notebook

May 17, 2024

[1]:
```
# Data Preprocessing
import os
import glob
import pandas as pd
# Feature Selection
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# Plots
import seaborn as sns
import matplotlib.pyplot as plt
# MLP
import torch
from torch.utils.data import Dataset, DataLoader

min_max_scaler = MinMaxScaler()
standard_scaler = StandardScaler()
pd.set_option('display.float_format', '{:.2f}'.format)
```

1 Data Preprocessing

1.1 Calculate Export Value of Crop Products

Two export value datasets are created. One which maps countries yearly exports to individual crop products, and another which maps each country to the sum of its yearly crop products.

[2]:
```
crop_indicators = pd.read_csv('Data/food_trade_indicators.csv')
print(crop_indicators['Item'].unique())
print()
print(crop_indicators.columns)
```

```
['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']

Index(['Domain Code', 'Domain', 'Area Code (M49)', 'Area', 'Element Code',
```

```
       'Element', 'Item Code (CPC)', 'Item', 'Year Code', 'Year', 'Unit',
       'Value', 'Flag', 'Flag Description', 'Note'],
      dtype='object')
```

[3]:
```
"""
Crop products

- Cereals and Preperations
    Edible grains fed to people and livestock
- Fats and Oils (excluding Butter)
    Fats and oils derived from plants such as rapeseed or olives
- Sugar and Honey
    80% of sugar comes from sugar cane, crops can be sowed specifically for␣
 ↪bees to pollinate and produce honey
- Fruit and Vegetables
    Edible crops for human and animal consumption
- Alcoholic Beverages
    Crops such as potatoes and wheat/barley are used to make drinks like vodka␣
 ↪and beer
- Non-alcoholic Beverages
    Crops such as lemons, oranges, apples and more are used to make juices
- Non-food
    Crops such as cotton and hemp are used to make textiles and clothing
- Tobacco
    Tobacco leaves are widely distributed around the world
"""
crop_products_dict = {
    'cereals': 'Cereals and Preparations',
    'fats': 'Fats and Oils (excluding Butter)',
    'sugar': 'Sugar and Honey',
    'fruit and veg': 'Fruit and Vegetables',
    'alcohol': 'Alcoholic Beverages',
    'drinks': 'Non-alcoholic Beverages',
    'materials': 'Non-food',
    'tobacco': 'Tobacco'
}

cols_to_drop = ['Domain Code', 'Domain', 'Element Code', 'Element',
                'Item Code (CPC)', 'Year Code', 'Flag', 'Flag Description',
                'Note']

# drop import values
crop_indicators = crop_indicators.
 ↪drop(crop_indicators[crop_indicators['Element'] == 'Import Value'].index)
# drop products that aren't crops (livestock products)
crop_indicators = crop_indicators.drop(crop_indicators[~crop_indicators['Item'].
 ↪isin(list(crop_products_dict.values()))].index)
```

```python
# drop unneccessary columns
crop_indicators = crop_indicators.drop(columns=cols_to_drop)
# retain data from 2002, this is the first common year between all features
crop_indicators = crop_indicators[crop_indicators['Year'] >= 2002]
# reset index
crop_indicators.reset_index(drop=True, inplace=True)
display(crop_indicators)
# save data with crops grouped
crop_indicators.to_csv('Data/crop_export_values/crop_exports_by_product.csv')
# sum values for each crop group
crop_indicators = crop_indicators.groupby(['Area Code (M49)', 'Area', 'Year',␣
↪'Unit'])['Value'].sum().reset_index()
display(crop_indicators)
# save data with total crop export values
crop_indicators.to_csv('Data/crop_export_values/crop_exports_summed.csv')
```

```
       Area Code (M49)         Area                     Item  Year      Unit  \
0                    4  Afghanistan  Cereals and Preparations  2009  1000 USD
1                    4  Afghanistan  Cereals and Preparations  2010  1000 USD
2                    4  Afghanistan  Cereals and Preparations  2011  1000 USD
3                    4  Afghanistan  Cereals and Preparations  2012  1000 USD
4                    4  Afghanistan  Cereals and Preparations  2013  1000 USD
…                  …          …                        …   …       …
30840              716     Zimbabwe                  Tobacco  2018  1000 USD
30841              716     Zimbabwe                  Tobacco  2019  1000 USD
30842              716     Zimbabwe                  Tobacco  2020  1000 USD
30843              716     Zimbabwe                  Tobacco  2021  1000 USD
30844              716     Zimbabwe                  Tobacco  2022  1000 USD

          Value
0         15.00
1         54.00
2          0.00
3          0.00
4          0.00
…          …
30840 893113.05
30841 828488.44
30842 794956.99
30843 836533.69
30844 998057.60

[30845 rows x 6 columns]

      Area Code (M49)         Area  Year      Unit      Value
0                   4  Afghanistan  2002  1000 USD   34952.00
1                   4  Afghanistan  2003  1000 USD   55146.00
2                   4  Afghanistan  2004  1000 USD   57772.00
```

```
3                    4  Afghanistan  2005  1000 USD   66899.00
4                    4  Afghanistan  2006  1000 USD   63787.00
…                 …           …    …        …          …
4071               894       Zambia  2018  1000 USD  687348.29
4072               894       Zambia  2019  1000 USD  594188.54
4073               894       Zambia  2020  1000 USD  674030.82
4074               894       Zambia  2021  1000 USD  888978.56
4075               894       Zambia  2022  1000 USD 1071355.53

[4076 rows x 5 columns]
```

1.2 Feature Selection

Each CSV file is a feature of the dataset. Some features have more dimensions than others, for example `consumer_prices` has two dimensions: `Consumer Prices, Food Indices (2015=100)`, a indication of the price levels of food since 2015, and `Food price inflation` which represents the rate of change of food prices over time.

Linear relationships of these features to export crop yields can be determined by computing their correlation. Features with low linearity are thought to have little to no effect on the value of crop yields, however argriculture and its economics has complex relationships between all given features, implying there's importance to all. To explore this further, Random Forest regression can be used to evaluate non-linear feature importance.

By combining linear (correlation) and non-linear (Random Forest regression) relationships between export crop yields and features, valuable inputs can be chosen for the model while minimising noise and keeping the parameter count as low as possible for faster training and convergence while requiring less computational expense.

[4]:
```
crop_exports_by_prod = pd.read_csv('Data/crop_export_values/
 ↪crop_exports_by_product.csv', index_col=[0])
crop_exports_by_prod = crop_exports_by_prod.rename(columns={'Value':␣
 ↪'Crop_Export_Value'})

crop_exports_summed = pd.read_csv('Data/crop_export_values/crop_exports_summed.
 ↪csv', index_col=[0])
crop_exports_summed = crop_exports_summed.rename(columns={'Value':␣
 ↪'Annual_Crop_Export_Value'})
```

[5]:
```
def display_heatmap(correlation, title, figsize=(12,8)):
    """
    Displays a heatmap to show linear correlations.
    """
    plt.figure(figsize=figsize)
    # Draw the heatmap with the mask
    sns.heatmap(correlation, annot=True, fmt=".2f", cmap='coolwarm',
                cbar_kws={"shrink": .8}, linewidths=.5, annot_kws={"size": 8})
    plt.xticks(rotation=45, ha='right')
    plt.yticks(rotation=0)
```

```python
    plt.title(f'{title}', size=15)
    plt.show()

def rank_feature_importance(X, y, feature_name, column_name=None):
    """
    Uses Random Forest regression to rank features based on non-linear
    correlation with crop export values.
    """
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

    model = RandomForestRegressor(
        n_estimators=200,
        max_depth=10,
        min_samples_split=10,
        min_samples_leaf=5,
    )

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f'MSE: {mse}')

    feature_importance_df = pd.DataFrame({
        'Feature': X.columns if column_name is None else [f'{column_name}'],
        'Importance': model.feature_importances_
    }).sort_values(by='Importance', ascending=False)

    display(feature_importance_df)
    sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
    plt.xlabel('Importance Against Export Crop Value')
    plt.title(f'{feature_name}')
    plt.show()
```

1.2.1 consumer_prices.csv

[ ]:

```python
consumer_prices = pd.read_csv('Data/raw_data/consumer_prices.csv')
print(f"Units: {consumer_prices['Unit'].unique()}\n")

# data is stored monthly. Change it to annually
consumer_prices = consumer_prices.groupby(['Year', 'Area', 'Item'])['Value'].
 ↪mean().reset_index()

consumer_prices = consumer_prices[(consumer_prices['Year'] >= 2002) &␣
 ↪(consumer_prices['Year'] <= 2022)]
unique_items = consumer_prices['Item'].unique()
print(f'Unique items: {unique_items}')
```

[ ]:
```
food_index_prices = consumer_prices[consumer_prices['Item'] == unique_items[0]]
food_index_prices = food_index_prices.rename(columns={'Value':␣
↪'Food_Indices_Value', 'Item': 'Food_Index_Item'})

price_inflation = consumer_prices[consumer_prices['Item'] == unique_items[1]]
price_inflation = price_inflation.rename(columns={'Value':␣
↪'Food_Price_Inflation_Value', 'Item': 'Inflation_Item'})

# normalise units by converting index values into percentages
# base year is 2015 (2015 = 100)
base_year = 2015
# get base year values
base_values = food_index_prices[food_index_prices['Year'] ==␣
↪base_year][['Area', 'Food_Indices_Value']].
↪rename(columns={'Food_Indices_Value': 'Base_Year_Value'})
food_index_prices = pd.merge(food_index_prices, base_values, on='Area')
# calculate percentage change with respect to base year
food_index_prices['Food_Indices_Percentage'] =␣
↪(food_index_prices['Food_Indices_Value'] /␣
↪food_index_prices['Base_Year_Value'] - 1) * 100
food_index_prices = food_index_prices.drop(columns=['Food_Indices_Value',␣
↪'Base_Year_Value'])

cpi_df = pd.merge(price_inflation, food_index_prices, on=['Area', 'Year'])
cpi_df = pd.merge(cpi_df, crop_exports_summed[['Year', 'Area',␣
↪'Annual_Crop_Export_Value']], on=['Area', 'Year'])
# normalise
value_cols = ['Food_Indices_Percentage', 'Food_Price_Inflation_Value',␣
↪'Annual_Crop_Export_Value']
cpi_df[value_cols] = standard_scaler.fit_transform(cpi_df[value_cols])

# find linear correlation between features and crop export values
correlation_export_crop_yield = cpi_df[value_cols].corr()
display_heatmap(correlation_export_crop_yield, title='Consumer Price Features␣
↪and Crop Values')

# find non-linear correlation between features and crop export values
X = cpi_df[['Food_Indices_Percentage', 'Food_Price_Inflation_Value']]
y = cpi_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Consumer Prices')
```

1.2.2 crops_production_indicators.csv

[ ]:
```
production_indicators = pd.read_csv('Data/raw_data/crops_production_indicators.
↪csv')
```

```
production_indicators = production_indicators.drop(columns=['Domain Code',␣
↪'Domain',
                                                            'Area Code (M49)',␣
↪'Element Code',
                                                            'Element', 'Item␣
↪Code (CPC)',
                                                            'Year Code', 'Unit',
                                                            'Flag', 'Flag␣
↪Description',
                                                            'Note'])
production_indicators = production_indicators[(production_indicators['Year'] >=␣
↪2002) & (production_indicators['Year'] <= 2022)]

unique_items = production_indicators['Item'].unique()
print(f'Unique Items: {unique_items}')
```

[ ]:
```
# sort all production indicators by product and save in separate dfs
cereals = production_indicators[production_indicators['Item'] ==␣
↪unique_items[0]].copy() # cereals and prep
citrus = production_indicators[production_indicators['Item'] ==␣
↪unique_items[1]].copy() # fruit and veg
materials = production_indicators[production_indicators['Item'] ==␣
↪unique_items[2]].copy() # non-food
fruit = production_indicators[production_indicators['Item'] == unique_items[3]].
↪copy() # fruit and veg
oil_crops_cake = production_indicators[production_indicators['Item'] ==␣
↪unique_items[4]].copy() # fats and oils
oil_crops_oil = production_indicators[production_indicators['Item'] ==␣
↪unique_items[5]].copy() # fats and oils
pulses = production_indicators[production_indicators['Item'] ==␣
↪unique_items[6]].copy() # fruit and veg
sugars = production_indicators[production_indicators['Item'] ==␣
↪unique_items[7]].copy() # sugar and honey
roots_and_tubers = production_indicators[production_indicators['Item'] ==␣
↪unique_items[8]].copy() # fruit and veg
vegetables = production_indicators[production_indicators['Item'] ==␣
↪unique_items[10]].copy() # fruit and veg
```

[ ]:
```
# match Item descriptions to crop_exports_by_prod df
cereals.loc[:, 'Item'] = crop_products_dict['cereals']
materials.loc[:, 'Item'] = crop_products_dict['materials']
sugars.loc[:, 'Item'] = crop_products_dict['sugar']
oil_crops_cake.loc[:, 'Item'] = crop_products_dict['fats']
oil_crops_oil.loc[:, 'Item'] = crop_products_dict['fats']
citrus.loc[:, 'Item'] = crop_products_dict['fruit and veg']
fruit.loc[:, 'Item'] = crop_products_dict['fruit and veg']
```

```
pulses.loc[:, 'Item'] = crop_products_dict['fruit and veg']
roots_and_tubers.loc[:, 'Item'] = crop_products_dict['fruit and veg']
vegetables.loc[:, 'Item'] = crop_products_dict['fruit and veg']

# merge production indicators that are in the same category
fruit_and_veg = pd.concat([citrus, fruit, pulses, roots_and_tubers, vegetables])
fats = pd.concat([oil_crops_cake, oil_crops_oil])

fruit_and_veg = fruit_and_veg.groupby(['Area', 'Year', 'Item'],␣
 ↪as_index=False)['Value'].sum()
fats = fats.groupby(['Area', 'Year', 'Item'], as_index=False)['Value'].sum()
```

[ ]:
```
# separate crop products into their own dfs
cereal_exports = crop_exports_by_prod[crop_exports_by_prod['Item'] ==␣
 ↪crop_products_dict['cereals']]
cereal_exports = cereal_exports.rename(columns={'Crop_Export_Value':␣
 ↪'Cereal_Export_Value'})

fruit_and_veg_exports = crop_exports_by_prod[crop_exports_by_prod['Item'] ==␣
 ↪crop_products_dict['fruit and veg']]
fruit_and_veg_exports = fruit_and_veg_exports.
 ↪rename(columns={'Crop_Export_Value': 'Fruit_And_Veg_Export_Value'})

non_food_exports = crop_exports_by_prod[crop_exports_by_prod['Item'] ==␣
 ↪crop_products_dict['materials']]
non_food_exports = non_food_exports.rename(columns={'Crop_Export_Value':␣
 ↪'Non_Food_Export_Value'})

fats_exports = crop_exports_by_prod[crop_exports_by_prod['Item'] ==␣
 ↪crop_products_dict['fats']]
fats_exports = fats_exports.rename(columns={'Crop_Export_Value':␣
 ↪'Fats_Export_Value'})

sugars_exports = crop_exports_by_prod[crop_exports_by_prod['Item'] ==␣
 ↪crop_products_dict['sugar']]
sugars_exports = sugars_exports.rename(columns={'Crop_Export_Value':␣
 ↪'Sugars_Export_Value'})
```

[ ]:
```
cereals = cereals.rename(columns={'Value': 'Cereals_Value'})
materials = materials.rename(columns={'Value': 'Materials_Value'})
sugars = sugars.rename(columns={'Value': 'Sugars_Value'})
fats = fats.rename(columns={'Value': 'Fats_Value'})
fruit_and_veg = fruit_and_veg.rename(columns={'Value': 'Fruit_And_Veg_Value'})
```

