```
from google.colab import drive
drive.mount('/content/drive')

# Import necessary libraries
import pandas as pd

# Load each CSV file into separate DataFrames
consumer_price_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Consumer_prices_indicators_-_FAOSTAT_data_en_2-22-2024[1]
crops_production_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Crops_production_indicators_-_FAOSTAT_data_en_2-22-2024
emissions_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Emissions_-_FAOSTAT_data_en_2-27-2024[1].csv")
employment_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Employment_-_FAOSTAT_data_en_2-27-2024[1].csv")
exchange_rate_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Exchange_rate_-_FAOSTAT_data_en_2-22-2024[1].csv")
fertilizers_use_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Fertilizers_use_-_FAOSTAT_data_en_2-27-2024[1].csv")
food_balances_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Food_balances_indicators_-_FAOSTAT_data_en_2-22-2024[1].cs
food_security_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Food_security_indicators__-_FAOSTAT_data_en_2-22-2024[1].c
food_trade_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Food_trade_indicators_-_FAOSTAT_data_en_2-22-2024[1].csv")
foreign_direct_investment_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Foreign_direct_investment_-_FAOSTAT_data_en_2-
land_temperature_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Land_temperature_change_-_FAOSTAT_data_en_2-27-2024[1].
land_use_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Land_use_-_FAOSTAT_data_en_2-22-2024[1].csv")
pesticides_use_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Pesticides_use_-_FAOSTAT_data_en_2-27-2024[1].csv")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
<ipython-input-214-7a37a39053e8>:19: DtypeWarning: Columns (14) have mixed types. Specify dtype option on import or set low_memory=
  land_use_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Land_use_-_FAOSTAT_data_en_2-22-2024[1].csv")
```

```
# Display the first few rows of the DataFrame
print("\nAnalysis of Consumer Prices Indicators DataFrame:")
print(consumer_price_df.head())

# Display the first few rows of the DataFrame
print("\nAnalysis of Crops Production Indicators DataFrame:")
print(crops_production_df.head())

# Display the first few rows of the DataFrame
print("\nAnalysis of Emissions DataFrame:")
print(emissions_df.head())

# Display the first few rows of the DataFrame
print("\nAnalysis of Employment DataFrame:")
print(employment_df.head())

# Display the first few rows of the DataFrame
print("\nAnalysis of Exchange Rate DataFrame:")
print(exchange_rate_df.head())

# Display the first few rows of the DataFrame
print("\nAnalysis of Fertilizers Use DataFrame:")
print(fertilizers_use_df.head())

# Display the first few rows of the DataFrame
print("\nAnalysis of Food Balances Indicators DataFrame:")
print(food_balances_df.head())

# Display the first few rows of the DataFrame
print("\nAnalysis of Food Security Indicators DataFrame:")
print(food_security_df.head())

# Display the first few rows of the DataFrame
print("\nAnalysis of Food Trade Indicators DataFrame:")
print(food_trade_df.head())

# Display the first few rows of the DataFrame
print("\nAnalysis of Foreign Direct Investment DataFrame:")
print(foreign_direct_investment_df.head())

# Display the first few rows of the DataFrame
print("\nAnalysis of Land Temperature Change DataFrame:")
print(land_temperature_df.head())

# Display the first few rows of the DataFrame
print("\nAnalysis of Land Use DataFrame:")
print(land_use_df.head())

# Display the first few rows of the DataFrame
print("\nAnalysis of Pesticides Use DataFrame:")
print(pesticides_use_df.head())
```

```
   Element Code             Element  Months Code       Months  Year Code  \
0          7271  Temperature change         7016  Dec–Jan–Feb       2000
1          7271  Temperature change         7016  Dec–Jan–Feb       2001
2          7271  Temperature change         7016  Dec–Jan–Feb       2002
3          7271  Temperature change         7016  Dec–Jan–Feb       2003
4          7271  Temperature change         7016  Dec–Jan–Feb       2004

   Year Unit  Value Flag Flag Description
0  2000   °c  0.618    E  Estimated value
1  2001   °c  0.365    E  Estimated value
2  2002   °c  1.655    E  Estimated value
3  2003   °c  0.997    E  Estimated value
4  2004   °c  1.883    E  Estimated value

Analysis of Land Use DataFrame:
  Domain Code    Domain  Area Code (M49)         Area  Element Code Element  \
0          RL  Land Use                4  Afghanistan          5110    Area
1          RL  Land Use                4  Afghanistan          5110    Area
2          RL  Land Use                4  Afghanistan          5110    Area
3          RL  Land Use                4  Afghanistan          5110    Area
4          RL  Land Use                4  Afghanistan          5110    Area

   Item Code          Item  Year Code  Year     Unit    Value Flag  \
0       6600  Country area       1980  1980  1000 ha  65286.0    A
1       6600  Country area       1981  1981  1000 ha  65286.0    A
2       6600  Country area       1982  1982  1000 ha  65286.0    A
3       6600  Country area       1983  1983  1000 ha  65286.0    A
4       6600  Country area       1984  1984  1000 ha  65286.0    A

  Flag Description Note
0  Official figure  NaN
1  Official figure  NaN
2  Official figure  NaN
3  Official figure  NaN
4  Official figure  NaN

Analysis of Pesticides Use DataFrame:
  Domain Code          Domain  Area Code (M49)     Area  Element Code  \
0          RP  Pesticides Use                8  Albania          5157
1          RP  Pesticides Use                8  Albania          5159
2          RP  Pesticides Use                8  Albania          5173
3          RP  Pesticides Use                8  Albania          5157
4          RP  Pesticides Use                8  Albania          5159

                                 Element  Item Code               Item  \
0                        Agricultural Use       1357  Pesticides (total)
1                 Use per area of cropland       1357  Pesticides (total)
2  Use per value of agricultural production       1357  Pesticides (total)
3                        Agricultural Use       1357  Pesticides (total)
4                 Use per area of cropland       1357  Pesticides (total)

   Year Code  Year    Unit   Value Flag Flag Description Note
0       2000  2000       t  307.98    E  Estimated value  NaN
1       2000  2000   kg/ha    0.44    E  Estimated value  NaN
2       2000  2000  g/Int$    0.23    E  Estimated value  NaN
3       2001  2001       t  319.38    E  Estimated value  NaN
4       2001  2001   kg/ha    0.46    E  Estimated value  NaN
```

```
import pandas as pd

# Load the dataset
consumer_price_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Consumer_prices_indicators_-_FAOSTAT_data_en_2-22-2024[1]

# Filter for 'Food price inflation'
cpi_food_inflation = consumer_price_df[consumer_price_df['Item'] == 'Food price inflation']

# Calculate the average annual inflation rate for each country
average_annual_inflation = cpi_food_inflation.groupby(['Year', 'Area'])['Value'].mean().reset_index()
average_annual_inflation.rename(columns={'Value': 'Average_Annual_Inflation'}, inplace=True)

# Calculate the standard deviation (volatility) of inflation rates for each country
inflation_volatility = cpi_food_inflation.groupby('Area')['Value'].std().reset_index()
inflation_volatility.rename(columns={'Value': 'Inflation_Volatility'}, inplace=True)

# Merge the average annual inflation and inflation volatility back into the original dataset
merged_df = pd.merge(consumer_price_df, average_annual_inflation, on=['Year', 'Area'], how='left')
merged_df = pd.merge(merged_df, inflation_volatility, on='Area', how='left')

# Select relevant columns for the new CSV
columns_to_keep = ['Year', 'Area', 'Months', 'Value', 'Average_Annual_Inflation', 'Inflation_Volatility']
consumer_price_new = merged_df[columns_to_keep]

# Save to a new CSV file
consumer_price_new.to_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Enhanced_Consumer_Prices_Indicators.csv", index=False)

# Print the info of the new DataFrame to ensure the columns are correct
print(consumer_price_new.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112890 entries, 0 to 112889
Data columns (total 6 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   Year                     112890 non-null  int64
 1   Area                     112890 non-null  object
 2   Months                   112890 non-null  object
 3   Value                    112890 non-null  float64
 4   Average_Annual_Inflation  110454 non-null  float64
 5   Inflation_Volatility     112890 non-null  float64
dtypes: float64(3), int64(1), object(2)
memory usage: 5.2+ MB
None
```

```
consumer_price_new
```

