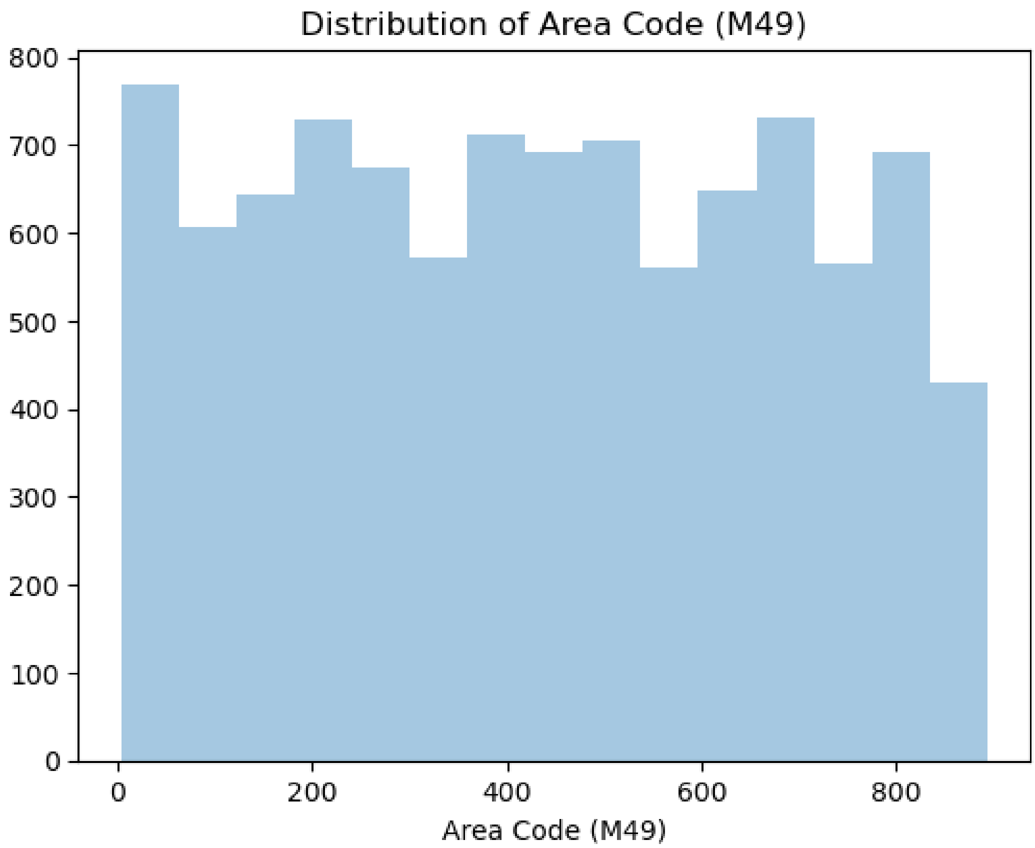
```
        df.boxplot(column=col, ax=axes[i], vert=False, patch_artist=True)

        # Calculate the IQR (Interquartile Range) for the column
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1

        # Define the lower and upper bounds for outlier detection
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Identify outliers
        outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]

        # Plot outliers
        if not outliers.empty:
            axes[i].scatter(outliers[col], [1] * len(outliers), color='red',␣
↪label='Outliers', marker='x')

        axes[i].set_title(col)
        axes[i].legend()

    plt.tight_layout()
    plt.show()

# Plot outliers using box plots in the merged table
outliers(df_filtered_nan_remove)
```