[ ]:
```
def production_indicators_heatmap(indicator_df, indicator_value, export_df,␣
 ↪heatmap_title):
```

```
    merged = pd.merge(indicator_df, export_df, on=['Area', 'Year'])
    merged[['Export_Value', 'Production_Value']] = min_max_scaler.
↪fit_transform(merged[['Crop_Export_Value', f'{indicator_value}']])
    correlation = merged[['Export_Value', 'Production_Value']].corr()
    display_heatmap(correlation, heatmap_title)

# # find linear correlation between features and crop export values
# production_indicators_heatmap(cereals, 'Cereals_Value', cereal_exports,␣
↪'Cereal Production and Cereal Export Value')
# production_indicators_heatmap(materials, 'Materials_Value', non_food_exports,␣
↪'Material Production and Material Export Value')
# production_indicators_heatmap(sugars, 'Sugars_Value', sugars_exports, 'Sugar␣
↪and Honey Production and Sugar and Honey Export Value')
# production_indicators_heatmap(fats, 'Fats_Value', fats_exports, 'Fats and␣
↪Oils Production and Fats and Oils Export Value')
# production_indicators_heatmap(fruit_and_veg, 'Fruit_And_Veg_Value',␣
↪fruit_and_veg_exports, 'Fruit and Vegetable Production and Fruit and␣
↪Vegetable Export Value')

# find non-linear correlation between features and crop export values
prod_indicators_df = pd.merge(cereals, materials, on=['Area', 'Year'])
prod_indicators_df = pd.merge(prod_indicators_df, sugars, on=['Area', 'Year'])
prod_indicators_df = pd.merge(prod_indicators_df, fats, on=['Area', 'Year'])
prod_indicators_df = pd.merge(prod_indicators_df, fruit_and_veg, on=['Area',␣
↪'Year'])
prod_indicators_df = pd.merge(prod_indicators_df, cereal_exports[['Area',␣
↪'Year', 'Cereal_Export_Value']], on=['Area', 'Year'])
prod_indicators_df = pd.merge(prod_indicators_df, non_food_exports[['Area',␣
↪'Year', 'Non_Food_Export_Value']], on=['Area', 'Year'])
prod_indicators_df = pd.merge(prod_indicators_df, sugars_exports[['Area',␣
↪'Year', 'Sugars_Export_Value']], on=['Area', 'Year'])
prod_indicators_df = pd.merge(prod_indicators_df, fats_exports[['Area', 'Year',␣
↪'Fats_Export_Value']], on=['Area', 'Year'])
prod_indicators_df = pd.merge(prod_indicators_df,␣
↪fruit_and_veg_exports[['Area', 'Year', 'Fruit_And_Veg_Export_Value']],␣
↪on=['Area', 'Year'])
prod_indicators_df = pd.merge(prod_indicators_df, crop_exports_summed[['Year',␣
↪'Area', 'Annual_Crop_Export_Value']], on=['Area', 'Year'])

# normalise
value_cols = ['Cereals_Value', 'Materials_Value', 'Sugars_Value', 'Fats_Value',
              'Fruit_And_Veg_Value', 'Cereal_Export_Value',␣
↪'Non_Food_Export_Value',
              'Sugars_Export_Value',  'Fats_Export_Value',␣
↪'Fruit_And_Veg_Export_Value', 'Annual_Crop_Export_Value']
```

```
prod_indicators_df[value_cols] = standard_scaler.
 ↪fit_transform(prod_indicators_df[value_cols])

# find linear correlation between features and crop export values
correlation = prod_indicators_df[value_cols].corr()
display_heatmap(correlation, title='Crop Production Indicators and Crop Values')

# find non-linear correlation between features and crop export values
X = prod_indicators_df[['Cereals_Value', 'Materials_Value', 'Sugars_Value',␣
 ↪'Fats_Value',
              'Fruit_And_Veg_Value', 'Cereal_Export_Value',␣
 ↪'Non_Food_Export_Value',
              'Sugars_Export_Value',  'Fats_Export_Value',␣
 ↪'Fruit_And_Veg_Export_Value']]
y = prod_indicators_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Crop Production Indicators')
```

1.2.3 emissions.csv

[ ]:
```
emissions = pd.read_csv('Data/raw_data/emissions.csv')
emissions = emissions.drop(columns=['Domain Code', 'Domain',
                                                            'Area Code (M49)',␣
 ↪'Element Code',
                                                            'Item Code (CPC)',␣
 ↪'Year Code',
                                                            'Unit', 'Flag',
                                                            'Flag Description',␣
 ↪'Note',
                                                            'Source Code',␣
 ↪'Source'])
emissions = emissions[(emissions['Year'] >= 2002) & (emissions['Year'] <= 2022)]

unique_items = emissions['Item'].unique()
print(f'Unique Items: {unique_items}')
```

[ ]:
```
crop_emissions = emissions[emissions['Item'] == unique_items[0]].copy()
unique_elements = crop_emissions['Element'].unique()
print(f'Unique Elements: {unique_elements}')
```

[ ]:
```
n20_emissions = crop_emissions[crop_emissions['Element'] == unique_elements[0]].
 ↪copy()
ch4_emissions = crop_emissions[crop_emissions['Element'] == unique_elements[1]].
 ↪copy()
n20_emissions = n20_emissions.rename(columns={'Value': 'N20_Emissions_Value'})
ch4_emissions = ch4_emissions.rename(columns={'Value': 'CH4_Emissions_Value'})
```

```
[ ]: emissions_df = pd.merge(n20_emissions, ch4_emissions, on=['Area', 'Year'])
emissions_df = pd.merge(emissions_df, crop_exports_summed[['Year', 'Area',␣
 ↪'Annual_Crop_Export_Value']], on=['Area', 'Year'])

# normalise
value_cols = ['N20_Emissions_Value', 'CH4_Emissions_Value',␣
 ↪'Annual_Crop_Export_Value']
emissions_df[value_cols] = standard_scaler.
 ↪fit_transform(emissions_df[value_cols])

# find linear correlation between features and crop export values
correlation = emissions_df[value_cols].corr()
display_heatmap(correlation, title='Emissions and Crop Values')

# find non-linear correlation between features and crop export values
X = emissions_df[['N20_Emissions_Value', 'CH4_Emissions_Value']]
y = emissions_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Emissions')
```

1.2.4 employment_data.csv

```
[ ]: employment_data = pd.read_csv('Data/raw_data/employment_data.csv')

employment_data = employment_data.drop(columns=['Domain Code', 'Domain',
                                                'Area Code (M49)', 'Indicator␣
 ↪Code',
                                                'Sex Code', 'Sex', 'Year Code',
                                                'Element Code', 'Element',␣
 ↪'Source Code',
                                                'Flag', 'Flag Description',␣
 ↪'Note', 'Source', 'Unit'])
employment_data = employment_data[(employment_data['Year'] >= 2002) &␣
 ↪(employment_data['Year'] <= 2022)]

unique_indicators = employment_data['Indicator'].unique()
print(f'Unique Indicators: {unique_items}')
```

```
[ ]: weekly_hours = employment_data[employment_data['Indicator'] ==␣
 ↪unique_indicators[0]]
weekly_hours = weekly_hours.rename(columns={'Value': 'Mean_Weekly_Hours_Value'})

employment_estimates = employment_data[employment_data['Indicator'] ==␣
 ↪unique_indicators[1]]
employment_estimates = employment_estimates.rename(columns={'Value':␣
 ↪'Employment_Estimates_Value'})
```

```
employment_df = pd.merge(weekly_hours, employment_estimates, on=['Area',␣
 ↪'Year'])
employment_df = pd.merge(employment_df, crop_exports_summed[['Area', 'Year',␣
 ↪'Annual_Crop_Export_Value']], on=['Area', 'Year'])

# normalise
value_cols = ['Mean_Weekly_Hours_Value', 'Employment_Estimates_Value',␣
 ↪'Annual_Crop_Export_Value']
employment_df[value_cols] = standard_scaler.
 ↪fit_transform(employment_df[value_cols])

# find linear correlation between features and crop export values
correlation = employment_df[value_cols].corr()
display_heatmap(correlation, title='Employment and Crop Values')

# find non-linear correlation between features and crop export values
X = employment_df[['Mean_Weekly_Hours_Value', 'Employment_Estimates_Value']]
y = employment_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Employment Features')
```

1.2.5 exchange_rate.csv

[ ]:
```
exchange_rates = pd.read_csv('Data/raw_data/exchange_rate.csv')
# change monthly data to annual data
exchange_rates = exchange_rates.groupby(['Year', 'Area'])['Value'].mean().
 ↪reset_index()
exchange_rates = exchange_rates[(exchange_rates['Year'] >= 2002) &␣
 ↪(exchange_rates['Year'] <= 2022)]
```

[ ]:
```
exchange_rates = exchange_rates.rename(columns={'Value': 'Exchange_Rate_Value'})
exchange_rates_df = pd.merge(exchange_rates, crop_exports_summed[['Area',␣
 ↪'Year', 'Annual_Crop_Export_Value']], on=['Year', 'Area'])

# normalise
value_cols = ['Exchange_Rate_Value', 'Annual_Crop_Export_Value']
exchange_rates_df[value_cols] = standard_scaler.
 ↪fit_transform(exchange_rates_df[value_cols])

# find linear correlation between features and crop export values
correlation = exchange_rates_df[value_cols].corr()
display_heatmap(correlation, title='Exchange Rates and Crop Values')

# find non-linear correlation between features and crop export values
X = exchange_rates_df['Exchange_Rate_Value'].values.reshape(-1, 1)
y = exchange_rates_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Employment Features', 'Exchange_Rate_Value')
```

1.2.6 fertilizers.csv

[ ]:
```python
fertilisers = pd.read_csv('Data/raw_data/fertilizers.csv')
fertilisers = fertilisers.drop(columns=['Domain Code', 'Domain', 'Area Code␣
↪(M49)', 'Element Code',
                                        'Element', 'Item Code', 'Year Code',
                                        'Unit', 'Flag', 'Flag Description'])
fertilisers = fertilisers[(fertilisers['Year'] >= 2002) & (fertilisers['Year']␣
↪<= 2022)]

unique_items = fertilisers['Item'].unique()
print(f'Unique Items: {unique_items}')
```

[ ]:
```python
npk = fertilisers[fertilisers['Item'] == unique_items[0]].copy()
npk = npk.rename(columns={'Value': 'NPK_Value', 'Item': 'NPK_Item'})

urea = fertilisers[fertilisers['Item'] == unique_items[1]].copy()
urea = urea.rename(columns={'Value': 'Urea_Value', 'Item': 'Urea_Item'})

an = fertilisers[fertilisers['Item'] == unique_items[2]].copy()
an = an.rename(columns={'Value': 'Ammonium_Nitrate_Value', 'Item': 'AN_Item'})

amm_sulphate = fertilisers[fertilisers['Item'] == unique_items[3]].copy()
amm_sulphate = amm_sulphate.rename(columns={'Value': 'Ammonium_Sulphate_Value',
                                            'Item': 'Ammonium_Sulphate_Item'})

can = fertilisers[fertilisers['Item'] == unique_items[4]].copy()
can = can.rename(columns={'Value': 'CAN_Value', 'Item': 'CAN_Item'})


dap = fertilisers[fertilisers['Item'] == unique_items[5]].copy()
dap = dap.rename(columns={'Value': 'DAP_Value', 'Item': 'DAP_Item'})

map = fertilisers[fertilisers['Item'] == unique_items[6]].copy()
map = map.rename(columns={'Value': 'MAP_Value', 'Item': 'MAP_Item'})

other_np = fertilisers[fertilisers['Item'] == unique_items[7]].copy()
other_np = other_np.rename(columns={'Value': 'Other_NP_Compounds_Value',
                                    'Item': 'Other_NP_Compounds_Item'})

pk_compounds = fertilisers[fertilisers['Item'] == unique_items[8]].copy()
pk_compounds = pk_compounds.rename(columns={'Value': 'PK_Compounds_Value',
                                            'Item': 'PK_Compounds_Item'})

mop = fertilisers[fertilisers['Item'] == unique_items[9]].copy()
mop = mop.rename(columns={'Value': 'MOP_Value', 'Item': 'MOP_Item'})
```

```
pot_nitrate = fertilisers[fertilisers['Item'] == unique_items[10]].copy()
pot_nitrate = pot_nitrate.rename(columns={'Value': 'Potassium_Nitrate_Value',
                                          'Item': 'Potassium_Nitrate_Item'})

sop = fertilisers[fertilisers['Item'] == unique_items[11]].copy()
sop = sop.rename(columns={'Value': 'SOP_Value', 'Item': 'SOP_Item'})

sod_nitrate = fertilisers[fertilisers['Item'] == unique_items[12]].copy()
sod_nitrate = sod_nitrate.rename(columns={'Value': 'Sodium_Nitrate_Value',
                                          'Item': 'Sodium_Nitrate_Item'})

superphosphates_35 = fertilisers[fertilisers['Item'] == unique_items[13]].copy()
superphosphates_35 = superphosphates_35.rename(columns={'Value':␣
 ↪'Superphosphates_Above_35%_Value',
                                                        'Item':␣
 ↪'Superphosphates_Above_35%_Item'})

superphosphates_other = fertilisers[fertilisers['Item'] == unique_items[14]].
 ↪copy()
superphosphates_other = superphosphates_other.rename(columns={'Value':␣
 ↪'Superphosphates_Other_Value',
                                                              'Item':␣
 ↪'Superphosphates_Other_Item'})

amm_anyhyrous = fertilisers[fertilisers['Item'] == unique_items[15]].copy()
amm_anyhyrous = amm_anyhyrous.rename(columns={'Value':␣
 ↪'Ammonia_Anhydrous_Value',
                                              'Item': 'Ammonia_Anhydrous_Item'})

phos_rock = fertilisers[fertilisers['Item'] == unique_items[16]].copy()
phos_rock = phos_rock.rename(columns={'Value': 'Phosphate_Rock_Value',
                                      'Item': 'Phosphate_Rock_Item'})

uan = fertilisers[fertilisers['Item'] == unique_items[17]].copy()
uan = uan.rename(columns={'Value': 'UAN_Value', 'Item': 'UAN_Item'})

nec = fertilisers[fertilisers['Item'] == unique_items[18]].copy()
nec = nec.rename(columns={'Value': 'Fertilisers_NEC_Value', 'Item':␣
 ↪'Fertilisers_NEC_Item'})

other_nit_nec = fertilisers[fertilisers['Item'] == unique_items[19]].copy()
other_nit_nec = other_nit_nec.rename(columns={'Value':␣
 ↪'Other_Nitrogenous_Value',
                                              'Item': 'Other_Nitrogenous_Item'})

other_phos_nec = fertilisers[fertilisers['Item'] == unique_items[20]].copy()
```

```
other_phos_nec = other_phos_nec.rename(columns={'Value':␣
↪'Other_Phosphatic_Value',
                                                'Item':␣
↪'Other_Phosphatic_Item'})

other_pots_nec = fertilisers[fertilisers['Item'] == unique_items[21]].copy()
other_pots_nec = other_pots_nec.rename(columns={'Value': 'Other_Potassic_Value',
                                                'Item': 'Other_Potassic_Item'})

other_nk = fertilisers[fertilisers['Item'] == unique_items[22]].copy()
other_nk = other_nk.rename(columns={'Value': 'Other_NK_Value',
                                    'Item': 'Other_NK_Item'})
```