	Year	Area	Months	Value	Average_Annual_Inflation	Inflation_Volatility
0	2000	Afghanistan	January	24.356332	NaN	11.473983
1	2000	Afghanistan	February	23.636242	NaN	11.473983
2	2000	Afghanistan	March	23.485345	NaN	11.473983
3	2000	Afghanistan	April	24.767194	NaN	11.473983
4	2000	Afghanistan	May	25.956912	NaN	11.473983
...	...	...	...	...	...	...
112885	2023	Zimbabwe	May	116.960656	126.397179	178.365835
112886	2023	Zimbabwe	June	255.596454	126.397179	178.365835
112887	2023	Zimbabwe	July	103.098144	126.397179	178.365835
112888	2023	Zimbabwe	August	70.758637	126.397179	178.365835
112889	2023	Zimbabwe	September	71.437761	126.397179	178.365835

112890 rows × 6 columns

```
import pandas as pd

# Load the crop production dataset
crop_production_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Crops_production_indicators_-_FAOSTAT_data_en_2-22-2024[

# Multiply 'Value' column with 100 (100gm/ha) to get the production value
crop_production_df['Production_Value'] = crop_production_df['Value'] * 100

# Calculate average production value per hectare per country and year
average_production_value = crop_production_df.groupby(['Area', 'Year'])['Production_Value'].mean().reset_index()

# Calculate total production value per country and year
total_production_value = crop_production_df.groupby(['Area', 'Year'])['Production_Value'].sum().reset_index()

# Merge the new variables back into the original dataset
crop_production_new = pd.merge(average_production_value, total_production_value, on=['Area', 'Year'], how='left')

crop_production_new.rename(columns={
    'Production_Value_x': 'Average_Production_Value',
    'Production_Value_y': 'Total_Production_Value'
}, inplace=True)

crop_production_new.head(10)
```

	Area	Year	Average_Production_Value	Total_Production_Value
0	Afghanistan	2000	6.017791e+06	66195700
1	Afghanistan	2001	6.070127e+06	66771400
2	Afghanistan	2002	6.113536e+06	67248900
3	Afghanistan	2003	6.120918e+06	67330100
4	Afghanistan	2004	6.144945e+06	67594400
5	Afghanistan	2005	5.508718e+06	60595900
6	Afghanistan	2006	5.637609e+06	62013700
7	Afghanistan	2007	5.676164e+06	62437800
8	Afghanistan	2008	5.146745e+06	56614200
9	Afghanistan	2009	7.228036e+06	79508400

Next steps: Generate code with `crop_production_new` View recommended plots

```
final_df = pd.merge(consumer_price_new, crop_production_new, on=['Year', 'Area'],how='outer')
```

```
final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113230 entries, 0 to 113229
Data columns (total 8 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   Year                      113230 non-null  int64  
 1   Area                      113230 non-null  object 
 2   Months                    112890 non-null  object 
 3   Value                     112890 non-null  float64
 4   Average_Annual_Inflation  110454 non-null  float64
 5   Inflation_Volatility      112890 non-null  float64
 6   Average_Production_Value  100048 non-null  float64
 7   Total_Production_Value    100048 non-null  float64
dtypes: float64(5), int64(1), object(2)
memory usage: 6.9+ MB
```