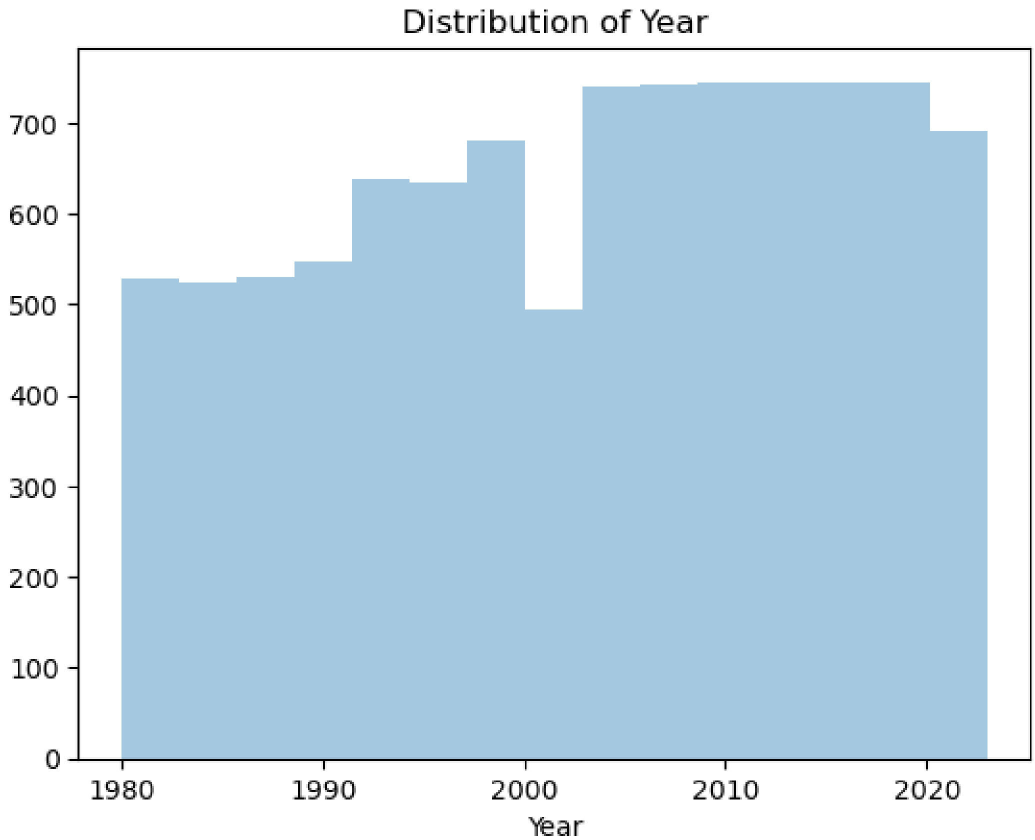
```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```

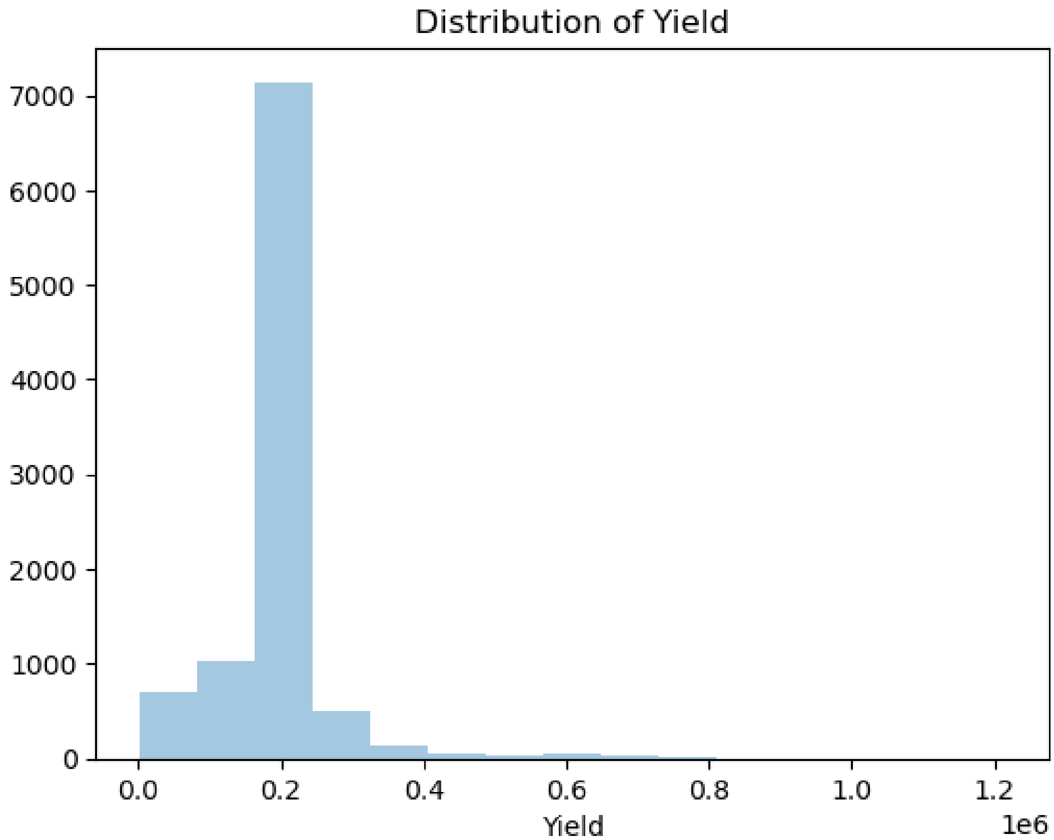

Area Code (M49)
Year
Yield
Outliers
EmissionCO2
Outliers
Average_exchange_rate
Outliers
Avg_Fertilizer
Outliers
food_balance_export
Outliers
Export_Value
Outliers
FDI_inflows
Outliers
FDI_outflows
Outliers
temp_change
Outliers
pesticides
Outliers
cropland_use
Outliers

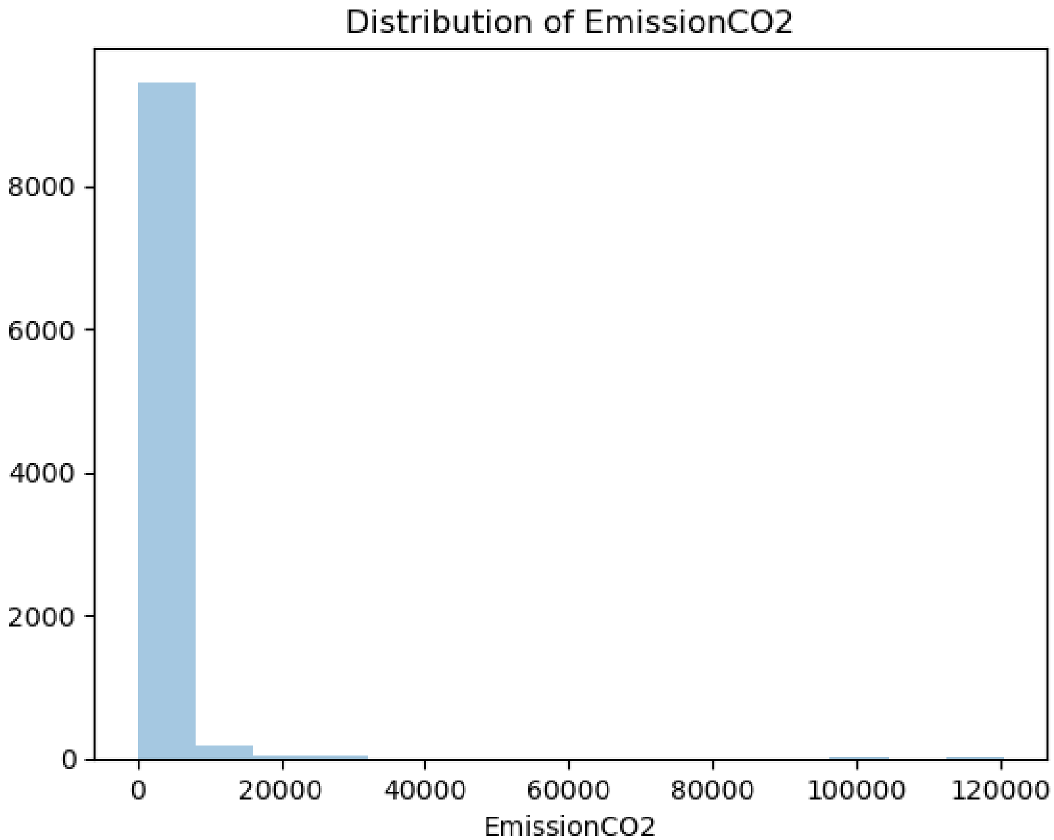
[53]:
```
columns = df_filtered_nan_remove.columns
```

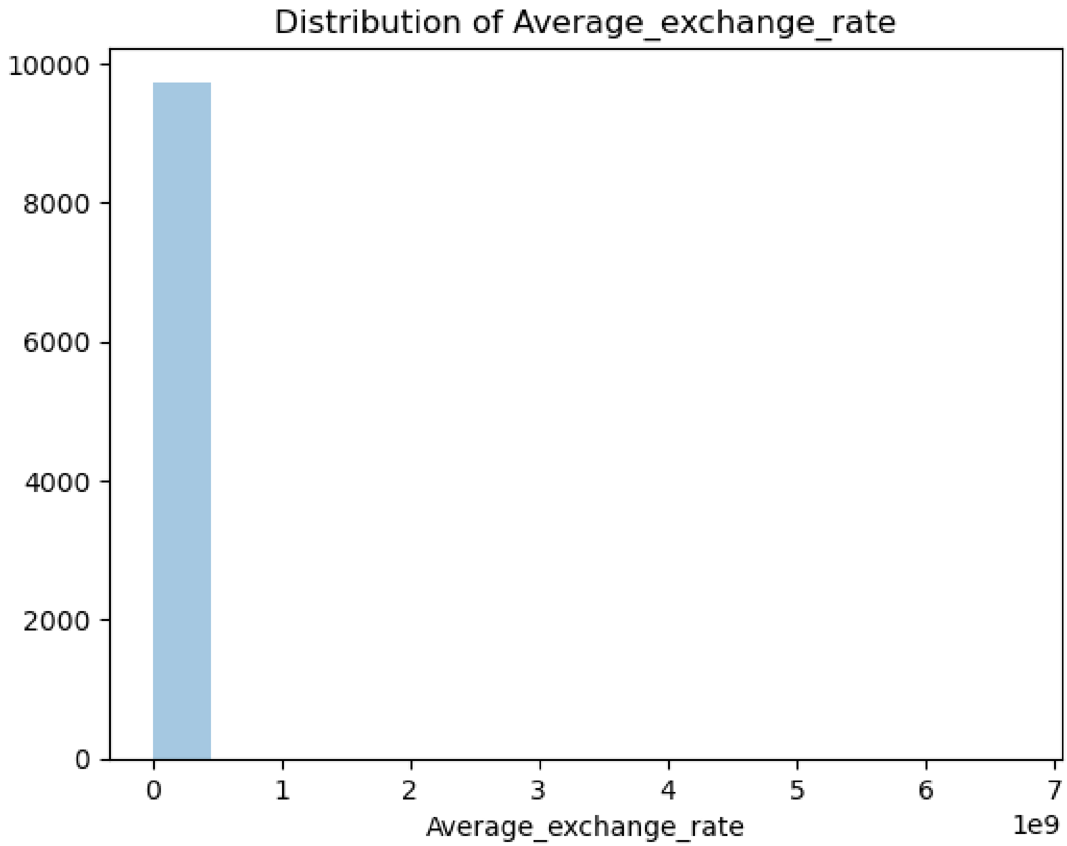
[54]:
```
#function for checking data distribution in each column- https://stackoverflow.
↪com/questions/52983362/multiple-distplots-from-pandas-columns
def distplot_df(df, columns):
    for col in columns:
        sns.distplot(df[col], bins = 15, kde = False)
        plt.title(f'Distribution of {col}')
        plt.show()
columns = df_filtered_nan_remove.columns
distplot_df(df_filtered_nan_remove, columns)
```

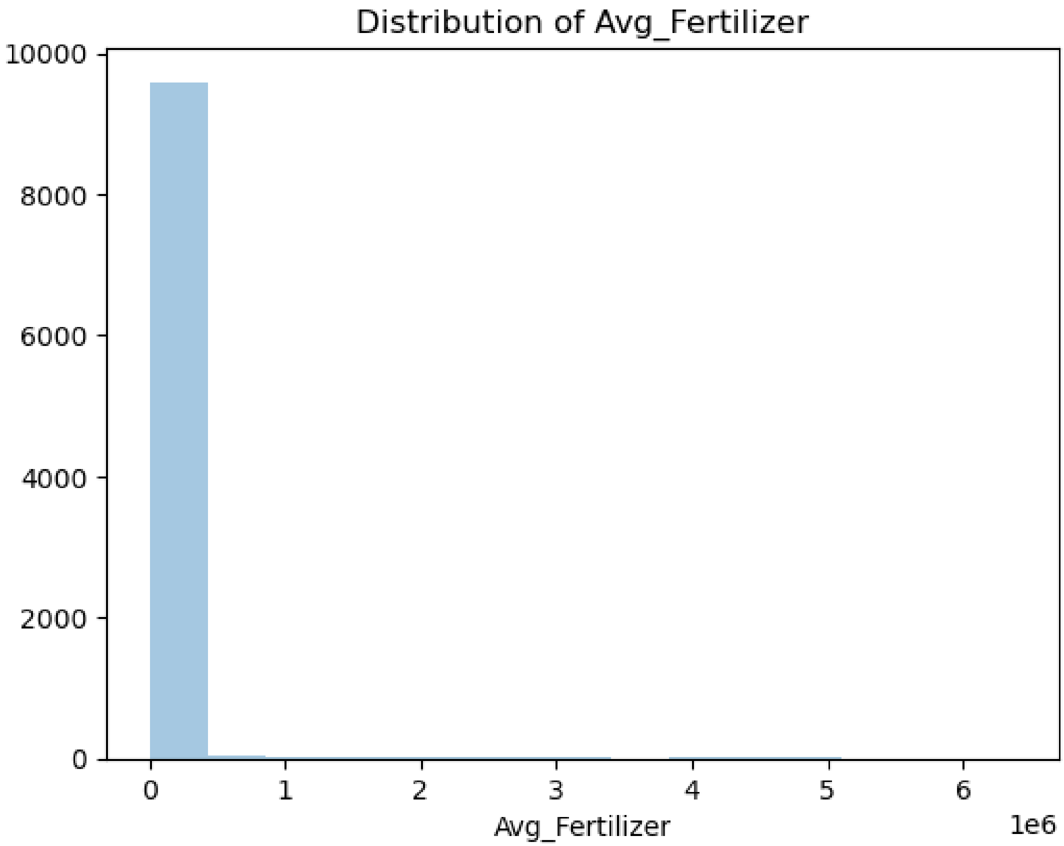


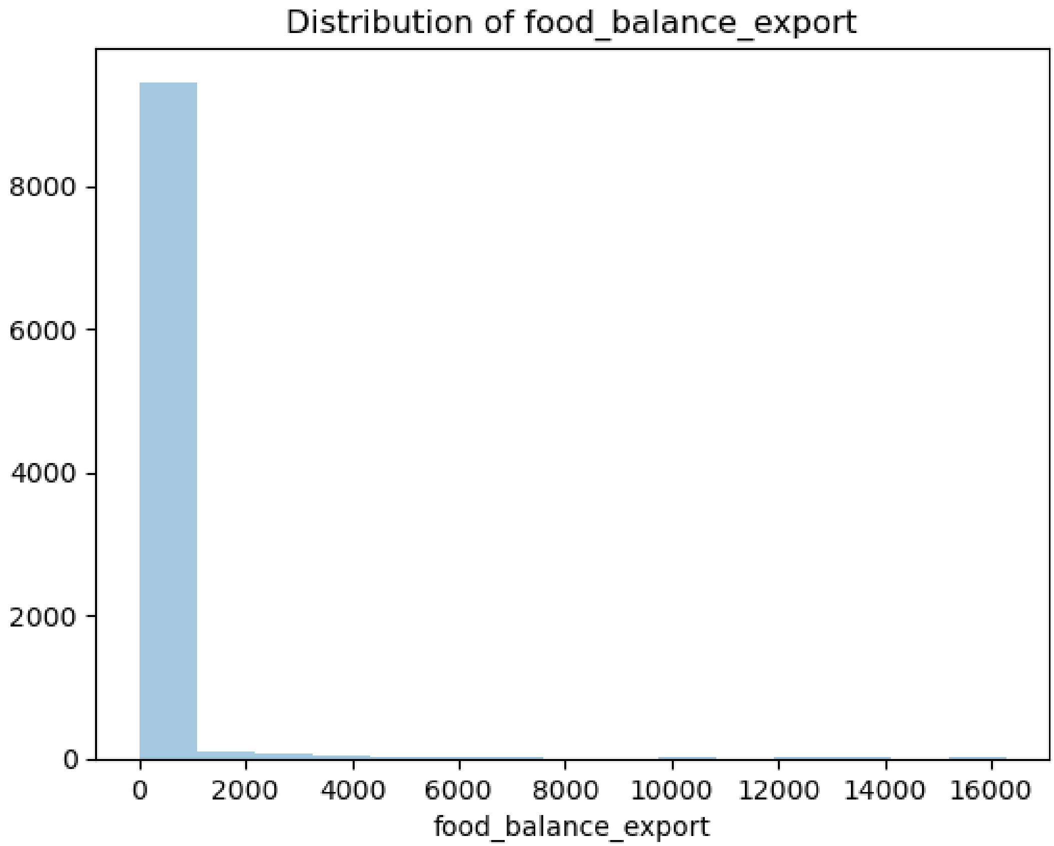

Distribution of Year
700
600
500
400
300
200
100
0
1980
1990
2000
2010
2020
Year


Distribution of Yield
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Yield
1e6

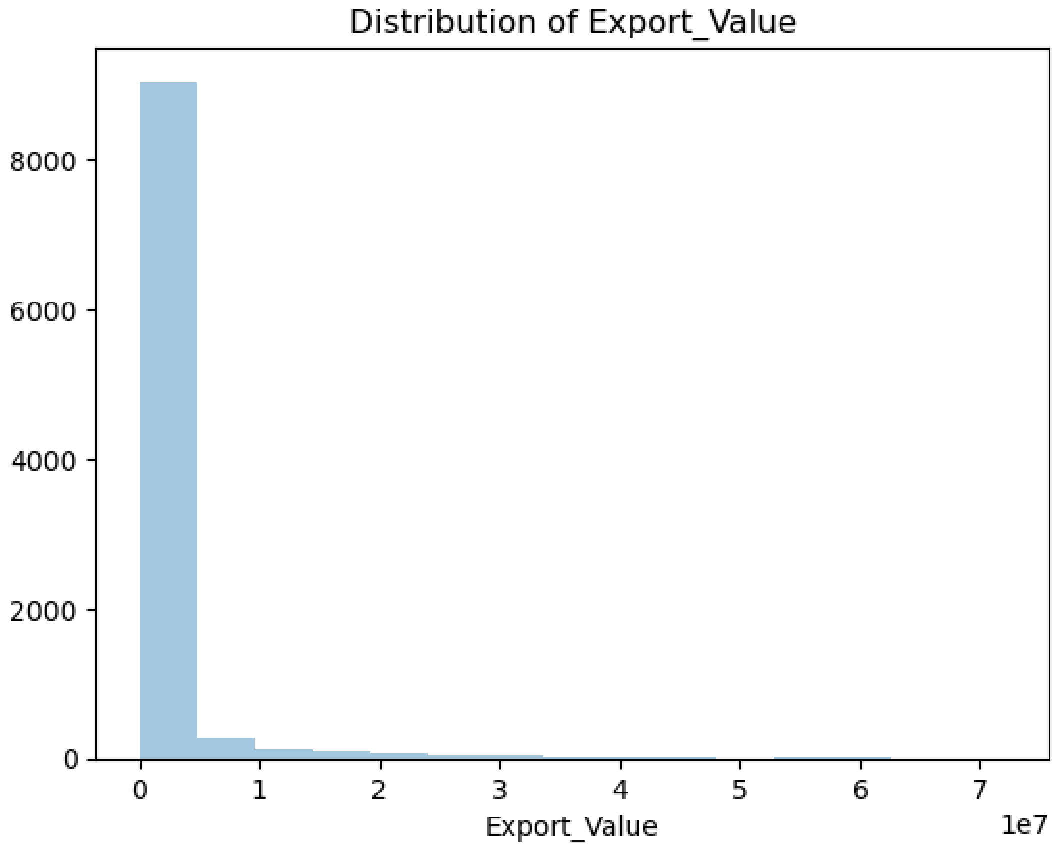

Distribution of EmissionCO2
8000
6000
4000
2000
0
0
20000
40000
60000
80000
100000
120000
EmissionCO2

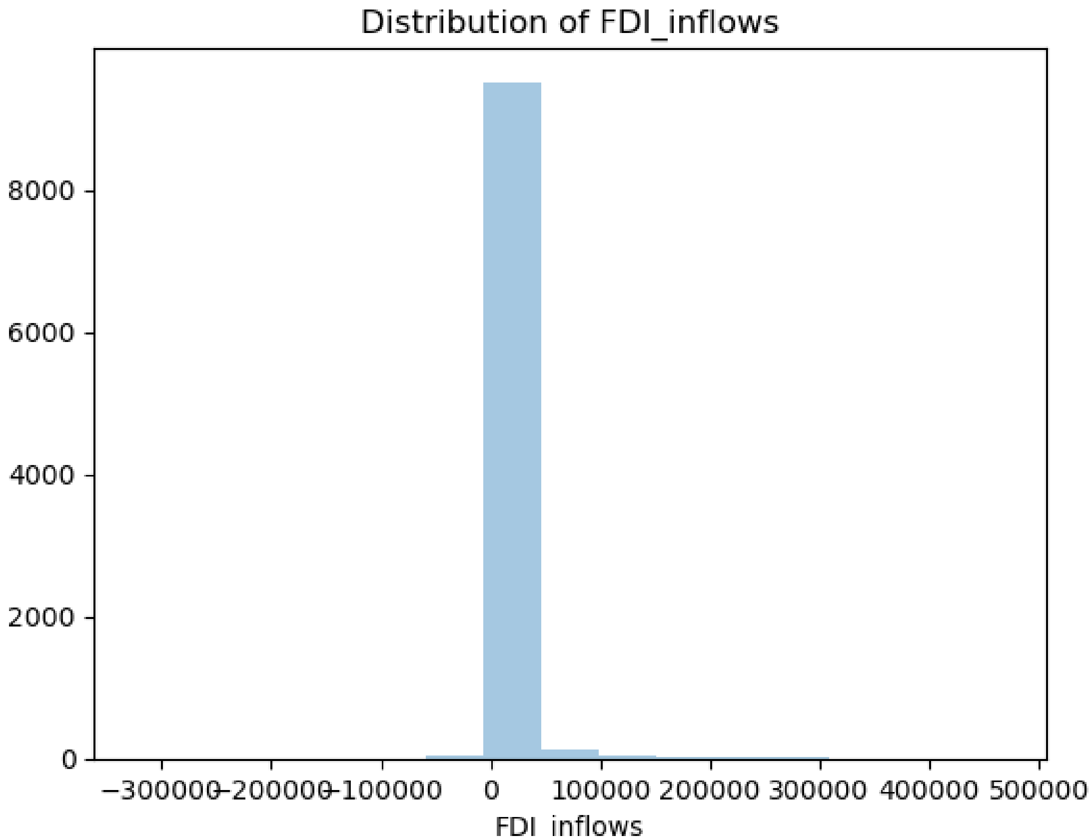

Distribution of Average_exchange_rate
10000
8000
6000
4000
2000
0
0
1
2
3
4
5
6
7
Average_exchange_rate
1e9

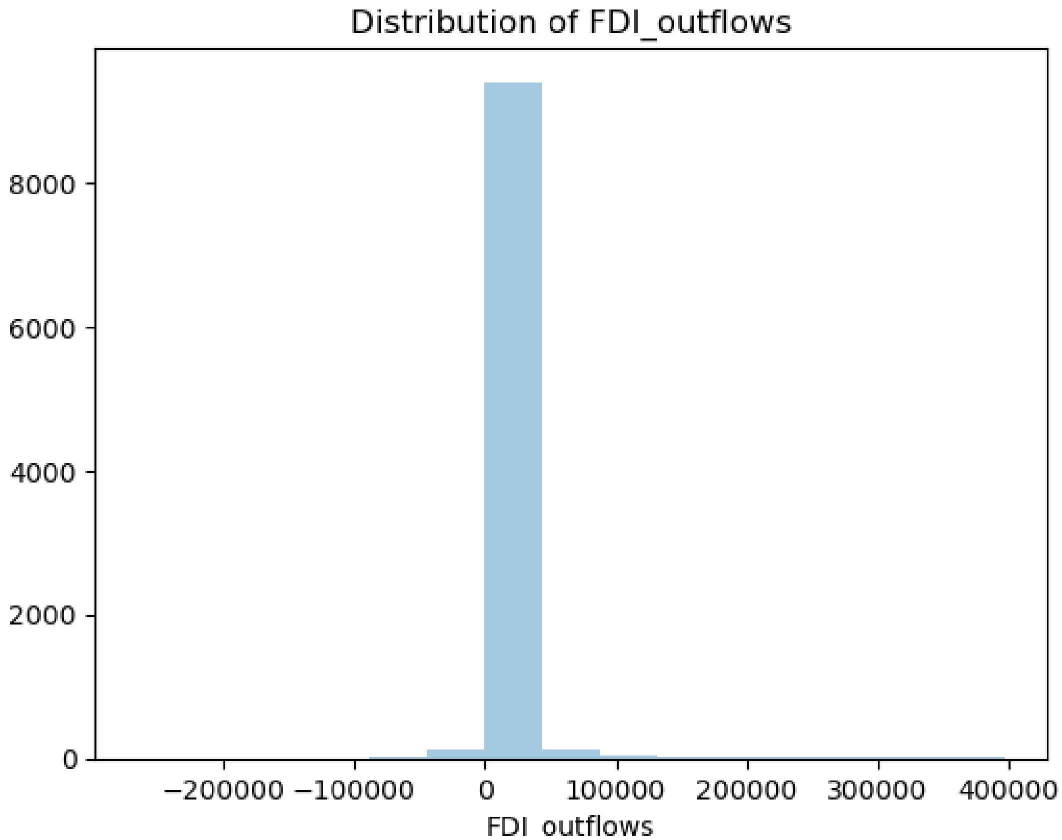

Distribution of Avg_Fertilizer
10000
8000
6000
4000
2000
0
0
1
2
3
4
5
6
Avg_Fertilizer
1e6

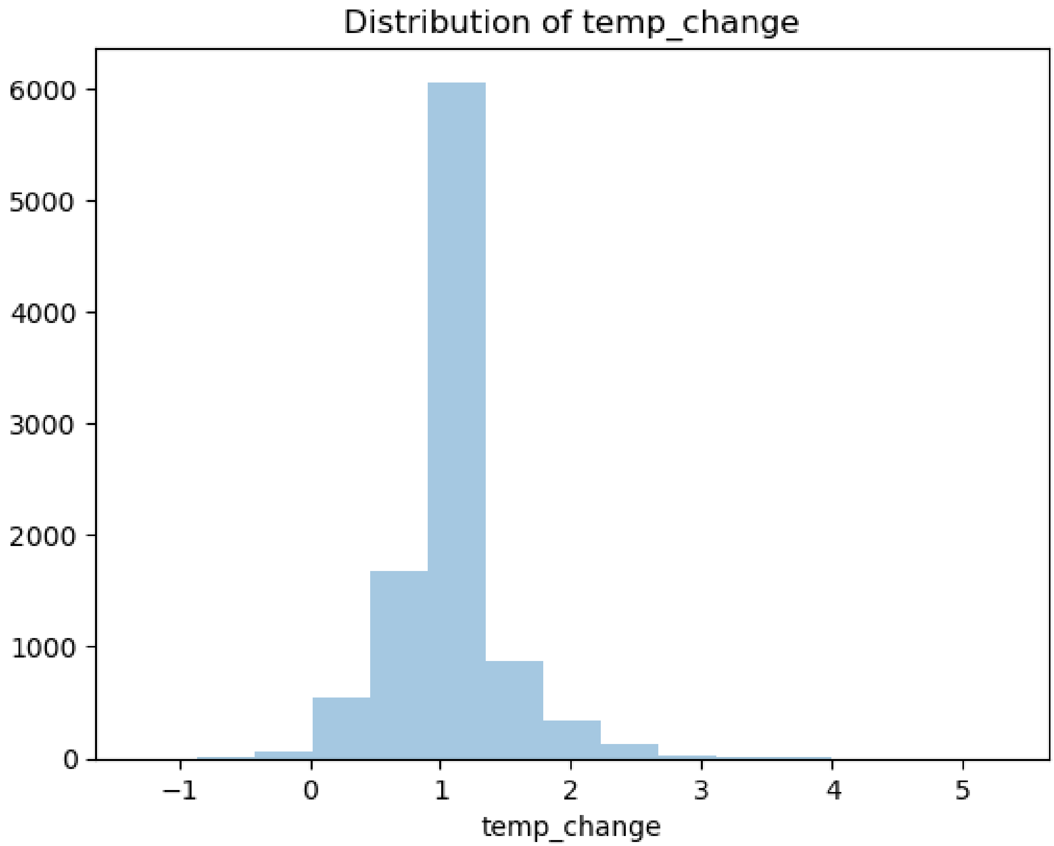