[ ]:
```
fertilisers_df = pd.merge(npk, urea, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, an, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, amm_sulphate, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, can, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, dap, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, map, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, other_np, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, pk_compounds, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, mop, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, pot_nitrate, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, sop, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, sod_nitrate, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, superphosphates_35, on=['Area',␣
↪'Year'])
fertilisers_df = pd.merge(fertilisers_df, superphosphates_other, on=['Area',␣
↪'Year'])
fertilisers_df = pd.merge(fertilisers_df, amm_anyhyrous, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, phos_rock, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, uan, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, nec, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, other_nit_nec, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, other_phos_nec, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, other_pots_nec, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, other_nk, on=['Area', 'Year'])
fertilisers_df = pd.merge(fertilisers_df, crop_exports_summed[['Area', 'Year',␣
↪'Annual_Crop_Export_Value']], on=['Area', 'Year'])

# normalise
value_cols = ['NPK_Value', 'Urea_Value', 'Ammonium_Nitrate_Value',
              'Ammonium_Sulphate_Value', 'CAN_Value', 'DAP_Value',
              'MAP_Value', 'Other_NP_Compounds_Value', 'PK_Compounds_Value',
              'MOP_Value', 'Potassium_Nitrate_Value', 'SOP_Value',
              'Sodium_Nitrate_Value', 'Superphosphates_Above_35%_Value',␣
↪'Superphosphates_Other_Value',
```

```
                'Ammonia_Anhydrous_Value', 'Phosphate_Rock_Value', 'UAN_Value',
                'Fertilisers_NEC_Value', 'Other_Nitrogenous_Value',␣
 ↪'Other_Phosphatic_Value',
                'Other_Potassic_Value', 'Other_NK_Value',␣
 ↪'Annual_Crop_Export_Value']

fertilisers_df[value_cols] = standard_scaler.
 ↪fit_transform(fertilisers_df[value_cols])

# find linear correlation between features and crop export values
correlation = fertilisers_df[value_cols].corr()
display_heatmap(correlation, title='Fertiliser Usage and Crop Values',␣
 ↪figsize=(16, 12))

# find non-linear correlation between features and crop export values
X = fertilisers_df[['NPK_Value', 'Urea_Value', 'Ammonium_Nitrate_Value',
                'Ammonium_Sulphate_Value', 'CAN_Value', 'DAP_Value',
                'MAP_Value', 'Other_NP_Compounds_Value', 'PK_Compounds_Value',
                'MOP_Value', 'Potassium_Nitrate_Value', 'SOP_Value',
                'Sodium_Nitrate_Value', 'Superphosphates_Above_35%_Value',␣
 ↪'Superphosphates_Other_Value',
                'Ammonia_Anhydrous_Value', 'Phosphate_Rock_Value', 'UAN_Value',
                'Fertilisers_NEC_Value', 'Other_Nitrogenous_Value',␣
 ↪'Other_Phosphatic_Value',
                'Other_Potassic_Value', 'Other_NK_Value']]
y = fertilisers_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Fertiliser Usage')
```

1.2.7 food_balance_indicators.csv

```
[ ]: balance_indicators = pd.read_csv('Data/raw_data/food_balance_indicators.csv')
balance_indicators = balance_indicators.drop(columns=['Domain Code', 'Domain',
                                                      'Area Code (M49)',␣
 ↪'Element Code',
                                                      'Item Code (FBS)', 'Year␣
 ↪Code',
                                                      'Unit', 'Flag', 'Flag␣
 ↪Description'])
balance_indicators = balance_indicators[(balance_indicators['Year'] >= 2002) &␣
 ↪(balance_indicators['Year'] <= 2022)]

unique_items = balance_indicators['Item'].unique()
print(f'Unique Items: {unique_items}\n')

unique_elements = balance_indicators['Element'].unique()
print(f'Elements: {unique_elements}')
```

[ ]:
```
cereals = balance_indicators[balance_indicators['Item'] == unique_items[0]].
    ↪copy()
roots = balance_indicators[balance_indicators['Item'] == unique_items[1]].copy()
sugars = balance_indicators[balance_indicators['Item'] == unique_items[2]].
    ↪copy()
pulses = balance_indicators[balance_indicators['Item'] == unique_items[4]].
    ↪copy()
oil_crops = balance_indicators[balance_indicators['Item'] == unique_items[6]].
    ↪copy()
vegetable_oils = balance_indicators[balance_indicators['Item'] ==␣
    ↪unique_items[7]].copy()
vegetables = balance_indicators[balance_indicators['Item'] == unique_items[8]].
    ↪copy()
fruits = balance_indicators[balance_indicators['Item'] == unique_items[9]].
    ↪copy()
alcohol = balance_indicators[balance_indicators['Item'] == unique_items[12]].
    ↪copy()
```

[ ]:
```
cereal_imports = cereals[cereals['Element'] == unique_elements[0]].copy()
cereal_exports = cereals[cereals['Element'] == unique_elements[1]].copy()
cereal_losses = cereals[cereals['Element'] == unique_elements[2]].copy()
cereal_other = cereals[cereals['Element'] == unique_elements[3]].copy()
cereal_food = cereals[cereals['Element'] == unique_elements[4]].copy()
cereal_imports = cereal_imports.rename(columns={'Value': 'Cereal_Imports_Value',
                                                'Element':␣
    ↪'Cereal_Imports_Element',
                                                'Item': 'Cereal_Imports_Item'})
cereal_exports = cereal_exports.rename(columns={'Value': 'Cereal_Exports_Value',
                                                'Element':␣
    ↪'Cereal_Exports_Element',
                                                'Item': 'Cereal_Exports_Item'})
cereal_losses = cereal_losses.rename(columns={'Value': 'Cereal_Losses_Value',
                                              'Element':␣
    ↪'Cereal_Losses_Element',
                                              'Item': 'Cereal_Losses_Item'})
cereal_other = cereal_other.rename(columns={'Value': 'Cereal_Other_Value',
                                            'Element': 'Cereal_Other_Element',
                                            'Item': 'Cereal_Other_Item'})
cereal_food = cereal_food.rename(columns={'Value': 'Cereal_Food_Value',
                                          'Element': 'Cereal_Food_Element',
                                          'Item': 'Cereal_Food_Item'})

roots_imports = roots[roots['Element'] == unique_elements[0]].copy()
roots_exports = roots[roots['Element'] == unique_elements[1]].copy()
roots_losses = roots[roots['Element'] == unique_elements[2]].copy()
roots_other = roots[roots['Element'] == unique_elements[3]].copy()
```

```
roots_food = roots[roots['Element'] == unique_elements[4]].copy()
roots_imports = roots_imports.rename(columns={'Value': 'Roots_Imports_Value',
                                              'Element':␣
 ↪'Roots_Imports_Element',
                                              'Item': 'Roots_Imports_Item'})
roots_exports = roots_exports.rename(columns={'Value': 'Roots_Exports_Value',
                                              'Element':␣
 ↪'Roots_Exports_Element',
                                              'Item': 'Roots_Exports_Item'})
roots_losses = roots_losses.rename(columns={'Value': 'Roots_Losses_Value',
                                            'Element': 'Roots_Losses_Element',
                                            'Item': 'Roots_Losses_Item'})
roots_other = roots_other.rename(columns={'Value': 'Roots_Other_Value',
                                          'Element': 'Roots_Other_Element',
                                          'Item': 'Roots_Other_Item'})
roots_food = roots_food.rename(columns={'Value': 'Roots_Food_Value',
                                        'Element': 'Roots_Food_Element',
                                        'Item': 'Roots_Food_Item'})


sugars_imports = sugars[sugars['Element'] == unique_elements[0]].copy()
sugars_exports = sugars[sugars['Element'] == unique_elements[1]].copy()
sugars_losses = sugars[sugars['Element'] == unique_elements[2]].copy()
sugars_other = sugars[sugars['Element'] == unique_elements[3]].copy()
sugars_food = sugars[sugars['Element'] == unique_elements[4]].copy()
sugars_imports = sugars_imports.rename(columns={'Value': 'Sugars_Imports_Value',
                                                'Element':␣
 ↪'Sugars_Imports_Element',
                                                'Item': 'Sugars_Imports_Item'})
sugars_exports = sugars_exports.rename(columns={'Value': 'Sugars_Exports_Value',
                                                'Element':␣
 ↪'Sugars_Exports_Element',
                                                'Item': 'Sugars_Exports_Item'})
sugars_losses = sugars_losses.rename(columns={'Value': 'Sugars_Losses_Value',
                                              'Element':␣
 ↪'Sugars_Losses_Element',
                                              'Item': 'Sugars_Losses_Item'})
sugars_other = sugars_other.rename(columns={'Value': 'Sugars_Other_Value',
                                            'Element': 'Sugars_Other_Element',
                                            'Item': 'Sugars_Other_Item'})
sugars_food = sugars_food.rename(columns={'Value': 'Sugars_Food_Value',
                                          'Element': 'Sugars_Food_Element',
                                          'Item': 'Sugars_Food_Item'})

pulses_imports = pulses[pulses['Element'] == unique_elements[0]].copy()
pulses_exports = pulses[pulses['Element'] == unique_elements[1]].copy()
pulses_losses = pulses[pulses['Element'] == unique_elements[2]].copy()
```

```
pulses_other = pulses[pulses['Element'] == unique_elements[3]].copy()
pulses_food = pulses[pulses['Element'] == unique_elements[4]].copy()
pulses_imports = pulses_imports.rename(columns={'Value': 'Pulses_Imports_Value',
                                               'Element':␣
↪'Pulses_Imports_Element',
                                               'Item': 'Pulses_Import_Item'})
pulses_exports = pulses_exports.rename(columns={'Value': 'Pulses_Exports_Value',
                                               'Element':␣
↪'Pulses_Exports_Element',
                                               'Item': 'Pulses_Exports_Item'})
pulses_losses = pulses_losses.rename(columns={'Value': 'Pulses_Losses_Value',
                                             'Element':␣
↪'Pulses_Losses_Element',
                                             'Item': 'Pulses_Losses_Item'})
pulses_other = pulses_other.rename(columns={'Value': 'Pulses_Other_Value',
                                           'Element': 'Pulses_Other_Element',
                                           'Item': 'Pulses_Other_Item'})
pulses_food = pulses_food.rename(columns={'Value': 'Pulses_Food_Value',
                                         'Element': 'Pulses_Food_Element',
                                         'Item': 'Pulses_Food_Item'})

oil_crops_imports = oil_crops[oil_crops['Element'] == unique_elements[0]].copy()
oil_crops_exports = oil_crops[oil_crops['Element'] == unique_elements[1]].copy()
oil_crops_losses = oil_crops[oil_crops['Element'] == unique_elements[2]].copy()
oil_crops_other = oil_crops[oil_crops['Element'] == unique_elements[3]].copy()
oil_crops_food = oil_crops[oil_crops['Element'] == unique_elements[4]].copy()
oil_crops_imports = oil_crops_imports.rename(columns={'Value':␣
↪'Oil_Crops_Imports_Value',
                                                     'Element':␣
↪'Oil_Crops_Imports_Element',
                                                     'Item':␣
↪'Oil_Crops_Import_Item'})
oil_crops_exports = oil_crops_exports.rename(columns={'Value':␣
↪'Oil_Crops_Exports_Value',
                                                     'Element':␣
↪'Oil_Crops_Exports_Element',
                                                     'Item':␣
↪'Oil_Crops_Exports_Item'})
oil_crops_losses = oil_crops_losses.rename(columns={'Value':␣
↪'Oil_Crops_Losses_Value',
                                                   'Element':␣
↪'Oil_Crops_Losses_Element',
                                                   'Item':␣
↪'Oil_Crops_Losses_Item'})
oil_crops_other = oil_crops_other.rename(columns={'Value':␣
↪'Oil_Crops_Other_Value',
```

```
                                                    'Element':␣
↪'Oil_Crops_Other_Element',
                                                    'Item':␣
↪'Oil_Crops_Other_Item'})
oil_crops_food = oil_crops_food.rename(columns={'Value': 'Oil_Crops_Food_Value',
                                                  'Element':␣
↪'Oil_Crops_Food_Element',
                                                  'Item': 'Oil_Crops_Food_Item'})

vegetable_oils_imports = vegetable_oils[vegetable_oils['Element'] ==␣
↪unique_elements[0]].copy()
vegetable_oils_exports = vegetable_oils[vegetable_oils['Element'] ==␣
↪unique_elements[1]].copy()
vegetable_oils_losses = vegetable_oils[vegetable_oils['Element'] ==␣
↪unique_elements[2]].copy()
vegetable_oils_other = vegetable_oils[vegetable_oils['Element'] ==␣
↪unique_elements[3]].copy()
vegetable_oils_food = vegetable_oils[vegetable_oils['Element'] ==␣
↪unique_elements[4]].copy()
vegetable_oils_imports = vegetable_oils_imports.rename(columns={'Value':␣
↪'Vegetable_Oils_Imports_Value',
                                                                'Element':␣
↪'Vegetable_Oils_Imports_Element',
                                                                'Item':␣
↪'Vegetable_Oils_Imports_Item'})
vegetable_oils_exports = vegetable_oils_exports.rename(columns={'Value':␣
↪'Vegetable_Oils_Exports_Value',
                                                                'Element':␣
↪'Vegetable_Oils_Exports_Element',
                                                                'Item':␣
↪'Vegetable_Oils_Exports_Item'})
vegetable_oils_losses = vegetable_oils_losses.rename(columns={'Value':␣
↪'Vegetable_Oils_Losses_Value',
                                                              'Element':␣
↪'Vegetable_Oils_Losses_Element',
                                                              'Item':␣
↪'Vegetable_Oils_Losses_Item'})
vegetable_oils_other = vegetable_oils_other.rename(columns={'Value':␣
↪'Vegetable_Oils_Other_Value',
                                                            'Element':␣
↪'Vegetable_Oils_Other_Element',
                                                            'Item':␣
↪'Vegetable_Oils_Other_Item'})
vegetable_oils_food = vegetable_oils_food.rename(columns={'Value':␣
↪'Vegetable_Oils_Food_Value',
```

```
                                                          'Element':␣
↪'Vegetable_Oils_Food_Element',
                                                          'Item':␣
↪'Vegetable_Oils_Food_Item'})

vegetables_imports = vegetables[vegetables['Element'] == unique_elements[0]].
↪copy()
vegetables_exports = vegetables[vegetables['Element'] == unique_elements[1]].
↪copy()
vegetables_losses = vegetables[vegetables['Element'] == unique_elements[2]].
↪copy()
vegetables_other = vegetables[vegetables['Element'] == unique_elements[3]].
↪copy()
vegetables_food = vegetables[vegetables['Element'] == unique_elements[4]].copy()
vegetables_imports = vegetables_imports.rename(columns={'Value':␣
↪'Vegetables_Imports_Value',
                                                        'Element':␣
↪'Vegetables_Imports_Element',
                                                        'Item':␣
↪'Vegetables_Imports_Item'})
vegetables_exports = vegetables_exports.rename(columns={'Value':␣
↪'Vegetables_Exports_Value',
                                                        'Element':␣
↪'Vegetables_Exports_Element',
                                                        'Item':␣
↪'Vegetables_Exports_Item'})
vegetables_losses = vegetables_losses.rename(columns={'Value':␣
↪'Vegetables_Losses_Value',
                                                      'Element':␣
↪'Vegetables_Losses_Element',
                                                      'Item':␣
↪'Vegetables_Losses_Item'})
vegetables_other = vegetables_other.rename(columns={'Value':␣
↪'Vegetables_Other_Value',
                                                    'Element':␣
↪'Vegetables_Other_Element',
                                                    'Item':␣
↪'Vegetables_Other_Item'})
vegetables_food = vegetables_food.rename(columns={'Value':␣
↪'Vegetables_Food_Value',
                                                  'Element':␣
↪'Vegetables_Food_Element',
                                                  'Item':␣
↪'Vegetables_Food_Item'})
```

```
fruits_imports = fruits[fruits['Element'] == unique_elements[0]].copy()
fruits_exports = fruits[fruits['Element'] == unique_elements[1]].copy()
fruits_losses = fruits[fruits['Element'] == unique_elements[2]].copy()
fruits_other = fruits[fruits['Element'] == unique_elements[3]].copy()
fruits_food = fruits[fruits['Element'] == unique_elements[4]].copy()
fruits_imports = fruits_imports.rename(columns={'Value': 'Fruits_Imports_Value',
                                                'Element':␣
 ↪'Fruits_Imports_Element',
                                                'Item': 'Fruits_Imports_Item'})
fruits_exports = fruits_exports.rename(columns={'Value': 'Fruits_Exports_Value',
                                                'Element':␣
 ↪'Fruits_Exports_Element',
                                                'Item': 'Fruits_Exports_Item'})
fruits_losses = fruits_losses.rename(columns={'Value': 'Fruits_Losses_Value',
                                              'Element':␣
 ↪'Fruits_Losses_Element',
                                              'Item': 'Fruits_Losses_Item'})
fruits_other = fruits_other.rename(columns={'Value': 'Fruits_Other_Value',
                                            'Element': 'Fruits_Other_Element',
                                            'Item': 'Fruits_Other_Item'})
fruits_food = fruits_food.rename(columns={'Value': 'Fruits_Food_Value',
                                          'Element': 'Fruits_Food_Element',
                                          'Item': 'Fruits_Food_Item'})

alcohol_imports = alcohol[alcohol['Element'] == unique_elements[0]].copy()
alcohol_exports = alcohol[alcohol['Element'] == unique_elements[1]].copy()
alcohol_losses = alcohol[alcohol['Element'] == unique_elements[2]].copy()
alcohol_other = alcohol[alcohol['Element'] == unique_elements[3]].copy()
alcohol_food = alcohol[alcohol['Element'] == unique_elements[4]].copy()
alcohol_imports = alcohol_imports.rename(columns={'Value':␣
 ↪'Alcohol_Imports_Value',
                                                  'Element':␣
 ↪'Alcohol_Imports_Element',
                                                  'Item':␣
 ↪'Alcohol_Imports_Item'})
alcohol_exports = alcohol_exports.rename(columns={'Value':␣
 ↪'Alcohol_Exports_Value',
                                                  'Element':␣
 ↪'Alcohol_Exports_Element',
                                                  'Item':␣
 ↪'Alcohol_Exports_Item'})
alcohol_losses = alcohol_losses.rename(columns={'Value': 'Alcohol_Losses_Value',
                                                'Element':␣
 ↪'Alcohol_Losses_Element',
                                                'Item': 'Alcohol_Losses_Item'})
alcohol_other = alcohol_other.rename(columns={'Value': 'Alcohol_Other_Value',
```

```
                                                     'Element':␣
↪'Alcohol_Other_Element',
                                                     'Item': 'Alcohol_Other_Item'})
alcohol_food = alcohol_food.rename(columns={'Value': 'Alcohol_Food_Value',
                                           'Element': 'Alcohol_Food_Element',
                                           'Item': 'Alcohol_Food_Item'})
```