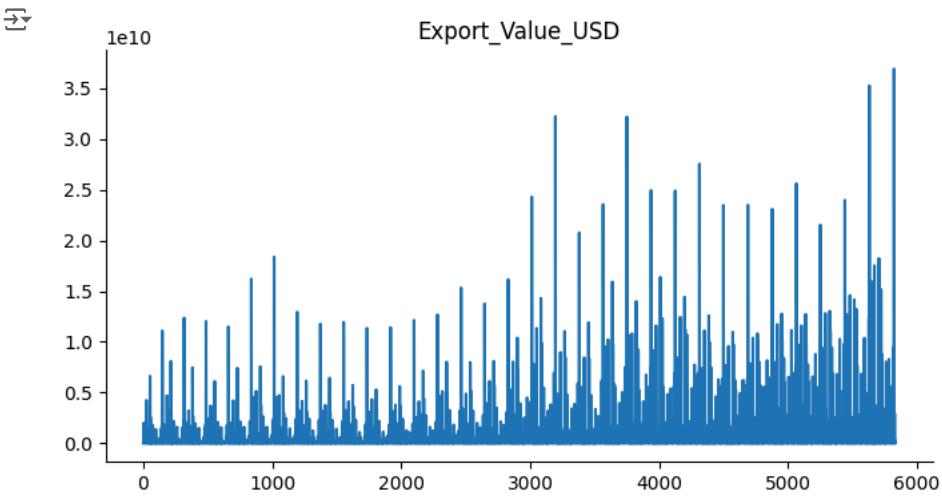
```
import pandas as pd

# Display the unique items in the 'Item' column to verify the available crop products
unique_items = food_trade_df["Item"].unique()
print(unique_items)

# Filter the dataset to include only export values related to crop products
crop_export_df = food_trade_df[
    (food_trade_df['Element'] == 'Export Value') &
    (food_trade_df['Domain Code'] == 'TCL') &  # Crops and livestock products
    (food_trade_df['Item'].isin(['Cereals and Preparations', 'Fruits and Vegetables', 'Vegetables']))  # Choose relevant crop items
]

# Create a new column for export value in USD
crop_export_df['Export_Value_USD'] = crop_export_df['Value'] * 1000

# Group by year, area, and area code to get total export value per year per area
crop_export_total_df = crop_export_df.groupby(['Year', 'Area'])['Export_Value_USD'].sum().reset_index()

# Display the first 10 rows to check the data
crop_export_total_df.head(10)
```

```
['Cereals and Preparations' 'Fats and Oils (excluding Butter)'
 'Meat and Meat Preparations' 'Sugar and Honey' 'Fruit and Vegetables'
 'Dairy Products and Eggs' 'Alcoholic Beverages' 'Non-alcoholic Beverages'
 'Other food' 'Non-food' 'Non-edible Fats and Oils' 'Tobacco']
<ipython-input-221-ff0c1b0e52ba>:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  crop_export_df['Export_Value_USD'] = crop_export_df['Value'] * 1000
```

	Year	Area	Export_Value_USD
0	1991	Albania	0.000000e+00
1	1991	Algeria	6.750000e+05
2	1991	Angola	0.000000e+00
3	1991	Antigua and Barbuda	0.000000e+00
4	1991	Argentina	1.152478e+09
5	1991	Australia	2.006318e+09
6	1991	Austria	1.847820e+08
7	1991	Bahamas	2.500000e+04
8	1991	Bahrain	1.620000e+05
9	1991	Bangladesh	0.000000e+00

Next steps: Generate code with crop_export_total_df View recommended plots

```
# Export_Value_USD

from matplotlib import pyplot as plt
crop_export_total_df['Export_Value_USD'].plot(kind='line', figsize=(8, 4), title='Export_Value_USD')
plt.gca().spines[['top', 'right']].set_visible(False)
```



```
final_df = pd.merge(final_df, crop_export_total_df, on=['Year', 'Area'],how='outer')
```

```
import pandas as pd

# Load the land temperature change dataset
land_temperature_df = pd.read_csv("/content/drive/MyDrive/ML_Coursework_Dataset/Land_temperature_change_-_FAOSTAT_data_en_2-27-2024[1].

# Calculate total temperature change per year per country
total_temperature_change = land_temperature_df.groupby(['Year', 'Area'])['Value'].sum().reset_index()
total_temperature_change.rename(columns={'Value': 'Total_Temperature_Change'}, inplace=True)

# Merge the new variables back into the original dataset
land_temperature_new = pd.merge(land_temperature_df, total_temperature_change, on=['Year', 'Area'], how='left')
# Select relevant columns
land_temperature_new = land_temperature_new[['Year', 'Area', 'Total_Temperature_Change']].drop_duplicates()

land_temperature_new.head(10)
```

	Year	Area	Total_Temperature_Change
0	2000	Afghanistan	9.128
1	2001	Afghanistan	10.718
2	2002	Afghanistan	10.988
3	2003	Afghanistan	7.098
4	2004	Afghanistan	11.029
5	2005	Afghanistan	6.170
6	2006	Afghanistan	12.762
7	2007	Afghanistan	7.540
8	2008	Afghanistan	7.685
9	2009	Afghanistan	8.637

Next steps: Generate code with land_temperature_new View recommended plots

```
land_temperature_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5481 entries, 0 to 54602
Data columns (total 3 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Year                     5481 non-null   int64
 1   Area                     5481 non-null   object
 2   Total_Temperature_Change  5481 non-null   float64
dtypes: float64(1), int64(1), object(1)
memory usage: 171.3+ KB
```

```
final_df = pd.merge(final_df, land_temperature_new, on=['Year', 'Area'],how='outer')
final_df
```

	Year	Area	Months	Value	Average_Annual_Inflation	Inflation_Volatility	Average_Production_Value	Total_Produc
0	2000	Afghanistan	January	24.356332	NaN	11.473983	6.017791e+06	
1	2000	Afghanistan	February	23.636242	NaN	11.473983	6.017791e+06	
2	2000	Afghanistan	March	23.485345	NaN	11.473983	6.017791e+06	
3	2000	Afghanistan	April	24.767194	NaN	11.473983	6.017791e+06	
4	2000	Afghanistan	May	25.956912	NaN	11.473983	6.017791e+06	
...	...	...	...	...	...	...	...	
115386	2018	Western Sahara	NaN	NaN	NaN	NaN	NaN	
115387	2019	Western Sahara	NaN	NaN	NaN	NaN	NaN	
115388	2020	Western Sahara	NaN	NaN	NaN	NaN	NaN	
115389	2021	Western Sahara	NaN	NaN	NaN	NaN	NaN	
115390	2022	Western Sahara	NaN	NaN	NaN	NaN	NaN	

