934G5_277822

May 17, 2024

1 Title: Building MLP Model for Forecasting Export Value of Crops by Classification Approach

1.1 Name: Aman Thapa Magar

1.2 Candidate No: 277822

1.3 Module Code & Title: 934G5 Machine Learning

2 1. Importing Required Libraries

In this section, I am importing libraries which will define how my dataset have been selected, modelled and visualised. They are as follows:

- Pandas: For data framing and manipulation **- Numpy:** Used for numerical computing **- Matplotlib:** For plotting graphs **- Sklearn:** For using machine learning model libraries. **- Torch:** Used for building and training neural networks with GPU acceleration. **- Random:** Provides functions for generating random numbers, often used for data shuffling and initialization. **- Seaborn:** Enables creation of visually appealing statistical plots for data analysis. **- Copy:** Used for creating copies of objects to prevent unintended data mutation. **- RandomizedSearchCV:** Conducts hyperparameter tuning using randomized search with cross-validation. **- MLPClassifier:** Implements a multi-layer perceptron classifier for classification tasks. **- Uniform, randint:** Probability distributions used for defining hyperparameter search spaces in optimization. **- Pickle:** For saving MLP model.

[ ]:
```
import pickle
import torch
import random
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from copy import deepcopy
from torch import nn
from torch import optim
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from sklearn.metrics import f1_score, accuracy_score, confusion_matrix,␣
↪ConfusionMatrixDisplay
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.neural_network import MLPClassifier
from scipy.stats import uniform, randint
```

3 2. Loading Datasets, Preprocessing and Exploration

In this section, I am going to load all the datasets to have a feel of what is actually inside it. To do this, I am using pandas to load it inside the dataframe for data manipulation. Additionally, I am also going to explore each dataset to produce suitable filtered data to see what can I make of it. This step is required as it seems the data provided is not optimal to be used. Hence, the following processes are executed in each of these datasets. - Exploring pathways in the dataset, what features can be extracted, what can be done in accordance to the assignment criteria. - Find any missing values, duplicates and null which can then be sorted out by dropping those elements. - Using general stats such as mean, total, average etc. to group one or more rows for data's that feel like it can be further refined.

3.1 2.1 Consumer Prices Data

The following steps are performed - Display the data - Check to see if any of the columns can be further classfied into two or more groups to create new dataframes to retain more datapoints after merge. - Describe the data - Print Data Types - Checking for Null Values, Duplicates - Filtering Data by Dropping the columns from previous checks

[330]:
```
# Loading the datatset
consumer_prices_df = pd.read_csv("content/Consumer prices indicators -␣
↪FAOSTAT_data_en_2-22-2024.csv", low_memory=False)
```

[331]:
```
# Displaying the first few rows of the consumer prices data
consumer_prices_df.head()
```

[331]:
```
  Domain Code                   Domain  Area Code (M49)         Area  \
0          CP  Consumer Price Indices                4  Afghanistan
1          CP  Consumer Price Indices                4  Afghanistan
2          CP  Consumer Price Indices                4  Afghanistan
3          CP  Consumer Price Indices                4  Afghanistan
4          CP  Consumer Price Indices                4  Afghanistan

   Year Code  Year  Item Code                                    Item  \
0       2000  2000      23013  Consumer Prices, Food Indices (2015 = 100)
1       2000  2000      23013  Consumer Prices, Food Indices (2015 = 100)
2       2000  2000      23013  Consumer Prices, Food Indices (2015 = 100)
3       2000  2000      23013  Consumer Prices, Food Indices (2015 = 100)
4       2000  2000      23013  Consumer Prices, Food Indices (2015 = 100)

   Months Code    Months  Element Code Element Unit      Value Flag  \
```

```
0        7001   January          6125   Value  NaN  24.356332    I
1        7002  February          6125   Value  NaN  23.636242    I
2        7003     March          6125   Value  NaN  23.485345    I
3        7004     April          6125   Value  NaN  24.767194    I
4        7005       May          6125   Value  NaN  25.956912    I

  Flag Description              Note
0    Imputed value  base year is 2015
1    Imputed value  base year is 2015
2    Imputed value  base year is 2015
3    Imputed value  base year is 2015
4    Imputed value  base year is 2015
```

[332]:
```
consumer_prices_df.shape
```

[332]:
```
(112890, 17)
```

[333]:
```
# Describing the consumer prices data
consumer_prices_df.describe()
```

[333]:
```
       Area Code (M49)      Year Code           Year     Item Code  \
count    112890.000000  112890.000000  112890.000000  112890.000000
mean        424.738719    2011.649588    2011.649588   23013.489211
std         249.672423       6.716990       6.716990       0.499886
min           4.000000    2000.000000    2000.000000   23013.000000
25%         212.000000    2006.000000    2006.000000   23013.000000
50%         426.000000    2012.000000    2012.000000   23013.000000
75%         638.000000    2017.000000    2017.000000   23014.000000
max         894.000000    2023.000000    2023.000000   23014.000000

         Months Code   Element Code         Value
count  112890.000000  112890.000000  1.128900e+05
mean     7006.451448    6123.043157  2.059421e+08
std         3.437632       1.999543  1.683090e+10
min      7001.000000    6121.000000 -2.498299e+01
25%      7003.000000    6121.000000  4.245692e+00
50%      7006.000000    6125.000000  3.087651e+01
75%      7009.000000    6125.000000  9.252795e+01
max      7012.000000    6125.000000  2.235770e+12
```

[334]:
```
# Printing information about the consumer price data
consumer_prices_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112890 entries, 0 to 112889
Data columns (total 17 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
```

```
 0   Domain Code       112890 non-null  object
 1   Domain            112890 non-null  object
 2   Area Code (M49)   112890 non-null  int64
 3   Area              112890 non-null  object
 4   Year Code         112890 non-null  int64
 5   Year              112890 non-null  int64
 6   Item Code         112890 non-null  int64
 7   Item              112890 non-null  object
 8   Months Code       112890 non-null  int64
 9   Months            112890 non-null  object
 10  Element Code      112890 non-null  int64
 11  Element           112890 non-null  object
 12  Unit              55227 non-null   object
 13  Value             112890 non-null  float64
 14  Flag              112890 non-null  object
 15  Flag Description  112890 non-null  object
 16  Note              57663 non-null   object
dtypes: float64(1), int64(6), object(10)
memory usage: 14.6+ MB
```

[335]:
```
#Assigning new labels to same dataset to extract required dataset based on item␣
↪or element filters
fpi_df = consumer_prices_df
```

[336]:
```
#Assigning new labels to same dataset to extract required dataset based on item␣
↪or element filters
cpi_df = consumer_prices_df
```

[337]:
```
# Filtering data to only take food price inflation and from after the year 2000
fpi_df = fpi_df[(fpi_df['Year'] >= 2000) & fpi_df['Item'].str.contains('Food␣
↪price inflation')]
cpi_df = cpi_df[(cpi_df['Year'] >= 2000) & cpi_df['Item'].str.
↪contains('Consumer Prices, Food Indices (2015 = 100)', regex=False)]

# Dropping not required columns
fpi_df = fpi_df.drop(columns=['Domain Code', 'Domain', 'Year Code', 'Item',␣
↪'Item Code','Months', 'Element Code', 'Element', 'Unit', 'Flag', 'Flag␣
↪Description', 'Months Code', 'Area Code (M49)', 'Note'])
cpi_df = cpi_df.drop(columns=['Domain Code', 'Domain', 'Year Code', 'Item',␣
↪'Item Code','Months', 'Element Code', 'Element', 'Unit', 'Flag', 'Flag␣
↪Description', 'Months Code', 'Area Code (M49)', 'Note'])

# Averaging the value column based on area and year.
fpi_df = fpi_df.groupby(["Area", "Year"])["Value"].mean().reset_index()
cpi_df = cpi_df.groupby(["Area", "Year"])["Value"].mean().reset_index()
```

```
# Renaming the column to avoid confusion of value tables after merging.
fpi_df = fpi_df.rename(columns={'Value': 'Total Food Price Inflation'})
cpi_df = cpi_df.rename(columns={'Value': 'Total Consumer Prices'})
```

[338]:
```
# checking rows and columns in the dataframe after splitting the data into two␣
↪categories
fpi_df.shape
```

[338]:
```
(4653, 3)
```

[339]:
```
fpi_df
```

[339]:
```
                  Area  Year  Total Food Price Inflation
0          Afghanistan  2001                   12.780692
1          Afghanistan  2002                   18.254516
2          Afghanistan  2003                   14.102244
3          Afghanistan  2004                   14.072172
4          Afghanistan  2005                   12.606240
…                    …     …                           …
4648     Åland Islands  2019                    1.797736
4649     Åland Islands  2020                    0.643114
4650     Åland Islands  2021                    1.164459
4651     Åland Islands  2022                    9.678792
4652     Åland Islands  2023                   13.303965

[4653 rows x 3 columns]
```

[340]:
```
# checking rows and columns in the dataframe after splitting the data into two␣
↪categories
cpi_df.shape
```

[340]:
```
(4856, 3)
```

[341]:
```
cpi_df
```

[341]:
```
                  Area  Year  Total Consumer Prices
0          Afghanistan  2000              26.629848
1          Afghanistan  2001              29.893548
2          Afghanistan  2002              35.344892
3          Afghanistan  2003              40.203113
4          Afghanistan  2004              45.840561
…                    …     …                      …
4851     Åland Islands  2019             102.928436
4852     Åland Islands  2020             103.585137
4853     Åland Islands  2021             104.784347
4854     Åland Islands  2022             114.944128
```

```
4855  Åland Islands  2023            127.303080

[4856 rows x 3 columns]
```

[342]:
```
# Checking for NaN values
fpi_df.isnull().sum()
cpi_df.isnull().sum()
```

[342]:
```
Area                      0
Year                      0
Total Consumer Prices     0
dtype: int64
```

[343]:
```
# Check for duplicates
duplicates_found_fpi = fpi_df.duplicated().sum()
duplicates_found_cpi = cpi_df.duplicated().sum()

# If duplicates are found, remove them
if duplicates_found_fpi > 0 and duplicates_found_cpi > 0:
    fpi_df.drop_duplicates(inplace=True)
    cpi_df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
else:
    print("No duplicates found.")
```

```
No duplicates found.
```

[344]:
```
fpi_df.shape
```

[344]:
```
(4653, 3)
```

[345]:
```
cpi_df.shape
```

[345]:
```
(4856, 3)
```

[346]:
```
fpi_df.head(100)
```

[346]:
```
           Area  Year  Total Food Price Inflation
0   Afghanistan  2001                   12.780692
1   Afghanistan  2002                   18.254516
2   Afghanistan  2003                   14.102244
3   Afghanistan  2004                   14.072172
4   Afghanistan  2005                   12.606240
..          …    …                           …
95       Angola  2004                   51.533639
96       Angola  2005                   23.399672
97       Angola  2006                   17.286257
98       Angola  2007                   14.186884
```

```
99       Angola  2008                   18.515094

[100 rows x 3 columns]
```

[347]:
```
cpi_df.head(100)
```

[347]:
```
           Area  Year  Total Consumer Prices
0   Afghanistan  2000              26.629848
1   Afghanistan  2001              29.893548
2   Afghanistan  2002              35.344892
3   Afghanistan  2003              40.203113
4   Afghanistan  2004              45.840561
..          …     …                    …
95      Andorra  2023             135.217328
96       Angola  2000               1.691044
97       Angola  2001               3.567232
98       Angola  2002               7.109792
99       Angola  2003              14.849055

[100 rows x 3 columns]
```

3.2 2.2 Crops Production Data

The following steps are performed - Display the data - Describe the data - Print Data Types - Checking for Null Values, Duplicates - Filtering Data by Dropping the columns from previous checks

[348]:
```
# load the dataset
crops_prod_df = pd.read_csv("content/Crops production indicators -
 ↪FAOSTAT_data_en_2-22-2024.csv", low_memory=False)

# Display the first few rows of the crops prod use data
crops_prod_df.head()
```

[348]:
```
  Domain Code                        Domain  Area Code (M49)         Area  \
0         QCL  Crops and livestock products                4  Afghanistan
1         QCL  Crops and livestock products                4  Afghanistan
2         QCL  Crops and livestock products                4  Afghanistan
3         QCL  Crops and livestock products                4  Afghanistan
4         QCL  Crops and livestock products                4  Afghanistan

   Element Code Element Item Code (CPC)              Item  Year Code  Year  \
0          5419   Yield           F1717  Cereals, primary       2000  2000
1          5419   Yield           F1717  Cereals, primary       2001  2001
2          5419   Yield           F1717  Cereals, primary       2002  2002
3          5419   Yield           F1717  Cereals, primary       2003  2003
4          5419   Yield           F1717  Cereals, primary       2004  2004
```

```
        Unit  Value Flag Flag Description  Note
0  100 g/ha   8063    A  Official figure   NaN
1  100 g/ha  10067    A  Official figure   NaN
2  100 g/ha  16698    A  Official figure   NaN
3  100 g/ha  14580    A  Official figure   NaN
4  100 g/ha  13348    A  Official figure   NaN
```

[349]:
```
# Describe the crops prod data
crops_prod_df.describe()
```

[349]:
```
       Area Code (M49)  Element Code     Year Code          Year  \
count     41649.000000       41649.0  41649.000000  41649.000000
mean        425.491777        5419.0   2010.900478   2010.900478
std         255.597188           0.0      6.614270      6.614270
min           4.000000        5419.0   2000.000000   2000.000000
25%         203.000000        5419.0   2005.000000   2005.000000
50%         417.000000        5419.0   2011.000000   2011.000000
75%         643.000000        5419.0   2017.000000   2017.000000
max         894.000000        5419.0   2022.000000   2022.000000

              Value  Note
count  4.164900e+04   0.0
mean   1.056544e+05   NaN
std    1.688875e+05   NaN
min    0.000000e+00   NaN
25%    8.469000e+03   NaN
50%    3.828200e+04   NaN
75%    1.289290e+05   NaN
max    1.359231e+06   NaN
```

[350]:
```
crops_prod_df.shape
```

[350]: (41649, 15)

[351]:
```
# Print information about the crops prod use data
crops_prod_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41649 entries, 0 to 41648
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Domain Code       41649 non-null  object
 1   Domain            41649 non-null  object
 2   Area Code (M49)   41649 non-null  int64
 3   Area              41649 non-null  object
 4   Element Code      41649 non-null  int64
 5   Element           41649 non-null  object
```

```
 6   Item Code (CPC)   41649 non-null  object
 7   Item              41649 non-null  object
 8   Year Code         41649 non-null  int64
 9   Year              41649 non-null  int64
 10  Unit              41649 non-null  object
 11  Value             41649 non-null  int64
 12  Flag              41649 non-null  object
 13  Flag Description  41649 non-null  object
 14  Note              0 non-null      float64
dtypes: float64(1), int64(5), object(9)
memory usage: 4.8+ MB
```

[352]:
```
# Filtering data to only take value from after the year 2000 to streamline data
crops_prod_df = crops_prod_df[(crops_prod_df['Year'] >= 2000)]

# Dropping not required columns
crops_prod_df = crops_prod_df.drop(columns=['Domain Code', 'Domain', 'Year␣
↪Code', 'Item', 'Item Code (CPC)', 'Element Code', 'Element', 'Unit', 'Flag',␣
↪'Flag Description', 'Area Code (M49)', 'Note'])

# Renaming the column to avoid confusion of value tables after merging.
crops_prod_df = crops_prod_df.rename(columns={'Value': 'Total Crops Yielded per␣
↪100g/ha'})
```

[353]:
```
crops_prod_df
```

[353]:

	Area	Year	Total Crops Yielded per 100g/ha
0	Afghanistan	2000	8063
1	Afghanistan	2001	10067
2	Afghanistan	2002	16698
3	Afghanistan	2003	14580
4	Afghanistan	2004	13348
…	…	…	…
41644	Zimbabwe	2018	66518
41645	Zimbabwe	2019	64830
41646	Zimbabwe	2020	65628
41647	Zimbabwe	2021	66126
41648	Zimbabwe	2022	65856

[41649 rows x 3 columns]

[354]:
```
# Checking for NaN values
crops_prod_df.isnull().sum()
```

[354]:
```
Area                               0
Year                               0
Total Crops Yielded per 100g/ha    0
dtype: int64
```

[355]:
```
# Check for duplicates
duplicates_found = crops_prod_df.duplicated().sum()

# If duplicates are found, remove them
if duplicates_found > 0:
    crops_prod_df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
else:
    print("No duplicates found.")
```

```
Duplicates removed.
```

[356]:
```
crops_prod_df.shape
```

[356]:
```
(41648, 3)
```

[357]:
```
crops_prod_df.head(100)
```

[357]:
```
           Area  Year  Total Crops Yielded per 100g/ha
0   Afghanistan  2000                             8063
1   Afghanistan  2001                            10067
2   Afghanistan  2002                            16698
3   Afghanistan  2003                            14580
4   Afghanistan  2004                            13348
..          …     …                              …
95  Afghanistan  2003                             3844
96  Afghanistan  2004                             3951
97  Afghanistan  2005                             3968
98  Afghanistan  2006                             3633
99  Afghanistan  2007                             3495

[100 rows x 3 columns]
```

3.3 2.3 Emissions Data

The following steps are performed - Display the data - Describe the data - Print Data Types - Checking for Null Values, Duplicates - Filtering Data by Dropping the columns from previous check

[358]:
```
# Load the dataset
emissions_df = pd.read_csv("content/Emissions - FAOSTAT_data_en_2-27-2024.csv",␣
 ↪low_memory=False)

# Display the first few rows of the emissions data
emissions_df.head()
```

[358]:
```
  Domain Code                Domain  Area Code (M49)         Area  \
0         GCE  Emissions from Crops                4  Afghanistan
1         GCE  Emissions from Crops                4  Afghanistan
```

```
2         GCE  Emissions from Crops                4  Afghanistan
3         GCE  Emissions from Crops                4  Afghanistan
4         GCE  Emissions from Crops                4  Afghanistan

   Element Code                      Element Item Code (CPC)       Item  \
0         72430  Crops total (Emissions N2O)            F1712  All Crops
1         72440  Crops total (Emissions CH4)            F1712  All Crops
2         72430  Crops total (Emissions N2O)            F1712  All Crops
3         72440  Crops total (Emissions CH4)            F1712  All Crops
4         72430  Crops total (Emissions N2O)            F1712  All Crops

   Year Code  Year  Source Code      Source Unit    Value Flag  \
0       2000  2000         3050  FAO TIER 1   kt   0.7056    E
1       2000  2000         3050  FAO TIER 1   kt  20.8471    E
2       2001  2001         3050  FAO TIER 1   kt   0.7054    E
3       2001  2001         3050  FAO TIER 1   kt  19.2605    E
4       2002  2002         3050  FAO TIER 1   kt   1.0656    E

  Flag Description  Note
0  Estimated value   NaN
1  Estimated value   NaN
2  Estimated value   NaN
3  Estimated value   NaN
4  Estimated value   NaN
```

[359]:

```python
# Describing the emissions data
emissions_df.describe()
```

[359]:

```
       Area Code (M49)  Element Code     Year Code          Year  Source Code  \
count     28910.000000  28910.000000  28910.000000  28910.000000      28910.0
mean        432.519543  26168.457281   2010.522414   2010.522414       3050.0
std         252.127600  29584.659513      6.342396      6.342396          0.0
min           4.000000   7230.000000   2000.000000   2000.000000       3050.0
25%         214.000000   7230.000000   2005.000000   2005.000000       3050.0
50%         428.000000   7273.000000   2011.000000   2011.000000       3050.0
75%         646.000000  72430.000000   2016.000000   2016.000000       3050.0
max         894.000000  72440.000000   2021.000000   2021.000000       3050.0

               Value  Note
count   28910.000000   0.0
mean      636.696462   NaN
std      6379.076614   NaN
min         0.000000   NaN
25%         0.000000   NaN
50%         0.021350   NaN
75%         3.655375   NaN
max    226389.853200   NaN
```