[ ]:
```
imports_df = pd.merge(cereal_imports, roots_imports, on=['Area', 'Year'])
imports_df = pd.merge(imports_df, sugars_imports, on=['Area', 'Year'])
imports_df = pd.merge(imports_df, pulses_imports, on=['Area', 'Year'])
imports_df = pd.merge(imports_df, oil_crops_imports, on=['Area', 'Year'])
imports_df = pd.merge(imports_df, vegetable_oils_imports, on=['Area', 'Year'])
imports_df = pd.merge(imports_df, fruits_imports, on=['Area', 'Year'])
imports_df = pd.merge(imports_df, alcohol_imports, on=['Area', 'Year'])
imports_df = pd.merge(imports_df, crop_exports_summed[['Area', 'Year',␣
↪'Annual_Crop_Export_Value']], on=['Area', 'Year'])

imports_value_cols = ['Cereal_Imports_Value', 'Roots_Imports_Value',␣
↪'Sugars_Imports_Value',
                   'Pulses_Imports_Value', 'Oil_Crops_Imports_Value',␣
↪'Vegetable_Oils_Imports_Value',
                   'Fruits_Imports_Value', 'Alcohol_Imports_Value',␣
↪'Annual_Crop_Export_Value']
imports_df[imports_value_cols] = standard_scaler.
↪fit_transform(imports_df[imports_value_cols])

# find linear correlation between features and crop export values
correlation = imports_df[imports_value_cols].corr()
display_heatmap(correlation, title='Imported Crops and Crop Values')
# find non-linear correlation between features and crop export values
X = imports_df[['Cereal_Imports_Value', 'Roots_Imports_Value',␣
↪'Sugars_Imports_Value',
                   'Pulses_Imports_Value', 'Oil_Crops_Imports_Value',␣
↪'Vegetable_Oils_Imports_Value',
                   'Fruits_Imports_Value', 'Alcohol_Imports_Value']]
y = imports_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Imported Crops')

exports_df = pd.merge(cereal_exports, roots_exports, on=['Area', 'Year'])
exports_df = pd.merge(exports_df, sugars_exports, on=['Area', 'Year'])
exports_df = pd.merge(exports_df, pulses_exports, on=['Area', 'Year'])
exports_df = pd.merge(exports_df, oil_crops_exports, on=['Area', 'Year'])
exports_df = pd.merge(exports_df, vegetable_oils_exports, on=['Area', 'Year'])
exports_df = pd.merge(exports_df, fruits_exports, on=['Area', 'Year'])
exports_df = pd.merge(exports_df, alcohol_exports, on=['Area', 'Year'])
```

```python
exports_df = pd.merge(exports_df, crop_exports_summed[['Area', 'Year', 'Annual_Crop_Export_Value']], on=['Area', 'Year'])

exports_value_cols = ['Cereal_Exports_Value', 'Roots_Exports_Value', 'Sugars_Exports_Value',
                      'Pulses_Exports_Value', 'Oil_Crops_Exports_Value', 'Vegetable_Oils_Exports_Value',
                      'Fruits_Exports_Value', 'Alcohol_Exports_Value', 'Annual_Crop_Export_Value']
exports_df[exports_value_cols] = standard_scaler.fit_transform(exports_df[exports_value_cols])

# find linear correlation between features and crop export values
correlation = exports_df[exports_value_cols].corr()
display_heatmap(correlation, title='Exported Crops and Export Crop Values')
# find non-linear correlation between features and crop export values
X = exports_df[['Cereal_Exports_Value', 'Roots_Exports_Value', 'Sugars_Exports_Value',
                      'Pulses_Exports_Value', 'Oil_Crops_Exports_Value', 'Vegetable_Oils_Exports_Value',
                      'Fruits_Exports_Value', 'Alcohol_Exports_Value']]
y = exports_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Exported Crops')

losses_df = pd.merge(cereal_losses, roots_losses, on=['Area', 'Year'])
losses_df = pd.merge(losses_df, sugars_losses, on=['Area', 'Year'])
losses_df = pd.merge(losses_df, pulses_losses, on=['Area', 'Year'])
losses_df = pd.merge(losses_df, oil_crops_losses, on=['Area', 'Year'])
losses_df = pd.merge(losses_df, vegetable_oils_losses, on=['Area', 'Year'])
losses_df = pd.merge(losses_df, fruits_losses, on=['Area', 'Year'])
losses_df = pd.merge(losses_df, alcohol_losses, on=['Area', 'Year'])
losses_df = pd.merge(losses_df, crop_exports_summed[['Area', 'Year', 'Annual_Crop_Export_Value']], on=['Area', 'Year'])

losses_value_cols = ['Cereal_Losses_Value', 'Roots_Losses_Value', 'Sugars_Losses_Value',
                      'Pulses_Losses_Value', 'Oil_Crops_Losses_Value', 'Vegetable_Oils_Losses_Value',
                      'Fruits_Losses_Value', 'Alcohol_Losses_Value', 'Annual_Crop_Export_Value']
losses_df[losses_value_cols] = standard_scaler.fit_transform(losses_df[losses_value_cols])

# find linear correlation between features and crop export values
correlation = losses_df[losses_value_cols].corr()
display_heatmap(correlation, title='Crop Losses and Export Crop Values')
```

```
# find non-linear correlation between features and crop export values
X = losses_df[['Cereal_Losses_Value', 'Roots_Losses_Value',␣
 ↪'Sugars_Losses_Value',
                   'Pulses_Losses_Value', 'Oil_Crops_Losses_Value',␣
 ↪'Vegetable_Oils_Losses_Value',
                   'Fruits_Losses_Value', 'Alcohol_Losses_Value']]
y = losses_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Crop Losses')

other_df = pd.merge(cereal_other, roots_other, on=['Area', 'Year'])
other_df = pd.merge(other_df, sugars_other, on=['Area', 'Year'])
other_df = pd.merge(other_df, pulses_other, on=['Area', 'Year'])
other_df = pd.merge(other_df, oil_crops_other, on=['Area', 'Year'])
other_df = pd.merge(other_df, vegetable_oils_other, on=['Area', 'Year'])
other_df = pd.merge(other_df, fruits_other, on=['Area', 'Year'])
other_df = pd.merge(other_df, alcohol_other, on=['Area', 'Year'])
other_df = pd.merge(other_df, crop_exports_summed[['Area', 'Year',␣
 ↪'Annual_Crop_Export_Value']], on=['Area', 'Year'])

other_value_cols = ['Cereal_Other_Value', 'Roots_Other_Value',␣
 ↪'Sugars_Other_Value',
                   'Pulses_Other_Value', 'Oil_Crops_Other_Value',␣
 ↪'Vegetable_Oils_Other_Value',
                   'Fruits_Other_Value', 'Alcohol_Other_Value',␣
 ↪'Annual_Crop_Export_Value']
other_df[other_value_cols] = standard_scaler.
 ↪fit_transform(other_df[other_value_cols])

# find linear correlation between features and crop export values
correlation = other_df[other_value_cols].corr()
display_heatmap(correlation, title='Crop (Other) and Export Crop Values')
# find non-linear correlation between features and crop export values
X = other_df[['Cereal_Other_Value', 'Roots_Other_Value', 'Sugars_Other_Value',
                   'Pulses_Other_Value', 'Oil_Crops_Other_Value',␣
 ↪'Vegetable_Oils_Other_Value',
                   'Fruits_Other_Value', 'Alcohol_Other_Value']]
y = other_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Crop (Other)')

food_df = pd.merge(cereal_food, roots_food, on=['Area', 'Year'])
food_df = pd.merge(food_df, sugars_food, on=['Area', 'Year'])
food_df = pd.merge(food_df, pulses_food, on=['Area', 'Year'])
food_df = pd.merge(food_df, oil_crops_food, on=['Area', 'Year'])
food_df = pd.merge(food_df, vegetable_oils_food, on=['Area', 'Year'])
food_df = pd.merge(food_df, fruits_food, on=['Area', 'Year'])
food_df = pd.merge(food_df, alcohol_food, on=['Area', 'Year'])
```

```
food_df = pd.merge(food_df, crop_exports_summed[['Area', 'Year',␣
↪'Annual_Crop_Export_Value']], on=['Area', 'Year'])

food_value_cols = ['Cereal_Food_Value', 'Roots_Food_Value', 'Sugars_Food_Value',
                   'Pulses_Food_Value', 'Oil_Crops_Food_Value',␣
↪'Vegetable_Oils_Food_Value',
                   'Fruits_Food_Value', 'Alcohol_Food_Value',␣
↪'Annual_Crop_Export_Value']
food_df[food_value_cols] = standard_scaler.
↪fit_transform(food_df[food_value_cols])

# find linear correlation between features and crop export values
correlation = food_df[food_value_cols].corr()
display_heatmap(correlation, title='Crops for Food and Crop Values')
# find non-linear correlation between features and crop export values
X = food_df[['Cereal_Food_Value', 'Roots_Food_Value', 'Sugars_Food_Value',
                   'Pulses_Food_Value', 'Oil_Crops_Food_Value',␣
↪'Vegetable_Oils_Food_Value',
                   'Fruits_Food_Value', 'Alcohol_Food_Value']]
y = food_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Crops for Food')
```

1.2.8 food_security_indicator.csv

[13]:
```
security_indicators = pd.read_csv('Data/raw_data/food_security_indicators.csv')
security_indicators = security_indicators.drop(columns=['Domain Code', 'Domain',
                                                        'Area Code (M49)',␣
↪'Element', 'Element Code',
                                                        'Item Code', 'Year␣
↪Code',
                                                        'Unit', 'Flag', 'Flag␣
↪Description', 'Note'])

# data has years in format str(x-y)
# take the y value so more rows fit with the years bounded 2002-2022
security_indicators['Year'] = security_indicators['Year'].apply(lambda x: int(x.
↪split('-')[1] if '-' in x else int(x)))
# convert from string to integer
security_indicators['Year'] = security_indicators['Year'].astype(int)

security_indicators = security_indicators[(security_indicators['Year'] >= 2002)␣
↪& (security_indicators['Year'] <= 2022)]

unique_items = security_indicators['Item'].unique()
print(f'Unique Items: {unique_items}\n')
```

```
Unique Items: ['Average dietary energy supply adequacy (percent) (3-year
```

```
average)'
 'Average protein supply (g/cap/day) (3-year average)'
 'Cereal import dependency ratio (percent) (3-year average)'
 'Percent of arable land equipped for irrigation (percent) (3-year average)'
 'Value of food imports in total merchandise exports (percent) (3-year average)'
 'Political stability and absence of violence/terrorism (index)'
 'Per capita food production variability (constant 2014-2016 thousand int$ per
capita)'
 'Per capita food supply variability (kcal/cap/day)'
 'Prevalence of anemia among women of reproductive age (15-49 years)'
 'Prevalence of low birthweight (percent)']
```

[18]:
```python
dietary_energy = security_indicators[security_indicators['Item'] ==␣
 ↪unique_items[0]].copy()
dietary_energy = dietary_energy.rename(columns={'Value': 'Dietary_Value',␣
 ↪'Item': 'Dietary_Item'})

cereal_dependency = security_indicators[security_indicators['Item'] ==␣
 ↪unique_items[1]].copy()
cereal_dependency = cereal_dependency.rename(columns={'Value': 'Cereal_Value',␣
 ↪'Item': 'Cereal_Item'})

irrigation_land = security_indicators[security_indicators['Item'] ==␣
 ↪unique_items[2]].copy()
irrigation_land = irrigation_land.rename(columns={'Value': 'Irrigation_Value',␣
 ↪'Item': 'Irrigation_Item'})

food_imports_in_merch = security_indicators[security_indicators['Item'] ==␣
 ↪unique_items[3]].copy()
food_imports_in_merch = food_imports_in_merch.rename(columns={'Value':␣
 ↪'Imports_Value', 'Item': 'Imports_Item'})

stability = security_indicators[security_indicators['Item'] == unique_items[4]].
 ↪copy()
stability = stability.rename(columns={'Value': 'Stability_Value', 'Item':␣
 ↪'Stability_Item'})

food_prod_variability = security_indicators[security_indicators['Item'] ==␣
 ↪unique_items[5]].copy()
food_prod_variability = food_prod_variability.rename(columns={'Value':␣
 ↪'Prod_Value', 'Item': 'Prod_Item'})

food_supply_variability = security_indicators[security_indicators['Item'] ==␣
 ↪unique_items[6]].copy()
```