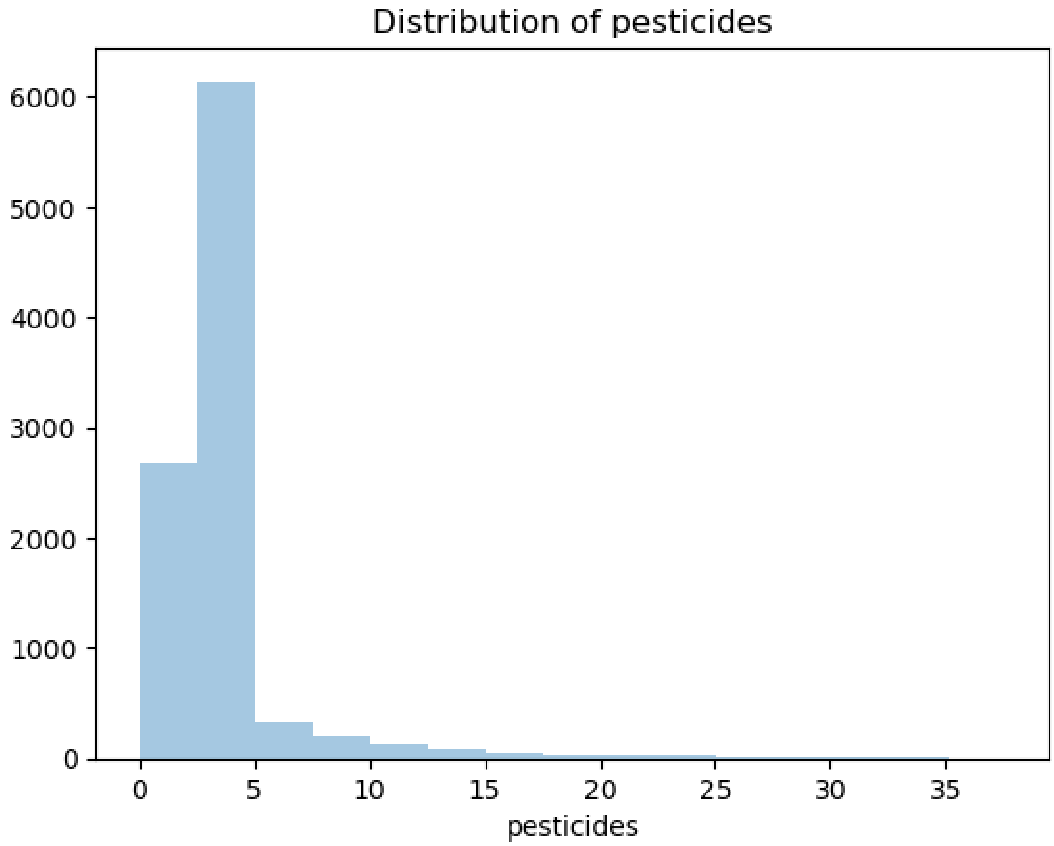
Distribution of food_balance_export
8000
6000
4000
2000
0
0
2000
4000
6000
8000
10000
12000
14000
16000
food_balance_export

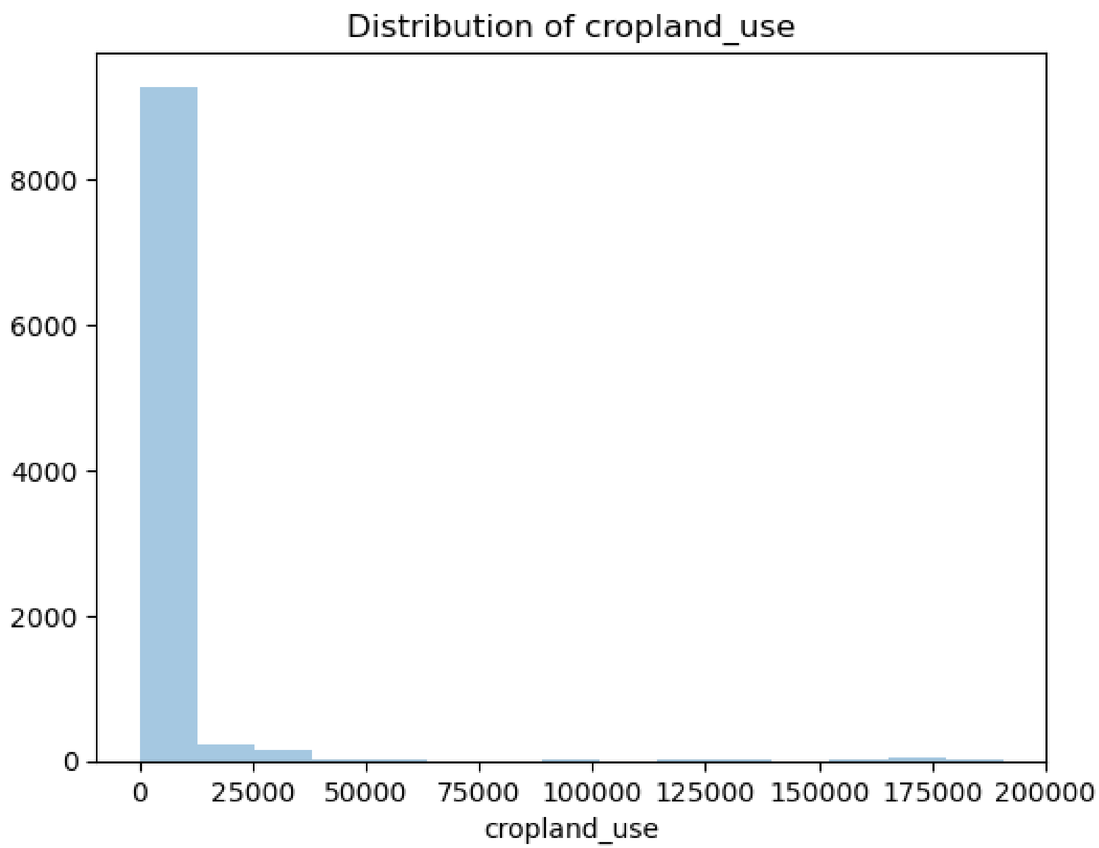

Distribution of Export_Value
8000
6000
4000
2000
0
0
1
2
3
4
5
6
7
Export_Value
1e7


Distribution of FDI_inflows
8000
6000
4000
2000
0
−300000
−200000
−100000
0
100000
200000
300000
400000
500000
FDI_inflows


Distribution of FDI_outflows
8000
6000
4000
2000
0
−200000
−100000
0
100000
200000
300000
400000
FDI_outflows


Distribution of temp_change
6000
5000
4000
3000
2000
1000
0
−1
0
1
2
3
4
5
temp_change


Distribution of pesticides
6000
5000
4000
3000
2000
1000
0
0
5
10
15
20
25
30
35
pesticides



[55]:
```
df_log = np.log(df_filtered_nan_remove['temp_change'])
```

[56]:
```
df_log.head(10)
```

[56]:
```
0   -0.007025
1    0.270790
2    0.311154
3   -0.532730
4    0.317144
5   -0.912797
6    0.542208
7   -0.392450
8   -0.350409
9   -0.111155
Name: temp_change, dtype: float64
```

[57]:
```
df_filtered_nan_remove['temp_change'].head()
```

[57]:
```
0    0.9930
1    1.3110
```

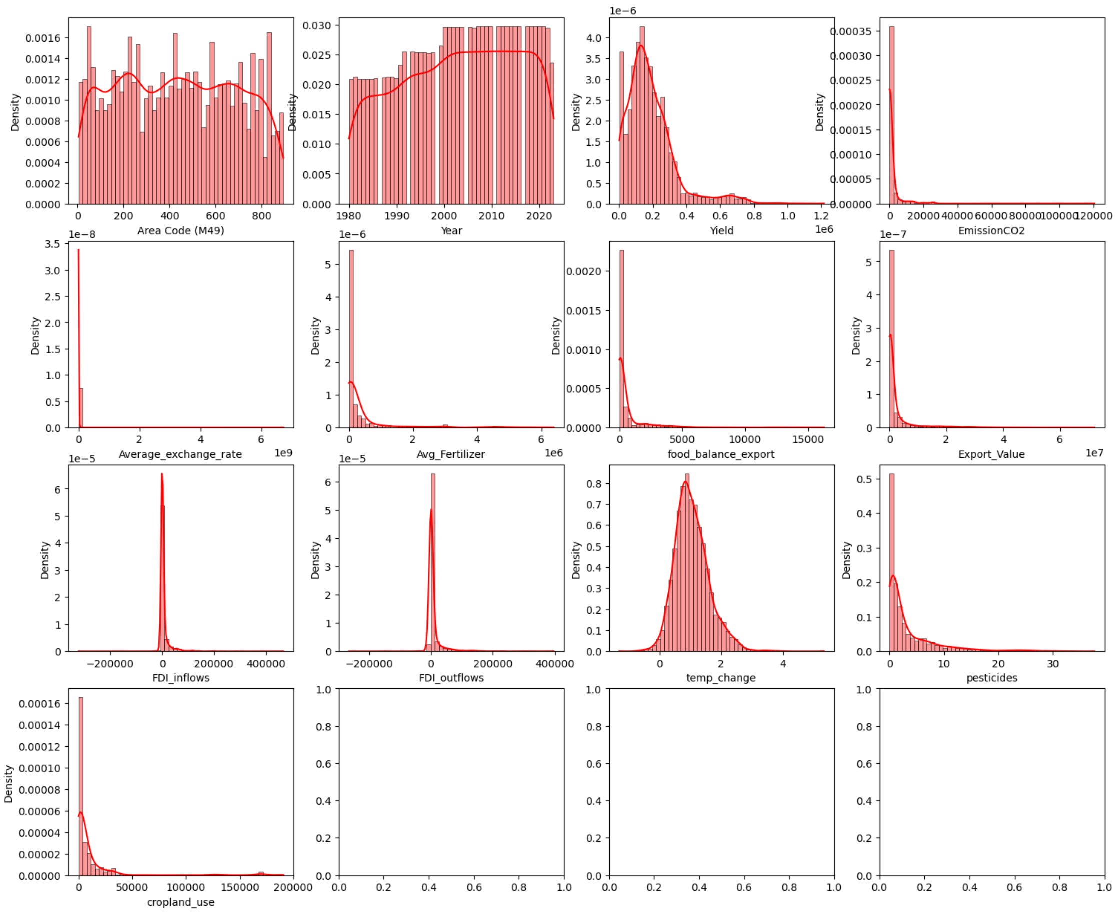
```
2    1.3650
3    0.5870
4    1.3732
Name: temp_change, dtype: float64
```

[58]:
```
df_filtered_nan_remove.head(5)
```

[58]:
```
   Area Code (M49)  Year         Yield  EmissionCO2  Average_exchange_rate  \
0                4  2000  64099.333333          0.0           47357.574730
1                4  2001  64704.666667          0.0           47500.014520
2                4  2002  66451.333333          0.0            3981.907750
3                4  2003  52071.750000          0.0              48.762754
4                4  2004  91307.000000          0.0              47.845312