[360]:
```python
# Printing information about the emissions data
emissions_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28910 entries, 0 to 28909
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Domain Code       28910 non-null  object 
 1   Domain            28910 non-null  object 
 2   Area Code (M49)   28910 non-null  int64  
 3   Area              28910 non-null  object 
 4   Element Code      28910 non-null  int64  
 5   Element           28910 non-null  object 
 6   Item Code (CPC)   28910 non-null  object 
 7   Item              28910 non-null  object 
 8   Year Code         28910 non-null  int64  
 9   Year              28910 non-null  int64  
 10  Source Code       28910 non-null  int64  
 11  Source            28910 non-null  object 
 12  Unit              28910 non-null  object 
 13  Value             28910 non-null  float64
 14  Flag              28910 non-null  object 
 15  Flag Description  28910 non-null  object 
 16  Note              0 non-null      float64
dtypes: float64(2), int64(5), object(10)
memory usage: 3.7+ MB
```

[361]:
```python
emissions_df.shape
```

[361]: (28910, 17)

[362]:
```python
emissions_n20_df = emissions_df
emissions_ch4_df = emissions_df
```

[363]:
```python
# Filtering data to only take  and from after the year 2000
emissions_n20_df = emissions_n20_df[(emissions_n20_df['Year'] >= 2000) &␣
 ↪emissions_n20_df['Element'].str.contains('Crops total (Emissions N2O)',␣
 ↪regex=False)]
emissions_ch4_df = emissions_ch4_df[(emissions_ch4_df['Year'] >= 2000) &␣
 ↪emissions_ch4_df['Element'].str.contains('Crops total (Emissions CH4)',␣
 ↪regex=False)]

# Dropping not required columns
emissions_n20_df = emissions_n20_df.drop(columns=['Domain Code', 'Domain',␣
 ↪'Year Code', 'Item', 'Item Code (CPC)', 'Element Code', 'Source', 'Source␣
 ↪Code', 'Element', 'Unit', 'Flag', 'Flag Description', 'Area Code (M49)',␣
 ↪'Note'])
```

```
emissions_ch4_df = emissions_ch4_df.drop(columns=['Domain Code', 'Domain',␣
↪'Year Code', 'Item', 'Item Code (CPC)', 'Element Code', 'Source', 'Source␣
↪Code', 'Element', 'Unit', 'Flag', 'Flag Description', 'Area Code (M49)',␣
↪'Note'])

# Renaming the column to avoid confusion of value tables after merging.
emissions_n20_df = emissions_n20_df.rename(columns={'Value': 'Total Crop␣
↪Emissions N20'})
emissions_ch4_df = emissions_ch4_df.rename(columns={'Value': 'Total Crop␣
↪Emissions CH4'})
```

[364]:

```
# Checking for NaN values
dataframes = [emissions_n20_df, emissions_ch4_df]

for df in dataframes:
    print(f"Null values in {df.columns[0]}:")
    print(df.isnull().sum())
    print()
```

```
Null values in Area:
Area                        0
Year                        0
Total Crop Emissions N20    0
dtype: int64

Null values in Area:
Area                        0
Year                        0
Total Crop Emissions CH4    0
dtype: int64
```

[365]:

```
# Check for duplicates
dataframes = [emissions_n20_df, emissions_ch4_df]

for df in dataframes:
    print(f"Null values in {df.columns[0]}:")
    print(df.duplicated().sum())
    print()
```

```
Null values in Area:
0

Null values in Area:
0
```

[366]:

```
dataframes = [emissions_n20_df, emissions_ch4_df]
```

```python
for i, df in enumerate(dataframes):
    print(f"Before dropping duplicates in {df.columns[0]}:")
    print(df.shape)  # Print shape before dropping duplicates
    dataframes[i] = df.drop_duplicates()
    print(f"After dropping duplicates in {df.columns[0]}:")
    print(dataframes[i].shape)  # Print shape after dropping duplicates
    print()
```

```
Before dropping duplicates in Area:
(4228, 3)
After dropping duplicates in Area:
(4228, 3)

Before dropping duplicates in Area:
(4162, 3)
After dropping duplicates in Area:
(4162, 3)
```

3.4 2.4 Employment Data

The following steps are performed - Display the data - Describe the data - Print Data Types - Checking for Null Values, Duplicates - Filtering Data by Dropping the columns from previous check and merging them based on Country (i.e., Area) and Year.

[367]:
```python
# Load the dataset
employment_df = pd.read_csv("content/Employment - FAOSTAT_data_en_2-27-2024.
 ↪csv", low_memory=False)

# Display the first few rows of the employment data
employment_df.head()
```

[367]:
```
  Domain Code                             Domain  Area Code (M49)  \
0         OEA  Employment Indicators: Agriculture                4
1         OEA  Employment Indicators: Agriculture                4
2         OEA  Employment Indicators: Agriculture                4
3         OEA  Employment Indicators: Agriculture                4
4         OEA  Employment Indicators: Agriculture                4

          Area  Indicator Code  \
0  Afghanistan           21150
1  Afghanistan           21150
2  Afghanistan           21144
3  Afghanistan           21144
4  Afghanistan           21144

                                           Indicator  Sex Code    Sex  \
0  Mean weekly hours actually worked per employed…         1  Total
1  Mean weekly hours actually worked per employed…         1  Total
```

```
2  Employment in agriculture, forestry and fishin…         1  Total
3  Employment in agriculture, forestry and fishin…         1  Total
4  Employment in agriculture, forestry and fishin…         1  Total

   Year Code  Year  Element Code Element  Source Code  \
0       2014  2014          6173    Value         3021
1       2017  2017          6173    Value         3021
2       2000  2000          6199    Value         3043
3       2001  2001          6199    Value         3043
4       2002  2002          6199    Value         3043

                                Source     Unit    Value Flag  \
0  Household income and expenditure survey       No    31.68    X
1  Household income and expenditure survey       No    29.66    X
2             ILO - ILO Modelled Estimates  1000 No  2765.95    X
3             ILO - ILO Modelled Estimates  1000 No  2805.54    X
4             ILO - ILO Modelled Estimates  1000 No  2897.51    X

                        Flag Description  \
0  Figure from international organizations
1  Figure from international organizations
2  Figure from international organizations
3  Figure from international organizations
4  Figure from international organizations

                                              Note
0  Job coverage: Main job currently held Reposito…
1  Job coverage: Main job currently held Reposito…
2                                              NaN
3                                              NaN
4                                              NaN
```

[368]:
```python
# Describing the employment data
employment_df.describe()
```

[368]:
```
       Area Code (M49)  Indicator Code  Sex Code    Year Code         Year  \
count      5917.000000     5917.000000    5917.0  5917.000000  5917.000000
mean        427.420145    21145.763394       1.0  2010.890992  2010.890992
std         250.847292        2.733508       0.0     6.270884     6.270884
min           4.000000    21144.000000       1.0  2000.000000  2000.000000
25%         208.000000    21144.000000       1.0  2006.000000  2006.000000
50%         418.000000    21144.000000       1.0  2011.000000  2011.000000
75%         642.000000    21150.000000       1.0  2016.000000  2016.000000
max         894.000000    21150.000000       1.0  2022.000000  2022.000000

       Element Code  Source Code         Value
count   5917.000000  5917.000000   5917.000000
```

```
mean    6191.358628  3037.136049    4536.367847
std       11.845202     9.126779   27086.237113
min     6173.000000  3018.000000       0.170000
25%     6173.000000  3023.000000      39.100000
50%     6199.000000  3043.000000     126.540000
75%     6199.000000  3043.000000    1386.380000
max     6199.000000  3043.000000  358919.780000
```

[369]:
```
# Printing information about the employment data
employment_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5917 entries, 0 to 5916
Data columns (total 19 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Domain Code        5917 non-null   object
 1   Domain             5917 non-null   object
 2   Area Code (M49)    5917 non-null   int64
 3   Area               5917 non-null   object
 4   Indicator Code     5917 non-null   int64
 5   Indicator          5917 non-null   object
 6   Sex Code           5917 non-null   int64
 7   Sex                5917 non-null   object
 8   Year Code          5917 non-null   int64
 9   Year               5917 non-null   int64
 10  Element Code       5917 non-null   int64
 11  Element            5917 non-null   object
 12  Source Code        5917 non-null   int64
 13  Source             5917 non-null   object
 14  Unit               5917 non-null   object
 15  Value              5917 non-null   float64
 16  Flag               5917 non-null   object
 17  Flag Description   5917 non-null   object
 18  Note               3762 non-null   object
dtypes: float64(1), int64(7), object(11)
memory usage: 878.4+ KB
```

[370]:
```
employment_df.shape
```

[370]:
```
(5917, 19)
```

[371]:
```
# Filtering data to only take  and from after the year 2000
employment_df = employment_df[(employment_df['Year'] >= 2000) &␣
 ↪employment_df['Indicator'].str.contains('Employment in agriculture, forestry␣
 ↪and fishing - ILO modelled estimates')]

# Dropping not required columns
```

```
employment_df = employment_df.drop(columns=['Domain Code', 'Domain', 'Year␣
↪Code', 'Indicator', 'Indicator Code', 'Sex', 'Sex Code', 'Element Code',␣
↪'Element', 'Source', 'Source Code', 'Unit', 'Flag', 'Flag Description',␣
↪'Area Code (M49)', 'Note'])

# employment_df = employment_df.groupby(["Area", "Year"])["Value"].mean().
↪reset_index()

# Renaming the column to avoid confusion of value tables after merging.
employment_df = employment_df.rename(columns={'Value': 'Total Employment'})
```

[372]:
```
# Checking for NaN values
employment_df.isnull().sum()
```

[372]:
```
Area                0
Year                0
Total Employment    0
dtype: int64
```

[373]:
```
# Check for duplicates
duplicates_found = employment_df.duplicated().sum()

# If duplicates are found, remove them
if duplicates_found > 0:
    employment_df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
else:
    print("No duplicates found.")
```

```
No duplicates found.
```

[374]:
```
employment_df.shape
```

[374]:
```
(4178, 3)
```

[375]:
```
# display final dataset
employment_df.head(100)
```

[375]:
```
            Area  Year  Total Employment
2    Afghanistan  2000           2765.95
3    Afghanistan  2001           2805.54
4    Afghanistan  2002           2897.51
5    Afghanistan  2003           3093.27
6    Afghanistan  2004           3212.46
..           …    …                 …
114    Argentina  2008           1670.15
115    Argentina  2009           1639.59
116    Argentina  2010           1617.47
```

```
117     Argentina  2011            1598.24
118     Argentina  2012            1566.35

[100 rows x 3 columns]
```

3.5 2.5 Exchange Rates Data

The following steps are performed - Display the data - Describe the data - Print Data Types - Checking for Null Values, Duplicates - Filtering Data by Dropping the columns from previous check and merging them based on Country (i.e., Area) and Year.

[376]:
```python
# Load the dataset
exchange_rates_df = pd.read_csv("content/Exchange rate -␣
↪FAOSTAT_data_en_2-22-2024.csv", low_memory=False)

# Display the first few rows of the exchange rates data
exchange_rates_df.head()
```

[376]:
```
  Domain Code          Domain  Area Code (M49)         Area  \
0          PE  Exchange rates                4  Afghanistan
1          PE  Exchange rates                4  Afghanistan
2          PE  Exchange rates                4  Afghanistan
3          PE  Exchange rates                4  Afghanistan
4          PE  Exchange rates                4  Afghanistan

  ISO Currency Code (FAO) Currency Element Code                      Element  \
0                     AFA  Afghani          LCU  Local currency units per USD
1                     AFA  Afghani          LCU  Local currency units per USD
2                     AFA  Afghani          LCU  Local currency units per USD
3                     AFA  Afghani          LCU  Local currency units per USD
4                     AFA  Afghani          LCU  Local currency units per USD

   Year Code  Year  Months Code    Months  Unit      Value Flag  \
0       1980  1980         7001   January   NaN  44.129167    X
1       1980  1980         7002  February   NaN  44.129167    X
2       1980  1980         7003     March   NaN  44.129167    X
3       1980  1980         7004     April   NaN  44.129167    X
4       1980  1980         7005       May   NaN  44.129167    X

                          Flag Description
0  Figure from international organizations
1  Figure from international organizations
2  Figure from international organizations
3  Figure from international organizations
4  Figure from international organizations
```

[377]:
```python
# Describe the exchange rates data
exchange_rates_df.describe()
```

[377]:
```
       Area Code (M49)      Year Code           Year   Months Code  Unit  \
count    103276.000000  103276.000000  103276.000000  103276.000000   0.0
mean        428.219887    2002.605959    2002.605959    7006.493329   NaN
std         249.825569      12.427199      12.427199       3.450808   NaN
min           4.000000    1980.000000    1980.000000    7001.000000   NaN
25%         218.000000    1992.000000    1992.000000    7003.000000   NaN
50%         426.000000    2003.000000    2003.000000    7006.000000   NaN
75%         642.000000    2013.000000    2013.000000    7009.000000   NaN
max         894.000000    2023.000000    2023.000000    7012.000000   NaN

              Value
count  1.032760e+05
mean   7.841324e+05
std    2.176740e+08
min    8.160000e-06
25%    1.508627e+00
50%    7.501877e+00
75%    1.122701e+02
max    6.907838e+10
```

[378]:
```python
# Print information about the exchange rates data
exchange_rates_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103276 entries, 0 to 103275
Data columns (total 16 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   Domain Code               103276 non-null  object
 1   Domain                    103276 non-null  object
 2   Area Code (M49)           103276 non-null  int64
 3   Area                      103276 non-null  object
 4   ISO Currency Code (FAO)   103276 non-null  object
 5   Currency                  103276 non-null  object
 6   Element Code              103276 non-null  object
 7   Element                   103276 non-null  object
 8   Year Code                 103276 non-null  int64
 9   Year                      103276 non-null  int64
 10  Months Code               103276 non-null  int64
 11  Months                    103276 non-null  object
 12  Unit                      0 non-null       float64
 13  Value                     103276 non-null  float64
 14  Flag                      103276 non-null  object
 15  Flag Description          103276 non-null  object
dtypes: float64(2), int64(4), object(10)
```

memory usage: 12.6+ MB

[379]:
```
exchange_rates_df.shape
```

[379]: (103276, 16)

[380]:
```
# Filtering data to only take food price inflation and from after the year 2000
exchange_rates_df = exchange_rates_df[(exchange_rates_df['Year'] >= 2000)]

# Dropping not required columns
exchange_rates_df = exchange_rates_df.drop(columns=['Domain Code', 'Domain',
↪'Year Code', 'ISO Currency Code (FAO)', 'Months', 'Element Code', 'Element',
↪'Unit', 'Flag', 'Flag Description', 'Months Code', 'Area Code (M49)'])

exchange_rates_df = exchange_rates_df.groupby(["Area", "Year"])["Value"].mean().
↪reset_index()

# Renaming the column to avoid confusion of value tables after merging.
exchange_rates_df = exchange_rates_df.rename(columns={'Value': 'Total Exchange
↪Rate per USD'})
```

[381]:
```
# Checking for NaN values
exchange_rates_df.isnull().sum()
```

[381]:
```
Area                           0
Year                           0
Total Exchange Rate per USD    0
dtype: int64
```

[382]:
```
# If duplicates are found, remove them
if duplicates_found > 0:
    exchange_rates_df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
else:
    print("No duplicates found.")
```

No duplicates found.

[383]:
```
exchange_rates_df.shape
```

[383]: (5069, 3)

[384]:
```
# display final dataset
exchange_rates_df.head(100)
```

[384]:

	Area	Year	Total Exchange Rate per USD
0	Afghanistan	2000	47357.574730
1	Afghanistan	2001	47500.014520

```
2   Afghanistan  2002                    3981.907750
3   Afghanistan  2003                      48.762754
4   Afghanistan  2004                      47.845312
..          …    …                              …
95       Angola  2001                      22.057862
96       Angola  2002                      43.530207
97       Angola  2003                      74.606301
98       Angola  2004                      83.541363
99       Angola  2005                      87.159142

[100 rows x 3 columns]
```

3.6 2.6 Fertilizers Used Data

The following steps are performed - Display the data - Describe the data - Print Data Types - Checking for Null Values, Duplicates - Filtering Data by Dropping the columns from previous check and merging them based on Country (i.e., Area) and Year.

[385]:
```
# Load the dataset
fertilizers_use_df = pd.read_csv("content/Fertilizers use -␣
↪FAOSTAT_data_en_2-27-2024.csv", low_memory=False)

# Display the first few rows of the Fertilizers Used data
fertilizers_use_df.head()
```

[385]:
```
  Domain Code                  Domain  Area Code (M49)         Area  \
0         RFB  Fertilizers by Product                4  Afghanistan
1         RFB  Fertilizers by Product                4  Afghanistan
2         RFB  Fertilizers by Product                4  Afghanistan
3         RFB  Fertilizers by Product                4  Afghanistan
4         RFB  Fertilizers by Product                4  Afghanistan

   Element Code           Element  Item Code            Item  Year Code  \
0          5157  Agricultural Use       4021  NPK fertilizers       2002
1          5157  Agricultural Use       4021  NPK fertilizers       2003
2          5157  Agricultural Use       4021  NPK fertilizers       2004
3          5157  Agricultural Use       4001             Urea       2004
4          5157  Agricultural Use       4001             Urea       2005

   Year Unit    Value Flag Flag Description
0  2002    t  17900.0    I    Imputed value
1  2003    t  33200.0    I    Imputed value
2  2004    t  47700.0    I    Imputed value
3  2004    t  42300.0    I    Imputed value
4  2005    t  20577.0    I    Imputed value
```

[386]:
```python
# Describe the Fertilizers Used data
fertilizers_use_df.describe()
```

[386]:
```
       Area Code (M49)  Element Code     Item Code     Year Code  \
count     17807.000000       17807.0  17807.000000  17807.000000
mean        428.095861        5157.0   4013.974224   2011.259224
std         252.862476           0.0      9.034514      5.443312
min           4.000000        5157.0   4001.000000   2002.000000
25%         208.000000        5157.0   4004.000000   2007.000000
50%         414.000000        5157.0   4016.000000   2011.000000
75%         620.000000        5157.0   4022.000000   2016.000000
max         894.000000        5157.0   4030.000000   2021.000000

               Year         Value
count  17807.000000  1.780700e+04
mean    2011.259224  2.124516e+05
std        5.443312  1.408350e+06
min     2002.000000  0.000000e+00
25%     2007.000000  1.000000e+02
50%     2011.000000  3.584000e+03
75%     2016.000000  4.573800e+04
max     2021.000000  9.621329e+07
```

[387]:
```python
# Print information about the Fertilizers Used data
fertilizers_use_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17807 entries, 0 to 17806
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Domain Code       17807 non-null  object
 1   Domain            17807 non-null  object
 2   Area Code (M49)   17807 non-null  int64
 3   Area              17807 non-null  object
 4   Element Code      17807 non-null  int64
 5   Element           17807 non-null  object
 6   Item Code         17807 non-null  int64
 7   Item              17807 non-null  object
 8   Year Code         17807 non-null  int64
 9   Year              17807 non-null  int64
 10  Unit              17807 non-null  object
 11  Value             17807 non-null  float64
 12  Flag              17807 non-null  object
 13  Flag Description  17807 non-null  object
dtypes: float64(1), int64(5), object(8)
memory usage: 1.9+ MB
```

[388]:
```python
fertilizers_use_df.shape
```

[388]: (17807, 14)

[389]:
```python
fertilizers_use_df = fertilizers_use_df[(fertilizers_use_df['Year'] >= 2000)]
fertilizers_use_df = fertilizers_use_df.drop(columns=['Domain Code', 'Domain',␣
↪'Year Code', 'Item', 'Item Code', 'Element Code', 'Element', 'Unit', 'Flag',␣
↪'Flag Description', 'Area Code (M49)'])

fertilizers_use_df = fertilizers_use_df.groupby(["Area", "Year"])["Value"].
↪mean().reset_index()

fertilizers_use_df = fertilizers_use_df.rename(columns={'Value': 'Total␣
↪Fertilizers Used in Tonnes'})
```

[390]:
```python
# Check for missing values
fertilizers_use_df.isnull().sum()
```

[390]:
```
Area                             0
Year                             0
Total Fertilizers Used in Tonnes    0
dtype: int64
```

[391]:
```python
# If duplicates are found, remove them
if duplicates_found > 0:
    exchange_rates_df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
else:
    print("No duplicates found.")
```

```
No duplicates found.
```

[392]:
```python
# display final dataset
fertilizers_use_df.head(100)
```

[392]:

	Area	Year	Total Fertilizers Used in Tonnes
0	Afghanistan	2002	17900.000000
1	Afghanistan	2003	33200.000000
2	Afghanistan	2004	45000.000000
3	Afghanistan	2005	20577.000000
4	Afghanistan	2006	68253.000000
..	…	…	…
95	Australia	2013	369481.000000
96	Australia	2014	433516.000000
97	Australia	2015	446052.416667
98	Australia	2016	397817.750000
99	Australia	2017	399294.600000

```
[100 rows x 3 columns]
```

[393]:
```
fertilizers_use_df.shape
```

[393]:
```
(1933, 3)
```

3.7 2.7 Food Balance Data

The following steps are performed - Display the data - Describe the data - Print Data Types - Checking for Null Values, Duplicates - Filtering Data by Dropping the columns from previous check and merging them based on Country (i.e., Area) and Year.