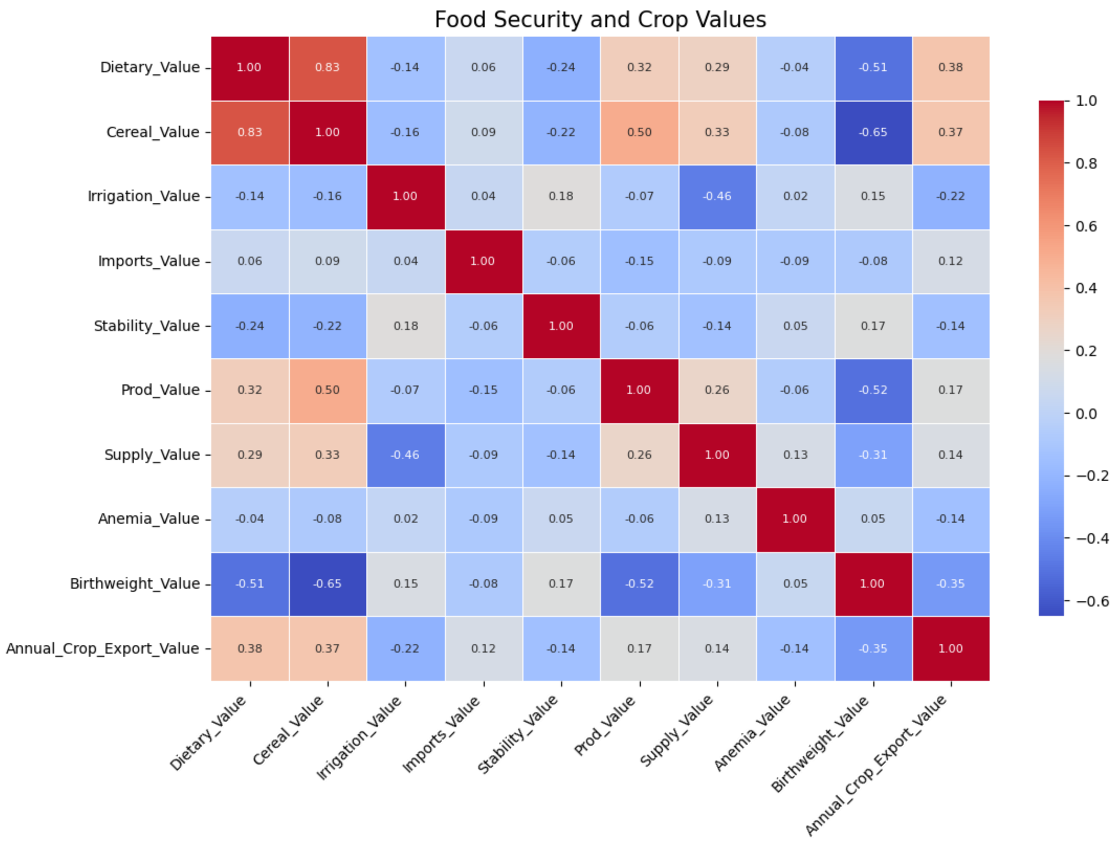
```
food_supply_variability = food_supply_variability.rename(columns={'Value':
 ↪'Supply_Value', 'Item': 'Supply_Item'})

anemia_among_women = security_indicators[security_indicators['Item'] ==
 ↪unique_items[7]].copy()
anemia_among_women = anemia_among_women.rename(columns={'Value':
 ↪'Anemia_Value', 'Item': 'Anemia_Item'})

low_birthweight = security_indicators[security_indicators['Item'] ==
 ↪unique_items[8]].copy()
low_birthweight = low_birthweight.rename(columns={'Value': 'Birthweight_Value',
 ↪'Item': 'Birthweight_Item'})
```

[15]:
```
display(stability)
```

```
             Area                                    Stability_Item  Year  \
79    Afghanistan  Value of food imports in total merchandise exp…  2002
80    Afghanistan  Value of food imports in total merchandise exp…  2003
81    Afghanistan  Value of food imports in total merchandise exp…  2004
82    Afghanistan  Value of food imports in total merchandise exp…  2005
83    Afghanistan  Value of food imports in total merchandise exp…  2006
…             …                                                …     …
36403    Zimbabwe  Value of food imports in total merchandise exp…  2017
36404    Zimbabwe  Value of food imports in total merchandise exp…  2018
36405    Zimbabwe  Value of food imports in total merchandise exp…  2019
36406    Zimbabwe  Value of food imports in total merchandise exp…  2020
36407    Zimbabwe  Value of food imports in total merchandise exp…  2021

       Stability_Value
79              240.00
80              281.00
81              199.00
82              187.00
83              175.00
…                  …
36403            25.00
36404            20.00
36405            13.00
36406            14.00
36407            15.00

[3858 rows x 4 columns]
```

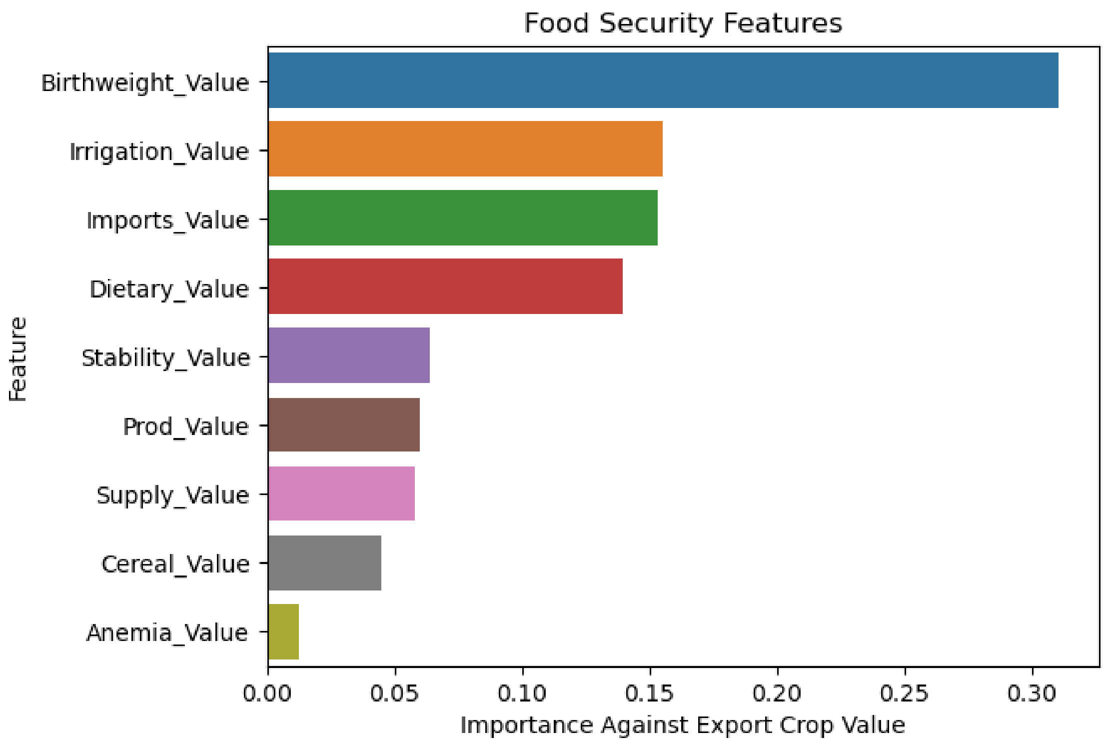
[21]:
```
food_security_df = pd.merge(dietary_energy, cereal_dependency, on=['Area',
 ↪'Year'])
food_security_df = pd.merge(food_security_df, irrigation_land, on=['Area',
 ↪'Year'])
```

```
food_security_df = pd.merge(food_security_df, food_imports_in_merch,␣
 ↪on=['Area', 'Year'])
food_security_df = pd.merge(food_security_df, stability, on=['Area', 'Year'])
food_security_df = pd.merge(food_security_df, food_prod_variability,␣
 ↪on=['Area', 'Year'])
food_security_df = pd.merge(food_security_df, food_supply_variability,␣
 ↪on=['Area', 'Year'])
food_security_df = pd.merge(food_security_df, anemia_among_women, on=['Area',␣
 ↪'Year'])
food_security_df = pd.merge(food_security_df, low_birthweight, on=['Area',␣
 ↪'Year'])
food_security_df = pd.merge(food_security_df, crop_exports_summed[['Area',␣
 ↪'Year', 'Annual_Crop_Export_Value']], on=['Area', 'Year'])

food_security_cols = ['Dietary_Value', 'Cereal_Value', 'Irrigation_Value',
                   'Imports_Value', 'Stability_Value', 'Prod_Value',
                   'Supply_Value', 'Anemia_Value', 'Birthweight_Value',
                   'Annual_Crop_Export_Value']
food_security_df[food_security_cols] = standard_scaler.
 ↪fit_transform(food_security_df[food_security_cols])

# find linear correlation between features and crop export values
correlation = food_security_df[food_security_cols].corr()
display_heatmap(correlation, title='Food Security and Crop Values')
# find non-linear correlation between features and crop export values
X = food_security_df[['Dietary_Value', 'Cereal_Value', 'Irrigation_Value',
                   'Imports_Value', 'Stability_Value', 'Prod_Value',
                   'Supply_Value', 'Anemia_Value', 'Birthweight_Value']]
y = food_security_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Food Security Features')
```



MSE: 0.11357499966049601

	Feature	Importance
8	Birthweight_Value	0.31
2	Irrigation_Value	0.16
3	Imports_Value	0.15
0	Dietary_Value	0.14
4	Stability_Value	0.06
5	Prod_Value	0.06
6	Supply_Value	0.06
1	Cereal_Value	0.04
7	Anemia_Value	0.01



1.2.9 foreign_direct_investment.csv

[29]:
```
foreign_direct_investment = pd.read_csv('Data/raw_data/
 ↪foreign_direct_investment.csv')
foreign_direct_investment = foreign_direct_investment.drop(columns=['Domain␣
 ↪Code',
                                                                     'Domain',
                                                                     'Area Code␣
 ↪(M49)',
                                                                     'Element␣
 ↪Code',
                                                                     'Element',
                                                                     'Item Code',
                                                                     'Year Code',
                                                                     'Unit',
                                                                     'Flag',
                                                                     'Flag␣
 ↪Description',
                                                                     'Note'])
foreign_direct_investment =␣
 ↪foreign_direct_investment[(foreign_direct_investment['Year'] >= 2002) &␣
 ↪(foreign_direct_investment['Year'] <= 2022)]
```

```
unique_items = foreign_direct_investment['Item'].unique()
print(f'Unique Items: {unique_items}\n')
```

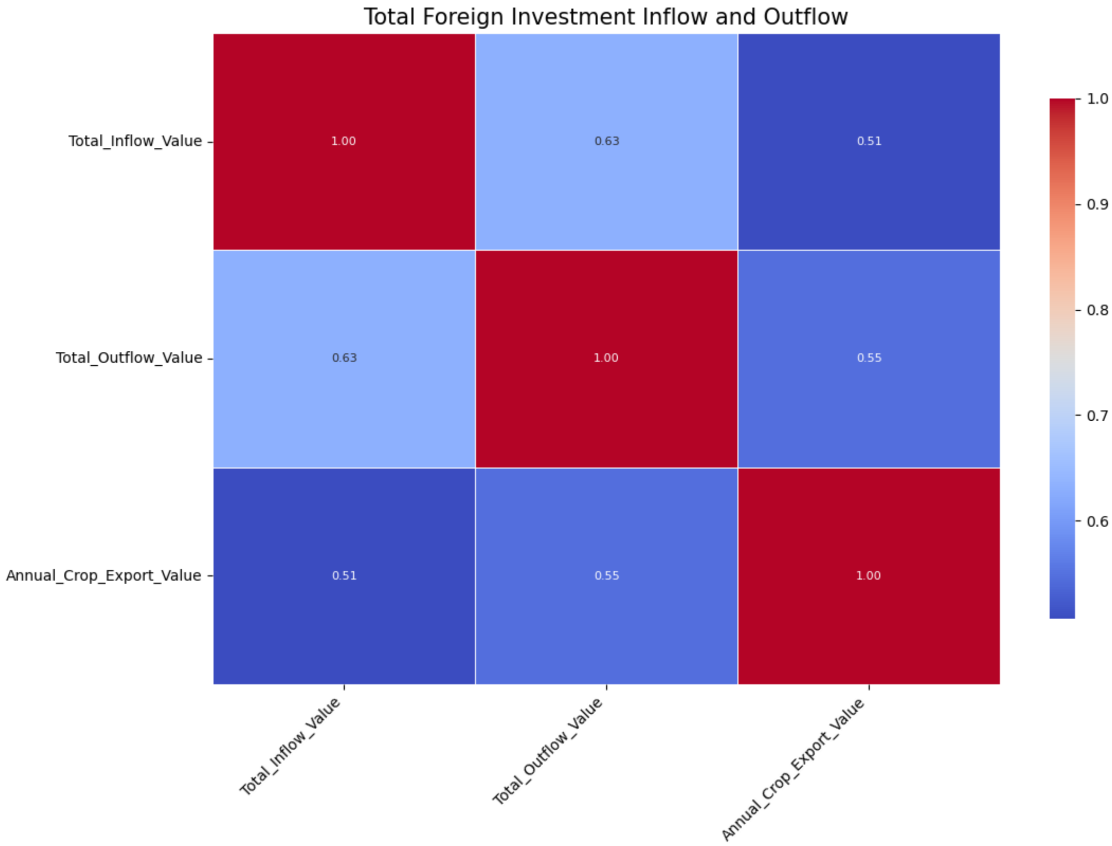
```
Unique Items: ['Total FDI inflows' 'Total FDI outflows'
 'FDI inflows to Agriculture, Forestry and Fishing'
 'FDI inflows to Food, Beverages and Tobacco'
 'FDI outflows to Agriculture, Forestry and Fishing'
 'FDI outflows to Food, Beverages and Tobacco']
```

[30]:
```
total_inflow = foreign_direct_investment[foreign_direct_investment['Item'] ==␣
 ↪unique_items[0]].copy()
total_inflow = total_inflow.rename(columns={'Value': 'Total_Inflow_Value',␣
 ↪'Item': 'Total_Inflow_Item'})

total_outflow = foreign_direct_investment[foreign_direct_investment['Item'] ==␣
 ↪unique_items[1]].copy()
total_outflow = total_outflow.rename(columns={'Value': 'Total_Outflow_Value',␣
 ↪'Item': 'Total_OutflowItem'})

aff_inflow = foreign_direct_investment[foreign_direct_investment['Item'] ==␣
 ↪unique_items[2]].copy()
aff_inflow = aff_inflow.rename(columns={'Value': 'Total_AFF_Inflow_Value',␣
 ↪'Item': 'Total_AFF_Inflow_Item'})

fbt_inflow = foreign_direct_investment[foreign_direct_investment['Item'] ==␣
 ↪unique_items[3]].copy()
fbt_inflow = fbt_inflow.rename(columns={'Value': 'Total_FBT_Inflow_Value',␣
 ↪'Item': 'Total_FBT_Inflow_Item'})

aff_outflow = foreign_direct_investment[foreign_direct_investment['Item'] ==␣
 ↪unique_items[4]].copy()
aff_outflow = aff_outflow.rename(columns={'Value': 'Total_AFF_Outflow_Value',␣
 ↪'Item': 'Total_AFF_Outflow_Item'})

fbt_outflow = foreign_direct_investment[foreign_direct_investment['Item'] ==␣
 ↪unique_items[5]].copy()
fbt_outflow = fbt_outflow.rename(columns={'Value': 'Total_FBT_Outflow_Value',␣
 ↪'Item': 'Total_FBT_Outflow_Item'})
```

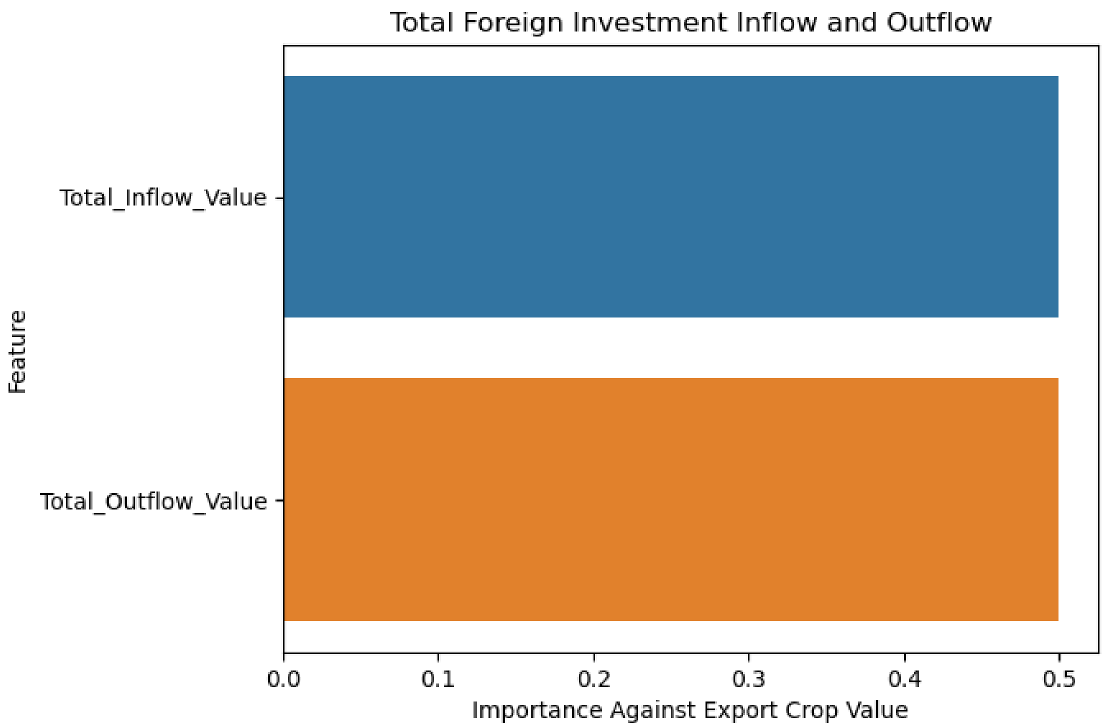
[37]:
```
totals_df = pd.merge(total_inflow, total_outflow, on=['Area', 'Year'])
totals_df = pd.merge(totals_df, crop_exports_summed[['Area', 'Year',␣
 ↪'Annual_Crop_Export_Value']], on=['Area', 'Year'])

totals_value_cols = ['Total_Inflow_Value', 'Total_Outflow_Value',␣
 ↪'Annual_Crop_Export_Value']
```