   Avg_Fertilizer  food_balance_export  Export_Value  FDI_inflows  \
0   342967.054575           590.871292       31080.0         0.17
1   342967.054575           590.871292       27110.0         0.68
2   342967.054575           590.871292       31153.0        50.00
3   342967.054575           590.871292       47612.0        57.80
4   342967.054575           590.871292       48633.0       186.90

   FDI_outflows  temp_change  pesticides  cropland_use
0   8826.181075       0.9930    3.410599    9080.03995
1   8826.181075       1.3110    3.410599    9080.03995
2   8826.181075       1.3650    3.410599    9080.03995
3      1.000000       0.5870    3.410599    9080.03995
4     -0.700000       1.3732    3.410599    9080.03995
```

[59]:
```
#Reference-data science research method module

n_bins = 50
histplot_hyperparams = {
    'kde':True,
    'alpha':0.4,
    'stat': 'density',
    'bins':n_bins
}
cols=['Area Code (M49)', 'Year', 'Yield', 'EmissionCO2',␣
 ↪'Average_exchange_rate',
       'Avg_Fertilizer', 'food_balance_export', 'Export_Value',
       'FDI_inflows', 'FDI_outflows', 'temp_change', 'pesticides',
       'cropland_use']
fig, ax = plt.subplots(4,4, figsize=(18, 15))
ax = ax.flatten()

for i, column in enumerate(cols):
    sns.histplot(
```

```
        merge_df[column], label='Train',
        ax=ax[i], color='red', **histplot_hyperparams
    )
```



[ ]:

[60]:
```
merge_df.fillna(0, inplace = True)
```

[61]:
```
merge_df.isnull().sum()
```

[61]:
```
Area Code (M49)          0
Area                     0
Year                     0
Yield                    0
EmissionCO2              0
Average_exchange_rate    0
Avg_Fertilizer           0
food_balance_export      0
```