[394]:
```
food_balances_df = pd.read_csv("content/Food balances indicators -␣
↪FAOSTAT_data_en_2-22-2024.csv", low_memory=False)

# Display the first few rows of the food balances data
food_balances_df.head()
```

[394]:
```
  Domain Code                 Domain  Area Code (M49)         Area  \
0         FBS  Food Balances (2010-)                4  Afghanistan
1         FBS  Food Balances (2010-)                4  Afghanistan
2         FBS  Food Balances (2010-)                4  Afghanistan
3         FBS  Food Balances (2010-)                4  Afghanistan
4         FBS  Food Balances (2010-)                4  Afghanistan

   Element Code          Element Item Code (FBS)                     Item  \
0          5611  Import Quantity           S2905  Cereals - Excluding Beer
1          5611  Import Quantity           S2905  Cereals - Excluding Beer
2          5611  Import Quantity           S2905  Cereals - Excluding Beer
3          5611  Import Quantity           S2905  Cereals - Excluding Beer
4          5611  Import Quantity           S2905  Cereals - Excluding Beer

   Year Code  Year    Unit   Value Flag Flag Description
0       2010  2010  1000 t  2000.0    E  Estimated value
1       2011  2011  1000 t  2448.0    E  Estimated value
2       2012  2012  1000 t  2001.0    E  Estimated value
3       2013  2013  1000 t  2155.0    E  Estimated value
4       2014  2014  1000 t  1840.0    E  Estimated value
```

[395]:
```
# Describing the food balances data
food_balances_df.describe()
```

[395]:
```
       Area Code (M49)   Element Code      Year Code           Year  \
count    148041.000000  148041.000000  148041.000000  148041.000000
mean        425.675185    5429.812417    2015.549274    2015.549274
std         251.359288     324.840991       3.452477       3.452477
min           4.000000    5123.000000    2010.000000    2010.000000
25%         204.000000    5142.000000    2013.000000    2013.000000
```

```
50%          417.000000    5154.000000    2016.000000    2016.000000
75%          642.000000    5611.000000    2019.000000    2019.000000
max          894.000000    5911.000000    2021.000000    2021.000000

               Value
count  148041.000000
mean      957.153400
std      9591.749593
min       -62.000000
25%         1.000000
50%        25.000000
75%       218.190000
max    573218.000000
```

[396]:

```python
# Printing information about the food balances data
food_balances_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148041 entries, 0 to 148040
Data columns (total 14 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Domain Code       148041 non-null  object
 1   Domain            148041 non-null  object
 2   Area Code (M49)   148041 non-null  int64
 3   Area              148041 non-null  object
 4   Element Code      148041 non-null  int64
 5   Element           148041 non-null  object
 6   Item Code (FBS)   148041 non-null  object
 7   Item              148041 non-null  object
 8   Year Code         148041 non-null  int64
 9   Year              148041 non-null  int64
 10  Unit              148041 non-null  object
 11  Value             148041 non-null  float64
 12  Flag              148041 non-null  object
 13  Flag Description  148041 non-null  object
dtypes: float64(1), int64(4), object(9)
memory usage: 15.8+ MB
```

[397]:

```python
food_balances_df.shape
```

[397]: (148041, 14)

[398]:

```python
# Filtering data to only take  and from after the year 2000
food_balances_df = food_balances_df[(food_balances_df['Year'] >= 2000) &␣
 ↪(food_balances_df['Element'].str.contains('Export Quantity'))]
```

```
# Further filter to exclude items that contain bewlo entries not produced from␣
↪crops
food_balances_df = food_balances_df[~food_balances_df['Item'].str.
↪contains('Fish, Seafood', case=False, na=False)]
food_balances_df = food_balances_df[~food_balances_df['Item'].str.
↪contains('Meat', case=False, na=False)]
food_balances_df = food_balances_df[~food_balances_df['Item'].str.
↪contains('Eggs', case=False, na=False)]
food_balances_df = food_balances_df[~food_balances_df['Item'].str.
↪contains('Milk - Excluding Butter', case=False, na=False)]

# Dropping not required columns
food_balances_df = food_balances_df.drop(columns=['Domain Code', 'Domain',␣
↪'Year Code', 'Item', 'Item Code (FBS)', 'Element Code', 'Element', 'Unit',␣
↪'Flag', 'Flag Description', 'Area Code (M49)'])

food_balances_df = food_balances_df.groupby(["Area", "Year"])["Value"].mean().
↪reset_index()

# Renaming the column to avoid confusion of value tables after merging.
food_balances_df = food_balances_df.rename(columns={'Value': 'Total Crop Based␣
↪Food Balance per 1000t'})
```

[399]:
```
# Checking for NaN values
food_balances_df.isnull().sum()
```

[399]:
```
Area                                      0
Year                                      0
Total Crop Based Food Balance per 1000t    0
dtype: int64
```

[400]:
```
# Check for duplicates
duplicates_found = food_balances_df.duplicated().sum()

# If duplicates are found, remove them
if duplicates_found > 0:
    food_balances_df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
else:
    print("No duplicates found.")
```

```
No duplicates found.
```

[401]:
```
food_balances_df.shape
```

[401]:
```
(2176, 3)
```

[402]:
```
# display final dataset
food_balances_df.head(100)
```

[402]:
```
          Area  Year  Total Crop Based Food Balance per 1000t
0   Afghanistan  2010                                 40.000000
1   Afghanistan  2011                                 30.777778
2   Afghanistan  2012                                 22.000000
3   Afghanistan  2013                                 31.222222
4   Afghanistan  2014                                 34.333333
..          …     …                                       …
95    Australia  2021                               3992.076923
96      Austria  2010                                445.846154
97      Austria  2011                                451.846154
98      Austria  2012                                462.076923
99      Austria  2013                                436.000000

[100 rows x 3 columns]
```

3.8 2.8 Food Security Data

The following steps are performed - Display the data - Describe the data - Print Data Types - Checking for Null Values, Duplicates - Filtering Data by Dropping the columns from previous check and merging them based on Country (i.e., Area) and Year.

[403]:
```
food_security_df = pd.read_csv("content/Food security indicators  -␣
↪FAOSTAT_data_en_2-22-2024.csv", low_memory=False)

# Display the first few rows of the food security data
food_security_df.head()
```

[403]:
```
  Domain Code                              Domain  Area Code (M49)  \
0          FS  Suite of Food Security Indicators                4
1          FS  Suite of Food Security Indicators                4
2          FS  Suite of Food Security Indicators                4
3          FS  Suite of Food Security Indicators                4
4          FS  Suite of Food Security Indicators                4

          Area  Element Code Element  Item Code  \
0  Afghanistan          6121   Value      21010
1  Afghanistan          6121   Value      21010
2  Afghanistan          6121   Value      21010
3  Afghanistan          6121   Value      21010
4  Afghanistan          6121   Value      21010

                                                Item  Year Code       Year  \
0  Average dietary energy supply adequacy (percen…   20002002  2000-2002
1  Average dietary energy supply adequacy (percen…   20012003  2001-2003
2  Average dietary energy supply adequacy (percen…   20022004  2002-2004
```

```
3  Average dietary energy supply adequacy (percen…   20032005  2003-2005
4  Average dietary energy supply adequacy (percen…   20042006  2004-2006

  Unit  Value Flag Flag Description Note
0    %   88.0    E  Estimated value  NaN
1    %   89.0    E  Estimated value  NaN
2    %   92.0    E  Estimated value  NaN
3    %   93.0    E  Estimated value  NaN
4    %   94.0    E  Estimated value  NaN
```

[404]:
```python
# Describing the food security data
food_security_df.describe()
```

[404]:
```
       Area Code (M49)  Element Code    Item Code     Year Code          Value
count     36512.000000  36512.000000  36512.000000  3.651200e+04  36512.000000
mean        424.835342   6122.999233  21030.970777  9.691701e+06     37.620671
std         252.424973      2.662834     11.014761  1.004127e+07     67.159815
min           4.000000   6121.000000  21010.000000  2.000000e+03   -654.900000
25%         204.000000   6121.000000  21030.000000  2.010000e+03      5.000000
50%         417.000000   6121.000000  21032.000000  2.020000e+03     18.700000
75%         642.000000   6125.000000  21035.000000  2.009201e+07     58.700000
max         894.000000   6128.000000  21049.000000  2.020202e+07   5735.000000
```

[405]:
```python
# Printing information about the food security data
food_security_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36512 entries, 0 to 36511
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Domain Code       36512 non-null  object
 1   Domain            36512 non-null  object
 2   Area Code (M49)   36512 non-null  int64
 3   Area              36512 non-null  object
 4   Element Code      36512 non-null  int64
 5   Element           36512 non-null  object
 6   Item Code         36512 non-null  int64
 7   Item              36512 non-null  object
 8   Year Code         36512 non-null  int64
 9   Year              36512 non-null  object
 10  Unit              36512 non-null  object
 11  Value             36512 non-null  float64
 12  Flag              36512 non-null  object
 13  Flag Description  36512 non-null  object
 14  Note              1 non-null      object
dtypes: float64(1), int64(4), object(10)
memory usage: 4.2+ MB
```

[406]:
```
food_security_df.shape
```

[406]:
```
(36512, 15)
```

[407]:
```
food_security_df = food_security_df[(food_security_df['Year'] >= "2000") &␣
↪(food_security_df['Item Code'] == 21031)]

# Dropping not required columns
food_security_df = food_security_df.drop(columns=['Domain Code', 'Domain',␣
↪'Year Code', 'Item', 'Item Code', 'Element Code', 'Element', 'Unit', 'Flag',␣
↪'Flag Description', 'Area Code (M49)', 'Note'])

# Renaming the column to avoid confusion of value tables after merging.
food_security_df = food_security_df.rename(columns={'Value': 'Total Food Supply␣
↪Variability per Capita'})
```

[408]:
```
# Checking for NaN values
food_security_df.isnull().sum()
```

[408]:
```
Area                                       0
Year                                       0
Total Food Supply Variability per Capita   0
dtype: int64
```

[409]:
```
# Check for duplicates
duplicates_found = food_security_df.duplicated().sum()

# If duplicates are found, remove them
if duplicates_found > 0:
    food_security_df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
else:
    print("No duplicates found.")
```

```
No duplicates found.
```

[410]:
```
food_security_df.shape
```

[410]:
```
(3776, 3)
```

[411]:
```
# display final dataset
food_security_df.head(100)
```

[411]:

	Area	Year	Total Food Supply Variability per Capita
140	Afghanistan	2000	58.0
141	Afghanistan	2001	47.0
142	Afghanistan	2002	71.0
143	Afghanistan	2003	72.0

```
144   Afghanistan  2004                                         50.0
…           …    …                                     …
1196    Argentina  2006                                         92.0
1197    Argentina  2007                                         61.0
1198    Argentina  2008                                         57.0
1199    Argentina  2009                                         47.0
1200    Argentina  2010                                         18.0

[100 rows x 3 columns]
```

3.9 2.9 Food Trade Data

The following steps are performed - Display the data - Describe the data - Print Data Types - Checking for Null Values, Duplicates - Filtering Data by Dropping the columns from previous check

[412]:
```
# Load the dataset
food_trade_df = pd.read_csv("content/Food trade indicators -␣
↪FAOSTAT_data_en_2-22-2024.csv", low_memory=False)

# Display the first few rows of the Food Trade data
food_trade_df.head()
```

[412]:
```
  Domain Code                        Domain  Area Code (M49)         Area  \
0         TCL  Crops and livestock products                4  Afghanistan
1         TCL  Crops and livestock products                4  Afghanistan
2         TCL  Crops and livestock products                4  Afghanistan
3         TCL  Crops and livestock products                4  Afghanistan
4         TCL  Crops and livestock products                4  Afghanistan

   Element Code       Element Item Code (CPC)                      Item  \
0          5622  Import Value           F1888  Cereals and Preparations
1          5622  Import Value           F1888  Cereals and Preparations
2          5622  Import Value           F1888  Cereals and Preparations
3          5622  Import Value           F1888  Cereals and Preparations
4          5622  Import Value           F1888  Cereals and Preparations

   Year Code  Year      Unit    Value Flag Flag Description  Note
0       1991  1991  1000 USD  41600.0    A  Official figure   NaN
1       1992  1992  1000 USD  25600.0    E  Estimated value   NaN
2       1993  1993  1000 USD  40000.0    E  Estimated value   NaN
3       1994  1994  1000 USD  25700.0    E  Estimated value   NaN
4       1995  1995  1000 USD  37720.0    E  Estimated value   NaN
```

[413]:
```
# Describe the Food Trade data
food_trade_df.describe()
```

[413]:
```
       Area Code (M49)  Element Code      Year Code           Year  \
count    141738.000000 141738.000000  141738.000000  141738.000000
mean        424.988359   5765.555010    2006.724273    2006.724273
std         253.512489    149.862005       9.168199       9.168199
min           4.000000   5622.000000    1991.000000    1991.000000
25%         204.000000   5622.000000    1999.000000    1999.000000
50%         414.000000   5622.000000    2007.000000    2007.000000
75%         643.000000   5922.000000    2015.000000    2015.000000
max         894.000000   5922.000000    2022.000000    2022.000000

              Value  Note
count  1.417380e+05   0.0
mean   4.572981e+05   NaN
std    1.876930e+06   NaN
min    0.000000e+00   NaN
25%    2.150000e+03   NaN
50%    2.406200e+04   NaN
75%    1.764239e+05   NaN
max    8.355806e+07   NaN
```

[414]:
```python
# Print information about the Food Trade data
food_trade_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 141738 entries, 0 to 141737
Data columns (total 15 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Domain Code       141738 non-null  object
 1   Domain            141738 non-null  object
 2   Area Code (M49)   141738 non-null  int64
 3   Area              141738 non-null  object
 4   Element Code      141738 non-null  int64
 5   Element           141738 non-null  object
 6   Item Code (CPC)   141738 non-null  object
 7   Item              141738 non-null  object
 8   Year Code         141738 non-null  int64
 9   Year              141738 non-null  int64
 10  Unit              141738 non-null  object
 11  Value             141738 non-null  float64
 12  Flag              141738 non-null  object
 13  Flag Description  141738 non-null  object
 14  Note              0 non-null       float64
dtypes: float64(2), int64(4), object(9)
memory usage: 16.2+ MB
```

[415]:
```python
food_trade_df.shape
```

[415]: (141738, 15)

[416]:
```
cereals_df = food_trade_df
sugar_honey_df = food_trade_df
fruits_vegetables_df = food_trade_df
tobacco_df = food_trade_df
fats_oils_df = food_trade_df
```

[417]:
```
# Filtering data to only take  and from after the year 2000
food_trade_df = food_trade_df[(food_trade_df['Year'] >= 2000) &␣
 ↪(food_trade_df['Element'].str.contains('Export Value'))]

cereals_df = cereals_df[(cereals_df['Year'] >= 2000) & (cereals_df['Element'].
 ↪str.contains('Export Value') ) & (cereals_df['Item'].str.contains('Cereals␣
 ↪and Preparations'))]

sugar_honey_df = sugar_honey_df[(sugar_honey_df['Year'] >= 2000) &␣
 ↪(sugar_honey_df['Element'].str.contains('Export Value')) &␣
 ↪(sugar_honey_df['Item'].str.contains('Sugar and Honey'))]

fruits_vegetables_df = fruits_vegetables_df[(fruits_vegetables_df['Year'] >=␣
 ↪2000) & (fruits_vegetables_df['Element'].str.contains('Export Value')) &␣
 ↪(fruits_vegetables_df['Item'].str.contains('Fruit and Vegetables'))]

tobacco_df = tobacco_df[(tobacco_df['Year'] >= 2000) & (tobacco_df['Element'].
 ↪str.contains('Export Value')) & (tobacco_df['Item'].str.contains('Tobacco'))]

fats_oils_df = fats_oils_df[(fats_oils_df['Year'] >= 2000) &␣
 ↪(fats_oils_df['Element'].str.contains('Export Value')) &␣
 ↪(fats_oils_df['Item'].str.contains('Fats and Oils (excluding Butter)',␣
 ↪regex=False))]

food_trade_df = food_trade_df.drop(columns=['Domain Code', 'Domain', 'Year␣
 ↪Code', 'Item', 'Item Code (CPC)', 'Element Code', 'Element', 'Unit', 'Flag',␣
 ↪'Flag Description', 'Area Code (M49)', 'Note'])
cereals_df = cereals_df.drop(columns=['Domain Code', 'Domain', 'Year Code',␣
 ↪'Item', 'Item Code (CPC)', 'Element Code', 'Element', 'Unit', 'Flag', 'Flag␣
 ↪Description', 'Area Code (M49)', 'Note'])
sugar_honey_df = sugar_honey_df.drop(columns=['Domain Code', 'Domain', 'Year␣
 ↪Code', 'Item', 'Item Code (CPC)', 'Element Code', 'Element', 'Unit', 'Flag',␣
 ↪'Flag Description', 'Area Code (M49)', 'Note'])
fruits_vegetables_df = fruits_vegetables_df.drop(columns=['Domain Code',␣
 ↪'Domain', 'Year Code', 'Item', 'Item Code (CPC)', 'Element Code', 'Element',␣
 ↪'Unit', 'Flag', 'Flag Description', 'Area Code (M49)', 'Note'])
```

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

food_trade_df = food_trade_df.rename(columns={'Value': 'Total Crop Export Value␣
↪per 1000 USD'})
cereals_df = cereals_df.rename(columns={'Value': 'Total Cereal Export Value per␣
↪1000 USD'})
sugar_honey_df = sugar_honey_df.rename(columns={'Value': 'Total Sugar and Honey␣
↪Export Value per 1000 USD'})
fruits_vegetables_df = fruits_vegetables_df.rename(columns={'Value': 'Total␣
↪Fruits and Vegetables Export Value per 1000 USD'})
tobacco_df = tobacco_df.rename(columns={'Value': 'Total Tobacco Export Value␣
↪per 1000 USD'})
fats_oils_df = fats_oils_df.rename(columns={'Value': 'Total Fats and Oils␣
↪Export Value per 1000 USD'})
```

[418]:

```
# Check for missing values
dataframes = [cereals_df, sugar_honey_df, fruits_vegetables_df, tobacco_df,␣
↪fats_oils_df]

for df in dataframes:
    print(f"Null values in {df.columns[0]}:")
    print(df.isnull().sum())
    print()
```

```
Null values in Area:
Area                                    0
Year                                    0
Total Cereal Export Value per 1000 USD    0
dtype: int64

Null values in Area:
Area                                              0
Year                                              0
Total Sugar and Honey Export Value per 1000 USD    0
dtype: int64

Null values in Area:
Area                                                     0
Year                                                     0
Total Fruits and Vegetables Export Value per 1000 USD    0
dtype: int64
```

```
Null values in Area:
Area                                    0
Year                                    0
Total Tobacco Export Value per 1000 USD    0
dtype: int64

Null values in Area:
Area                                          0
Year                                          0
Total Fats and Oils Export Value per 1000 USD    0
dtype: int64
```

[419]:
```python
# Check for duplicates
# Check for missing values
dataframes = [cereals_df, sugar_honey_df, fruits_vegetables_df, tobacco_df,
 ↪fats_oils_df]

for df in dataframes:
    print(f"Null values in {df.columns[0]}:")
    print(df.duplicated().sum())
    print()
```

```
Null values in Area:
0

Null values in Area:
0

Null values in Area:
0

Null values in Area:
0

Null values in Area:
0
```

[420]:
```python
dataframes = [cereals_df, sugar_honey_df, fruits_vegetables_df, tobacco_df,
 ↪fats_oils_df]
for i, df in enumerate(dataframes):
    print(f"Before dropping duplicates in {df.columns[0]}:")
    print(df.shape)  # Print shape before dropping duplicates
    dataframes[i] = df.drop_duplicates()
    print(f"After dropping duplicates in {df.columns[0]}:")
    print(dataframes[i].shape)  # Print shape after dropping duplicates
    print()
```

```
Before dropping duplicates in Area:
```

```
(4268, 3)
After dropping duplicates in Area:
(4268, 3)

Before dropping duplicates in Area:
(4164, 3)
After dropping duplicates in Area:
(4164, 3)

Before dropping duplicates in Area:
(4404, 3)
After dropping duplicates in Area:
(4404, 3)

Before dropping duplicates in Area:
(4045, 3)
After dropping duplicates in Area:
(4045, 3)

Before dropping duplicates in Area:
(4197, 3)
After dropping duplicates in Area:
(4197, 3)
```

3.10 2.10 FDI Data

The following steps are performed - Display the data - Describe the data - Print Data Types - Checking for Null Values, Duplicates - Filtering Data by Dropping the columns from previous check and merging them based on Country (i.e., Area) and Year.

[421]:
```
# Load the dataset
fdi_df = pd.read_csv("content/Foreign direct investment -␣
↪FAOSTAT_data_en_2-27-2024.csv", low_memory=False)

# Display the first few rows of the FDI data
fdi_df.head()
```

[421]:
```
  Domain Code                                Domain  Area Code (M49)         Area  \
0         FDI   Foreign Direct Investment (FDI)                    4  Afghanistan
1         FDI   Foreign Direct Investment (FDI)                    4  Afghanistan
2         FDI   Foreign Direct Investment (FDI)                    4  Afghanistan
3         FDI   Foreign Direct Investment (FDI)                    4  Afghanistan
4         FDI   Foreign Direct Investment (FDI)                    4  Afghanistan

   Element Code    Element  Item Code             Item  Year Code  Year  \
0          6110  Value US$      23082  Total FDI inflows       2000  2000
1          6110  Value US$      23082  Total FDI inflows       2001  2001
2          6110  Value US$      23082  Total FDI inflows       2002  2002
```

```
3          6110  Value US$      23082  Total FDI inflows       2003  2003
4          6110  Value US$      23082  Total FDI inflows       2004  2004

          Unit   Value Flag                       Flag Description    Note
0  million USD    0.17    X  Figure from international organizations  UNCTAD
1  million USD    0.68    X  Figure from international organizations  UNCTAD
2  million USD   50.00    X  Figure from international organizations  UNCTAD
3  million USD   57.80    X  Figure from international organizations  UNCTAD
4  million USD  186.90    X  Figure from international organizations  UNCTAD
```

[422]:

```python
# Describe the FDI data
fdi_df.describe()
```

[422]:

```
       Area Code (M49)  Element Code     Item Code     Year Code  \
count     12276.000000       12276.0  12276.000000  12276.000000
mean        420.778674        6110.0  23082.692978   2011.305148
std         248.237052           0.0      1.745190      6.470153
min           4.000000        6110.0  23080.000000   2000.000000
25%         204.000000        6110.0  23082.000000   2006.000000
50%         410.000000        6110.0  23082.000000   2012.000000
75%         626.000000        6110.0  23085.000000   2017.000000
max         894.000000        6110.0  23085.000000   2022.000000

               Year          Value
count  12276.000000   12276.000000
mean    2011.305148    5230.433618
std        6.470153   23875.653754
min     2000.000000 -322053.781300
25%     2006.000000       4.914940
50%     2012.000000      93.866445
75%     2017.000000    1116.813653
max     2022.000000  467625.000000
```

[423]:

```python
# Print information about the FDI data
fdi_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12276 entries, 0 to 12275
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Domain Code       12276 non-null  object
 1   Domain            12276 non-null  object
 2   Area Code (M49)   12276 non-null  int64
 3   Area              12276 non-null  object
 4   Element Code      12276 non-null  int64
 5   Element           12276 non-null  object
 6   Item Code         12276 non-null  int64
```

```
 7   Item               12276 non-null  object
 8   Year Code          12276 non-null  int64
 9   Year               12276 non-null  int64
 10  Unit               12276 non-null  object
 11  Value              12276 non-null  float64
 12  Flag               12276 non-null  object
 13  Flag Description   12276 non-null  object
 14  Note               12276 non-null  object
dtypes: float64(1), int64(5), object(9)
memory usage: 1.4+ MB
```

[424]:
```
fdi_df.shape
```

[424]: (12276, 15)

[425]:
```
fdi_df = fdi_df[(fdi_df['Year'] >= 2000) & fdi_df['Item'].str.contains('Total␣
↪FDI inflows')]

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

fdi_df = fdi_df.rename(columns={'Value': 'Total FDI Inflows per million USD'})
```

[426]:
```
# Checking for NaN values
fdi_df.isnull().sum()
```

[426]:
```
Area                                 0
Year                                 0
Total FDI Inflows per million USD    0
dtype: int64
```

[427]:
```
# Check for duplicates
duplicates_found = fdi_df.duplicated().sum()

# If duplicates are found, remove them
if duplicates_found > 0:
    fdi_df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
else:
    print("No duplicates found.")
```

No duplicates found.

[428]:
```
fdi_df.shape
```

[428]: (4566, 3)

[429]:
```
# display final dataset
fdi_df.head(100)
```

[429]:
```
            Area  Year  Total FDI Inflows per million USD
0    Afghanistan  2000                           0.170000
1    Afghanistan  2001                           0.680000
2    Afghanistan  2002                          50.000000
3    Afghanistan  2003                          57.800000
4    Afghanistan  2004                         186.900000
..           …     …                                 …
200     Anguilla  2004                          91.750519
201     Anguilla  2005                         118.584133
202     Anguilla  2006                         143.182989
203     Anguilla  2007                         120.132137
204     Anguilla  2008                         100.849210

[100 rows x 3 columns]
```

3.11 2.11 Land Temp Change Data

The following steps are performed - Display the data - Describe the data - Print Data Types - Checking for Null Values, Duplicates - Filtering Data by Dropping the columns from previous check and merging them based on Country (i.e., Area) and Year.

[430]:
```
# Load the dataset
land_temp_change_df = pd.read_csv("content/Land temperature change -␣
↪FAOSTAT_data_en_2-27-2024.csv", low_memory=False)

# Display the first few rows of the Land Temp Change data
land_temp_change_df.head()
```

[430]:
```
  Domain Code                      Domain  Area Code (M49)         Area  \
0          ET  Temperature change on land                4  Afghanistan
1          ET  Temperature change on land                4  Afghanistan
2          ET  Temperature change on land                4  Afghanistan
3          ET  Temperature change on land                4  Afghanistan
4          ET  Temperature change on land                4  Afghanistan

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

   Year Unit  Value Flag Flag Description
0  2000   °c  0.618    E  Estimated value
1  2001   °c  0.365    E  Estimated value
```

```
2  2002   °c  1.655    E  Estimated value
3  2003   °c  0.997    E  Estimated value
4  2004   °c  1.883    E  Estimated value
```

[431]:
```
# Describe the Land Temp Change data
land_temp_change_df.describe()
```

[431]:
```
       Area Code (M49)  Element Code   Months Code     Year Code  \
count     54810.000000  54810.000000  54810.000000  54810.000000
mean        434.977194   6674.500000   7018.000000   2011.021346
std         253.978304    596.505442      1.414226      6.629795
min           4.000000   6078.000000   7016.000000   2000.000000
25%         214.000000   6078.000000   7017.000000   2005.000000
50%         434.000000   6674.500000   7018.000000   2011.000000
75%         654.000000   7271.000000   7019.000000   2017.000000
max         894.000000   7271.000000   7020.000000   2022.000000

               Year         Value
count  54810.000000  48255.000000
mean    2011.021346      0.802197
std        6.629795      0.669648
min     2000.000000     -4.176000
25%     2005.000000      0.364000
50%     2011.000000      0.643000
75%     2017.000000      1.084000
max     2022.000000      8.200000
```

[432]:
```
# Print information about the Land Temp Change data
land_temp_change_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54810 entries, 0 to 54809
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Domain Code       54810 non-null  object
 1   Domain            54810 non-null  object
 2   Area Code (M49)   54810 non-null  int64
 3   Area              54810 non-null  object
 4   Element Code      54810 non-null  int64
 5   Element           54810 non-null  object
 6   Months Code       54810 non-null  int64
 7   Months            54810 non-null  object
 8   Year Code         54810 non-null  int64
 9   Year              54810 non-null  int64
 10  Unit              54810 non-null  object
 11  Value             48255 non-null  float64
 12  Flag              54810 non-null  object
```

```
 13  Flag Description  54810 non-null  object
dtypes: float64(1), int64(5), object(8)
memory usage: 5.9+ MB
```

[433]:
```python
land_temp_change_df.shape
```

[433]:
```
(54810, 14)
```

[434]:
```python
land_temp_change_df = land_temp_change_df[(land_temp_change_df['Year'] >= 2000) & (land_temp_change_df['Months'].str.contains('Meteorological year')) & (land_temp_change_df['Element Code'] == 7271)]

land_temp_change_df = land_temp_change_df.drop(columns=['Domain Code', 'Domain', 'Year Code', 'Months', 'Element Code', 'Element', 'Unit', 'Flag', 'Flag Description', 'Months Code', 'Area Code (M49)'])

land_temp_change_df = land_temp_change_df.rename(columns={'Value': 'Total Land Temperature Change in °C'})
```

[435]:
```python
# Check for missing values
land_temp_change_df.isnull().sum()
```

[435]:
```
Area                                   0
Year                                   0
Total Land Temperature Change in °C    262
dtype: int64
```

[436]:
```python
# Check for duplicates
duplicates_found = land_temp_change_df.duplicated().sum()

# If duplicates are found, remove them
if duplicates_found > 0:
    land_temp_change_df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
else:
    print("No duplicates found.")
```

```
No duplicates found.
```

[437]:
```python
land_temp_change_df.shape
```

[437]:
```
(5481, 3)
```

[438]:
```python
# display final dataset
land_temp_change_df.head(100)
```

[438]:
```
            Area  Year  Total Land Temperature Change in °C
184  Afghanistan  2000                                0.993
```

```
185   Afghanistan  2001                                         1.311
186   Afghanistan  2002                                         1.365
187   Afghanistan  2003                                         0.587
188   Afghanistan  2004                                         1.373
…             …    …                                       …
1107      Andorra  2003                                         1.949
1108      Andorra  2004                                         0.936
1109      Andorra  2005                                         0.851
1110      Andorra  2006                                         1.485
1111      Andorra  2007                                         1.024

[100 rows x 3 columns]
```

3.12 2.12 Land Use Data

The following steps are performed - Display the data - Describe the data - Print Data Types - Checking for Null Values, Duplicates - Filtering Data by Dropping the columns from previous check and merging them based on Country (i.e., Area) and Year.

[439]:
```python
# Load the dataset
land_use_df = pd.read_csv("content/Land use - FAOSTAT_data_en_2-22-2024.csv",␣
↪low_memory=False)

# Display the first few rows of the Land Use data
land_use_df.head()
```

[439]:
```
  Domain Code    Domain  Area Code (M49)         Area  Element Code Element  \
0          RL  Land Use                4  Afghanistan          5110    Area
1          RL  Land Use                4  Afghanistan          5110    Area
2          RL  Land Use                4  Afghanistan          5110    Area
3          RL  Land Use                4  Afghanistan          5110    Area
4          RL  Land Use                4  Afghanistan          5110    Area

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

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

[440]:
```
# Describe the Land Use data
land_use_df.describe()
```

[440]:
```
       Area Code (M49)  Element Code     Item Code     Year Code  \
count     97995.000000       97995.0  97995.000000  97995.000000
mean        430.530884        5110.0   6627.879984   2002.966988
std         255.076689           0.0     26.601230     11.828224
min           4.000000        5110.0   6600.000000   1980.000000
25%         208.000000        5110.0   6602.000000   1993.000000
50%         426.000000        5110.0   6621.000000   2004.000000
75%         646.000000        5110.0   6650.000000   2013.000000
max         894.000000        5110.0   6695.000000   2021.000000

               Year         Value
count  97995.000000  9.799500e+04
mean    2002.966988  2.044488e+04
std       11.828224  9.502952e+04
min     1980.000000  0.000000e+00
25%     1993.000000  3.000000e+01
50%     2004.000000  7.037793e+02
75%     2013.000000  6.500000e+03
max     2021.000000  2.241237e+06
```

[441]:
```
# Print information about the Land Use data
land_use_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97995 entries, 0 to 97994
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Domain Code       97995 non-null  object
 1   Domain            97995 non-null  object
 2   Area Code (M49)   97995 non-null  int64
 3   Area              97995 non-null  object
 4   Element Code      97995 non-null  int64
 5   Element           97995 non-null  object
 6   Item Code         97995 non-null  int64
 7   Item              97995 non-null  object
 8   Year Code         97995 non-null  int64
 9   Year              97995 non-null  int64
 10  Unit              97995 non-null  object
 11  Value             97995 non-null  float64
 12  Flag              97995 non-null  object
 13  Flag Description  97995 non-null  object
 14  Note              1 non-null      object
dtypes: float64(1), int64(5), object(9)
memory usage: 11.2+ MB
```

[442]:
```
land_use_df.shape
```

[442]: (97995, 15)

[443]:
```
land_use_df = land_use_df[(land_use_df['Year'] >= 2000) & land_use_df['Item'].
 ↪str.contains('Cropland')]

land_use_df = land_use_df[~land_use_df['Item'].str.contains('Cropland area␣
 ↪actually irrigated', case=False, na=False)]

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

land_use_df = land_use_df.rename(columns={'Value': 'Total Land Use per 1000/
 ↪ha'})
```

[444]:
```
# Check for NaN values
land_use_df.isnull().sum()
```

[444]:
```
Area                           0
Year                           0
Total Land Use per 1000/ha     0
dtype: int64
```

[445]:
```
# Check for duplicates
duplicates_found = land_use_df.duplicated().sum()

# If duplicates are found, remove them
if duplicates_found > 0:
    land_use_df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
else:
    print("No duplicates found.")
```

```
No duplicates found.
```

[446]:
```
land_use_df.shape
```

[446]: (4921, 3)

[447]:
```
# display final dataset
land_use_df.head(100)
```

[447]:

	Area	Year	Total Land Use per 1000/ha
188	Afghanistan	2000	7794.00
189	Afghanistan	2001	7795.00
190	Afghanistan	2002	7790.00

```
191   Afghanistan  2003                          7884.00
192   Afghanistan  2004                          7928.00
…             …    …                        …
2067       Andorra  2007                             0.77
2068       Andorra  2008                             0.76
2069       Andorra  2009                             0.77
2070       Andorra  2010                             0.77
2071       Andorra  2011                             0.77

[100 rows x 3 columns]
```

3.13 2.13 Pesticides Use Data

The following steps are performed - Display the data - Describe the data - Print Data Types - Checking for Null Values, Duplicates - Filtering Data by Dropping the columns from previous check

[448]:
```
# Load the dataset
pesticides_use_df = pd.read_csv("content/Pesticides use -
↪FAOSTAT_data_en_2-27-2024.csv", low_memory=False)

# Display the first few rows of the pesticides use data
pesticides_use_df.head()
```

[448]:
```
  Domain Code          Domain  Area Code (M49)     Area  Element Code  \
0          RP  Pesticides Use                8  Albania          5157
1          RP  Pesticides Use                8  Albania          5159
2          RP  Pesticides Use                8  Albania          5173
3          RP  Pesticides Use                8  Albania          5157
4          RP  Pesticides Use                8  Albania          5159

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

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

[449]:
```
# Describing the pesticides use data
pesticides_use_df.describe()
```

[449]:
```
       Area Code (M49)  Element Code    Item Code    Year Code  \
count     35202.000000  35202.000000  35202.000000  35202.000000
mean        424.550423   5158.978694   1340.576445   2010.510852
std         248.441525      4.950079     17.776129      6.341302
min           8.000000   5157.000000   1309.000000   2000.000000
25%         208.000000   5157.000000   1320.000000   2005.000000
50%         418.000000   5157.000000   1345.000000   2011.000000
75%         626.000000   5157.000000   1357.000000   2016.000000
max         894.000000   5173.000000   1357.000000   2021.000000

               Year          Value
count  35202.000000   35202.000000
mean    2010.510852    3855.186176
std        6.341302   24198.051890
min     2000.000000       0.000000
25%     2005.000000       0.430000
50%     2011.000000       8.075000
75%     2016.000000     366.012500
max     2021.000000  719507.440000
```

[450]:
```python
# Printing information about the pesticides use data
pesticides_use_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35202 entries, 0 to 35201
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Domain Code        35202 non-null  object
 1   Domain             35202 non-null  object
 2   Area Code (M49)    35202 non-null  int64
 3   Area               35202 non-null  object
 4   Element Code       35202 non-null  int64
 5   Element            35202 non-null  object
 6   Item Code          35202 non-null  int64
 7   Item               35202 non-null  object
 8   Year Code          35202 non-null  int64
 9   Year               35202 non-null  int64
 10  Unit               35202 non-null  object
 11  Value              35202 non-null  float64
 12  Flag               35202 non-null  object
 13  Flag Description   35202 non-null  object
 14  Note               198 non-null    object
dtypes: float64(1), int64(5), object(9)
memory usage: 4.0+ MB
```

[451]:
```python
pesticides_use_df.shape
```

[451]: (35202, 15)

[452]:
```
# Filtering data to only take  and from after the year 2000
pesticides_use_df = pesticides_use_df[(pesticides_use_df['Year'] >= 2000) &␣
↪(pesticides_use_df['Element'].str.contains('Agricultural Use')) &␣
↪(pesticides_use_df['Item'].str.contains('Pesticides (total)', regex=False))]

# Dropping not required columns
pesticides_use_df = pesticides_use_df.drop(columns=['Domain Code', 'Domain',␣
↪'Year Code', 'Item', 'Item Code', 'Element Code', 'Element', 'Unit', 'Flag',␣
↪'Flag Description', 'Area Code (M49)', 'Note'])

# Renaming the column to avoid confusion of value tables after merging.
pesticides_use_df = pesticides_use_df.rename(columns={'Value': 'Total␣
↪Pesticides Used in Tonnes'})
```

[453]:
```
# Checking for NaN values
pesticides_use_df.isnull().sum()
```

[453]:
```
Area                            0
Year                            0
Total Pesticides Used in Tonnes 0
dtype: int64
```

[454]:
```
# Check for duplicates
duplicates_found = pesticides_use_df.duplicated().sum()