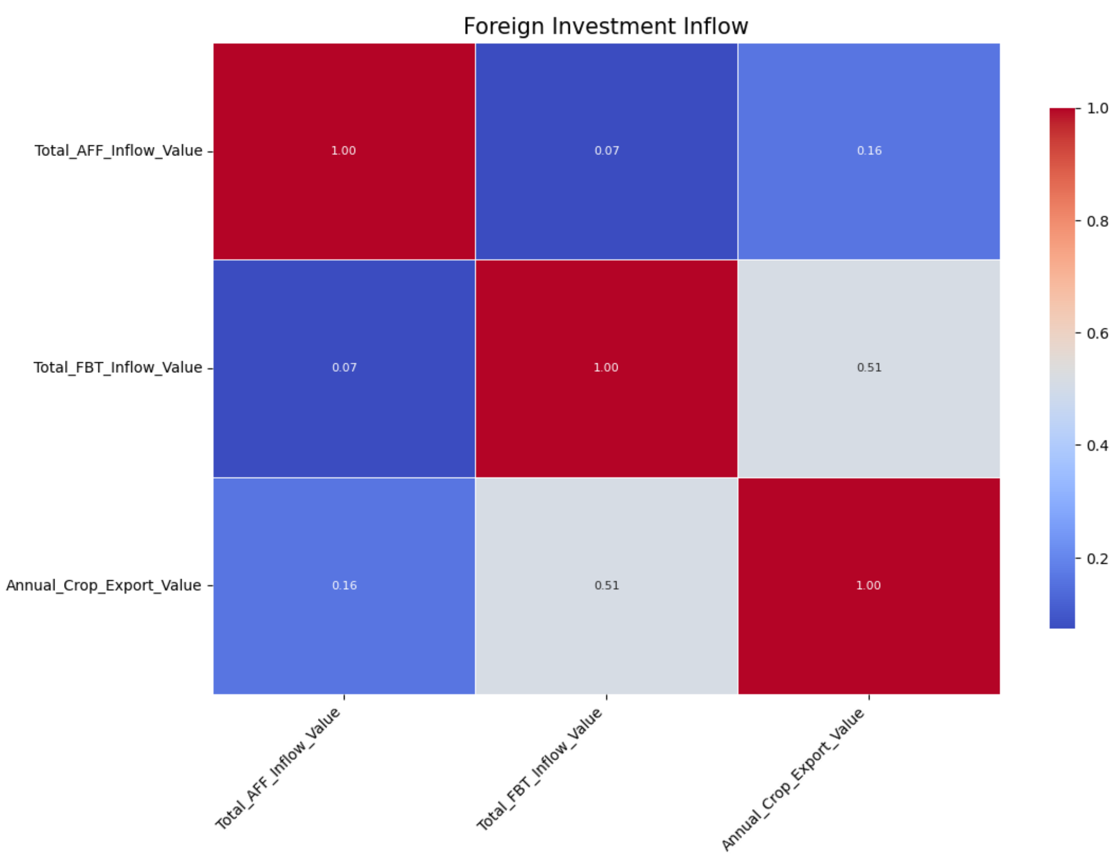
```
totals_df[totals_value_cols] = standard_scaler.
 ↪fit_transform(totals_df[totals_value_cols])

# find linear correlation between features and crop export values
correlation = totals_df[totals_value_cols].corr()
display_heatmap(correlation, title='Total Foreign Investment Inflow and␣
 ↪Outflow')
# find non-linear correlation between features and crop export values
X = totals_df[['Total_Inflow_Value', 'Total_Outflow_Value']]
y = totals_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Total Foreign Investment Inflow and Outflow')

foreign_investment_inflow = pd.merge(aff_inflow, fbt_inflow, on=['Area',␣
 ↪'Year'])
foreign_investment_inflow = pd.merge(foreign_investment_inflow,␣
 ↪crop_exports_summed[['Area', 'Year', 'Annual_Crop_Export_Value']],␣
 ↪on=['Area', 'Year'])

totals_value_cols = ['Total_AFF_Inflow_Value', 'Total_FBT_Inflow_Value',␣
 ↪'Annual_Crop_Export_Value']
foreign_investment_inflow[totals_value_cols] = standard_scaler.
 ↪fit_transform(foreign_investment_inflow[totals_value_cols])

# find linear correlation between features and crop export values
correlation = foreign_investment_inflow[totals_value_cols].corr()
display_heatmap(correlation, title='Foreign Investment Inflow')
# find non-linear correlation between features and crop export values
X = foreign_investment_inflow[['Total_AFF_Inflow_Value',␣
 ↪'Total_FBT_Inflow_Value']]
y = foreign_investment_inflow['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Foreign Investment Inflow')

foreign_investment_outflow = pd.merge(aff_outflow, fbt_outflow, on=['Area',␣
 ↪'Year'])
foreign_investment_outflow = pd.merge(foreign_investment_outflow,␣
 ↪crop_exports_summed[['Area', 'Year', 'Annual_Crop_Export_Value']],␣
 ↪on=['Area', 'Year'])

totals_value_cols = ['Total_AFF_Outflow_Value', 'Total_FBT_Outflow_Value',␣
 ↪'Annual_Crop_Export_Value']
foreign_investment_outflow[totals_value_cols] = standard_scaler.
 ↪fit_transform(foreign_investment_outflow[totals_value_cols])

# find linear correlation between features and crop export values
correlation = foreign_investment_outflow[totals_value_cols].corr()
display_heatmap(correlation, title='Foreign Investment Outflow')
```

```
# find non-linear correlation between features and crop export values
X = foreign_investment_outflow[['Total_AFF_Outflow_Value',␣
 ↪'Total_FBT_Outflow_Value']]
y = foreign_investment_outflow['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Foreign Investment Outflow')
```

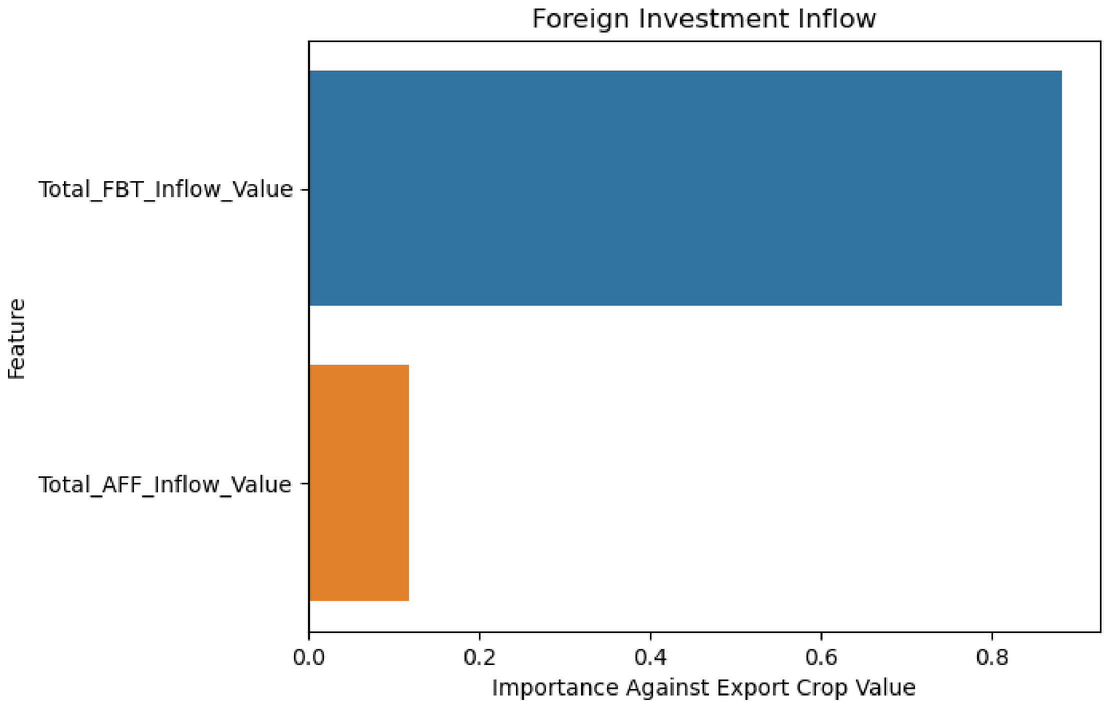


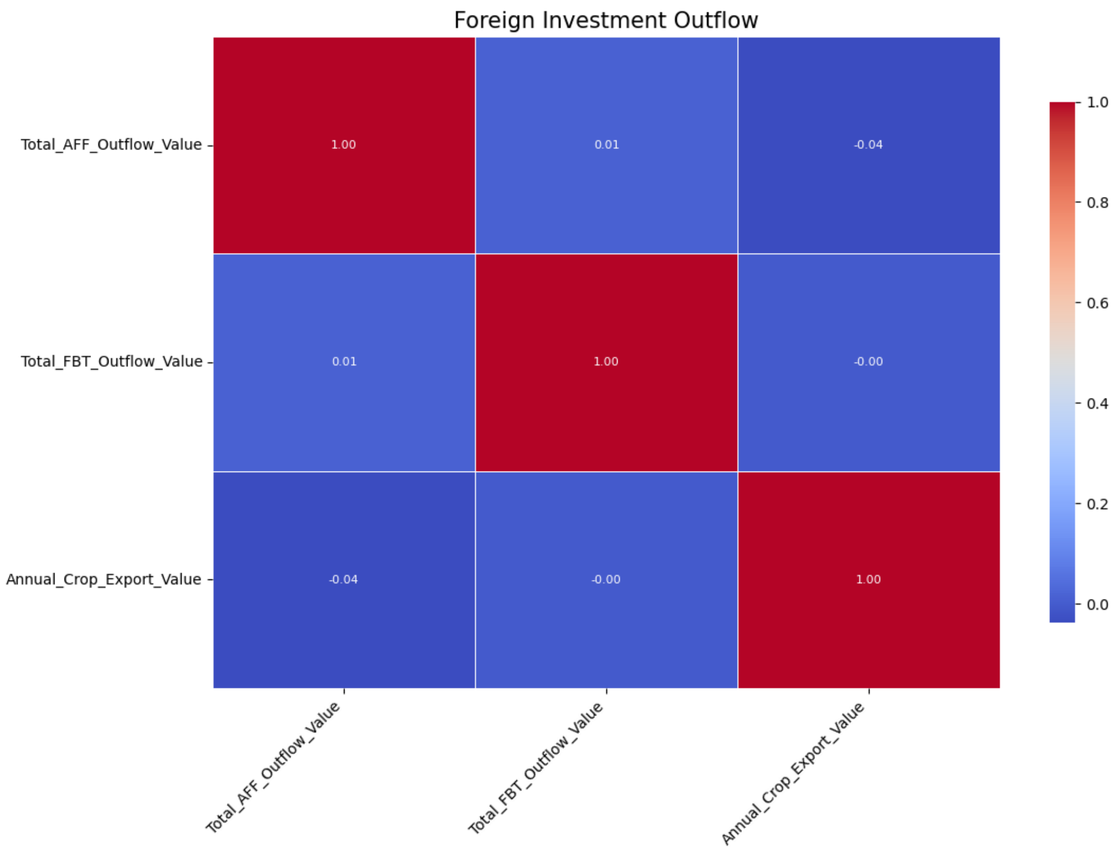

```
MSE: 0.4739602848150201

              Feature  Importance
0   Total_Inflow_Value        0.50
1  Total_Outflow_Value        0.50
```


Total Foreign Investment Inflow and Outflow
Total_Inflow_Value
Total_Outflow_Value
Feature
0.0
0.1
0.2
0.3
0.4
0.5
Importance Against Export Crop Value

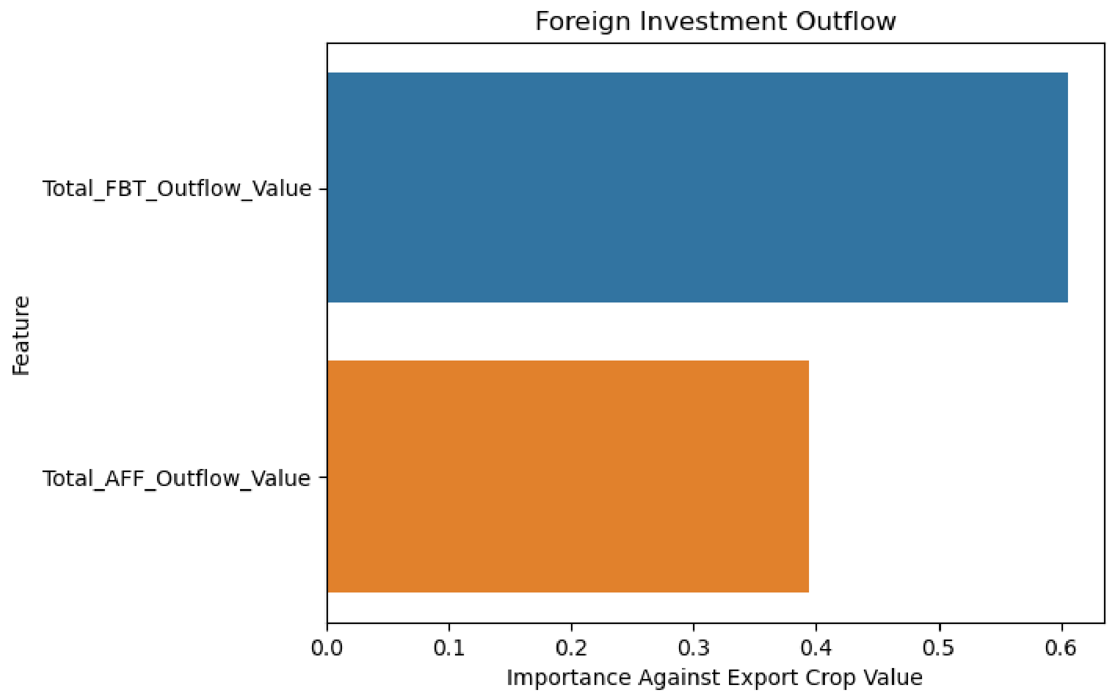

Foreign Investment Inflow
Total_AFF_Inflow_Value
Total_FBT_Inflow_Value
Annual_Crop_Export_Value
1.00
0.07
0.16
0.07
1.00
0.51
0.16
0.51
1.00
1.0
0.8
0.6
0.4
0.2
Total_AFF_Inflow_Value
Total_FBT_Inflow_Value
Annual_Crop_Export_Value

```
MSE: 0.424669501189604
                   Feature  Importance
1  Total_FBT_Inflow_Value        0.88
0  Total_AFF_Inflow_Value        0.12
```






```
MSE: 0.6711274558690102

                    Feature  Importance
1   Total_FBT_Outflow_Value        0.61
0   Total_AFF_Outflow_Value        0.39
```