```
Export_Value            0
FDI_inflows             0
FDI_outflows            0
temp_change             0
pesticides              0
cropland_use            0
dtype: int64
```

[62]:

```python
merge_df.drop(columns = ['Area'], inplace = True)
```

[63]:

```python
##reference-https://www.geeksforgeeks.org/z-score-for-outlier-detection-python/
import numpy as np

def out_zscore(data):
    outliers = []
    zscore = []
    threshold = 3
    mean = np.mean(data)
    std = np.std(data)
    for i in data:
        z_score = (i - mean) / std
        zscore.append(z_score)
        if np.abs(z_score) > threshold:
            outliers.append(i)
    print("Total number of outliers are", len(outliers))
    return outliers

columns = merge_df.columns
filtered_df = merge_df.copy()  # Make a copy to preserve the original DataFrame

for col in columns:
    outliers = out_zscore(df_filtered_nan_remove[col])
    filtered_df = filtered_df[~filtered_df[col].isin(outliers)] # new dataframe␣
↪after outlier

print("Shape of DataFrame after removing outliers:", filtered_df.shape)
```

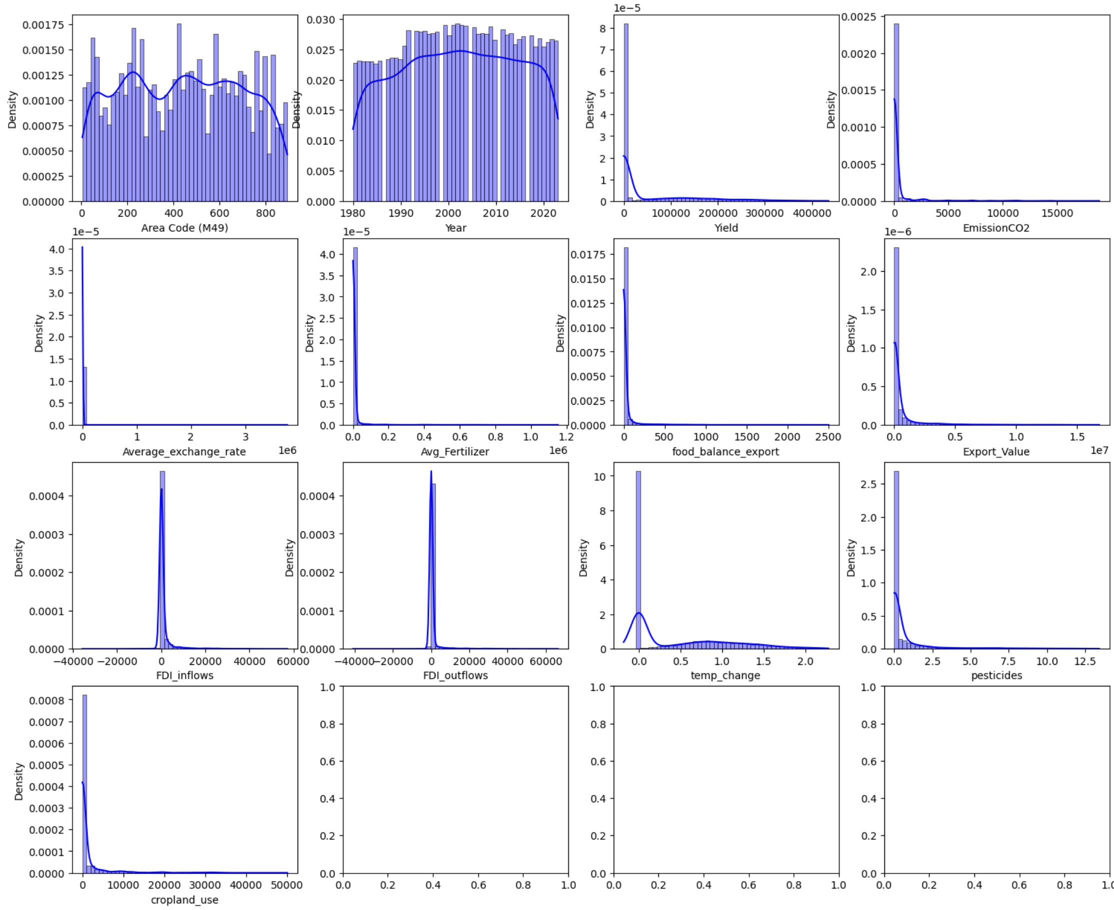
```
Total number of outliers are 0
Total number of outliers are 0
Total number of outliers are 197
Total number of outliers are 88
Total number of outliers are 1
Total number of outliers are 70
Total number of outliers are 133
Total number of outliers are 253
Total number of outliers are 141
Total number of outliers are 129
Total number of outliers are 177
```

```
Total number of outliers are 223
Total number of outliers are 76
Shape of DataFrame after removing outliers: (8723, 13)
```

[64]:
```python
merge_df.columns
```

[64]:
```
Index(['Area Code (M49)', 'Year', 'Yield', 'EmissionCO2',
       'Average_exchange_rate', 'Avg_Fertilizer', 'food_balance_export',
       'Export_Value', 'FDI_inflows', 'FDI_outflows', 'temp_change',
       'pesticides', 'cropland_use'],
      dtype='object')
```

[65]:
```python
n_bins = 50
histplot_hyperparams = {
    'kde':True,
    'alpha':0.4,
    'stat': 'density',
    'bins':n_bins
}
cols=['Area Code (M49)', 'Year', 'Yield', 'EmissionCO2',␣
 ↪'Average_exchange_rate',
      'Avg_Fertilizer', 'food_balance_export', 'Export_Value',
      'FDI_inflows', 'FDI_outflows', 'temp_change', 'pesticides',
      'cropland_use']
fig, ax = plt.subplots(4,4, figsize=(18, 15))
ax = ax.flatten()

for i, column in enumerate(cols):
    sns.histplot(
        filtered_df[column], label='Train',
        ax=ax[i], color='blue', **histplot_hyperparams
    )
```



[66]:
```
for col in filtered_df.columns:
    if filtered_df[col].dtype == 'float64':
        filtered_df[col] = filtered_df[col].map(lambda i: np.log(i) if i > 0␣
↪else 0)
```

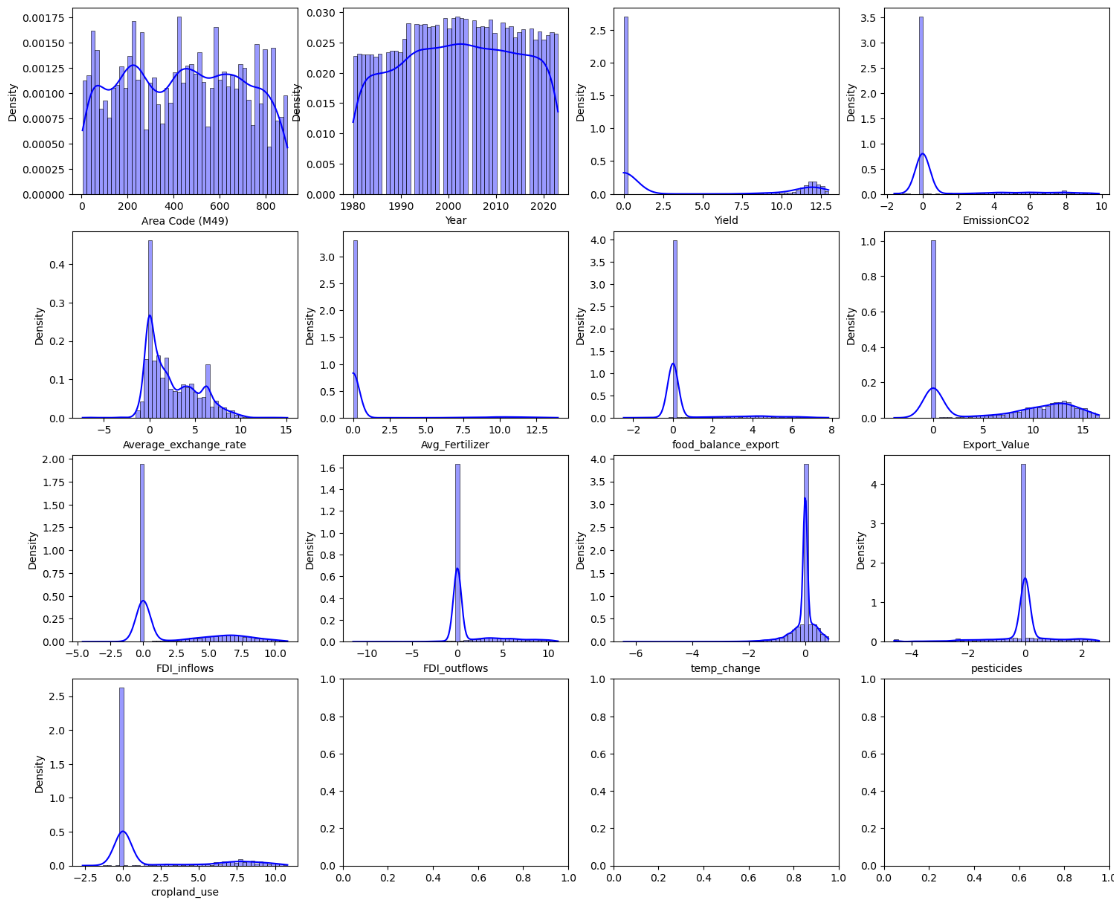
[67]:
```
skewness_values = filtered_df.skew()

print("Skewness values for each column:")
print(skewness_values)
```

```
Skewness values for each column:
Area Code (M49)           0.010590
Year                     -0.046583
Yield                     0.918584
EmissionCO2               2.075457
Average_exchange_rate     0.762560
Avg_Fertilizer            3.455697
food_balance_export       2.388590
Export_Value              0.011696
```

```
FDI_inflows              0.887779
FDI_outflows             1.862131
temp_change             -3.253943
pesticides              -1.209815
cropland_use             1.237673
dtype: float64
```

[68]:

```python
# Looking for Distribution of cols
#reference-https://stackoverflow.com/questions/53646710/
↪plot-a-histogram-through-all-the-columns
n_bins = 50
histplot_hyperparams = {
    'kde':True,
    'alpha':0.4,
    'stat': 'density',
    'bins':n_bins
}
cols=['Area Code (M49)', 'Year', 'Yield', 'EmissionCO2',␣
↪'Average_exchange_rate',
       'Avg_Fertilizer', 'food_balance_export', 'Export_Value',
       'FDI_inflows', 'FDI_outflows', 'temp_change', 'pesticides',
       'cropland_use']
fig, ax = plt.subplots(4,4, figsize=(18, 15))
ax = ax.flatten()

for i, column in enumerate(cols):
    sns.histplot(
        filtered_df[column], label='Train',
        ax=ax[i], color='blue', **histplot_hyperparams
    )
```



[69]:
```
filtered_df.shape
```

[69]: (8723, 13)

[70]:
```
filtered_df.head(10)
```

[70]:

	Area Code (M49)	Year	Yield	EmissionCO2	Average_exchange_rate	\
0	4	2000	11.068189	0.0	10.765482	
1	4	2001	11.077589	0.0	10.768485	
2	4	2002	11.104225	0.0	8.289516	
3	4	2003	10.860378	0.0	3.886967	
4	4	2004	11.421983	0.0	3.867973	
5	4	2005	11.283710	0.0	3.901864	
6	4	2006	11.341333	0.0	3.910529	
7	4	2007	11.347774	0.0	3.911263	
8	4	2008	11.255152	0.0	3.917003	
9	4	2009	12.012929	0.0	3.918502	

Avg_Fertilizer food_balance_export Export_Value FDI_inflows \