# If duplicates are found, remove them
if duplicates_found > 0:
    pesticides_use_df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
else:
    print("No duplicates found.")
```

```
No duplicates found.
```

[455]:
```
pesticides_use_df.shape
```

[455]: (4636, 3)

[456]:
```
# display final dataset
pesticides_use_df.head(100)
```

[456]:

	Area	Year	Total Pesticides Used in Tonnes
0	Albania	2000	307.98
3	Albania	2001	319.38
6	Albania	2002	330.78
9	Albania	2003	342.17

```
12     Albania  2004                              353.57
..          …    …                              …
667   Anguilla  2007                               48.62
668   Anguilla  2008                               58.04
669   Anguilla  2009                               65.91
670   Anguilla  2010                               65.91
671   Anguilla  2011                               65.91

[100 rows x 3 columns]
```

4 3. Merging Datasets

Based on the target variable **"food trade indicators"**, I am going to correlate which dataset has the highest matching rate with my target variable so that I can use that to predict the export value and get higher precision and accuracy. I will also be discarding datasets that have negative correlation or not near to my target variable (i.e., it's not that it is not useful, negative correlation can be taken as well but for this model not taking has its own reasons which is described as we go down further below.)

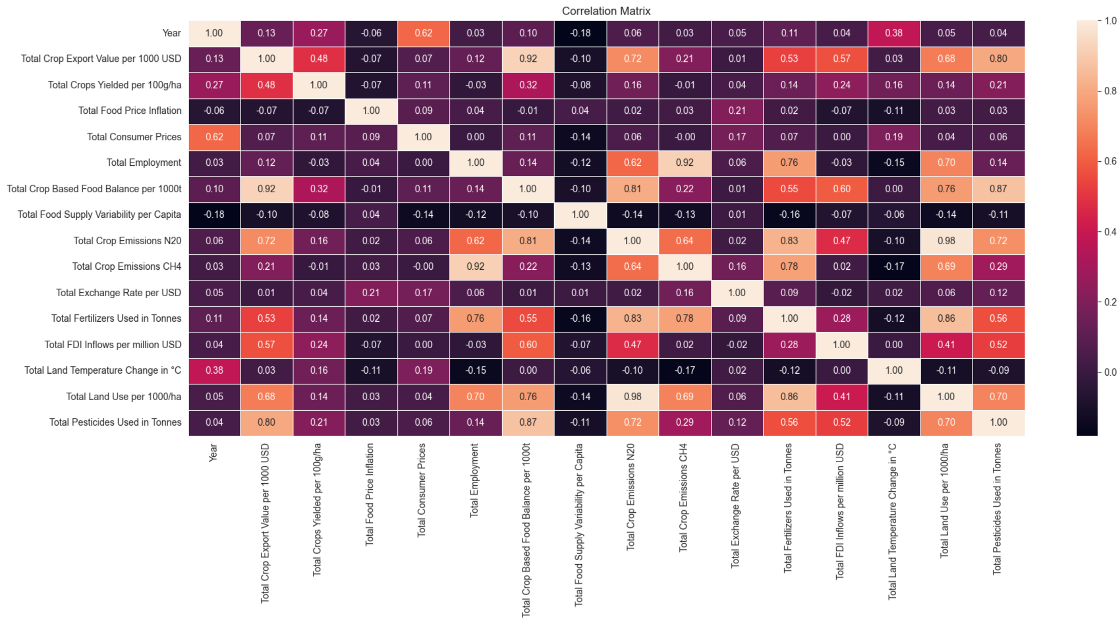
4.1 3.1 Visualising Features

4.1.1 3.1.1 Correlation Matrix

Here, I am taking target variable as food trade indicators against all the feature datasets and checking which dataframes have either strong or weak relations with respect to my target variable.

[457]:
```
food_security_df['Year'] = food_security_df['Year'].astype(int)
target = food_trade_df
required_dataframes = [crops_prod_df, fpi_df, cpi_df, employment_df,␣
 ↪food_balances_df, food_security_df, emissions_n20_df, emissions_ch4_df,␣
 ↪exchange_rates_df, fertilizers_use_df, fdi_df, land_temp_change_df,␣
 ↪land_use_df, pesticides_use_df]
for df in required_dataframes:
    target = pd.merge(target, df, left_on=['Year', 'Area'], right_on=['Year',␣
 ↪'Area'], how='inner')
target = target.groupby(["Area", "Year"]).mean().reset_index()
required_features = target.drop('Area', axis=1)
# Calculate correlation matrix
required_features_corr = required_features.corr()
```

[458]:
```
# Plot correlation matrix
plt.figure(figsize=(20, 8))
sns.heatmap(required_features_corr, annot=True, fmt=".2f", linewidths=0.5)
plt.title(f'Correlation Matrix')
plt.show()
```



The above correlation shows that all the other dataframes except for dataframes such as **(exchange_rate_df, land_temp_use_df, food_security_df, cpi_df, fpi_df, employment_df)**, shows high correlation with respect to total export value. This resulted in a lot of datapoints being discarded due to inner join properties. Hence, we can say that the labels extracted as features will also have a similar properties and some of these can even be correlated based on economic factors.

[459]:
```
target.shape
```

[459]:
```
(823, 17)
```

[460]:
```
# Print the correlation values with the target variable
required_features_corr['Total Crop Export Value per 1000 USD'].
    ↪sort_values(ascending=False)
```

[460]:
```
Total Crop Export Value per 1000 USD        1.000000
Total Crop Based Food Balance per 1000t     0.924971
Total Pesticides Used in Tonnes             0.798576
Total Crop Emissions N20                    0.716750
Total Land Use per 1000/ha                  0.677397
Total FDI Inflows per million USD           0.568236
Total Fertilizers Used in Tonnes            0.527127
Total Crops Yielded per 100g/ha             0.484189
Total Crop Emissions CH4                    0.207045
Year                                        0.132974
Total Employment                            0.123728
Total Consumer Prices                       0.070113
```

```
Total Land Temperature Change in °C         0.028800
Total Exchange Rate per USD                 0.011766
Total Food Price Inflation                 -0.067997
Total Food Supply Variability per Capita   -0.103731
Name: Total Crop Export Value per 1000 USD, dtype: float64
```

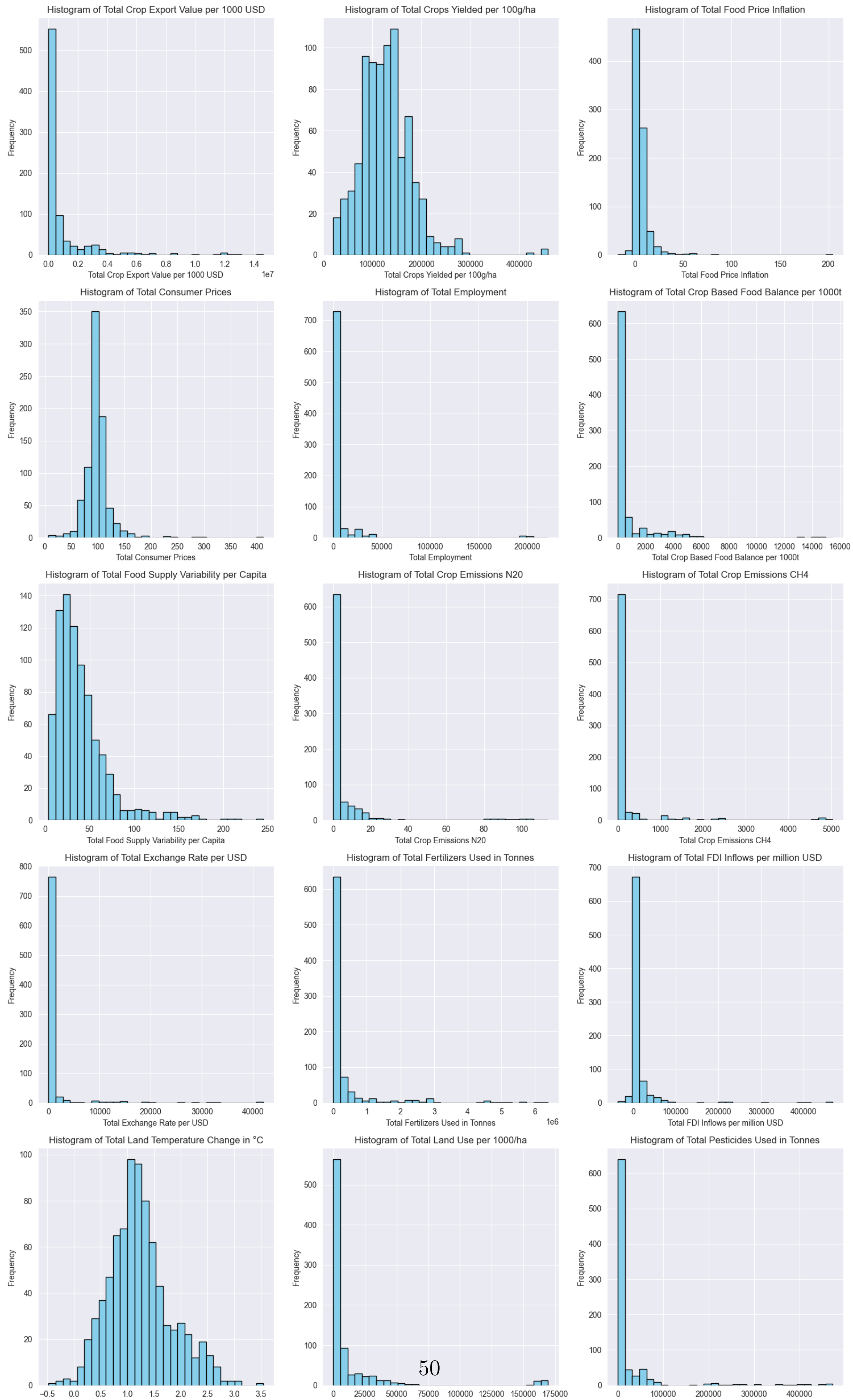
As you can see from the table above, taking into consideration any dataframe above year positively impacts the export value for crops in some form, higher correlation means it is likely to affect the export value more and is better for predicting than ones with lower values, there is also the fact that negative correlations do also contribute to the factor of export values but for this model we will not use any negative correlations or either ones that are less likely affecting the target variable. The reason for not selecting the negative correlations were it is not high enough to potentially affect the model in a drastical way as the maximum value is of around -0.1… so it is pretty negligible in that sense.

4.1.2 3.1.2 Histogram

The histogram plot shows the data distribution for each dataframe where some are evenly distributed whereas some have high skewness pointing towards high gap in the dataset. This indicates the datasets need to be transformed using log transformations or scaling to make it evenly distribute across the dataset.

[461]:

```
num_plots = len(target.columns)
num_cols = 3
num_rows = (num_plots // num_cols)
start_index = 2
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows))

for i, column in enumerate(target.columns[start_index:], start=start_index):
    if target[column].dtype in ['int64', 'float64']:
        row = (i - start_index) // num_cols
        col = (i - start_index) % num_cols
        axes[row, col].hist(target[column], bins=30, color='skyblue',␣
 ↪edgecolor='black')
        axes[row, col].set_title(f'Histogram of {column}')
        axes[row, col].set_xlabel(column)
        axes[row, col].set_ylabel('Frequency')
        axes[row, col].grid(True)

plt.tight_layout()
plt.show()
```

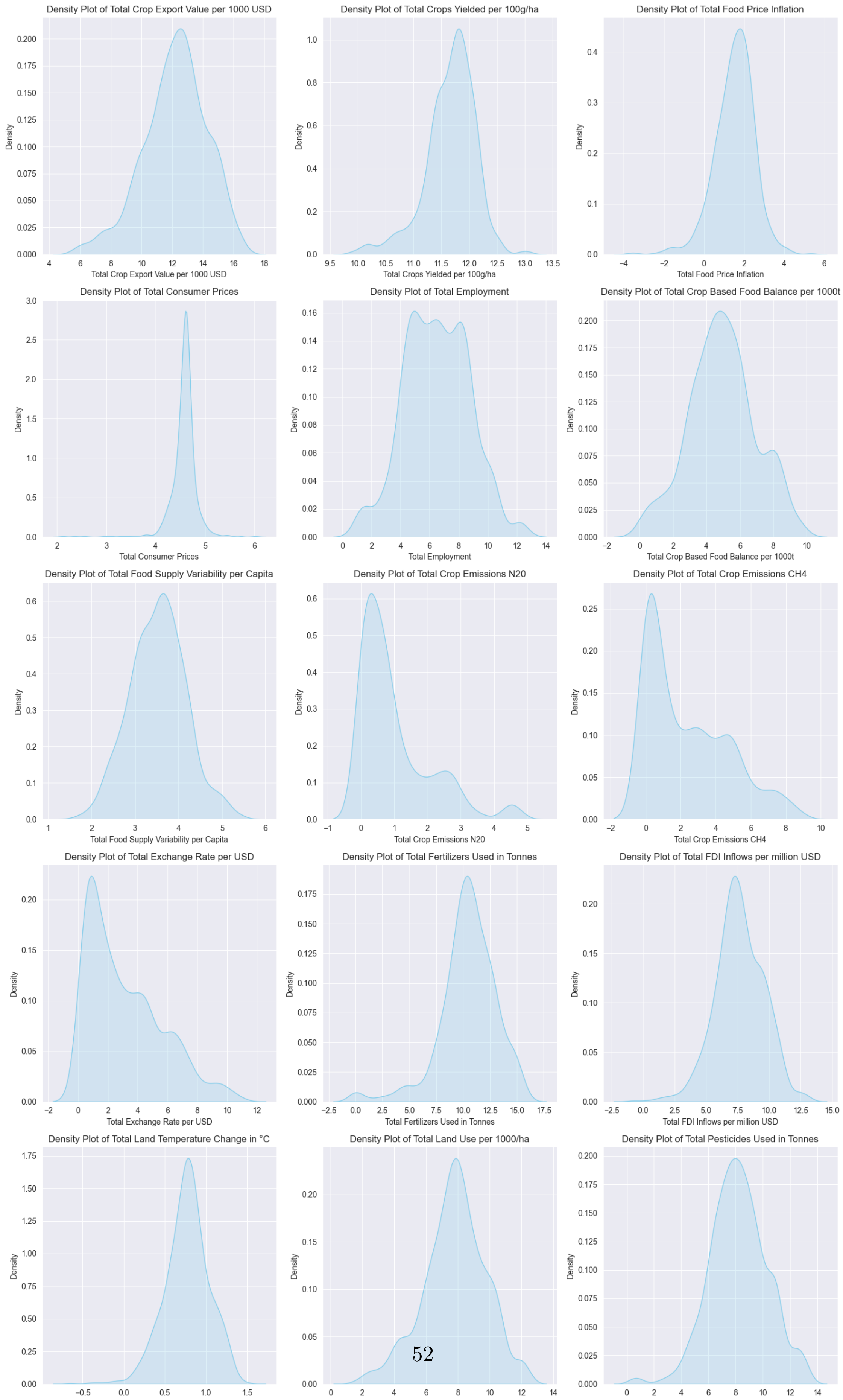

Histogram of Total Crop Export Value per 1000 USD
Frequency
Total Crop Export Value per 1000 USD
1e7
Histogram of Total Crops Yielded per 100g/ha
Frequency
Total Crops Yielded per 100g/ha
Histogram of Total Food Price Inflation
Frequency
Total Food Price Inflation
Histogram of Total Consumer Prices
Frequency
Total Consumer Prices
Histogram of Total Employment
Frequency
Total Employment
Histogram of Total Crop Based Food Balance per 1000t
Frequency
Total Crop Based Food Balance per 1000t
Histogram of Total Food Supply Variability per Capita
Frequency
Total Food Supply Variability per Capita
Histogram of Total Crop Emissions N20
Frequency
Total Crop Emissions N20
Histogram of Total Crop Emissions CH4
Frequency
Total Crop Emissions CH4
Histogram of Total Exchange Rate per USD
Frequency
Total Exchange Rate per USD
Histogram of Total Fertilizers Used in Tonnes
Frequency
Total Fertilizers Used in Tonnes
1e6
Histogram of Total FDI Inflows per million USD
Frequency
Total FDI Inflows per million USD
Histogram of Total Land Temperature Change in °C
Frequency
Total Land Temperature Change in °C
Histogram of Total Land Use per 1000/ha
Frequency
Total Land Use per 1000/ha
Histogram of Total Pesticides Used in Tonnes
Frequency
Total Pesticides Used in Tonnes

4.1.3 3.1.3 Density Plot

After applying log transformation, checking out how the data distribution across all the dataframes looks like using density plot. If the bell curves are observed, it signifies the values have been evenly spaced out to some extent giving you some freedom when building model to avoid sensitivity.

[269]:
```
for column in target.select_dtypes(include=['int64', 'float64']).columns:
    target[column] = np.log1p(target[column])

num_plots = len(target.columns)
num_cols = 3
num_rows = (num_plots // num_cols)
start_index = 2
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows))

for i, column in enumerate(target.columns[start_index:], start=start_index):
    if target[column].dtype in ['int64', 'float64']:
        row = (i - start_index) // num_cols
        col = (i - start_index) % num_cols
        sns.kdeplot(data=target[column], ax=axes[row, col], color='skyblue',␣
 ↪fill=True)
        axes[row, col].set_title(f'Density Plot of {column}')
        axes[row, col].set_xlabel(column)
        axes[row, col].set_ylabel('Density')

plt.tight_layout()
plt.show()
```

```
C:\Users\amany\MLP\lib\site-packages\pandas\core\arraylike.py:399:
RuntimeWarning: invalid value encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
```


Density Plot of Total Crop Export Value per 1000 USD
Total Crop Export Value per 1000 USD
Density
Density Plot of Total Crops Yielded per 100g/ha
Total Crops Yielded per 100g/ha
Density
Density Plot of Total Food Price Inflation
Total Food Price Inflation
Density
Density Plot of Total Consumer Prices
Total Consumer Prices
Density
Density Plot of Total Employment
Total Employment
Density
Density Plot of Total Crop Based Food Balance per 1000t
Total Crop Based Food Balance per 1000t
Density
Density Plot of Total Food Supply Variability per Capita
Total Food Supply Variability per Capita
Density
Density Plot of Total Crop Emissions N20
Total Crop Emissions N20
Density
Density Plot of Total Crop Emissions CH4
Total Crop Emissions CH4
Density
Density Plot of Total Exchange Rate per USD
Total Exchange Rate per USD
Density
Density Plot of Total Fertilizers Used in Tonnes
Total Fertilizers Used in Tonnes
Density
Density Plot of Total FDI Inflows per million USD
Total FDI Inflows per million USD
Density
Density Plot of Total Land Temperature Change in °C
Total Land Temperature Change in °C
Density
Density Plot of Total Land Use per 1000/ha
Total Land Use per 1000/ha
Density
Density Plot of Total Pesticides Used in Tonnes
Total Pesticides Used in Tonnes
Density

4.2 3.2 Merging DataFrames with Strong Correlations

Merging dataframes with all my target variables again with ones that have strong correlations. I have discarded any dataframes whose correlations were below 0.4 as it will just create more uneven spaced data points reducing my final dataset.

[270]:
```
# list of target variables extracted that will be made as labels.
target_dataframes = [cereals_df, sugar_honey_df, fruits_vegetables_df,␣
↪tobacco_df, fats_oils_df]

# List of features
required_dataframes = [crops_prod_df, food_balances_df, fertilizers_use_df,␣
↪fdi_df, emissions_n20_df, land_use_df, pesticides_use_df]