115391 rows × 10 columns

```
#food security

# Calculate total and average food security index per year per country
average_food_security_index = food_security_df.groupby(['Year', 'Area'])['Value'].mean().reset_index()
average_food_security_index.rename(columns={'Value': 'Average_Food_Security_Index'}, inplace=True)

total_food_security_index = food_security_df.groupby(['Year', 'Area'])['Value'].sum().reset_index()
total_food_security_index.rename(columns={'Value': 'Total_Food_Security_Index'}, inplace=True)

# Merge the new variables back into the original dataset
food_security_new = pd.merge(average_food_security_index, total_food_security_index, on=['Year', 'Area'], how='left')

food_security_new.head(10)
```

	Year	Area	Average_Food_Security_Index	Total_Food_Security_Index
0	2000	Afghanistan	30.4200	91.26
1	2000	Albania	25.8400	103.36
2	2000	Algeria	18.7425	74.97
3	2000	Andorra	6.8900	20.67
4	2000	Angola	27.5400	110.16
5	2000	Antigua and Barbuda	9.9525	39.81
6	2000	Argentina	23.4000	93.60
7	2000	Armenia	16.4050	65.62
8	2000	Australia	11.1325	44.53
9	2000	Austria	16.3800	65.52

Next steps: Generate code with `food_security_new` View recommended plots

```
# Convert 'Year' column to string type
food_security_new['Year'] = food_security_new['Year'].astype(str)

# Extract the first year from the range if it's a range
food_security_new['Year'] = food_security_new['Year'].apply(lambda x: x.split('-')[0] if '-' in x else x)

# Convert 'Year' column to integer type
food_security_new['Year'] = food_security_new['Year'].astype(int)
```

```
final_df = pd.merge(final_df, food_security_new, on=['Year', 'Area'],how='outer')
final_df
```

	Year	Area	Months	Value	Average_Annual_Inflation	Inflation_Volatility	Average_Production_Value	Total_Produc
0	2000	Afghanistan	January	24.356332	NaN	11.473983	6.017791e+06	
1	2000	Afghanistan	January	24.356332	NaN	11.473983	6.017791e+06	
2	2000	Afghanistan	February	23.636242	NaN	11.473983	6.017791e+06	
3	2000	Afghanistan	February	23.636242	NaN	11.473983	6.017791e+06	
4	2000	Afghanistan	March	23.485345	NaN	11.473983	6.017791e+06	
...	...	...	...	...	...	...	...	
206485	2006	Sudan	NaN	NaN	NaN	NaN	NaN	
206486	2006	Sudan	NaN	NaN	NaN	NaN	NaN	
206487	2007	South Sudan	NaN	NaN	NaN	NaN	NaN	
206488	2007	Sudan	NaN	NaN	NaN	NaN	NaN	
206489	2007	Sudan	NaN	NaN	NaN	NaN	NaN	

206490 rows × 12 columns

```
import pandas as pd

# Load the emissions dataset

# Filter the DataFrame for emissions from crops
filtered_emission = emissions_df[
    (emissions_df['Domain'] == 'Emissions from Crops') &
    (emissions_df['Element'].isin(['Crops total (Emissions CH4)','Crops total (Emissions N2O)']))
]

# Calculate total emissions per year per country

total_emissions = filtered_emission.groupby(['Year', 'Area'])['Value'].sum().reset_index()
total_emissions.rename(columns={'Value': 'Total_Emissions'}, inplace=True)

# Merge the new variables back into the original dataset
emissions_new = pd.merge(filtered_emission, total_emissions, on=['Year', 'Area'], how='left')
# Select relevant columns
emissions_new = emissions_new[['Year', 'Area', 'Total_Emissions']].drop_duplicates()

emissions_new.head(10)
```

	Year	Area	Total_Emissions
0	2000	Afghanistan	21.5527
2	2001	Afghanistan	19.9659
4	2002	Afghanistan	22.3209
6	2003	Afghanistan	25.0134
8	2004	Afghanistan	31.3945
10	2005	Afghanistan	27.4134
12	2006	Afghanistan	26.9968
14	2007	Afghanistan	28.6996
16	2008	Afghanistan	30.7308
18	2009	Afghanistan	33.2563

Next steps: Generate code with `emissions_new` View recommended plots

```
final_df = pd.merge(final_df, emissions_new, on=['Year', 'Area'],how='outer')
```

```
# Load the employment dataset

# Calculate total employment per year per country

total_employment = employment_df.groupby(['Year', 'Area'])['Value'].sum().reset_index()
total_employment.rename(columns={'Value': 'Total_Employment'}, inplace=True)

# Merge the total employment back into the original dataset (if needed)
employment_new = pd.merge(employment_df, total_employment, on=['Year', 'Area'], how='left')
# Select relevant columns
employment_new = employment_new[['Year', 'Area', 'Total_Employment']].drop_duplicates()

employment_new.head(10)
```

	Year	Area	Total_Employment
0	2014	Afghanistan	3449.80
1	2017	Afghanistan	3645.52
2	2000	Afghanistan	2765.95
3	2001	Afghanistan	2805.54
4	2002	Afghanistan	2897.51
5	2003	Afghanistan	3093.27
6	2004	Afghanistan	3212.46
7	2005	Afghanistan	3287.47
8	2006	Afghanistan	3406.43
9	2007	Afghanistan	3352.45

Next steps: Generate code with employment_new View recommended plots

```
final_df = pd.merge(final_df, employment_new, on=['Year', 'Area'],how='outer')
```

```
# Load the exchange rate dataset

total_exchange_rate = exchange_rate_df.groupby(['Year', 'Area'])['Value'].sum().reset_index()
total_exchange_rate.rename(columns={'Value': 'Total_Exchange_Rate'}, inplace=True)

# Merge the total exchange_rate back into the original dataset (if needed)
exchange_rate_new = pd.merge(exchange_rate_df, total_exchange_rate, on=['Year', 'Area'], how='left')
# Select relevant columns
exchange_rate_new = exchange_rate_new[['Year', 'Area', 'Total_Exchange_Rate']].drop_duplicates()

exchange_rate_new.head(10)
```