```
0              0.0                    0.0     10.344320    -1.771957
1              0.0                    0.0     10.207658    -0.385662
2              0.0                    0.0     10.346666     3.912023
3              0.0                    0.0     10.770840     4.056989
4              0.0                    0.0     10.792058     5.230574
5              0.0                    0.0     11.026955     5.602119
6              0.0                    0.0     10.939071     5.472271
7              0.0                    0.0     11.731796     5.240105
8              0.0                    0.0     11.991213     3.829375
9              0.0                    0.0     12.383637     5.285803

   FDI_outflows  temp_change  pesticides  cropland_use
0      0.000000    -0.007025         0.0      0.000000
1      0.000000     0.270790         0.0      0.000000
2      0.000000     0.311154         0.0      0.000000
3      0.000000    -0.532730         0.0      0.000000
4      0.000000     0.317144         0.0      0.000000
5      0.405465    -0.912797         0.0      0.000000
6      0.000000     0.542208         0.0      8.975883
7      0.000000    -0.392450         0.0      0.000000
8      0.000000    -0.350409         0.0      0.000000
9     -1.093747    -0.111155         0.0      0.000000
```

[71]:

```
filtered_df.isnull().sum()
```

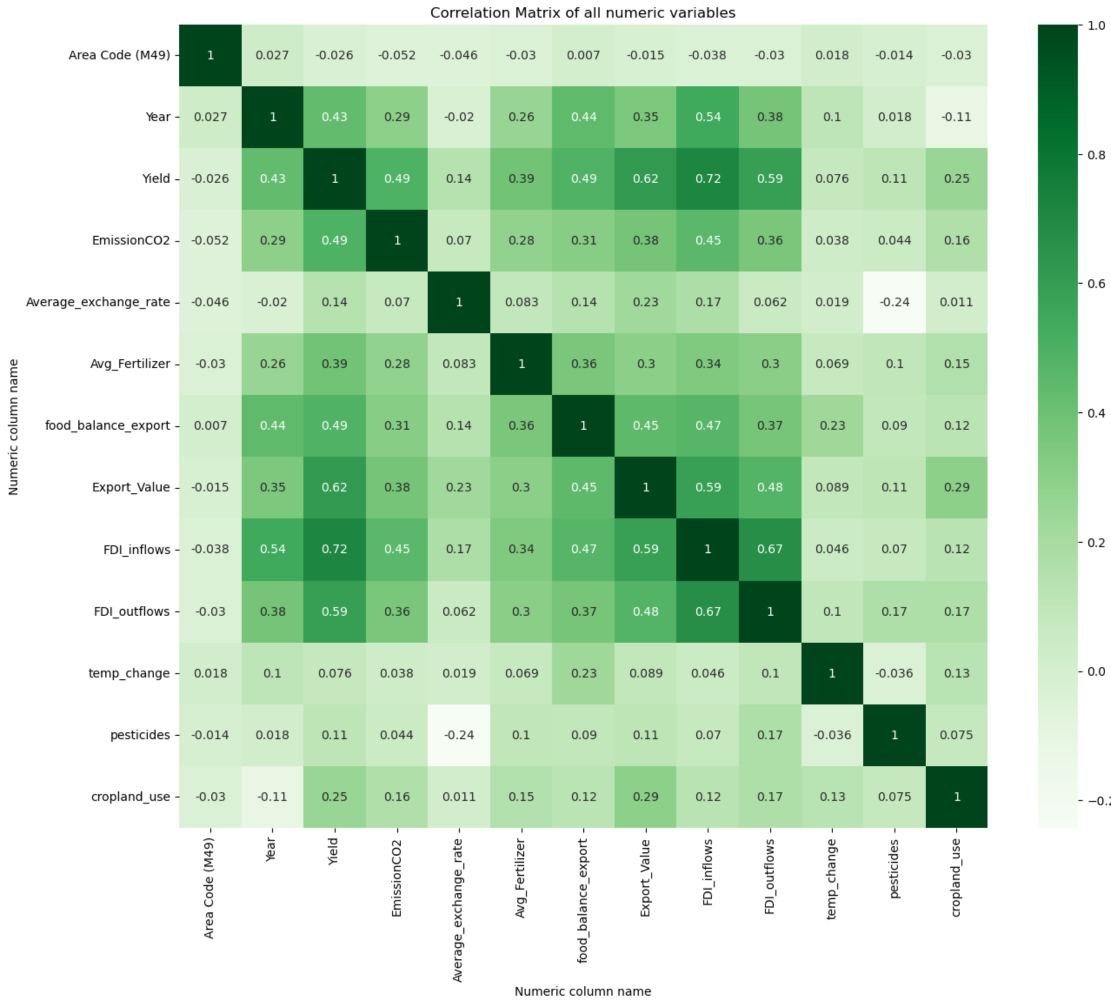
[71]:

```
Area Code (M49)          0
Year                     0
Yield                    0
EmissionCO2              0
Average_exchange_rate    0
Avg_Fertilizer           0
food_balance_export      0
Export_Value             0
FDI_inflows              0
FDI_outflows             0
temp_change              0
pesticides               0
cropland_use             0
dtype: int64
```

[72]:

```
correlationMatrix1= filtered_df.corr(method='spearman')
plt.figure(figsize=(15,12))
plt.title('Correlation Matrix of all numeric variables')
sns.heatmap(correlationMatrix1, cmap="Greens",annot=True)
plt.xlabel('Numeric column name')
plt.ylabel('Numeric column name')
plt.plot()
```

[72]: []



[73]:
```
correlationMatrix1
```

[73]:

	Area Code (M49)	Year	Yield	EmissionCO2	\
Area Code (M49)	1.000000	0.026879	-0.025867	-0.052394	
Year	0.026879	1.000000	0.431069	0.292164	
Yield	-0.025867	0.431069	1.000000	0.493283	
EmissionCO2	-0.052394	0.292164	0.493283	1.000000	
Average_exchange_rate	-0.045788	-0.019563	0.137864	0.070377	
Avg_Fertilizer	-0.029774	0.255120	0.386371	0.278004	
food_balance_export	0.007003	0.441290	0.486658	0.313410	
Export_Value	-0.014697	0.350949	0.615184	0.377138	
FDI_inflows	-0.037934	0.540780	0.717974	0.452287	
FDI_outflows	-0.029848	0.377733	0.590438	0.355239	
temp_change	0.017995	0.103418	0.076118	0.037740	

```
pesticides               -0.014309  0.018133  0.108726     0.044089
cropland_use             -0.029794 -0.106979  0.249516     0.164482

                      Average_exchange_rate  Avg_Fertilizer  \
Area Code (M49)                   -0.045788       -0.029774
Year                              -0.019563        0.255120
Yield                              0.137864        0.386371
EmissionCO2                        0.070377        0.278004
Average_exchange_rate              1.000000        0.082728
Avg_Fertilizer                     0.082728        1.000000
food_balance_export                0.138897        0.358153
Export_Value                       0.227844        0.299250
FDI_inflows                        0.172026        0.336883
FDI_outflows                       0.062007        0.303758
temp_change                        0.019409        0.068959
pesticides                        -0.242511        0.100244
cropland_use                       0.011004        0.151758

                      food_balance_export  Export_Value  FDI_inflows  \
Area Code (M49)                  0.007003     -0.014697    -0.037934
Year                             0.441290      0.350949     0.540780
Yield                            0.486658      0.615184     0.717974
EmissionCO2                      0.313410      0.377138     0.452287
Average_exchange_rate            0.138897      0.227844     0.172026
Avg_Fertilizer                   0.358153      0.299250     0.336883
food_balance_export              1.000000      0.446344     0.468340
Export_Value                     0.446344      1.000000     0.586791
FDI_inflows                      0.468340      0.586791     1.000000
FDI_outflows                     0.371432      0.478181     0.674103
temp_change                      0.229393      0.088673     0.045569
pesticides                       0.089756      0.111019     0.069889
cropland_use                     0.119457      0.293928     0.122512

                      FDI_outflows  temp_change  pesticides  cropland_use
Area Code (M49)          -0.029848     0.017995   -0.014309     -0.029794
Year                      0.377733     0.103418    0.018133     -0.106979
Yield                     0.590438     0.076118    0.108726      0.249516
EmissionCO2               0.355239     0.037740    0.044089      0.164482
Average_exchange_rate     0.062007     0.019409   -0.242511      0.011004
Avg_Fertilizer            0.303758     0.068959    0.100244      0.151758
food_balance_export       0.371432     0.229393    0.089756      0.119457
Export_Value              0.478181     0.088673    0.111019      0.293928
FDI_inflows               0.674103     0.045569    0.069889      0.122512
FDI_outflows              1.000000     0.099542    0.165760      0.168356
temp_change               0.099542     1.000000   -0.035626      0.128484
pesticides                0.165760    -0.035626    1.000000      0.075121
cropland_use              0.168356     0.128484    0.075121      1.000000
```

[74]:
```python
#filtered_df['Year'].unique()
```

1.1.1 Train-test split

[75]:
```python
# Define test years
testyears = [2020, 2021, 2022]

test_df = filtered_df[filtered_df['Year'].isin(testyears)]
```

[76]:
```python
 # Define test years
trainyears = [2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,␣
 ↪2011, 2012, 2013,2014,2015, 2016, 2017, 2018,2019]

# Filter the DataFrame for training and testing
train_df = filtered_df[filtered_df['Year'].isin(trainyears)]
```

[77]:
```python
train_df.shape
```

[77]: (4151, 13)

[78]:
```python
test_df.shape
```

[78]: (588, 13)

[79]:
```python
#from sklearn.preprocessing import RobustScaler
feature = train_df.drop(columns=['Export_Value'])
target = train_df['Export_Value']

# Display the shapes of the training set
print("Training set shape (feature, target):", feature.shape, target.shape)
```

```
Training set shape (feature, target): (4151, 12) (4151,)
```

[80]:
```python
from sklearn.model_selection import train_test_split

X_train_val, X_test, y_train_val, y_test = train_test_split(feature, target,␣
 ↪test_size=0.2, random_state=42)

# Split the training and validation data into 75% training and 25% validation
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,␣
 ↪test_size=0.25, random_state=42)

# Display the shapes of the training, validation, and testing sets
print("Training set shape (X_train, y_train):", X_train.shape, y_train.shape)
print("Validation set shape (X_val, y_val):", X_val.shape, y_val.shape)
print("Testing set shape (X_test, y_test):", X_test.shape, y_test.shape)
```

```
Training set shape (X_train, y_train): (2490, 12) (2490,)
```

```
Validation set shape (X_val, y_val): (830, 12) (830,)
Testing set shape (X_test, y_test): (831, 12) (831,)
```

[81]:
```
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler

scaler = StandardScaler()

# Fit the scaler to the training data and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the validation and testing data using the same scaler
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Display the shapes of the scaled training, validation, and testing sets
print("Scaled Training set shape (X_train_scaled):", X_train_scaled.shape)
print("Scaled Validation set shape (X_val_scaled):", X_val_scaled.shape)
print("Scaled Testing set shape (X_test_scaled):", X_test_scaled.shape)
```