# Initialising an empty list to store the merged dataframes
merged_dfs = []

# Merging all required dataframes first
temp_df = required_dataframes[0].copy()
for req_df in required_dataframes[1:]:
    temp_df = pd.merge(temp_df, req_df, left_on=['Year', 'Area'],␣
↪right_on=['Year', 'Area'], how='inner')

# Looping through each target dataframe and merging with the combined required␣
↪dataframes
for target_df in target_dataframes:
    merged_df = pd.merge(temp_df, target_df, left_on=['Year', 'Area'],␣
↪right_on=['Year', 'Area'], how='inner')
    merged_dfs.append(merged_df)

# Concatenate all merged dataframes into a single dataframe and take mean␣
↪average.
final_merged_df = pd.concat(merged_dfs)
final_merged_df = final_merged_df.groupby(["Area", "Year"]).mean().reset_index()
```

[271]:
```
final_merged_df
```

[271]:
```
          Area  Year  Total Crops Yielded per 100g/ha  \
0      Albania  2010                    102745.727273
1      Albania  2011                    108613.272727
2      Albania  2012                    112625.363636
3      Albania  2013                    107375.090909
4      Albania  2014                    113253.272727
..         …     …                              …
```

```
864    Zambia  2021                     148215.800000
865  Zimbabwe  2010                      85084.454545
866  Zimbabwe  2011                      89278.727273
867  Zimbabwe  2017                      96210.363636
868  Zimbabwe  2018                      99997.818182

     Total Crop Based Food Balance per 1000t  \
0                                   3.230769
1                                   4.307692
2                                   5.615385
3                                   6.846154
4                                  10.222222
..                                       …
864                                78.615385
865                                18.461538
866                                15.538462
867                                18.909091
868                                16.153846

     Total Fertilizers Used in Tonnes  Total FDI Inflows per million USD  \
0                        22947.400000                        1050.714858
1                        26066.800000                         876.271104
2                        25001.600000                         855.435093
3                        19481.713333                        1254.930606
4                        19481.666667                        1154.690314
..                                …                                  …
864                     194259.350000                        -122.000000
865                      11731.214286                         166.000000
866                       9578.428571                         387.000000
867                      10817.000000                         349.000000
868                      13372.690000                         745.007943

     Total Crop Emissions N20  Total Land Use per 1000/ha  \
0                      0.1551                       696.0
1                      0.1575                       696.0
2                      0.1567                       696.0
3                      0.1569                       696.3
4                      0.1562                       696.0
..                        …                          …
864                    0.9896                      3839.0
865                    0.5099                      4100.0
866                    0.4903                      4300.0
867                    0.4894                      4100.0
868                    0.4981                      4100.0

     Total Pesticides Used in Tonnes  Total Cereal Export Value per 1000 USD  \
0                             590.50                                  624.00
```

```
1                        582.68                                  2911.00
2                        361.62                                  4814.00
3                        450.60                                  6596.00
4                        457.47                                  1751.15
..                          …                                       …
864                     4196.64                                155725.58
865                     3305.17                                  2911.00
866                     3340.35                                  4588.00
867                     2185.07                                 13515.02
868                     2185.07                                  7562.86

     Total Sugar and Honey Export Value per 1000 USD  \
0                                               160.00
1                                               556.00
2                                               491.00
3                                               324.00
4                                                55.66
..                                                 …
864                                          126878.56
865                                           52535.00
866                                           53142.00
867                                           53981.45
868                                           44521.70

     Total Fruits and Vegetables Export Value per 1000 USD  \
0                                              11791.00
1                                              18571.00
2                                              20612.00
3                                              32438.00
4                                              37161.31
..                                                 …
864                                            45484.57
865                                             9131.00
866                                            12677.00
867                                            30800.25
868                                            45693.61

     Total Tobacco Export Value per 1000 USD  \
0                                    4235.00
1                                    4163.00
2                                    4661.00
3                                    6104.00
4                                        NaN
..                                      …
864                                129116.43
865                                478055.00
866                                718045.00
```

```
867                                  837638.52
868                                  893113.05

     Total Fats and Oils Export Value per 1000 USD
0                                          1005.00
1                                          2380.00
2                                          2723.00
3                                          2092.00
4                                              NaN
..                                             …
864                                       13831.14
865                                        2030.00
866                                        8761.00
867                                        2451.03
868                                        1415.42

[869 rows x 14 columns]
```

[272]:
```
final_merged_df.shape
```

[272]:
```
(869, 14)
```

[273]:
```
# Checking for NaN values after inner join
nan_counts = final_merged_df.isna().sum()

print(nan_counts)
```

```
Area                                                   0
Year                                                   0
Total Crops Yielded per 100g/ha                        0
Total Crop Based Food Balance per 1000t                0
Total Fertilizers Used in Tonnes                       0
Total FDI Inflows per million USD                      0
Total Crop Emissions N20                               0
Total Land Use per 1000/ha                             0
Total Pesticides Used in Tonnes                        0
Total Cereal Export Value per 1000 USD                 2
Total Sugar and Honey Export Value per 1000 USD        6
Total Fruits and Vegetables Export Value per 1000 USD  2
Total Tobacco Export Value per 1000 USD               19
Total Fats and Oils Export Value per 1000 USD         11
dtype: int64
```

[274]:
```
# Dropping any NaN value if exists and then checking again.
final_merged_df = final_merged_df.dropna()
print("NaN values dropped")
nan_counts = final_merged_df.isna().sum()
```

```
print(nan_counts)
```

```
NaN values dropped
Area                                                  0
Year                                                  0
Total Crops Yielded per 100g/ha                       0
Total Crop Based Food Balance per 1000t               0
Total Fertilizers Used in Tonnes                      0
Total FDI Inflows per million USD                     0
Total Crop Emissions N20                              0
Total Land Use per 1000/ha                            0
Total Pesticides Used in Tonnes                       0
Total Cereal Export Value per 1000 USD                0
Total Sugar and Honey Export Value per 1000 USD       0
Total Fruits and Vegetables Export Value per 1000 USD 0
Total Tobacco Export Value per 1000 USD               0
Total Fats and Oils Export Value per 1000 USD         0
dtype: int64
```

[275]:
```
# getting the shape of the dataset after NaN drops
final_merged_df.shape
```

[275]: (841, 14)

[276]:
```
# Saving the final preprocessed data into a csv file, making it easier to just␣
↪run the code from below this point.
final_merged_df.to_csv('curated_data/
↪export_value_crops_2010_2021_nonans_restructured.csv', index=False)
```

5 4. The MLP model

The MLP model will mostly comprise and draw inspirations from lab module 4 to predict the export value of crops using my preprocessed data.

5.1 4.1 Loading Curated Dataset

[277]:
```
# Loading the datatset
curated_data = pd.read_csv("curated_data/
↪export_value_crops_2010_2021_nonans_restructured.csv", low_memory=False)
```

[278]:
```
# checking the shape
curated_data.shape
```

[278]: (841, 14)

[279]:
```
# look inside the data
curated_data
```

[279]:
```
         Area  Year  Total Crops Yielded per 100g/ha  \
0     Albania  2010                    102745.727273
1     Albania  2011                    108613.272727
2     Albania  2012                    112625.363636
3     Albania  2013                    107375.090909
4     Albania  2016                    132529.727273
..        …     …                              …
836    Zambia  2021                    148215.800000
837  Zimbabwe  2010                     85084.454545
838  Zimbabwe  2011                     89278.727273
839  Zimbabwe  2017                     96210.363636
840  Zimbabwe  2018                     99997.818182

     Total Crop Based Food Balance per 1000t  \
0                                   3.230769
1                                   4.307692
2                                   5.615385
3                                   6.846154
4                                  13.666667
..                                       …
836                                78.615385
837                                18.461538
838                                15.538462
839                                18.909091
840                                16.153846

     Total Fertilizers Used in Tonnes  Total FDI Inflows per million USD  \
0                        22947.400000                        1050.714858
1                        26066.800000                         876.271104
2                        25001.600000                         855.435093
3                        19481.713333                        1254.930606
4                        28082.000000                        1043.180470
..                                …                                  …
836                     194259.350000                        -122.000000
837                      11731.214286                         166.000000
838                       9578.428571                         387.000000
839                      10817.000000                         349.000000
840                      13372.690000                         745.007943

     Total Crop Emissions N20  Total Land Use per 1000/ha  \
0                      0.1551                       696.0
1                      0.1575                       696.0
2                      0.1567                       696.0
3                      0.1569                       696.3
4                      0.1558                       703.5
..                        …                         …
836                    0.9896                      3839.0
```

```
837                  0.5099                      4100.0
838                  0.4903                      4300.0
839                  0.4894                      4100.0
840                  0.4981                      4100.0

     Total Pesticides Used in Tonnes  Total Cereal Export Value per 1000 USD  \
0                             590.50                                  624.00
1                             582.68                                 2911.00
2                             361.62                                 4814.00
3                             450.60                                 6596.00
4                             584.49                                15198.79
..                              …                                      …
836                          4196.64                               155725.58
837                          3305.17                                 2911.00
838                          3340.35                                 4588.00
839                          2185.07                                13515.02
840                          2185.07                                 7562.86

     Total Sugar and Honey Export Value per 1000 USD  \
0                                             160.00
1                                             556.00
2                                             491.00
3                                             324.00
4                                             258.09
..                                              …
836                                        126878.56
837                                         52535.00
838                                         53142.00
839                                         53981.45
840                                         44521.70

     Total Fruits and Vegetables Export Value per 1000 USD  \
0                                             11791.00
1                                             18571.00
2                                             20612.00
3                                             32438.00
4                                             69861.95
..                                                …
836                                           45484.57
837                                            9131.00
838                                           12677.00
839                                           30800.25
840                                           45693.61

     Total Tobacco Export Value per 1000 USD  \
0                                     4235.00
1                                     4163.00
```

```
2                                    4661.00
3                                    6104.00
4                                    4989.45
..                                       ...
836                                129116.43
837                                478055.00
838                                718045.00
839                                837638.52
840                                893113.05

     Total Fats and Oils Export Value per 1000 USD
0                                          1005.00
1                                          2380.00
2                                          2723.00
3                                          2092.00
4                                          1603.33
..                                             ...
836                                       13831.14
837                                        2030.00
838                                        8761.00
839                                        2451.03
840                                        1415.42

[841 rows x 14 columns]
```

[280]:

```python
# describing the data
curated_data.describe()
```

[280]:

```
              Year  Total Crops Yielded per 100g/ha  \
count   841.000000                       841.000000
mean   2014.565993                    130541.978338
std       3.595067                     65599.300800
min    2010.000000                     20197.777778
25%    2011.000000                     92303.909091
50%    2014.000000                    124475.428571
75%    2018.000000                    156664.454545
max    2021.000000                    718138.000000

       Total Crop Based Food Balance per 1000t  \
count                               841.000000
mean                                789.649793
std                                1953.249116
min                                   0.000000
25%                                  35.076923
50%                                 124.692308
75%                                 451.846154
max                               15482.076923
```

```
       Total Fertilizers Used in Tonnes  Total FDI Inflows per million USD  \
count                      8.410000e+02                         841.000000
mean                       3.076233e+05                       10377.379923
std                        8.064419e+05                       37772.415660
min                        0.000000e+00                      -35743.719060
25%                        9.814444e+03                         434.075669
50%                        3.838630e+04                        1675.084894
75%                        1.782221e+05                        7114.000000
max                        6.388340e+06                      467625.000000

       Total Crop Emissions N20  Total Land Use per 1000/ha  \
count                841.000000                  841.000000
mean                   5.514041                11573.477137
std                   15.973152                28507.002777
min                    0.000000                    4.100000
25%                    0.223200                  793.000000
50%                    0.800800                 2399.500000
75%                    2.825800                 8720.000000
max                  113.927600               169463.000000

       Total Pesticides Used in Tonnes  \
count                       841.000000
mean                      25547.934376
std                       70853.242208
min                           0.690000
25%                         917.040000
50%                        3461.100000
75%                       13697.000000
max                      472977.150000

       Total Cereal Export Value per 1000 USD  \
count                            8.410000e+02
mean                             1.639247e+06
std                              4.025218e+06
min                              0.000000e+00
25%                              2.239700e+04
50%                              1.771898e+05
75%                              1.025758e+06
max                              3.526993e+07

       Total Sugar and Honey Export Value per 1000 USD  \
count                                     8.410000e+02
mean                                      3.433331e+05
std                                       1.036881e+06
min                                       0.000000e+00
25%                                       7.983760e+03
```

```
50%                                        8.606400e+04
75%                                        2.949167e+05
max                                        1.522496e+07

       Total Fruits and Vegetables Export Value per 1000 USD  \
count                                          8.410000e+02
mean                                           1.735224e+06
std                                            3.969750e+06
min                                            0.000000e+00
25%                                            7.621100e+04
50%                                            3.032200e+05
75%                                            1.369088e+06
max                                            2.557505e+07

       Total Tobacco Export Value per 1000 USD  \
count                                8.410000e+02
mean                                 2.323533e+05
std                                  4.565925e+05
min                                  0.000000e+00
25%                                  2.821810e+03
50%                                  5.091395e+04
75%                                  2.195717e+05
max                                  3.272139e+06

       Total Fats and Oils Export Value per 1000 USD
count                                      8.410000e+02
mean                                       7.357524e+05
std                                        2.699210e+06
min                                        0.000000e+00
25%                                        6.893000e+03
50%                                        6.762900e+04
75%                                        2.961504e+05
max                                        3.203634e+07
```

5.2 4.1 Generalising Labels

Generalising columns Area and Year to numbered indexes. This will result in the area and year getting indexed as 1,2,3,4……and so on which will better help on classification later on the code.

[281]:
```
offset = 1
curated_data['Area'] = pd.factorize(curated_data['Area'])[0] + offset

offset = 1
curated_data['Year'] = pd.factorize(curated_data['Year'])[0] + offset

curated_data
```

[281]:

```
     Area  Year  Total Crops Yielded per 100g/ha  \
0       1     1                    102745.727273
1       1     2                    108613.272727
2       1     3                    112625.363636
3       1     4                    107375.090909
4       1     5                    132529.727273
..    …   …                              …
836   142    12                    148215.800000
837   143     1                     85084.454545
838   143     2                     89278.727273
839   143     8                     96210.363636
840   143     9                     99997.818182

     Total Crop Based Food Balance per 1000t  \
0                                   3.230769
1                                   4.307692
2                                   5.615385
3                                   6.846154
4                                  13.666667
..                                     …
836                                78.615385
837                                18.461538
838                                15.538462
839                                18.909091
840                                16.153846

     Total Fertilizers Used in Tonnes  Total FDI Inflows per million USD  \
0                        22947.400000                        1050.714858
1                        26066.800000                         876.271104
2                        25001.600000                         855.435093
3                        19481.713333                        1254.930606
4                        28082.000000                        1043.180470
..                                …                                  …
836                     194259.350000                        -122.000000
837                      11731.214286                         166.000000
838                       9578.428571                         387.000000
839                      10817.000000                         349.000000
840                      13372.690000                         745.007943

     Total Crop Emissions N20  Total Land Use per 1000/ha  \
0                      0.1551                       696.0
1                      0.1575                       696.0
2                      0.1567                       696.0
3                      0.1569                       696.3
4                      0.1558                       703.5
..                       …                          …
836                    0.9896                      3839.0
```

```
837                   0.5099                     4100.0
838                   0.4903                     4300.0
839                   0.4894                     4100.0
840                   0.4981                     4100.0

     Total Pesticides Used in Tonnes  Total Cereal Export Value per 1000 USD  \
0                             590.50                                  624.00
1                             582.68                                 2911.00
2                             361.62                                 4814.00
3                             450.60                                 6596.00
4                             584.49                                15198.79
..                               …                                    …
836                          4196.64                               155725.58
837                          3305.17                                 2911.00
838                          3340.35                                 4588.00
839                          2185.07                                13515.02
840                          2185.07                                 7562.86

     Total Sugar and Honey Export Value per 1000 USD  \
0                                             160.00
1                                             556.00
2                                             491.00
3                                             324.00
4                                             258.09
..                                              …
836                                        126878.56
837                                         52535.00
838                                         53142.00
839                                         53981.45
840                                         44521.70

     Total Fruits and Vegetables Export Value per 1000 USD  \
0                                             11791.00
1                                             18571.00
2                                             20612.00
3                                             32438.00
4                                             69861.95
..                                                 …
836                                           45484.57
837                                            9131.00
838                                           12677.00
839                                           30800.25
840                                           45693.61

     Total Tobacco Export Value per 1000 USD  \
0                                     4235.00
1                                     4163.00
```

```
2                                        4661.00
3                                        6104.00
4                                        4989.45
..                                           ...
836                                    129116.43
837                                    478055.00
838                                    718045.00
839                                    837638.52
840                                    893113.05

     Total Fats and Oils Export Value per 1000 USD
0                                              1005.00
1                                              2380.00
2                                              2723.00
3                                              2092.00
4                                              1603.33
..                                                 ...
836                                           13831.14
837                                            2030.00
838                                            8761.00
839                                            2451.03
840                                            1415.42

[841 rows x 14 columns]
```

[282]:
```
# converting the curated data to numpy array
data = curated_data.to_numpy()
data.shape
```

[282]:
```
(841, 14)
```

5.3 4.2 Getting Features, Labels and Recoding to Classes

Here, I am going to take 5 features: **cereal, sugar and honey, fruits and vegetables, tobacco, fats and oils** and then I will be creating seperate arrays of equal length of curated_data to then populate with my classification values. The classification for the labeling will go as follows: - 0: Low Export Value - 1: High Export Value

[283]:
```
# getting features and labels from the data
label_col_cereal_export_value = 9
label_col_sugar_honey_export_value = 10
label_col_fruits_vegetables_export_value = 11
label_col_tobacco_export_value = 12
label_col_fats_oils_export_value = 13

# recoding labels into classes, we will classify them to either low or high␣
↪export value.
```

```
# cereal_export - 2 classes
# sugar_honey_export - 2 classes
# fruits_vegetables_export - 2 classes
# tobacco_export - 2 classes
# fats_oils_export - 2 classes
cereal_export_value = np.zeros(data.shape[0])
sugar_honey_value = np.zeros(data.shape[0])
fruits_vegetables_value = np.zeros(data.shape[0])
tobacco_value = np.zeros(data.shape[0])
fats_oils_value = np.zeros(data.shape[0])
```

In the below code, I am just using IntelliJ inbuilt feature that shows my min, 25th percentile, 50th percentile, 75th percentile and max values for a column to determine how to divide my columns into 2 classes. Also, will be rounding up to whole numbers for simplicity. Using those features I am classifying my data in below fashion.

[284]:
```
# using Intellij inbuilt feature to determine best division of my dataset into␣
↪2 parts for grouping them.

# for cereal exports
for i in np.arange(data.shape[0]):

    if data[i, label_col_cereal_export_value] < 550000:

        cereal_export_value[i] = 0

    else:

        cereal_export_value[i] = 1

# for sugar and honey exports
for i in np.arange(data.shape[0]):

    if data[i, label_col_sugar_honey_export_value] < 180000:

        sugar_honey_value[i] = 0

    else:

        sugar_honey_value[i] = 1

# for fruits and vegetable exports
for i in np.arange(data.shape[0]):

    if data[i, label_col_fruits_vegetables_export_value] < 1070000:

        fruits_vegetables_value[i] = 0
```

```
    else:

        fruits_vegetables_value[i] = 1

# for tobacco exports
for i in np.arange(data.shape[0]):

    if data[i, label_col_tobacco_export_value] < 150000:

        tobacco_value[i] = 0

    else:

        tobacco_value[i] = 1

# for oil and fats exports
for i in np.arange(data.shape[0]):

    if data[i, label_col_fats_oils_export_value] < 250000:

        fats_oils_value[i] = 0


    else:

        fats_oils_value[i] = 1


# Looking at the lables after succesful classification.
print ("\n Cereal Export Labels: \n"+str(cereal_export_value))
print ("\n Sugar and Honey Export Labels: \n"+str(sugar_honey_value))
print ("\n Fruits and Vegetables Export Labels:␣
↪\n"+str(fruits_vegetables_value))
print ("\n Tobacco Export Labels: \n"+str(tobacco_value))
print ("\n Fats and Oils Export Labels: \n"+str(fats_oils_value))
```

```
 Cereal Export Labels:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 1.
 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1.
 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0.]