	Year	Area	Total_Exchange_Rate
0	1980	Afghanistan	529.550000
12	1981	Afghanistan	593.758826
24	1982	Afghanistan	607.195295
36	1983	Afghanistan	607.195295
48	1984	Afghanistan	607.195277
60	1985	Afghanistan	607.195264
72	1986	Afghanistan	607.195264
84	1987	Afghanistan	607.195264
96	1988	Afghanistan	607.195264
108	1989	Afghanistan	607.195264

Next steps: Generate code with exchange_rate_new View recommended plots

```
final_df = pd.merge(final_df, exchange_rate_new, on=['Year', 'Area'],how='outer')
```

```
# Load the fertilizers use dataset

total_fertilizers_use = fertilizers_use_df.groupby(['Year', 'Area'])['Value'].sum().reset_index()
total_fertilizers_use.rename(columns={'Value': 'Total_Fertilizers_Use'}, inplace=True)

# Merge the new variables back into the original dataset
fertilizers_use_new = pd.merge(fertilizers_use_df, total_fertilizers_use, on=['Year', 'Area'], how='left')
# Select relevant columns
fertilizers_use_new = fertilizers_use_new[['Year', 'Area', 'Total_Fertilizers_Use']].drop_duplicates()

fertilizers_use_new.head(10)
```

	Year	Area	Total_Fertilizers_Use
0	2002	Afghanistan	17900.0
1	2003	Afghanistan	33200.0
2	2004	Afghanistan	90000.0
4	2005	Afghanistan	20577.0
5	2006	Afghanistan	68253.0
6	2007	Afghanistan	61315.0
7	2008	Afghanistan	50628.0
8	2009	Afghanistan	76599.0
9	2010	Afghanistan	72084.0
10	2011	Afghanistan	107228.0

Next steps: Generate code with `fertilizers_use_new` View recommended plots

```
final_df = pd.merge(final_df, fertilizers_use_new, on=['Year', 'Area'],how='outer')
```

```
# Load the food balances dataset

item = food_balances_df['Item'].unique()

food_balances_df = food_balances_df[
    (food_balances_df['Element'] == 'Export Quantity') &
    (food_balances_df['Item'].isin(item))
]

# Calculate total food balances filtering on the basis of export quantities per year per country
total_food_balances = food_balances_df.groupby(['Year', 'Area', 'Area Code (M49)'])['Value'].sum().reset_index()
total_food_balances.rename(columns={'Value': 'Total_Food_Balances'}, inplace=True)

# Merge the new variables back into the original dataset
food_balances_new = pd.merge(food_balances_df, total_food_balances, on=['Year', 'Area'], how='left')
# Select relevant columns
food_balances_new = food_balances_new[['Year', 'Area', 'Total_Food_Balances']].drop_duplicates()

food_balances_new.head(10)
```

	Year	Area	Total_Food_Balances
0	2010	Afghanistan	360.0
1	2011	Afghanistan	277.0
2	2012	Afghanistan	198.0
3	2013	Afghanistan	281.0
4	2014	Afghanistan	412.0
5	2015	Afghanistan	420.0
6	2016	Afghanistan	536.0
7	2017	Afghanistan	625.0
8	2018	Afghanistan	1331.0
9	2019	Afghanistan	981.0

Next steps: Generate code with `food_balances_new` View recommended plots

```
final_df = pd.merge(final_df, food_balances_new, on=['Year', 'Area'],how='outer')
```

```
# Load the foreign direct investment dataset

# Calculate total FDI per year per country
total_fdi = foreign_direct_investment_df.groupby(['Year', 'Area', 'Area Code (M49)'])['Value'].sum().reset_index()
total_fdi.rename(columns={'Value': 'Total_FDI'}, inplace=True)

# Merge the new variables back into the original dataset
foreign_direct_investment_new = pd.merge(foreign_direct_investment_df, total_fdi, on=['Year', 'Area'], how='left')
# Select relevant columns
foreign_direct_investment_new = foreign_direct_investment_new[['Year', 'Area', 'Total_FDI']].drop_duplicates()

foreign_direct_investment_new.head(10)
```

	Year	Area	Total_FDI
0	2000	Afghanistan	0.170000
1	2001	Afghanistan	0.680000
2	2002	Afghanistan	50.000000
3	2003	Afghanistan	58.800000
4	2004	Afghanistan	186.200000
5	2005	Afghanistan	272.500000
6	2006	Afghanistan	238.000000
7	2007	Afghanistan	188.690000
8	2008	Afghanistan	44.115704
9	2009	Afghanistan	197.847687

Next steps: Generate code with `foreign_direct_investment_new` View recommended plots

```
final_df = pd.merge(final_df, foreign_direct_investment_new, on=['Year', 'Area'],how='outer')
```

```
# Load the land use dataset

# Calculate total land use per year per country
total_land_use = land_use_df.groupby(['Year', 'Area', 'Area Code (M49)'])['Value'].sum().reset_index()
total_land_use.rename(columns={'Value': 'Total_Land_Use'}, inplace=True)

# Merge the new variables back into the original dataset
land_use_new = pd.merge(land_use_df, total_land_use, on=['Year', 'Area'], how='left')
# Select relevant columns
land_use_new = land_use_new[['Year', 'Area', 'Total_Land_Use']].drop_duplicates()

land_use_new.head(10)
```

	Year	Area	Total_Land_Use
0	1980	Afghanistan	255210.0
1	1981	Afghanistan	255241.0
2	1982	Afghanistan	255260.0
3	1983	Afghanistan	255275.0
4	1984	Afghanistan	255306.0
5	1985	Afghanistan	255311.0
6	1986	Afghanistan	255444.0
7	1987	Afghanistan	255439.0
8	1988	Afghanistan	255519.0
9	1989	Afghanistan	255569.0