1.2.10 land_temperature_change.csv

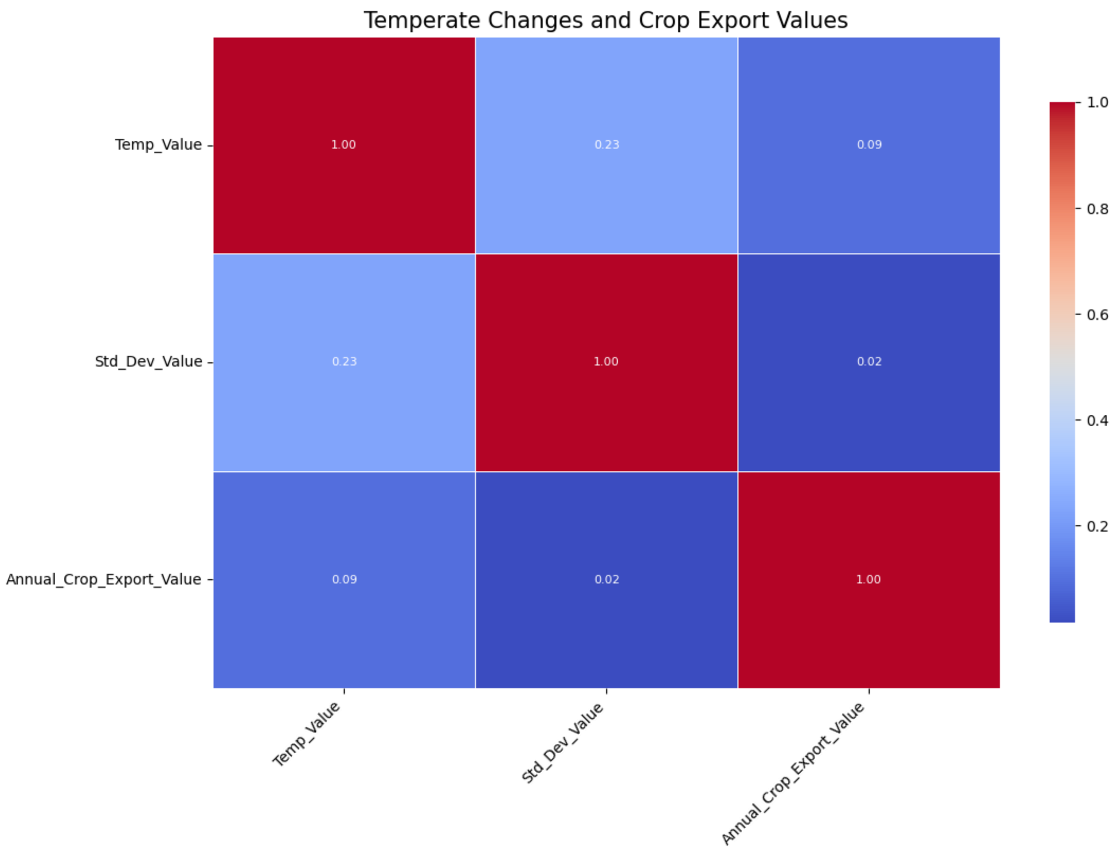
[64]:
```
land_temp_change = pd.read_csv('Data/raw_data/land_temperature_change.csv')

# use the Flag Description column to find NaN values and fix them
for i in range(len(land_temp_change)):
    if 'Missing' in land_temp_change.loc[i, 'Flag Description']:
        if i > 0 and i < len(land_temp_change) - 1:
            year_before = land_temp_change.loc[i - 1, 'Value']
            year_after = land_temp_change.loc[i + 1, 'Value']
            # if values on both sides are available use an average
            if pd.notna(year_before) and pd.notna(year_after):
                land_temp_change.loc[i, 'Value'] = (year_before + year_after) /␣
 ↪2
            # otherwise use the year before
            elif pd.notna(year_before):
                land_temp_change.loc[i, 'Value'] = year_before
            # otherwise use the year after
            elif pd.notna(year_after):
                land_temp_change.loc[i, 'Value'] = year_after

land_temp_change = land_temp_change.drop(columns=['Domain Code', 'Domain',␣
 ↪'Area Code (M49)',
```

```
                                                  'Element Code', 'Months␣
↪Code', 'Months',
                                                  'Year Code', 'Unit', 'Flag',␣
↪'Flag Description'])

land_temp_change = land_temp_change[(land_temp_change['Year'] >= 2002) &␣
↪(land_temp_change['Year'] <= 2022)]

unique_items = land_temp_change['Element'].unique()
print(f'Unique elements: {unique_items}\n')
```

```
Unique elements: ['Temperature change' 'Standard Deviation']
```

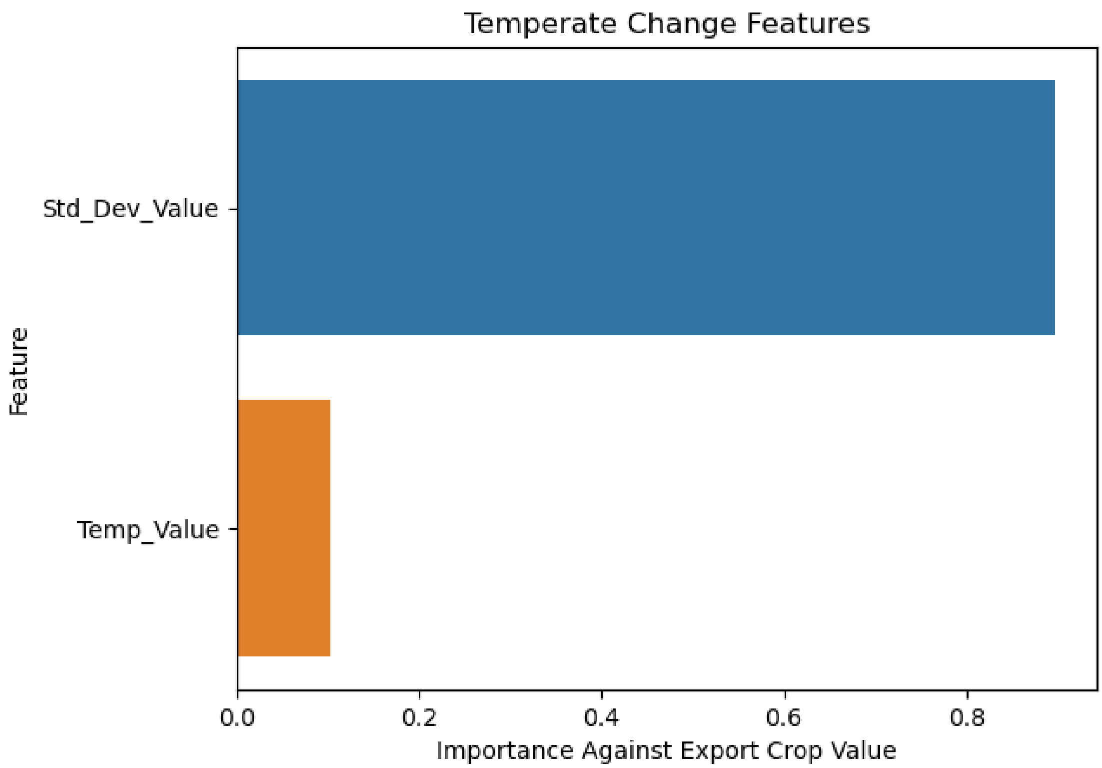
[65]:
```
temp_change = land_temp_change[land_temp_change['Element'] == unique_items[0]].
↪copy()
temp_change = temp_change.rename(columns={'Value': 'Temp_Value', 'Element':␣
↪'Temp_Element'})

std_dev = land_temp_change[land_temp_change['Element'] == unique_items[1]].
↪copy()
std_dev = std_dev.rename(columns={'Value': 'Std_Dev_Value', 'Element':␣
↪'Std_Dev_Element'})
```

[67]:
```
temp_df = pd.merge(temp_change, std_dev, on=['Area', 'Year'])
temp_df = pd.merge(temp_df, crop_exports_summed[['Area', 'Year',␣
↪'Annual_Crop_Export_Value']], on=['Area', 'Year'])

# normalise
value_cols = ['Temp_Value', 'Std_Dev_Value', 'Annual_Crop_Export_Value']
temp_df[value_cols] = standard_scaler.fit_transform(temp_df[value_cols])

# find linear correlation between features and crop export values
correlation = temp_df[value_cols].corr()
display_heatmap(correlation, title='Temperate Changes and Crop Export Values')

# find non-linear correlation between features and crop export values
X = temp_df[['Temp_Value', 'Std_Dev_Value']]
y = temp_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Temperate Change Features')
```



MSE: 0.5481454883379271

	Feature	Importance
1	Std_Dev_Value	0.90
0	Temp_Value	0.10



1.2.11 land_use.csv

[72]:
```
land_use = pd.read_csv('Data/raw_data/land_use.csv')

land_use = land_use.drop(columns=['Domain Code', 'Domain', 'Area Code (M49)',
                                                    'Element Code', 'Element',␣
 ↪'Item Code',
                                                    'Year Code', 'Unit', 'Flag',␣
 ↪'Flag Description', 'Note'])

land_use = land_use[(land_use['Year'] >= 2002) & (land_use['Year'] <= 2022)]

unique_items = land_use['Item'].unique()
print(f'Unique items: {unique_items}\n')
```

```
Unique items: ['Country area' 'Land area' 'Agriculture' 'Agricultural land'
'Cropland'
 'Arable land' 'Temporary crops' 'Temporary meadows and pastures'
 'Temporary fallow' 'Permanent crops' 'Permanent meadows and pastures'
 'Perm. meadows & pastures - Nat. growing'
 'Land area equipped for irrigation' 'Land area actually irrigated'
 'Agriculture area actually irrigated' 'Farm buildings and Farmyards'
```

```
 'Cropland area actually irrigated'
 'Perm. meadows & pastures - Cultivated'
 'Perm. meadows & pastures area actually irrig.'
 'Forestry area actually irrigated']
```

```
/var/folders/r7/pmx4mq9n359ff5rbmwjtfsqh0000gn/T/ipykernel_94689/2003263151.py:1
: DtypeWarning: Columns (14) have mixed types. Specify dtype option on import or
set low_memory=False.
  land_use = pd.read_csv('Data/raw_data/land_use.csv')
```

[73]:

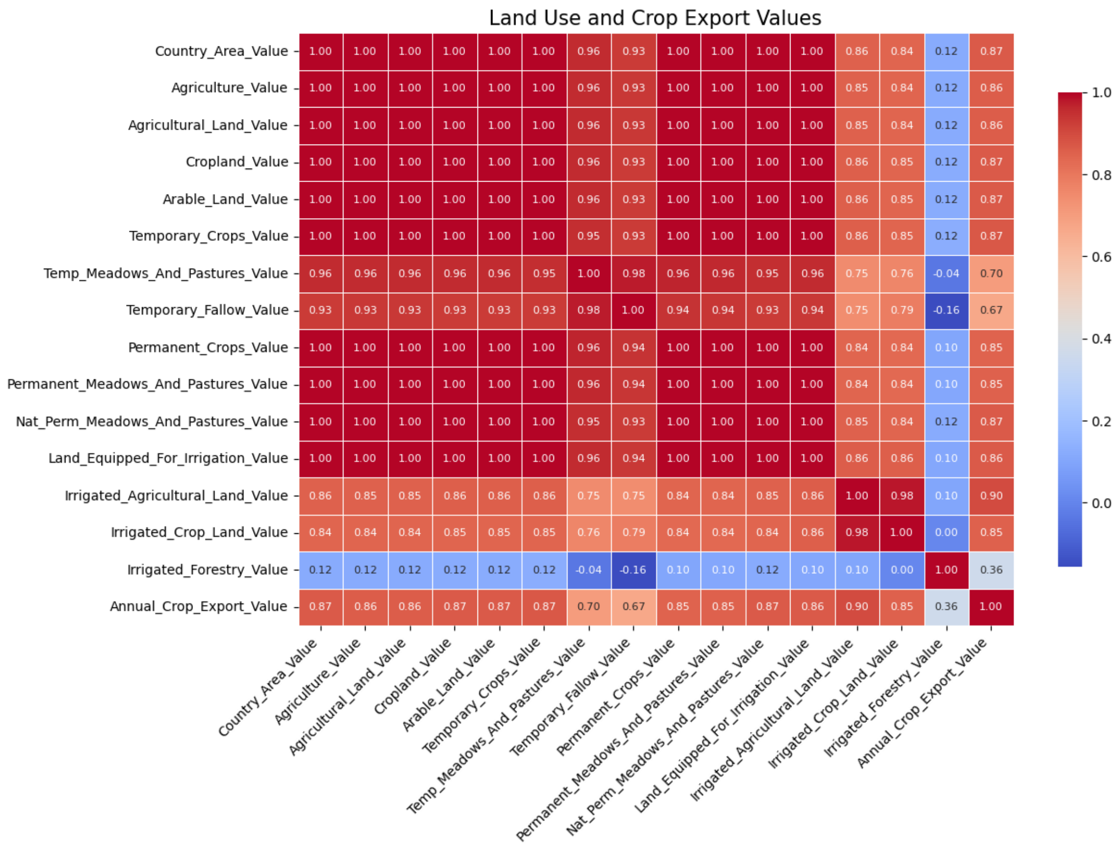
```
country_area = land_use[land_use['Item'] == unique_items[0]].copy()
country_area = country_area.rename(columns={'Value': 'Country_Area_Value',
                                            'Item': 'Country_Area_Item'})

land_area = land_use[land_use['Item'] == unique_items[1]].copy()
land_area = land_area.rename(columns={'Value': 'Land_Use_Value',
                                      'Item': 'Land_Use_Item'})

agriculture = land_use[land_use['Item'] == unique_items[2]].copy()
agriculture = agriculture.rename(columns={'Value': 'Agriculture_Value',
                                          'Item': 'Agriculture_Item'})

agricultural_land = land_use[land_use['Item'] == unique_items[3]].copy()
agricultural_land = agricultural_land.rename(columns={'Value':␣
 ↪'Agricultural_Land_Value',
                                                      'Item':␣
 ↪'Agricultural_Land_Item'})

cropland = land_use[land_use['Item'] == unique_items[4]].copy()
cropland = cropland.rename(columns={'Value': 'Cropland_Value',
                                    'Item': 'Cropland_Item'})

arable_land = land_use[land_use['Item'] == unique_items[5]].copy()
arable_land = arable_land.rename(columns={'Value': 'Arable_Land_Value',
                                          'Item': 'Arable_Land_Item'})

temporary_crops = land_use[land_use['Item'] == unique_items[6]].copy()
temporary_crops = temporary_crops.rename(columns={'Value':␣
 ↪'Temporary_Crops_Value',
                                                  'Item':␣
 ↪'Temporary_Crops_Item'})

temp_meadows = land_use[land_use['Item'] == unique_items[7]].copy()
temp_meadows = temp_meadows.rename(columns={'Value':␣
 ↪'Temp_Meadows_And_Pastures_Value',
```

```
                                            'Item':␣
↪'Temp_Meadows_And_Pastures_Item'})

temp_fallow = land_use[land_use['Item'] == unique_items[8]].copy()
temp_fallow = temp_fallow.rename(columns={'Value': 'Temporary_Fallow_Value',
                                          'Item': 'Temporary_Fallow_Item'})

permanent_crops = land_use[land_use['Item'] == unique_items[9]].copy()
permanent_crops = permanent_crops.rename(columns={'Value':␣
↪'Permanent_Crops_Value',
                                                  'Item':␣
↪'Permanent_Crops_Item'})

permanent_meadows = land_use[land_use['Item'] == unique_items[10]].copy()
permanent_meadows = permanent_meadows.rename(columns={'Value':␣
↪'Permanent_Meadows_And_Pastures_Value',
                                                      'Item':␣
↪'Permanent_Meadows_And_Pastures_Item'})

nat_perm_meadows = land_use[land_use['Item'] == unique_items[11]].copy()
nat_perm_meadows = nat_perm_meadows.rename(columns={'Value':␣
↪'Nat_Perm_Meadows_And_Pastures_Value',
                                                    'Item':␣
↪'Nat_Perm_Meadows_And_Pastures_Item'})

land_equipped_irr = land_use[land_use['Item'] == unique_items[12]].copy()
land_equipped_irr = land_equipped_irr.rename(columns={'Value':␣
↪'Land_Equipped_For_Irrigation_Value',
                                                      'Item':␣
↪'Land_Equipped_For_Irrigation_Item'})

irrigated_land = land_use[land_use['Item'] == unique_items[13]].copy()
irrigated_land = irrigated_land.rename(columns={'Value': 'Irrigated_Land_Value',
                                                'Item': 'Irrigated_Land_Item'})

agricultural_land_irr = land_use[land_use['Item'] == unique_items[14]].copy()
agricultural_land_irr = agricultural_land_irr.rename(columns={'Value':␣
↪'Irrigated_Agricultural_Land_Value',
                                                              'Item':␣
↪'Irrigated_Agricultural_Land_Item'})

farm_buildings = land_use[land_use['Item'] == unique_items[15]].copy()
farm_buildings = farm_buildings.rename(columns={'Value': 'Farm_Buildings_Value',
                                                'Item': 'Farm_Buildings_Item'})

irrigated_crop_land = land_use[land_use['Item'] == unique_items[16]].copy()
```

```
irrigated_crop_land = irrigated_crop_land.rename(columns={'Value':␣
↪'Irrigated_Crop_Land_Value',
                                                           'Item':␣
↪'Irrigated_Crop_Land_Item'})

permanent_meadows_irr = land_use[land_use['Item'] == unique_items[17]].copy()
permanent_meadows_irr = permanent_meadows_irr.rename(columns={'Value':␣
↪'Permanent_Meadows_And_Pastures_Irrigated_Value',
                                                               'Item':␣
↪'Permanent_Meadows_And_Pastures_Irrigated_Item'})

irrigated_foresty = land_use[land_use['Item'] == unique_items[18]].copy()
irrigated_foresty = irrigated_foresty.rename(columns={'Value':␣
↪'Irrigated_Forestry_Value',
                                                           'Item':␣
↪'Irrigated_Forestry_Item'})
```