 Sugar and Honey Export Labels:
[0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 0. 0.
 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.
 1. 1. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 0. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0. 0. 1. 0. 1. 0. 0.
 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 1. 1. 1.
 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0.]

 Fruits and Vegetables Export Labels:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0.
 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1.
 1. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0.]

 Tobacco Export Labels:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1.
 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1.
 1.]

 Fats and Oils Export Labels:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
```

```
 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 1. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1.
 1. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0.]
```

5.4 4.3 Training, Validation and Testing Sets

Splitting data into training, testing and validation sets for the MLP model. Here I am going to be performing the following: - randomly splitting with 75:25 ratio to obtain test set with remaining 80% being remnant from test split. - splitting remaining 80% set again to 75:25 ratio to obtain training dataset and validation sets

The reason to do 75:25 split was there was not lot of data after preprocessing so needed more on the testing side to obtain better model accuracy.

[285]:
```python
all_dfs = np.arange(0, data.shape[0])
```

```
random_seed = 23

# first 80/20 split to the all_dfs to obtain test set with remaining being a␣
↪remnant.
rem_set, test_set = train_test_split(all_dfs, test_size=0.25, train_size=0.75,
                                             random_state=random_seed,␣
↪shuffle=True)


# splitting the remaining 80% again to 80/20 to obtain training and validation␣
↪sets
train_set, val_set = train_test_split(rem_set, test_size=0.25, train_size=0.75,
                                             random_state=random_seed,␣
↪shuffle=True)

# storing data into their respective sets.
training_data = data[train_set, :]
train_label_col_cereal_export_value = cereal_export_value[train_set]
train_label_col_sugar_honey_export_value = sugar_honey_value[train_set]
train_label_col_fruits_vegetables_export_value =␣
↪fruits_vegetables_value[train_set]
train_label_col_tobacco_export_value = tobacco_value[train_set]
train_label_col_fats_oils_export_value = fats_oils_value[train_set]


validation_data = data[val_set, :]
val_label_col_cereal_export_value = cereal_export_value[val_set]
val_label_col_sugar_honey_export_value = sugar_honey_value[val_set]
val_label_col_fruits_vegetables_export_value = fruits_vegetables_value[val_set]
val_label_col_tobacco_export_value = tobacco_value[val_set]
val_label_col_fats_oils_export_value = fats_oils_value[val_set]


test_data = data[test_set, :]
test_label_col_cereal_export_value = cereal_export_value[test_set]
test_label_col_sugar_honey_export_value = sugar_honey_value[test_set]
test_label_col_fruits_vegetables_export_value =␣
↪fruits_vegetables_value[test_set]
test_label_col_tobacco_export_value = tobacco_value[test_set]
test_label_col_fats_oils_export_value = fats_oils_value[test_set]
```

[462]:
```
training_data.shape
```

[462]: (472, 14)

[463]:
```
validation_data.shape
```

[463]: (158, 14)

[464]:
```
test_data.shape
```

[464]: (211, 14)

5.5 4.4 Scaling Training, Validation and Test Datasets

Now we scale the training, validation and test datasets to make it more evenly spaced inside the dataframe. This step will also ensure the fair treatment of all my features and get a optimal performance of my MLP model.

[286]:
```
# Using StandardScaler from Scikit Learn Library.
scaler = StandardScaler()
scaler.fit(training_data)
scaled_training_data = scaler.transform(training_data)
scaled_validation_data = scaler.transform(validation_data)
scaled_test_data = scaler.transform(test_data)

scaled_training_data
```

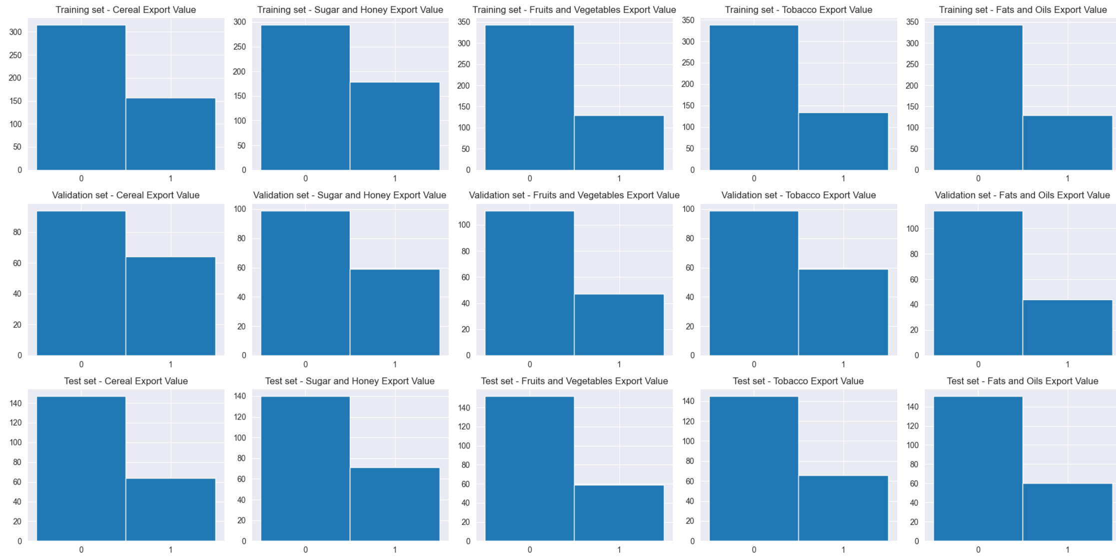
[286]:
```
array([[ 1.65257158,  1.81388141,  0.27464447, …, -0.41395004,
        -0.21470884, -0.26012188],
       [-1.44892344, -0.9505746 , -0.96625102, …, -0.38763001,
        -0.49101838, -0.21129544],
       [-1.61987198,  1.26099021,  1.80993973, …, -0.10084609,
        -0.48969271, -0.16966467],
       …,
       [-1.400081  ,  0.15520781, -1.18072448, …, -0.40684659,
        -0.46812431, -0.24466737],
       [-0.03249264,  1.53743581, -0.17254649, …, -0.35895766,
        -0.34247057, -0.25749784],
       [-1.20471123, -0.674129  ,  0.36518941, …,  0.84333075,
        -0.06943138,  1.02095968]])
```

5.6 4.5 Visualising the Labels

Here, we visualise the labels using histogram graphs based on class frequencies and how evenly distributed it is.

[287]:
```
def plot_label_distr(ax, labels, plot_title):
    the_bin_centres = np.unique(labels)
    ax.hist(labels, bins=the_bin_centres.shape[0], range=(the_bin_centres[0]-0.
↪5, the_bin_centres[-1]+0.5))
    ax.set_xticks(the_bin_centres)
    ax.set_title(plot_title)

# list of training labels
```

```python
train_labels = [
    train_label_col_cereal_export_value,
    train_label_col_sugar_honey_export_value,
    train_label_col_fruits_vegetables_export_value,
    train_label_col_tobacco_export_value,
    train_label_col_fats_oils_export_value
]

# list of validation labels
val_labels = [
    val_label_col_cereal_export_value,
    val_label_col_sugar_honey_export_value,
    val_label_col_fruits_vegetables_export_value,
    val_label_col_tobacco_export_value,
    val_label_col_fats_oils_export_value
]

# list of testing labels
test_labels = [
    test_label_col_cereal_export_value,
    test_label_col_sugar_honey_export_value,
    test_label_col_fruits_vegetables_export_value,
    test_label_col_tobacco_export_value,
    test_label_col_fats_oils_export_value
]

# Titles for the histogram plot.
titles = [
    'Cereal Export Value',
    'Sugar and Honey Export Value',
    'Fruits and Vegetables Export Value',
    'Tobacco Export Value',
    'Fats and Oils Export Value'
]

fig, axes = plt.subplots(3, len(train_labels), figsize=(20, 10))

# Plotting for each set of labels
for i in range(len(train_labels)):
    plot_label_distr(axes[0, i], train_labels[i], f'Training set - {titles[i]}')
    plot_label_distr(axes[1, i], val_labels[i], f'Validation set - {titles[i]}')
    plot_label_distr(axes[2, i], test_labels[i], f'Test set - {titles[i]}')

plt.tight_layout()
plt.show()
```



5.7 4.6 Building MLP Model and Evaluation

Here based on the train/test/val split data we will now commence to build our Multi Layer Perceptron Model to predict export values of crops and evaluate the model using techniques such as f1_scores, confusion matrix etc. I am going to build 2 Layer MLP model as my datasets are not that huge and making more layered MLP model will just increase the complexity and increase the risk of overfitting. So, I will start with 2 Layer and then if I require more than will add accordingly.

[316]:
```
# To ensure reproducibility of the same results, setting up the random seed for
↪the state.
random.seed(random_seed)

# for PyTorch operations that use random numbers internally
torch.manual_seed(random_seed)

# Creating the 2-Layer MLP network structure
class two_layer_MLP(nn.Module):
    def __init__(self,
                 input_size,
                 hidden_layer_sizes,
                 output_size):
        super().__init__()
        self.hidden_l1 = nn.Linear(input_size, hidden_layer_sizes[0])
        self.output_l2 = nn.Linear(hidden_layer_sizes[0], output_size)


    def forward(self, inputs):
        out = self.hidden_l1(inputs)
        out = self.output_l2(out)
```

```
        out = torch.softmax(out, 1)
        return out


# Creating method for computing stats such as labels, prediction and confusion␣
 ↪matrix.
def my_stats(labels, predictions, show_confusion_matrix=False):

    predictions_numpy = predictions.detach().numpy()
    predicted_classes = np.argmax(predictions_numpy, axis=1)

    f1_scores = f1_score(labels, predicted_classes, average=None)
    acc = accuracy_score(labels, predicted_classes)

    if show_confusion_matrix:
        import os
        # Plotting the confusion matrix graph if the requirements are set to␣
 ↪true.
        print("\n Confusion matrix:")
        confuse_mat = confusion_matrix(labels, predicted_classes)
        disp = ConfusionMatrixDisplay(confuse_mat)
        disp.plot(cmap='Blues')
        disp.ax_.set_facecolor('blue')
        plt.show()

        # Adding states to the output file of the prediction to make it easier␣
 ↪to understand the confusion matrix count for each states.
        states = []
        for true_label, predicted_label in zip(labels.numpy(),␣
 ↪predicted_classes):
            if true_label == 0 and predicted_label == 0:
                states.append('True Positive')
            elif true_label == 0 and predicted_label == 1:
                states.append('False Negative')
            elif true_label == 1 and predicted_label == 0:
                states.append('False Positive')
            elif true_label == 1 and predicted_label == 1:
                states.append('True Negative')

        # Here, adding counter to make it unique in a sense that if the model␣
 ↪is ran multiple times it will bear different outcomes and is easier to read␣
 ↪in the prediction output file.
        columns = ['Data Instance ID', 'True Label', 'Predicted Label',␣
 ↪'Result']
        data = {
            'Data Instance ID': unique_id,
```

```
            'True Label': labels.numpy(),
            'Predicted Label': predicted_classes,
            'Result': states
        }
        predictions_df = pd.DataFrame(data, columns=columns)
        predictions_df = predictions_df.groupby(['Data Instance ID', 'True␣
↪Label', 'Predicted Label', 'Result']).size().reset_index(name='Count')

        # saving the output prediction file to better understand how the model␣
↪is peforming in each instance.

        file_path = f'predictions.csv'
        if os.path.isfile(file_path):
            predictions_df.to_csv(file_path, mode='a', header=False,␣
↪index=False)
        else:
            predictions_df.to_csv(file_path, index=False)

        print(f"Prediction outputs appended to 'predictions.csv'")

    return f1_scores, acc


# Creating a class to manage the dataset for training the model
class ExportDataset(Dataset):
    def __init__(self, feats, labels):
        # Converting features and labels from numpy arrays to PyTorch tensors
        self.feats = torch.tensor(feats, dtype=torch.float32)
        self.labels = torch.tensor(labels, dtype=torch.long)

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):

        return self.feats[idx, :], self.labels[idx]
```

5.8 4.7 Using Random Search for Best Hyperparameters

Here, we will use random search cross validation function to determine the best hyperparameter to tune my MLP model. The resulted output will be then used in my MLP model to predict export values of crops. For now, I am going to be using it for two feature labels: - Cereal Export Label - Tobacco Export Label

5.8.1 4.7.1 For Cereal

[ ]:

[417]:
```
from sklearn.exceptions import ConvergenceWarning
import warnings
warnings.filterwarnings("ignore", category=ConvergenceWarning)

param_dist = {
    'learning_rate_init': uniform(0.001, 0.1),
    'batch_size': randint(32, 256),
    'max_iter': randint(10, 100)
}

# Loop over each export value label
random_search = RandomizedSearchCV(
    estimator=MLPClassifier(hidden_layer_sizes=(30,)),
    param_distributions=param_dist,
    n_iter=10,
    cv=5,
    scoring='accuracy',
    random_state=42
)

# Performing random search for the current label
random_search.fit(scaled_training_data, train_label_col_cereal_export_value)

# Get the best hyperparameters and score
best_params = random_search.best_params_
best_score = random_search.best_score_

# Printing the results
print("Best Hyperparameters for Cereal Export Value is :", best_params)
print("Best Accuracy:", best_score)
```

```
Best Hyperparameters for Cereal Export Value is : {'batch_size': 119,
'learning_rate_init': 0.08424426408004218, 'max_iter': 47}
Best Accuracy: 0.9830907054871221
```

5.8.2 4.7.2 For Tobacco

[418]:
```
import warnings
warnings.filterwarnings("ignore", category=ConvergenceWarning)

param_dist = {
    'learning_rate_init': uniform(0.001, 0.1),
    'batch_size': randint(32, 256),
    'max_iter': randint(10, 50)
```

```
}

# Loop over each export value label
random_search = RandomizedSearchCV(
    estimator=MLPClassifier(hidden_layer_sizes=(30,)),
    param_distributions=param_dist,
    n_iter=10,
    cv=5,
    scoring='accuracy',
    random_state=23
)

# Performing random search for the current label
random_search.fit(scaled_training_data, train_label_col_tobacco_export_value)

# Get the best hyperparameters and score
best_params = random_search.best_params_
best_score = random_search.best_score_

# Printing the results
print("Best Hyperparameters for Tobacco Export Value is :", best_params)
print("Best Accuracy:", best_score)
```

```
Best Hyperparameters for Tobacco Export Value is : {'batch_size': 222,
'learning_rate_init': 0.06915574306629138, 'max_iter': 43}
Best Accuracy: 0.9957670772676372
```

5.9 4.8 Running Experiments on MLP Model - Cereals

Now we will input the train data into the MLP model to predict and check the accuracy of the model to predict the export value of Cereals. I will be using the exact hyper parameter values as provided to me from the random search cross validation folds.

[318]:
```
# Creating an instance of the MLP network
feature_count = training_data.shape[1]
hidden_layer_sizes = [30]
class_count = np.unique(train_label_col_cereal_export_value).shape[0]
model = two_layer_MLP(feature_count, hidden_layer_sizes, class_count)


# Setting best hyperparameters based on my random search CV function
num_epochs = 47
learning_rate = 0.08424426408004218
batch_size = 119


# Setting up the data loading by batch with the test and validation sets having␣
↪only one batch
```

```
unique_id = "Cereal Export Value"

train_set = ExportDataset(scaled_training_data,␣
↪train_label_col_cereal_export_value)
train_dataloader = DataLoader(train_set, batch_size=batch_size)

val_set = ExportDataset(scaled_validation_data,␣
↪val_label_col_cereal_export_value)
val_dataloader = DataLoader(val_set, batch_size=len(val_set))

test_set = ExportDataset(scaled_test_data, test_label_col_cereal_export_value)
test_dataloader = DataLoader(test_set, batch_size=len(test_set))


# Setting up the SGD optimizer for updating the model weights
optimizer = optim.SGD(model.parameters(), lr=learning_rate)


# Computing cross entropy loss against the training labels
loss_function = nn.CrossEntropyLoss()

best_model_acc = 0
losses = []

# Iterating over the dataset at two different stages:
# - First, Iterating over the batches in the dataset
# - Second, Iterating over the specified number of epochs
for epoch in range(0, num_epochs):

    # Setting the model to training mode
    model.train()

    if epoch == 0:  best_model = deepcopy(model)

    for batch, (X_train, y_train) in enumerate(train_dataloader):

        optimizer.zero_grad()

        # Computing the forward pass and then the loss
        train_pred = model.forward(X_train)
        train_loss = loss_function(train_pred, y_train)
        train_avg_f1_score, train_acc = my_stats(y_train, train_pred)

        # Computing the model parameters' gradients and propagating the loss␣
↪backwards through the network.
        train_loss.backward()
```

```
        # Updating the model parameters using those gradients
        optimizer.step()

    # Evaluating on the validation set
    model.eval()
    for batch, (X_val, y_val) in enumerate(val_dataloader):
        val_pred = model.forward(X_val)
        val_loss = loss_function(val_pred, y_val)
        val_avg_f1_score, val_acc = my_stats(y_val, val_pred)

    if val_acc > best_model_acc:
        best_model_acc = val_acc
        best_model = deepcopy(model)
        print('Found Improvements, Saving the New Model.')