Next steps: Generate code with `land_use_new` View recommended plots

```
final_df = pd.merge(final_df, land_use_new, on=['Year', 'Area'],how='outer')
```

```
# Load the pesticides use dataset
pesticides_use_df['Unit'].unique()
```

```
array(['t', 'kg/ha', 'g/Int$'], dtype=object)
```

```
# Define conversion factors for different units to tons
conversion_factors = {
    'kg/ha': 1 / 1000,  # kg/ha to tons/ha
    'g/Int$': 1 / 1_000_000_000,  # g/Int$ to tons/Int$
    'kg': 1 / 1000,  # kg to tons
    'g': 1 / 1_000_000,  # g to tons
    't': 1  # tons remain unchanged
}

# Convert values based on unit
for unit, conversion_factor in conversion_factors.items():
    pesticides_use_df.loc[pesticides_use_df['Unit'] == unit, 'Value'] *= conversion_factor

# Update all units to 'tons'
pesticides_use_df['Unit'] = 'tons'

total_pesticides_use = pesticides_use_df.groupby(['Year', 'Area'])['Value'].sum().reset_index()
total_pesticides_use.rename(columns={'Value': 'Total_Pesticides_Use'}, inplace=True)

# Merge the new variables back into the original dataset
pesticides_use_new = pd.merge(pesticides_use_df, total_pesticides_use, on=['Year', 'Area'], how='left')
# Select relevant columns
pesticides_use_new = pesticides_use_new[['Year', 'Area', 'Total_Pesticides_Use']].drop_duplicates()

pesticides_use_new.head(10)
```

	Year	Area	Total_Pesticides_Use
0	2000	Albania	607.90044
3	2001	Albania	629.09046
6	2002	Albania	650.27047
9	2003	Albania	671.44049
12	2004	Albania	692.62051
15	2005	Albania	713.81055
18	2006	Albania	734.99053
21	2007	Albania	756.17056
24	2008	Albania	777.35057
27	2009	Albania	798.53059

Next steps: Generate code with `pesticides_use_new` View recommended plots

```
final_df = pd.merge(final_df, pesticides_use_new, on=['Year', 'Area'],how='outer')
```

```
final_df
```

	Year	Area	Months	Value	Average_Annual_Inflation	Inflation_Volatility	Average_Production_Value	Total_Produc
0	2000	Afghanistan	January	24.356332	NaN	11.473983	6.017791e+06	
1	2000	Afghanistan	January	24.356332	NaN	11.473983	6.017791e+06	
2	2000	Afghanistan	February	23.636242	NaN	11.473983	6.017791e+06	
3	2000	Afghanistan	February	23.636242	NaN	11.473983	6.017791e+06	
4	2000	Afghanistan	March	23.485345	NaN	11.473983	6.017791e+06	
...	...	...	...	...	...	...	...	
209557	1985	Yemen	NaN	NaN	NaN	NaN	NaN	
209558	1986	Yemen	NaN	NaN	NaN	NaN	NaN	
209559	1987	Yemen	NaN	NaN	NaN	NaN	NaN	
209560	1988	Yemen	NaN	NaN	NaN	NaN	NaN	
209561	1989	Yemen	NaN	NaN	NaN	NaN	NaN	

209562 rows × 20 columns

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

processed_df = final_df.copy()
```

```
# Make a copy of the original dataframe
preprocessed_df = processed_df.copy()

# 1. Handling Missing Values
# For numerical columns, fill missing values with the mean
imputer = SimpleImputer(strategy='mean')
numerical_cols = preprocessed_df.select_dtypes(include=['float64', 'int64']).columns
preprocessed_df[numerical_cols] = imputer.fit_transform(preprocessed_df[numerical_cols])

# For categorical columns, fill missing values with a constant or most frequent value
categorical_cols = preprocessed_df.select_dtypes(include=['object']).columns
preprocessed_df[categorical_cols] = preprocessed_df[categorical_cols].fillna('Unknown')

# 2. Feature Scaling
scaler = StandardScaler()
preprocessed_df[numerical_cols] = scaler.fit_transform(preprocessed_df[numerical_cols])

# 3. Encoding Categorical Variables (if any)
# If 'Area' is a categorical variable
preprocessed_df = pd.get_dummies(preprocessed_df, columns=['Area'])

# 4. Train-Test Split
X = preprocessed_df.drop(columns=['Export_Value_USD'])  # Features
y = preprocessed_df['Export_Value_USD']  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
X_train
```

	Year	Months	Value	Average_Annual_Inflation	Inflation_Volatility	Average_Production_Value	Total_Production_
207699	-3.742402	Unknown	1.206094e-18	0.000000	-9.664608e-18	0.000000	0.(
56811	-0.219206	November	-9.313316e-03	-0.021848	-7.068028e-02	-0.115512	0.2
75384	0.344505	October	-9.313314e-03	-0.023389	-7.482193e-02	0.626688	1.2
111828	0.767289	October	-9.313311e-03	-0.011499	-7.299362e-02	0.735226	1.3
82041	-0.360134	August	-9.313315e-03	-0.013734	-7.220950e-02	-0.434283	-0.3
...	...	...	...	...	...	...	
119879	1.190072	September	-9.313321e-03	-0.021433	-7.513033e-02	0.767906	1.4
103694	1.612856	September	-9.313319e-03	-0.013812	-7.374474e-02	-0.187354	-1.(
131932	-0.641990	March	-9.313318e-03	-0.018451	-7.412644e-02	0.249373	0.4
146867	1.612856	December	-9.313309e-03	-0.016551	-7.446570e-02	2.131319	1.2
121958	-0.641990	August	-9.313321e-03	-0.022510	-7.433965e-02	-0.534132	-0.1

167649 rows × 276 columns

```
y_train
```

```
207699   -5.064062e-17
56811    -3.402963e-01
75384    -1.880316e-01
111828   -3.438749e-01
82041    -5.064062e-17
              ...
```

```
119879    7.876884e-01
103694    2.024004e-01
131932   -3.385304e-01
146867    2.864079e+00
121958   -3.445953e-01
Name: Export_Value_USD, Length: 167649, dtype: float64
```

```
from sklearn.model_selection import train_test_split

# Assuming you have a DataFrame named preprocessed_df containing your preprocessed data
# Split the data into features (X) and target variable (y)
X = preprocessed_df.drop(columns=['Export_Value_USD'])  # Features
y = preprocessed_df['Export_Value_USD']  # Target variable