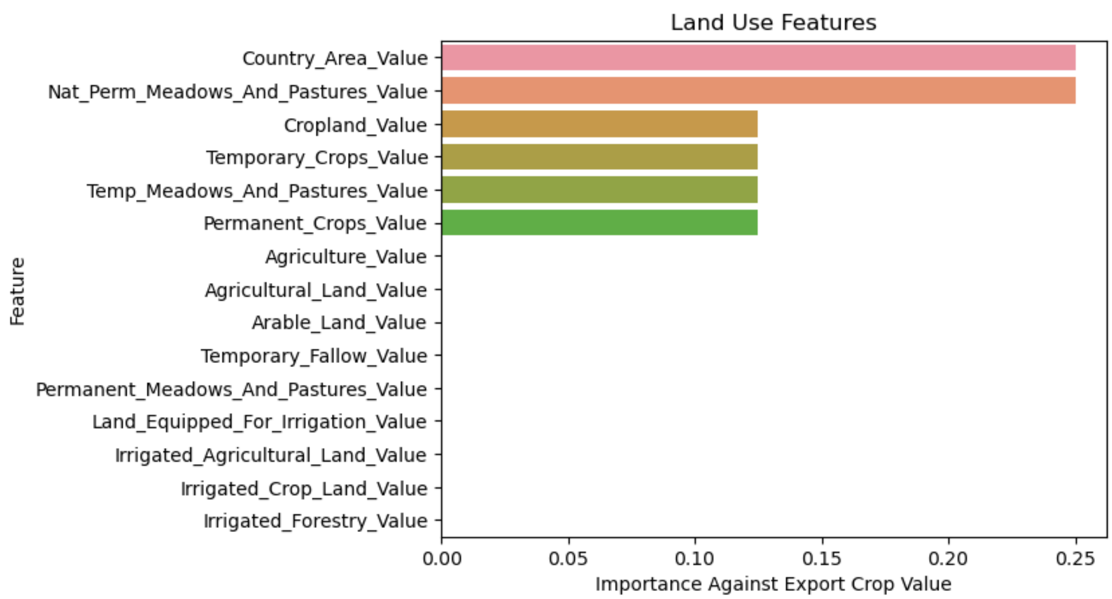
[74]:
```
land_use_df = pd.merge(country_area, land_area, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, agriculture, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, agricultural_land, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, cropland, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, arable_land, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, temporary_crops, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, temp_meadows, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, temp_fallow, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, permanent_crops, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, permanent_meadows, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, nat_perm_meadows, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, land_equipped_irr, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, irrigated_land, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, agricultural_land_irr, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, farm_buildings, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, irrigated_crop_land, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, permanent_meadows_irr, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, irrigated_foresty, on=['Year', 'Area'])
land_use_df = pd.merge(land_use_df, crop_exports_summed[['Area', 'Year',␣
↪'Annual_Crop_Export_Value']], on=['Area', 'Year'])

# normalise
value_cols = ['Country_Area_Value', 'Agriculture_Value',␣
↪'Agricultural_Land_Value',
              'Cropland_Value', 'Arable_Land_Value', 'Temporary_Crops_Value',
              'Temp_Meadows_And_Pastures_Value', 'Temporary_Fallow_Value',␣
↪'Permanent_Crops_Value',
              'Permanent_Meadows_And_Pastures_Value',␣
↪'Nat_Perm_Meadows_And_Pastures_Value',
```

```
              'Land_Equipped_For_Irrigation_Value',␣
↪'Irrigated_Agricultural_Land_Value',
              'Irrigated_Crop_Land_Value', 'Irrigated_Forestry_Value',␣
↪'Annual_Crop_Export_Value']
land_use_df[value_cols] = standard_scaler.fit_transform(land_use_df[value_cols])

# find linear correlation between features and crop export values
correlation = land_use_df[value_cols].corr()
display_heatmap(correlation, title='Land Use and Crop Export Values')

# find non-linear correlation between features and crop export values
X = land_use_df[['Country_Area_Value', 'Agriculture_Value',␣
↪'Agricultural_Land_Value',
              'Cropland_Value', 'Arable_Land_Value', 'Temporary_Crops_Value',
              'Temp_Meadows_And_Pastures_Value', 'Temporary_Fallow_Value',␣
↪'Permanent_Crops_Value',
              'Permanent_Meadows_And_Pastures_Value',␣
↪'Nat_Perm_Meadows_And_Pastures_Value',
              'Land_Equipped_For_Irrigation_Value',␣
↪'Irrigated_Agricultural_Land_Value',
              'Irrigated_Crop_Land_Value', 'Irrigated_Forestry_Value']]
y = land_use_df['Annual_Crop_Export_Value']
rank_feature_importance(X, y, 'Land Use Features')
```



MSE: 0.5671844578990466

	Feature	Importance
0	Country_Area_Value	0.25
10	Nat_Perm_Meadows_And_Pastures_Value	0.25
3	Cropland_Value	0.12
5	Temporary_Crops_Value	0.12
6	Temp_Meadows_And_Pastures_Value	0.12
8	Permanent_Crops_Value	0.12
1	Agriculture_Value	0.00
2	Agricultural_Land_Value	0.00
4	Arable_Land_Value	0.00
7	Temporary_Fallow_Value	0.00
9	Permanent_Meadows_And_Pastures_Value	0.00
11	Land_Equipped_For_Irrigation_Value	0.00
12	Irrigated_Agricultural_Land_Value	0.00
13	Irrigated_Crop_Land_Value	0.00
14	Irrigated_Forestry_Value	0.00



1.2.12 pesticide_use.csv

[77]:
```
pesticide_use = pd.read_csv('Data/raw_data/pesticide_use.csv')

pesticide_use = pesticide_use.drop(columns=['Domain Code', 'Domain', 'Area Code␣
↪(M49)',
                                              'Element Code', 'Item Code', 'Year␣
↪Code',
                                              'Unit', 'Flag', 'Flag Description',␣
↪'Note'])

pesticide_use = pesticide_use[(pesticide_use['Year'] >= 2002) &␣
↪(pesticide_use['Year'] <= 2022)]

unique_elements = pesticide_use['Element'].unique()
unique_items = pesticide_use['Item'].unique()
print(f'Unique elements: {unique_elements}\n')
print(f'Unique items: {unique_items}\n')
```

```
Unique elements: ['Agricultural Use' 'Use per area of cropland'
 'Use per value of agricultural production']

Unique items: ['Pesticides (total)' 'Insecticides' 'Herbicides'
 'Fungicides and Bactericides' 'Fungicides - Seed treatments'
 'Insecticides - Seed Treatments' 'Rodenticides']
```

[80]:
```
algricultural_use = pesticide_use[pesticide_use['Element'] ==␣
 ↪unique_elements[0]].copy()
algricultural_use = algricultural_use.rename(columns={'Element':␣
 ↪'Agricultural_Use_Element'})

use_per_area = pesticide_use[pesticide_use['Element'] == unique_elements[1]].
 ↪copy()
use_per_area = use_per_area.rename(columns={'Element': 'Use_Per_Area_Element'})

use_per_value_of_agr_prod = pesticide_use[pesticide_use['Element'] ==␣
 ↪unique_elements[2]].copy()
use_per_value_of_agr_prod = use_per_value_of_agr_prod.rename(columns={'Element':
 ↪ 'Use_Per_Value_Of_Agricultural_Produce_Element'})


agr_use_pesticides = algricultural_use[algricultural_use['Item'] ==␣
 ↪unique_items[0]].copy()
agr_use_pesticides = agr_use_pesticides.rename(columns={'Item':␣
 ↪'Pesticides_Total_Value'})

agr_use_insecticides = algricultural_use[algricultural_use['Item'] ==␣
 ↪unique_items[1]].copy()
agr_use_insecticides = agr_use_insecticides.rename(columns={'Item':␣
 ↪'Insecticides_Total_Value'})

agr_use_herbicides = algricultural_use[algricultural_use['Item'] ==␣
 ↪unique_items[2]].copy()
agr_use_herbicides = agr_use_herbicides.rename(columns={'Item':␣
 ↪'Herbicides_Total_Value'})

agr_use_fungicides = algricultural_use[algricultural_use['Item'] ==␣
 ↪unique_items[3]].copy()
agr_use_fungicides = agr_use_fungicides.rename(columns={'Item':␣
 ↪'Fungicides_And_Bactericides_Total_Value'})

agr_use_fungicides_seed_treatments =␣
 ↪algricultural_use[algricultural_use['Item'] == unique_items[4]].copy()
agr_use_fungicides_seed_treatments = agr_use_fungicides_seed_treatments.
 ↪rename(columns={'Item': 'Fungicides_Seed_Treatments_Total_Value'})
```

	Area	Agricultural_Use_Element	Item	Year	Value
6	Albania	Agricultural Use	Pesticides (total)	2002	330.78
9	Albania	Agricultural Use	Pesticides (total)	2003	342.17
12	Albania	Agricultural Use	Pesticides (total)	2004	353.57
15	Albania	Agricultural Use	Pesticides (total)	2005	364.97
18	Albania	Agricultural Use	Pesticides (total)	2006	376.36
…	…	…	…	…	…

```
35055  Zimbabwe        Agricultural Use  Pesticides (total)  2017 2185.07
35058  Zimbabwe        Agricultural Use  Pesticides (total)  2018 2185.07
35061  Zimbabwe        Agricultural Use  Pesticides (total)  2019 2185.07
35064  Zimbabwe        Agricultural Use  Pesticides (total)  2020 2185.07
35067  Zimbabwe        Agricultural Use  Pesticides (total)  2021 2185.07

[4216 rows x 5 columns]
```

[ ]:

```
# faostat_data = glob.glob(os.path.join('Data/raw_data/', '*.csv'))

# # Inspect csv files before selecting features
# for file_name in faostat_data:
#     df = pd.read_csv(file_name)
#     print(file_name.strip('Data/raw_data/'))
#     # display(df) # output hidden after inspection for brevity

# drop_cols = ['Element Code', 'Months', 'Months Code', 'Flag','Note',␣
↪'Source',
#               'Source Code', 'Item Code', 'Item Code (CPC)', 'Year Code',
#               'ISO Currency Code (FAO)', 'Domain Code','Element', 'Sex',
#               'Sex Code', 'Indicator', 'Item Code (FBS)', 'Item', 'Indicator␣
↪Code']

# for file_name in faostat_data:
#     df = pd.read_csv(file_name)
#     cols_to_drop = [col for col in drop_cols if col in df.columns]
#     df = df.drop(columns=cols_to_drop)

#     if 'Year' in df.columns:
#         # food_security_indicators.csv has dates in form '2000-2002'
#         df['Year'] = df['Year'].astype(str)
#         if df['Year'].str.contains('-').any():
#             # keep only the first part of the year
#             df['Year'] = df['Year'].apply(lambda x: int(x.split('-')[0]))
#         # convert from string to integer
#         df['Year'] = df['Year'].astype(int)
#         # only keep years >= 2002 as all files have data from this year
#         df = df[df['Year'] >= 2002]

#     file_name = os.path.join('Data/feature_selected/', os.path.
↪basename(file_name))
#     df.to_csv(file_name, index=False)
#     print(file_name.strip('Data/raw_data/'))
#     # display(df) # output hidden after inspection for brevity
```

1.3 Handle NaN values

Identify which features have NaN values and handle them by finding the average from the year before and the year after the missing value. If both of those values are missing, use whichever is not NaN.

`land_temperature_change.csv` was altered as the `Value` column had missing values.

`consumer_prices.csv` and `exchange_rate.csv` are unchanged as the `NaN` values are in the `Unit` column which is not going to fed to the model.

[ ]:
```
# Check if feature selected data has NaN values
faostat_data = glob.glob(os.path.join('Data/feature_selected/', '*.csv'))

for file_name in faostat_data:
    df = pd.read_csv(file_name)
    if df.isnull().values.any():
        print(f'{file_name} contains NaN values')
        nan_cols = [(c, df[c].hasnans) for c in df]
        print(nan_cols)
        print()
```

[ ]:
```
# Handle NaN values in land_temperate_change.csv Value column
land_temp_df = pd.read_csv('Data/feature_selected/land_temperature_change.csv')

for i in range(len(land_temp_df)):
    if 'Missing' in land_temp_df.loc[i, 'Flag Description']:
        if i > 0 and i < len(land_temp_df) - 1:
            year_before = land_temp_df.loc[i - 1, 'Value']
            year_after = land_temp_df.loc[i + 1, 'Value']
            # if values on both sides are available use an average
            if pd.notna(year_before) and pd.notna(year_after):
                land_temp_df.loc[i, 'Value'] = (year_before + year_after) / 2
            # otherwise use the year before
            elif pd.notna(year_before):
                land_temp_df.loc[i, 'Value'] = year_before
            # otherwise use the year after
            elif pd.notna(year_after):
                land_temp_df.loc[i, 'Value'] = year_after

if land_temp_df.isnull().any().any():
    print(f'{file_name} contains NaN values')
else:
    # no more NaN values so overwrite file
    land_temp_df.to_csv('Data/feature_selected/land_temperature_change.csv')
```

1.4 Normalise Data

`exchange_rate.csv` and `consumer_prices.csv` store data monthly rather than yearly like the other feature sets. For each year in each country the monthly data is summed and averaged. The original features are then overwritten. This standardises the data so all features now have yearly values.

[ ]:
```
exchange_rates = pd.read_csv('Data/raw_data/exchange_rate.csv')
exchange_rate_df_normalised = exchange_rates.groupby(['Year', 'Area Code␣
 ↪(M49)', 'Area', 'Domain', 'Flag Description'])['Value'].mean().reset_index()
exchange_rate_df_normalised.to_csv('Data/feature_selected/exchange_rate.csv')

consumer_prices = pd.read_csv('Data/raw_data/consumer_prices.csv')
consumer_prices_df_normalised = consumer_prices.groupby(['Year', 'Area Code␣
 ↪(M49)', 'Area', 'Domain', 'Flag Description'])['Value'].mean().reset_index()
consumer_prices_df_normalised.to_csv('Data/feature_selected/consumer_prices.
 ↪csv')
```

2 Create Dataset and DataLoaders

[ ]:
```
class CropForecastDataset(Dataset):
    def __init__(self, data):
        self.data = data

    def __getitem__(self, idx):
        return self.data[idx]

    def __len__(self):
        return len(self.data)
```