```
Scaled Training set shape (X_train_scaled): (2490, 12)
Scaled Validation set shape (X_val_scaled): (830, 12)
Scaled Testing set shape (X_test_scaled): (831, 12)
```

1.1.2 MLP regressor Model

[82]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import mean_squared_error
from torch.utils.data import DataLoader, TensorDataset
from torch.optim.lr_scheduler import ReduceLROnPlateau
from sklearn.preprocessing import RobustScaler
```

[83]:
```
# Convert NumPy arrays to PyTorch tensors
X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.float32)
X_val_tensor = torch.tensor(X_val_scaled, dtype=torch.float32)
y_val_tensor = torch.tensor(y_val.values, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)
```

[84]:
```
# Create DataLoader for training data
#reference-https://machinelearningmastery.com/
 ↪building-a-regression-model-in-pytorch/
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=256, shuffle=True)
```

```python
# Define the neural network model
class MLPRegressor(nn.Module):
    def __init__(self, input_size):
        super(MLPRegressor, self).__init__()
        self.fc1 = nn.Linear(input_size, 50)
        self.dropout1 = nn.Dropout(0.2)
        self.fc2 = nn.Linear(50, 100)
        self.dropout2 = nn.Dropout(0.2)
        self.fc3 = nn.Linear(100, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.dropout1(x)
        x = torch.relu(self.fc2(x))
        x = self.dropout2(x)
        x = self.fc3(x)
        return x

# Initialize the model
model = MLPRegressor(input_size=X_train_scaled.shape[1])

# Define the loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Learning rate reduction scheduler
scheduler = ReduceLROnPlateau(optimizer, 'min', factor=0.2, patience=5,
    min_lr=0.001)

train_losses = []
val_losses = []

# Training loop
for epoch in range(800):
    model.train()
    train_loss = 0.0
    for inputs, targets in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets.unsqueeze(1))
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * inputs.size(0)
    train_loss /= len(train_loader.dataset)
    train_losses.append(train_loss)

    # Evaluate on validation set
```

```
    model.eval()
    with torch.no_grad():
        y_val_pred = model(X_val_tensor)
        mse_val = criterion(y_val_pred, y_val_tensor.unsqueeze(1))
        scheduler.step(mse_val)
    val_losses.append(mse_val.item())
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/800], Train Loss: {train_loss:.4f}, Validation␣
 ↪Loss: {mse_val.item():.4f}')

# Predict on the test set
model.eval()
with torch.no_grad():
    y_test_pred = model(X_test_tensor)

# Convert predictions and targets to NumPy arrays
y_test_pred_np = y_test_pred.numpy().flatten()
y_test_np = y_test.values.flatten()

# Evaluate the model using Mean Squared Error (MSE) on the test set
mse_test = mean_squared_error(y_test_np, y_test_pred_np)
print("Mean Squared Error on Test Set:", mse_test)
```

```
Epoch [10/800], Train Loss: 12.7751, Validation Loss: 11.1409
Epoch [20/800], Train Loss: 9.5254, Validation Loss: 8.3803
Epoch [30/800], Train Loss: 8.2229, Validation Loss: 7.1409
Epoch [40/800], Train Loss: 7.8038, Validation Loss: 6.4891
Epoch [50/800], Train Loss: 7.2547, Validation Loss: 6.0352
Epoch [60/800], Train Loss: 6.7579, Validation Loss: 5.6640
Epoch [70/800], Train Loss: 6.5118, Validation Loss: 5.4320
Epoch [80/800], Train Loss: 6.3932, Validation Loss: 5.1981
Epoch [90/800], Train Loss: 5.9601, Validation Loss: 5.0030
Epoch [100/800], Train Loss: 5.8074, Validation Loss: 4.8558
Epoch [110/800], Train Loss: 5.6744, Validation Loss: 4.6947
Epoch [120/800], Train Loss: 5.4629, Validation Loss: 4.5449
Epoch [130/800], Train Loss: 5.4674, Validation Loss: 4.3929
Epoch [140/800], Train Loss: 5.2589, Validation Loss: 4.3568
Epoch [150/800], Train Loss: 5.0369, Validation Loss: 4.2103
Epoch [160/800], Train Loss: 5.0600, Validation Loss: 4.1949
Epoch [170/800], Train Loss: 4.7938, Validation Loss: 4.0928
Epoch [180/800], Train Loss: 4.7270, Validation Loss: 4.0067
Epoch [190/800], Train Loss: 4.6306, Validation Loss: 3.9867
Epoch [200/800], Train Loss: 4.5885, Validation Loss: 3.8745
Epoch [210/800], Train Loss: 4.5127, Validation Loss: 3.7827
Epoch [220/800], Train Loss: 4.4807, Validation Loss: 3.7624
Epoch [230/800], Train Loss: 4.4448, Validation Loss: 3.7014
Epoch [240/800], Train Loss: 4.3183, Validation Loss: 3.6770
Epoch [250/800], Train Loss: 4.1035, Validation Loss: 3.6697
```

```
Epoch [260/800], Train Loss: 4.1572, Validation Loss: 3.6650
Epoch [270/800], Train Loss: 4.0782, Validation Loss: 3.6147
Epoch [280/800], Train Loss: 4.0512, Validation Loss: 3.5864
Epoch [290/800], Train Loss: 4.0008, Validation Loss: 3.6221
Epoch [300/800], Train Loss: 3.8104, Validation Loss: 3.4870
Epoch [310/800], Train Loss: 3.8364, Validation Loss: 3.4754
Epoch [320/800], Train Loss: 3.9548, Validation Loss: 3.4611
Epoch [330/800], Train Loss: 4.0423, Validation Loss: 3.5935
Epoch [340/800], Train Loss: 3.6956, Validation Loss: 3.4088
Epoch [350/800], Train Loss: 3.7310, Validation Loss: 3.3690
Epoch [360/800], Train Loss: 3.7727, Validation Loss: 3.3343
Epoch [370/800], Train Loss: 3.6542, Validation Loss: 3.3173
Epoch [380/800], Train Loss: 3.6531, Validation Loss: 3.4186
Epoch [390/800], Train Loss: 3.4440, Validation Loss: 3.3607
Epoch [400/800], Train Loss: 3.4870, Validation Loss: 3.3760
Epoch [410/800], Train Loss: 3.5841, Validation Loss: 3.3044
Epoch [420/800], Train Loss: 3.5631, Validation Loss: 3.3778
Epoch [430/800], Train Loss: 3.4165, Validation Loss: 3.2487
Epoch [440/800], Train Loss: 3.4862, Validation Loss: 3.3465
Epoch [450/800], Train Loss: 3.4619, Validation Loss: 3.2653
Epoch [460/800], Train Loss: 3.4283, Validation Loss: 3.1670
Epoch [470/800], Train Loss: 3.3921, Validation Loss: 3.2313
Epoch [480/800], Train Loss: 3.3567, Validation Loss: 3.2538
Epoch [490/800], Train Loss: 3.3109, Validation Loss: 3.2416
Epoch [500/800], Train Loss: 3.3045, Validation Loss: 3.3106
Epoch [510/800], Train Loss: 3.2562, Validation Loss: 3.3782
Epoch [520/800], Train Loss: 3.1436, Validation Loss: 3.2661
Epoch [530/800], Train Loss: 3.2481, Validation Loss: 3.3063
Epoch [540/800], Train Loss: 3.0221, Validation Loss: 3.2979
Epoch [550/800], Train Loss: 3.1551, Validation Loss: 3.1287
Epoch [560/800], Train Loss: 3.1196, Validation Loss: 3.1497
Epoch [570/800], Train Loss: 3.0519, Validation Loss: 3.2616
Epoch [580/800], Train Loss: 3.1032, Validation Loss: 3.3073
Epoch [590/800], Train Loss: 3.1564, Validation Loss: 3.1250
Epoch [600/800], Train Loss: 3.1792, Validation Loss: 3.1370
Epoch [610/800], Train Loss: 3.0882, Validation Loss: 3.1586
Epoch [620/800], Train Loss: 2.9839, Validation Loss: 3.1243
Epoch [630/800], Train Loss: 2.9902, Validation Loss: 3.1811
Epoch [640/800], Train Loss: 3.1088, Validation Loss: 3.1529
Epoch [650/800], Train Loss: 3.0189, Validation Loss: 3.2395
Epoch [660/800], Train Loss: 2.9407, Validation Loss: 3.1237
Epoch [670/800], Train Loss: 2.9885, Validation Loss: 3.2692
Epoch [680/800], Train Loss: 2.9086, Validation Loss: 3.2181
Epoch [690/800], Train Loss: 3.0526, Validation Loss: 3.1882
Epoch [700/800], Train Loss: 2.8672, Validation Loss: 3.0603
Epoch [710/800], Train Loss: 2.8603, Validation Loss: 3.1187
Epoch [720/800], Train Loss: 2.8540, Validation Loss: 3.1758
Epoch [730/800], Train Loss: 2.9763, Validation Loss: 3.1709
```

```
Epoch [740/800], Train Loss: 2.7905, Validation Loss: 3.1594
Epoch [750/800], Train Loss: 2.8831, Validation Loss: 3.2220
Epoch [760/800], Train Loss: 2.8253, Validation Loss: 3.1749
Epoch [770/800], Train Loss: 2.7217, Validation Loss: 3.2505
Epoch [780/800], Train Loss: 2.8567, Validation Loss: 3.2253
Epoch [790/800], Train Loss: 2.7167, Validation Loss: 3.2750
Epoch [800/800], Train Loss: 2.7912, Validation Loss: 3.1191
Mean Squared Error on Test Set: 2.856959493841862
```

[ ]:

```

```

[85]:

```python
from sklearn.metrics import r2_score

# Calculate R-squared for the predictions
r2 = r2_score(y_test_np, y_test_pred_np)
print(f'R-squared: {r2}')
```

```
R-squared: 0.9069877498164576
```

[86]:

```python
from sklearn.metrics import mean_absolute_error

# Assuming y_true and y_pred are your true and predicted target values␣
↪respectively
mae = mean_absolute_error(y_test_np, y_test_pred_np)

print("Mean Absolute Error:", mae)
```

```
Mean Absolute Error: 1.099837110535382
```

1.1.3 Testing

[87]:

```python
test_df_with_predictions = X_test.copy()
test_df_with_predictions['Predicted Export Value (USD)'] = y_test_pred
test_df_with_predictions['Actual Export Value (USD)'] = y_test

test_df_with_predictions = train_df[['Year', 'Area Code (M49)']].
↪merge(test_df_with_predictions, left_index=True, right_index=True)
```

[88]:

```python
test_df_with_predictions.head()
```

[88]:

```
    Year_x  Area Code (M49)_x  Area Code (M49)_y  Year_y      Yield  \
6     2006                  4                  4    2006  11.341333
8     2008                  4                  4    2008  11.255152
12    2012                  4                  4    2012  11.652110
14    2014                  4                  4    2014  11.704245
17    2017                  4                  4    2017  11.911438

    EmissionCO2  Average_exchange_rate  Avg_Fertilizer  food_balance_export  \
```

```
6          0.0              3.910529           0.0            0.000000
8          0.0              3.917003           0.0            0.000000
12         0.0              3.930283           0.0            3.091042
14         0.0              4.047384           0.0            3.623128
17         0.0              4.219903           0.0            3.952845

    FDI_inflows  FDI_outflows  temp_change  pesticides  cropland_use  \
6      5.472271      0.000000     0.542208         0.0      8.975883
8      3.829375      0.000000    -0.350409         0.0      0.000000
12     3.710248      0.000000    -1.499687         0.0      0.000000
14     3.760625      0.000000    -0.785701         0.0      8.975883
17     3.942240      2.421322     0.431912         0.0      8.975883

    Predicted Export Value (USD)  Actual Export Value (USD)
6                      11.226238                  10.939071
8                      10.359215                  11.991213
12                     11.350175                  11.774720
14                     11.924175                  12.566183
17                     12.145748                  13.115620
```