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Now X_val and y_val contain the validation data
```

```
# Check the shape of the processed data
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (167649, 276)
X_test shape: (41913, 276)
y_train shape: (167649,)
y_test shape: (41913,)
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Assuming X_train, y_train, X_val, y_val are already defined as provided

# Drop the 'Months' column from X_train and X_val
X_train = X_train.drop(columns=['Months'])
X_val = X_val.drop(columns=['Months'])

# Ensure all data is numeric
X_train = X_train.apply(pd.to_numeric, errors='coerce')
X_val = X_val.apply(pd.to_numeric, errors='coerce')
y_train = pd.to_numeric(y_train, errors='coerce')
y_val = pd.to_numeric(y_val, errors='coerce')

# Handle missing values if any
X_train = X_train.fillna(0)
X_val = X_val.fillna(0)
y_train = y_train.fillna(0)
y_val = y_val.fillna(0)
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

# Define the MLP model architecture
mlp_model = Sequential([
    Dense(100, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.2),
    Dense(50, activation='relu'),
    Dropout(0.2),
    Dense(1)  # Output layer
])

# Compile the model
mlp_model.compile(optimizer='adam', loss='mean_squared_error')

# Print the model summary
mlp_model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_21 (Dense)            (None, 100)               27600

 dropout_14 (Dropout)        (None, 100)               0

 dense_22 (Dense)            (None, 50)                5050

 dropout_15 (Dropout)        (None, 50)                0

 dense_23 (Dense)            (None, 1)                 51

=================================================================
Total params: 32701 (127.74 KB)
Trainable params: 32701 (127.74 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Scale the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

```
# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Learning rate reduction callback
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.0001)
```

```
# Define the MLP model
model = Sequential()
model.add(Dense(100, input_dim=X_train_scaled.shape[1], activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(50, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')

# Early stopping and learning rate reduction callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.0001)

# Train the model
model.fit(X_train_scaled, y_train, epochs=500, batch_size=256, validation_data=(X_val_scaled, y_val), callbacks=[early_stopping, reduce
```

```
Epoch 1/500
655/655 [==============================] - 7s 6ms/step - loss: 0.2133 - val_loss: 0.0642 - lr: 0.0010
Epoch 2/500
655/655 [==============================] - 3s 5ms/step - loss: 0.0935 - val_loss: 0.0575 - lr: 0.0010
Epoch 3/500
655/655 [==============================] - 4s 6ms/step - loss: 0.0763 - val_loss: 0.0446 - lr: 0.0010
Epoch 4/500
655/655 [==============================] - 4s 6ms/step - loss: 0.0681 - val_loss: 0.0738 - lr: 0.0010
Epoch 5/500
655/655 [==============================] - 4s 5ms/step - loss: 0.0581 - val_loss: 0.0327 - lr: 0.0010
Epoch 6/500
655/655 [==============================] - 3s 5ms/step - loss: 0.0540 - val_loss: 0.0287 - lr: 0.0010
Epoch 7/500
655/655 [==============================] - 4s 7ms/step - loss: 0.0532 - val_loss: 0.0236 - lr: 0.0010
Epoch 8/500
655/655 [==============================] - 4s 6ms/step - loss: 0.0482 - val_loss: 0.0213 - lr: 0.0010
Epoch 9/500
655/655 [==============================] - 3s 5ms/step - loss: 0.0443 - val_loss: 0.0224 - lr: 0.0010
Epoch 10/500
655/655 [==============================] - 4s 6ms/step - loss: 0.0447 - val_loss: 0.0215 - lr: 0.0010
Epoch 11/500
655/655 [==============================] - 5s 8ms/step - loss: 0.0422 - val_loss: 0.0268 - lr: 0.0010
Epoch 12/500
655/655 [==============================] - 4s 5ms/step - loss: 0.0417 - val_loss: 0.0236 - lr: 0.0010
Epoch 13/500
655/655 [==============================] - 3s 5ms/step - loss: 0.0401 - val_loss: 0.0189 - lr: 0.0010
Epoch 14/500
655/655 [==============================] - 4s 6ms/step - loss: 0.0409 - val_loss: 0.0281 - lr: 0.0010
Epoch 15/500
655/655 [==============================] - 5s 8ms/step - loss: 0.0392 - val_loss: 0.0184 - lr: 0.0010
Epoch 16/500
```

```
655/655 [==============================] - 3s 5ms/step - loss: 0.0393 - val_loss: 0.0147 - lr: 0.0010
Epoch 17/500
655/655 [==============================] - 3s 5ms/step - loss: 0.0371 - val_loss: 0.0162 - lr: 0.0010
Epoch 18/500
655/655 [==============================] - 3s 5ms/step - loss: 0.0365 - val_loss: 0.0157 - lr: 0.0010
Epoch 19/500
655/655 [==============================] - 5s 8ms/step - loss: 0.0356 - val_loss: 0.0245 - lr: 0.0010
Epoch 20/500
655/655 [==============================] - 4s 6ms/step - loss: 0.0369 - val_loss: 0.0191 - lr: 0.0010
Epoch 21/500
655/655 [==============================] - 3s 5ms/step - loss: 0.0350 - val_loss: 0.0145 - lr: 0.0010
Epoch 22/500
655/655 [==============================] - 4s 6ms/step - loss: 0.0357 - val_loss: 0.0134 - lr: 0.0010
Epoch 23/500
655/655 [==============================] - 5s 7ms/step - loss: 0.0351 - val_loss: 0.0123 - lr: 0.0010
Epoch 24/500
655/655 [==============================] - 3s 5ms/step - loss: 0.0342 - val_loss: 0.0180 - lr: 0.0010
Epoch 25/500
655/655 [==============================] - 3s 5ms/step - loss: 0.0340 - val_loss: 0.0134 - lr: 0.0010
Epoch 26/500
655/655 [==============================] - 4s 6ms/step - loss: 0.0350 - val_loss: 0.0117 - lr: 0.0010
Epoch 27/500
655/655 [==============================] - 5s 7ms/step - loss: 0.0316 - val_loss: 0.0118 - lr: 0.0010
Epoch 28/500
655/655 [==============================] - 3s 5ms/step - loss: 0.0336 - val_loss: 0.0110 - lr: 0.0010
Epoch 29/500
655/655 [==============================] - 3s 5ms/step - loss: 0.0318 - val_loss: 0.0110 - lr: 0.0010
```

```
# Predict on the validation set
y_val_pred = model.predict(X_val_scaled)
mse_val = mean_squared_error(y_val, y_val_pred)