[89]:

```python
# Extracting specific columns
selected_columns = test_df_with_predictions[['Year_x', 'Area Code (M49)_x',␣
↪'Predicted Export Value (USD)', 'Actual Export Value (USD)']]

# Displaying the selected columns
selected_columns.head()
```

[89]:

```
    Year_x  Area Code (M49)_x  Predicted Export Value (USD)  \
6     2006                  4                     11.226238
8     2008                  4                     10.359215
12    2012                  4                     11.350175
14    2014                  4                     11.924175
17    2017                  4                     12.145748

    Actual Export Value (USD)
6                   10.939071
8                   11.991213
12                  11.774720
14                  12.566183
17                  13.115620
```

[90]:

```python
# # Save selected columns to a CSV file
# selected_columns.to_csv('selected_columns.csv', index=False)
```

[91]:

```python
test_df[(test_df['Year']==2021) & (test_df['Area Code (M49)']==4)]
```

[91]:
```
    Area Code (M49)  Year      Yield  EmissionCO2  Average_exchange_rate  \
21                4  2021  11.849886          0.0               4.348242

    Avg_Fertilizer  food_balance_export  Export_Value  FDI_inflows  \
21             0.0             4.820953     13.494983     3.025338

    FDI_outflows  temp_change  pesticides  cropland_use
21      3.427221     0.283071         0.0           0.0
```

1.1.4 Test on unseen data (test_df)

[92]:
```python
# Prepare the test_df by dropping the 'Export Value (USD)' column
X_test_df = test_df.drop(columns=['Export_Value'])
```

[93]:
```python
X_test_df_scaled = scaler.transform(X_test_df)
```

[94]:
```python
X_test_df_scaled = torch.tensor(X_test_df_scaled, dtype=torch.float32)
```

[95]:
```python
model.eval()
with torch.no_grad():
    test_predictions = model(X_test_df_scaled)
```

[ ]:

[ ]:

[ ]:

[ ]:

[96]:
```python
test_df_with_predictions1 = X_test_df.copy()
test_df_with_predictions1['Predicted Export Value (USD)'] = test_predictions
test_df_with_predictions1['Actual Export Value (USD)'] =
 ↪test_df['Export_Value'].values
```

[97]:
```python
test_df_with_predictions1.head(10)
```

[97]:
```
    Area Code (M49)  Year      Yield  EmissionCO2  Average_exchange_rate  \
20                4  2020  11.718623     0.000000               4.341381
21                4  2021  11.849886     0.000000               4.348242
22                4  2022  12.294566     0.000000               0.000000
43                8  2020  12.316384     3.966805               4.688132
44                8  2021  12.322925     3.966805               4.639765
45                8  2022  12.352476     0.000000               4.727756
66               12  2020  11.849028     0.000000               4.842428
88               24  2020  11.172466     5.062986               6.360021
89               24  2021  11.168471     5.062986               6.448006
90               24  2022  11.202334     0.000000               6.132459
```

```
    Avg_Fertilizer  food_balance_export  FDI_inflows  FDI_outflows  \
20        0.000000             4.314149     2.562650      3.617074
21        0.000000             4.820953     3.025338      3.427221
22        0.000000             0.000000     0.000000      0.000000
43        9.729149             2.914763     6.975284      4.471942
44        0.000000             2.811591     7.111188      4.144521
45        0.000000             0.000000     7.268311      5.095437
66        0.000000             4.284735     7.041097      2.685819
88        0.000000             1.446919     0.000000      4.505510
89        0.000000             1.452252     0.000000      0.000000
90        0.000000             0.000000     0.000000      3.715189

    temp_change  pesticides  cropland_use  Predicted Export Value (USD)  \
20    -0.697959     0.00000      0.000000                     12.092794
21     0.283071     0.00000      0.000000                     12.685706
22     0.699229     0.00000      0.000000                      3.866697
43     0.403997     0.09531      6.533142                     11.465244
44     0.429051     0.09531      6.533105                     11.063963
45     0.417130     0.00000      0.000000                     11.013435
66     0.655653    -0.34249      0.000000                     12.528455
88     0.150487    -4.60517      0.000000                     10.039722
89     0.440317    -4.60517      0.000000                      9.054209
90     0.191942     0.00000      0.000000                      9.816685

    Actual Export Value (USD)
20                  13.453783
21                  13.494983
22                  13.202831
43                  11.644769
44                  11.641084
45                  12.004654
66                  13.117803
88                   9.904791
89                   9.710446
90                  10.199824
```

[98]:
```
selected_columns1=test_df_with_predictions1[['Area Code␣
↪(M49)','Year','Predicted Export Value (USD)','Actual Export Value (USD)']]
```

[99]:
```
for col in selected_columns1.columns:
    if col not in ['Year', 'Area Code (M49)']:  #excluding categorical cols
        selected_columns1[col] = np.exp(selected_columns1[col])
```

[112]:
```
selected_columns1.head(50)
```

[112]:

	Area Code (M49)	Year	Predicted Export Value (USD)	\
20	4	2020	1.785804e+05	
21	4	2021	3.230964e+05	
22	4	2022	4.778431e+01	
43	8	2020	9.534377e+04	
44	8	2021	6.382900e+04	
45	8	2022	6.068400e+04	
66	12	2020	2.760824e+05	
88	24	2020	2.291902e+04	
89	24	2021	8.554466e+03	
90	24	2022	1.833716e+04	
91	28	2020	1.912007e+04	
92	28	2021	2.831741e+04	
93	28	2022	2.122155e+04	
114	31	2020	2.939306e+05	
115	31	2021	3.284794e+05	
116	31	2022	1.713806e+03	
160	36	2020	5.215681e+06	
184	40	2021	1.889546e+06	
202	48	2020	4.545663e+04	
226	50	2022	5.018307e+04	
247	51	2020	7.440373e+04	
248	51	2021	6.804184e+04	
249	51	2022	4.492889e+04	
271	52	2022	1.330366e+04	
314	64	2020	1.476587e+04	
315	64	2021	1.751413e+04	
316	64	2022	1.584961e+04	
337	68	2020	3.591897e+05	
338	68	2021	4.094098e+05	
339	68	2022	1.554462e+03	
359	70	2020	1.121529e+05	
360	70	2021	1.053429e+05	
361	70	2022	1.577124e+04	
374	72	2020	4.238220e+04	
419	84	2020	1.837403e+05	
420	84	2021	2.290540e+05	
421	84	2022	2.188338e+04	
469	100	2020	1.138985e+06	
470	100	2021	1.149586e+06	
471	100	2022	8.139400e+03	
492	104	2020	1.562962e+06	
493	104	2021	1.065905e+06	
536	112	2022	1.273912e+04	
566	120	2020	8.320879e+04	
607	132	2020	2.692961e+03	
608	132	2021	4.126063e+03	

609	132	2022	2.360905e+04
617	140	2020	8.115875e+01
618	140	2021	2.561225e+01
619	140	2022	9.442923e+03

	Actual Export Value (USD)
20	696471.98
21	725765.72
22	541896.88
43	114093.03
44	113673.36
45	163513.95
66	497724.92
88	20026.08
89	16488.96
90	26898.46
91	233.44
92	127.02
93	255.12
114	702462.24
115	751750.30
116	814505.65
160	9818924.08
184	7762181.93
202	243903.98
226	604924.12
247	404271.63
248	442041.24
249	556849.99
271	44216.01
314	20441.88
315	25476.53
316	17405.35
337	713106.42
338	1092259.73
339	1439547.59
359	270211.67
360	288795.53
361	302201.63
374	16405.97
419	145347.59
420	164909.18
421	173711.43
469	3025625.98
470	4136054.89
471	5236547.10
492	3448094.55

```
493                 3526694.94
536                 1659373.02
566                   99638.07
607                     509.91
608                     294.00
609                     440.00
617                     253.88
618                      41.32
619                     134.36
```

[101]:
```
selected_columns1.columns
```

[101]:
```
Index(['Area Code (M49)', 'Year', 'Predicted Export Value (USD)',
       'Actual Export Value (USD)'],
      dtype='object')
```

[102]:
```
fltered_df=filtered_food_trade[['Area','Area Code (M49)']]
fltered_df.head(5)
```

[102]:

	Area	Area Code (M49)
0	Afghanistan	4
1	Afghanistan	4
2	Afghanistan	4
3	Afghanistan	4
4	Afghanistan	4

[103]:
```
merging=pd.merge(fltered_df,selected_columns1,on='Area Code (M49)')
```

[104]:
```
merging.isnull().sum()
```

[104]:
```
Area                            0
Area Code (M49)                 0
Year                            0
Predicted Export Value (USD)    0
Actual Export Value (USD)       0
dtype: int64
```

[105]:
```
merging['Area-Year']=merging['Area']+'-'+merging['Year'].astype(str)
```

[106]:
```
merging.head(10)
```

[106]:

	Area	Area Code (M49)	Year	Predicted Export Value (USD)	\
0	Afghanistan	4	2020	178580.437500	
1	Afghanistan	4	2021	323096.437500	
2	Afghanistan	4	2022	47.784309	
3	Afghanistan	4	2020	178580.437500	
4	Afghanistan	4	2021	323096.437500	
5	Afghanistan	4	2022	47.784309	

```
6  Afghanistan                  4  2020                 178580.437500
7  Afghanistan                  4  2021                 323096.437500
8  Afghanistan                  4  2022                     47.784309
9  Afghanistan                  4  2020                 178580.437500

   Actual Export Value (USD)         Area-Year
0                  696471.98  Afghanistan-2020
1                  725765.72  Afghanistan-2021
2                  541896.88  Afghanistan-2022
3                  696471.98  Afghanistan-2020
4                  725765.72  Afghanistan-2021
5                  541896.88  Afghanistan-2022
6                  696471.98  Afghanistan-2020
7                  725765.72  Afghanistan-2021
8                  541896.88  Afghanistan-2022
9                  696471.98  Afghanistan-2020
```

[107]:
```
merging=merging.drop(columns=['Area','Area Code (M49)','Year'])
```

[108]:
```
merging.head(10)
```

[108]:

	Predicted Export Value (USD)	Actual Export Value (USD)	Area-Year
0	178580.437500	696471.98	Afghanistan-2020
1	323096.437500	725765.72	Afghanistan-2021
2	47.784309	541896.88	Afghanistan-2022
3	178580.437500	696471.98	Afghanistan-2020
4	323096.437500	725765.72	Afghanistan-2021
5	47.784309	541896.88	Afghanistan-2022
6	178580.437500	696471.98	Afghanistan-2020
7	323096.437500	725765.72	Afghanistan-2021
8	47.784309	541896.88	Afghanistan-2022
9	178580.437500	696471.98	Afghanistan-2020

[109]:
```
last=merging.pop('Area-Year')
merging.insert(0,'Area-Year',last)
```

[110]:
```
merging.head()
```

[110]:

	Area-Year	Predicted Export Value (USD)	Actual Export Value (USD)
0	Afghanistan-2020	178580.437500	696471.98
1	Afghanistan-2021	323096.437500	725765.72
2	Afghanistan-2022	47.784309	541896.88
3	Afghanistan-2020	178580.437500	696471.98
4	Afghanistan-2021	323096.437500	725765.72

[111]:
```
merging.to_csv('277244_output.csv',index=False)
```

[ ]:
```

```

```
[ ]:
```