r2_val = r2_score(y_val, y_val_pred)

print("Mean Squared Error on Validation Set:", mse_val)
print("R-squared on Validation Set:", r2_val)
```

```
1310/1310 [==============================] - 2s 1ms/step
Mean Squared Error on Validation Set: 0.005462604625227736
R-squared on Validation Set: 0.9945472646855494
```

```
# Assuming X_test is defined
# Drop the 'Months' column from X_test if necessary
X_test = X_test.drop(columns=['Months'])
X_test = X_test.apply(pd.to_numeric, errors='coerce')
X_test = X_test.fillna(0)

# Scale the test data
X_test_scaled = scaler.transform(X_test)

# Predict on the test set
y_test_pred = model.predict(X_test_scaled)

# Clip the predictions to be non-negative
y_test_pred = np.clip(y_test_pred, 0, None)

# Evaluate the model using Mean Squared Error (MSE) on the test set
mse_test_clipped = mean_squared_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)

print("Mean Squared Error on Test Set with Clipped Predictions:", mse_test_clipped)
print("R-squared on Test Set:", r2_test)

# Add the year and area code to the predictions for the test set
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

# Display the first few rows of the DataFrame with predictions for the test set
print(test_df_with_predictions.head(10))
```

```
1310/1310 [==============================] - 5s 4ms/step
Mean Squared Error on Test Set with Clipped Predictions: 0.07640475699173555
R-squared on Test Set: 0.9237332838044297
            Year     Value  Average_Annual_Inflation  Inflation_Volatility  \
108916  1.471928 -0.009313                 -0.022149             -0.073553
9838    0.908217 -0.009313                 -0.024279             -0.074533
145571  0.767289 -0.009313                 -0.022686             -0.075079
74792  -1.346629 -0.009313                 -0.021101             -0.074822
59297   0.908217 -0.009313                 -0.006093             -0.071652
4979   -0.641990 -0.009313                 -0.015702             -0.064760
199463  1.331000 -0.009311                  1.767297             14.090680
17905   0.344505 -0.009313                 -0.018050             -0.067528
69998  -0.078278 -0.009313                 -0.023466             -0.075022
167898 -1.346629 -0.009313                 -0.020675             -0.074106
```

```
        Average_Production_Value  Total_Production_Value  \
108916                  0.609729                0.001894
9838                    0.000000                0.000000
145571                 -0.234619                0.098843
74792                   0.786094                1.435157
59297                   1.180254                1.951190
4979                   -0.555405               -0.321130
199463                  0.349881                0.594403
17905                  -0.065513                0.100009
69998                  -0.216593               -0.282048
167898                 -0.427198               -0.330461

        Total_Temperature_Change  Average_Food_Security_Index  \
108916                  0.536405                -6.085528e-01
9838                   -0.169461                 1.837445e-16
145571                  0.614451                 1.664634e-01
74792                   0.816619                -4.627039e-01
59297                  -0.418332                 6.991739e-01
4979                   -0.960581                 2.113168e-02
199463                  0.358684                 1.578922e+00
17905                  -0.372256                 1.949091e-01
69998                  -0.184192                -4.521014e-01
167898                 -0.334016                -2.454822e-01

        Total_Food_Security_Index  Total_Emissions  ...  Area_Wake Island  \
108916              -1.158132e+00       -0.243628  ...             False
9838                -2.533178e-16        0.000000  ...             False
145571               5.894952e-01        2.453853  ...             False
74792               -6.823654e-01       -0.224450  ...             False
59297                1.507516e+00        0.039224  ...             False
4979                 3.390447e-01       -0.237832  ...             False
199463              -5.415427e-01       -0.167859  ...             False
17905                6.385157e-01       -0.242576  ...             False
69998               -4.764791e-01       -0.247475  ...             False
167898              -3.828944e-01       -0.244945  ...             False

        Area_Wallis and Futuna Islands  Area_Western Sahara  Area_Yemen  \
108916                           False                False       False
9838                             False                False       False
145571                           False                False       False
74792                            False                False       False
59297                            False                False       False
4979                             False                False       False
```

```
test_df_with_predictions
```

	Year	Value	Average_Annual_Inflation	Inflation_Volatility	Average_
108916	1.471928	-0.009313	-0.022149	-0.073553	
9838	0.908217	-0.009313	-0.024279	-0.074533	
145571	0.767289	-0.009313	-0.022686	-0.075079	
74792	-1.346629	-0.009313	-0.021101	-0.074822	
59297	0.908217	-0.009313	-0.006093	-0.071652	
...	...	...	...	...	
169557	0.485433	-0.009313	-0.023523	-0.074567	
22634	-0.360134	-0.009313	-0.018187	-0.074703	
152471	0.626361	-0.009313	-0.023149	-0.075501	
77467	-1.064773	-0.009313	-0.022040	-0.075387	
139671	-1.205701	-0.009313	-0.022231	-0.074583	

41913 rows × 277 columns

```
# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Remove the 'Months' column from X_train
X_train = X_train.drop(columns=['Months'])
X_train
```

	Year	Value	Average_Annual_Inflation	Inflation_Volatility	Averag
207699	-3.742402	1.206094e-18	0.000000	-9.664608e-18	
56811	-0.219206	-9.313316e-03	-0.021848	-7.068028e-02	
75384	0.344505	-9.313314e-03	-0.023389	-7.482193e-02	
111828	0.767289	-9.313311e-03	-0.011499	-7.299362e-02	
82041	-0.360134	-9.313315e-03	-0.013734	-7.220950e-02	