    # Printing how well does the network does on batches which results on how␣
↪well the mode is progressing.

    print("epoch: {} - train loss: {:.4f} train acc: {:.2f} val loss: {:.4f}␣
↪val acc: {:.2f}".format(
        epoch,
        train_loss.item(),
        train_acc,
        val_loss.item(),
        val_acc ))

    losses.append([train_loss.item(), val_loss.item()])

model = best_model

cereal_model = model
```

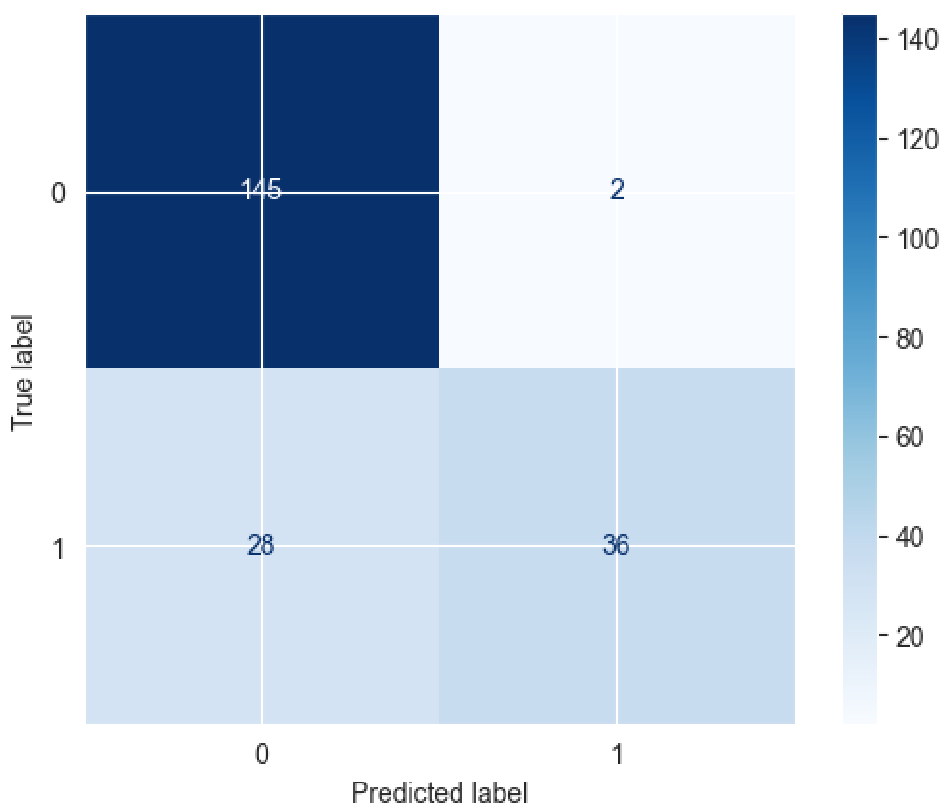
```
Found Improvements, Saving the New Model.
epoch: 0 - train loss: 0.6758 train acc: 0.61 val loss: 0.6596 val acc: 0.65
Found Improvements, Saving the New Model.
epoch: 1 - train loss: 0.6210 train acc: 0.74 val loss: 0.6183 val acc: 0.71
Found Improvements, Saving the New Model.
epoch: 2 - train loss: 0.5953 train acc: 0.77 val loss: 0.5993 val acc: 0.72
epoch: 3 - train loss: 0.5793 train acc: 0.77 val loss: 0.5871 val acc: 0.72
Found Improvements, Saving the New Model.
epoch: 4 - train loss: 0.5674 train acc: 0.77 val loss: 0.5783 val acc: 0.72
Found Improvements, Saving the New Model.
epoch: 5 - train loss: 0.5581 train acc: 0.77 val loss: 0.5716 val acc: 0.73
epoch: 6 - train loss: 0.5506 train acc: 0.78 val loss: 0.5662 val acc: 0.73
epoch: 7 - train loss: 0.5444 train acc: 0.78 val loss: 0.5618 val acc: 0.73
epoch: 8 - train loss: 0.5392 train acc: 0.78 val loss: 0.5581 val acc: 0.73
epoch: 9 - train loss: 0.5347 train acc: 0.78 val loss: 0.5550 val acc: 0.73
```

```
epoch: 10 - train loss: 0.5308 train acc: 0.78 val loss: 0.5522 val acc: 0.73
Found Improvements, Saving the New Model.
epoch: 11 - train loss: 0.5274 train acc: 0.78 val loss: 0.5498 val acc: 0.74
epoch: 12 - train loss: 0.5244 train acc: 0.78 val loss: 0.5475 val acc: 0.74
epoch: 13 - train loss: 0.5217 train acc: 0.78 val loss: 0.5455 val acc: 0.74
Found Improvements, Saving the New Model.
epoch: 14 - train loss: 0.5193 train acc: 0.78 val loss: 0.5437 val acc: 0.75
epoch: 15 - train loss: 0.5171 train acc: 0.78 val loss: 0.5419 val acc: 0.75
epoch: 16 - train loss: 0.5150 train acc: 0.78 val loss: 0.5403 val acc: 0.75
epoch: 17 - train loss: 0.5131 train acc: 0.78 val loss: 0.5388 val acc: 0.75
epoch: 18 - train loss: 0.5114 train acc: 0.78 val loss: 0.5373 val acc: 0.75
epoch: 19 - train loss: 0.5097 train acc: 0.78 val loss: 0.5359 val acc: 0.75
epoch: 20 - train loss: 0.5081 train acc: 0.78 val loss: 0.5345 val acc: 0.75
Found Improvements, Saving the New Model.
epoch: 21 - train loss: 0.5066 train acc: 0.78 val loss: 0.5332 val acc: 0.75
epoch: 22 - train loss: 0.5052 train acc: 0.79 val loss: 0.5319 val acc: 0.75
epoch: 23 - train loss: 0.5038 train acc: 0.79 val loss: 0.5306 val acc: 0.75
epoch: 24 - train loss: 0.5025 train acc: 0.79 val loss: 0.5294 val acc: 0.75
Found Improvements, Saving the New Model.
epoch: 25 - train loss: 0.5013 train acc: 0.79 val loss: 0.5282 val acc: 0.76
epoch: 26 - train loss: 0.5001 train acc: 0.79 val loss: 0.5270 val acc: 0.76
epoch: 27 - train loss: 0.4989 train acc: 0.80 val loss: 0.5258 val acc: 0.76
epoch: 28 - train loss: 0.4978 train acc: 0.80 val loss: 0.5247 val acc: 0.76
Found Improvements, Saving the New Model.
epoch: 29 - train loss: 0.4967 train acc: 0.80 val loss: 0.5235 val acc: 0.77
epoch: 30 - train loss: 0.4956 train acc: 0.80 val loss: 0.5224 val acc: 0.77
Found Improvements, Saving the New Model.
epoch: 31 - train loss: 0.4946 train acc: 0.80 val loss: 0.5213 val acc: 0.78
epoch: 32 - train loss: 0.4936 train acc: 0.81 val loss: 0.5202 val acc: 0.78
epoch: 33 - train loss: 0.4927 train acc: 0.81 val loss: 0.5192 val acc: 0.78
epoch: 34 - train loss: 0.4917 train acc: 0.81 val loss: 0.5181 val acc: 0.78
Found Improvements, Saving the New Model.
epoch: 35 - train loss: 0.4908 train acc: 0.81 val loss: 0.5171 val acc: 0.79
epoch: 36 - train loss: 0.4899 train acc: 0.81 val loss: 0.5161 val acc: 0.79
epoch: 37 - train loss: 0.4891 train acc: 0.82 val loss: 0.5150 val acc: 0.79
epoch: 38 - train loss: 0.4882 train acc: 0.83 val loss: 0.5141 val acc: 0.79
epoch: 39 - train loss: 0.4874 train acc: 0.83 val loss: 0.5131 val acc: 0.79
epoch: 40 - train loss: 0.4866 train acc: 0.83 val loss: 0.5121 val acc: 0.79
epoch: 41 - train loss: 0.4859 train acc: 0.83 val loss: 0.5112 val acc: 0.79
epoch: 42 - train loss: 0.4851 train acc: 0.83 val loss: 0.5102 val acc: 0.79
epoch: 43 - train loss: 0.4844 train acc: 0.83 val loss: 0.5093 val acc: 0.79
epoch: 44 - train loss: 0.4837 train acc: 0.84 val loss: 0.5084 val acc: 0.79
epoch: 45 - train loss: 0.4830 train acc: 0.84 val loss: 0.5075 val acc: 0.79
epoch: 46 - train loss: 0.4823 train acc: 0.84 val loss: 0.5067 val acc: 0.79
```

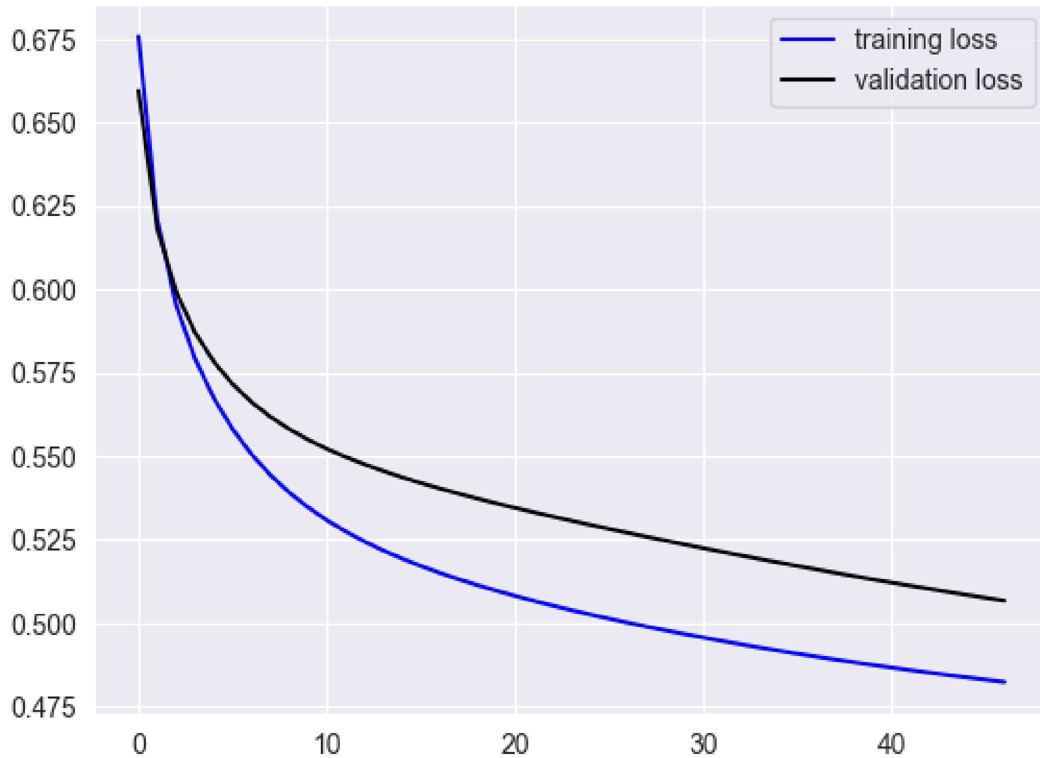
5.9.1 4.8.1 Evaluating Model to Estimate Performance

[319]:
```
# Setting to Evaluation Mode and testing our model on the test set to get␣
↪estimate of its performance.
model.eval()
for batch, (X_test, y_test) in enumerate(test_dataloader):
    test_pred = model.forward(X_test)
    test_f1_scores, test_accuracy = my_stats(y_test, test_pred,␣
↪show_confusion_matrix=True)
    print("\n test accuracy: {:2.2f}".format(test_accuracy))
    test_pred_numpy = test_pred.detach().numpy()
    print('\n The F1 scores for each of the classes are: '+str(test_f1_scores))

    print("\n Loss graph:")
    fig, ax = plt.subplots()
    losses = np.array(losses)
    ax.plot(losses[:, 0], 'b-', label='training loss')
    ax.plot(losses[:, 1], 'k-', label='validation loss')
    plt.legend(loc='upper right')
```

```
Confusion matrix:
```



```
Prediction outputs appended to 'predictions.csv'

 test accuracy: 0.86

 The F1 scores for each of the classes are: [0.90625    0.70588235]

 Loss graph:
```



5.10 4.9 Running Experiments on MLP Model - Tobacco

Now we will input the another training data into the MLP model to predict and check the accuracy of the model to predict the export value of Tobacco. I will be using the exact hyper parameter values as provided to me from the random search cross validation folds.

[321]:
```
# Creating an instance of the MLP network
feature_count = training_data.shape[1]
hidden_layer_sizes = [30]
class_count = np.unique(train_label_col_tobacco_export_value).shape[0]
model = two_layer_MLP(feature_count, hidden_layer_sizes, class_count)



# Setting best hyperparameters based on my random search CV function
```

```
num_epochs = 35
learning_rate = 0.06962220852374669
batch_size = 123


# Setting up the data loading by batch
# With the test and validation sets having only one batch
unique_id = "Tobacco Export Value"

train_set = ExportDataset(scaled_training_data,␣
 ↪train_label_col_tobacco_export_value)
train_dataloader = DataLoader(train_set, batch_size=batch_size)

val_set = ExportDataset(scaled_validation_data,␣
 ↪val_label_col_tobacco_export_value)
val_dataloader = DataLoader(val_set, batch_size=len(val_set))

test_set = ExportDataset(scaled_test_data, test_label_col_tobacco_export_value)
test_dataloader = DataLoader(test_set, batch_size=len(test_set))


# Setting up the SGD optimizer for updating the model weights
optimizer = optim.SGD(model.parameters(), lr=learning_rate)


# Computing cross entropy loss against the training labels
loss_function = nn.CrossEntropyLoss()

best_model_acc = 0
losses = []

# Iterating over the dataset at two different stages:
# - First, Iterating over the batches in the dataset
# - Second, Iterating over the specified number of epochs
for epoch in range(0, num_epochs):

    # Setting the model to training mode
    model.train()

    if epoch == 0:  best_model = deepcopy(model)

    for batch, (X_train, y_train) in enumerate(train_dataloader):

        optimizer.zero_grad()

        # Computing the forward pass and then the loss
        train_pred = model.forward(X_train)
```

```
        train_loss = loss_function(train_pred, y_train)
        train_avg_f1_score, train_acc = my_stats(y_train, train_pred)

        # Computing the model parameters' gradients and propagating the loss␣
↪backwards through the network.
        train_loss.backward()

        # Updating the model parameters using those gradients
        optimizer.step()

    # Evaluating on the validation set
    model.eval()
    for batch, (X_val, y_val) in enumerate(val_dataloader):
        val_pred = model.forward(X_val)
        val_loss = loss_function(val_pred, y_val)
        val_avg_f1_score, val_acc = my_stats(y_val, val_pred)

    if val_acc > best_model_acc:
        best_model_acc = val_acc
        best_model = deepcopy(model)
        print('Found improvement in performance. New model saved.')

    # Printing how well does the network does on batches which results on how␣
↪well the mode is progressing.
    print("epoch: {} - train loss: {:.4f} train acc: {:.2f} val loss: {:.4f}␣
↪val acc: {:.2f}".format(
        epoch,
        train_loss.item(),
        train_acc,
        val_loss.item(),
        val_acc ))

    losses.append([train_loss.item(), val_loss.item()])

model = best_model
tobacco_model = model
```

```
Found improvement in performance. New model saved.
epoch: 0 - train loss: 0.6675 train acc: 0.69 val loss: 0.6577 val acc: 0.68
Found improvement in performance. New model saved.
epoch: 1 - train loss: 0.6343 train acc: 0.77 val loss: 0.6326 val acc: 0.73
epoch: 2 - train loss: 0.6108 train acc: 0.78 val loss: 0.6136 val acc: 0.73
Found improvement in performance. New model saved.
epoch: 3 - train loss: 0.5927 train acc: 0.79 val loss: 0.5987 val acc: 0.75
Found improvement in performance. New model saved.
epoch: 4 - train loss: 0.5782 train acc: 0.80 val loss: 0.5872 val acc: 0.75
Found improvement in performance. New model saved.
```

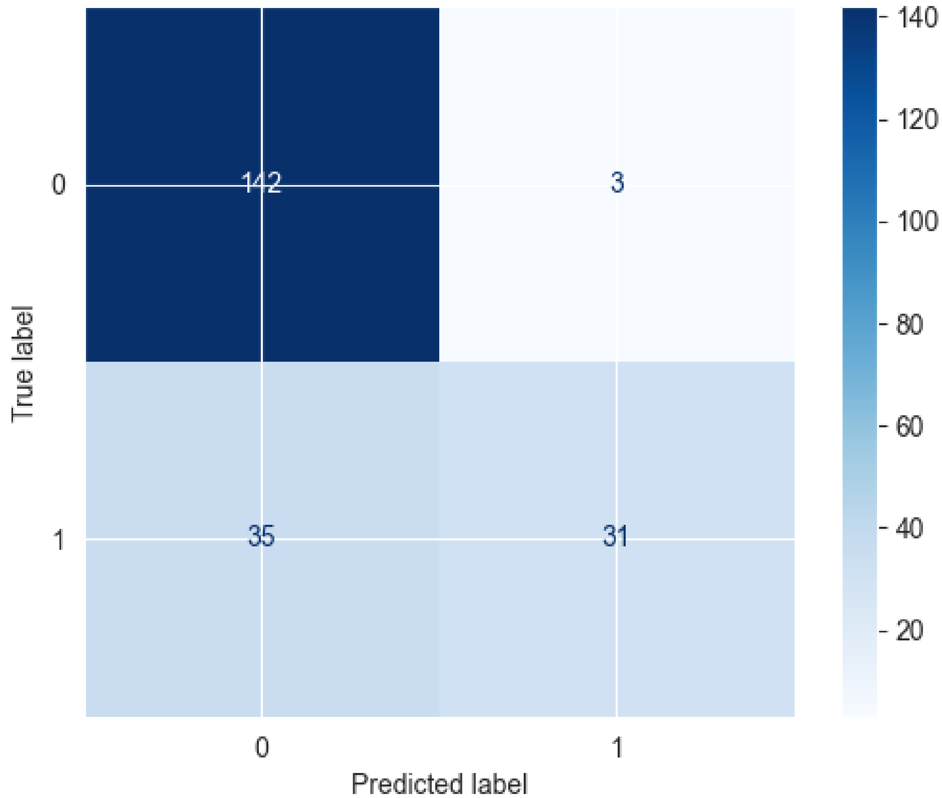
```
epoch: 5 - train loss: 0.5666 train acc: 0.80 val loss: 0.5783 val acc: 0.76
epoch: 6 - train loss: 0.5572 train acc: 0.80 val loss: 0.5713 val acc: 0.76
epoch: 7 - train loss: 0.5494 train acc: 0.80 val loss: 0.5656 val acc: 0.76
epoch: 8 - train loss: 0.5429 train acc: 0.80 val loss: 0.5609 val acc: 0.76
epoch: 9 - train loss: 0.5372 train acc: 0.80 val loss: 0.5568 val acc: 0.76
epoch: 10 - train loss: 0.5323 train acc: 0.80 val loss: 0.5532 val acc: 0.76
epoch: 11 - train loss: 0.5280 train acc: 0.80 val loss: 0.5501 val acc: 0.76
epoch: 12 - train loss: 0.5242 train acc: 0.80 val loss: 0.5472 val acc: 0.76
epoch: 13 - train loss: 0.5207 train acc: 0.80 val loss: 0.5446 val acc: 0.76
epoch: 14 - train loss: 0.5175 train acc: 0.80 val loss: 0.5422 val acc: 0.76
epoch: 15 - train loss: 0.5146 train acc: 0.80 val loss: 0.5399 val acc: 0.76
epoch: 16 - train loss: 0.5119 train acc: 0.80 val loss: 0.5377 val acc: 0.76
epoch: 17 - train loss: 0.5093 train acc: 0.80 val loss: 0.5356 val acc: 0.75
epoch: 18 - train loss: 0.5070 train acc: 0.80 val loss: 0.5336 val acc: 0.75
epoch: 19 - train loss: 0.5047 train acc: 0.80 val loss: 0.5317 val acc: 0.75
epoch: 20 - train loss: 0.5026 train acc: 0.80 val loss: 0.5298 val acc: 0.75
epoch: 21 - train loss: 0.5005 train acc: 0.80 val loss: 0.5279 val acc: 0.76
epoch: 22 - train loss: 0.4986 train acc: 0.80 val loss: 0.5261 val acc: 0.76
epoch: 23 - train loss: 0.4967 train acc: 0.80 val loss: 0.5242 val acc: 0.76
epoch: 24 - train loss: 0.4949 train acc: 0.80 val loss: 0.5224 val acc: 0.76
epoch: 25 - train loss: 0.4931 train acc: 0.80 val loss: 0.5206 val acc: 0.76
epoch: 26 - train loss: 0.4914 train acc: 0.81 val loss: 0.5188 val acc: 0.76
epoch: 27 - train loss: 0.4897 train acc: 0.81 val loss: 0.5170 val acc: 0.76
Found improvement in performance. New model saved.
epoch: 28 - train loss: 0.4881 train acc: 0.81 val loss: 0.5152 val acc: 0.77
epoch: 29 - train loss: 0.4865 train acc: 0.81 val loss: 0.5134 val acc: 0.77
epoch: 30 - train loss: 0.4849 train acc: 0.81 val loss: 0.5116 val acc: 0.77
epoch: 31 - train loss: 0.4833 train acc: 0.81 val loss: 0.5098 val acc: 0.77
Found improvement in performance. New model saved.
epoch: 32 - train loss: 0.4818 train acc: 0.81 val loss: 0.5080 val acc: 0.78
epoch: 33 - train loss: 0.4803 train acc: 0.81 val loss: 0.5061 val acc: 0.78
Found improvement in performance. New model saved.
epoch: 34 - train loss: 0.4787 train acc: 0.81 val loss: 0.5043 val acc: 0.79
```

5.10.1 4.9.1 Evaluating Model to Estimate Performance

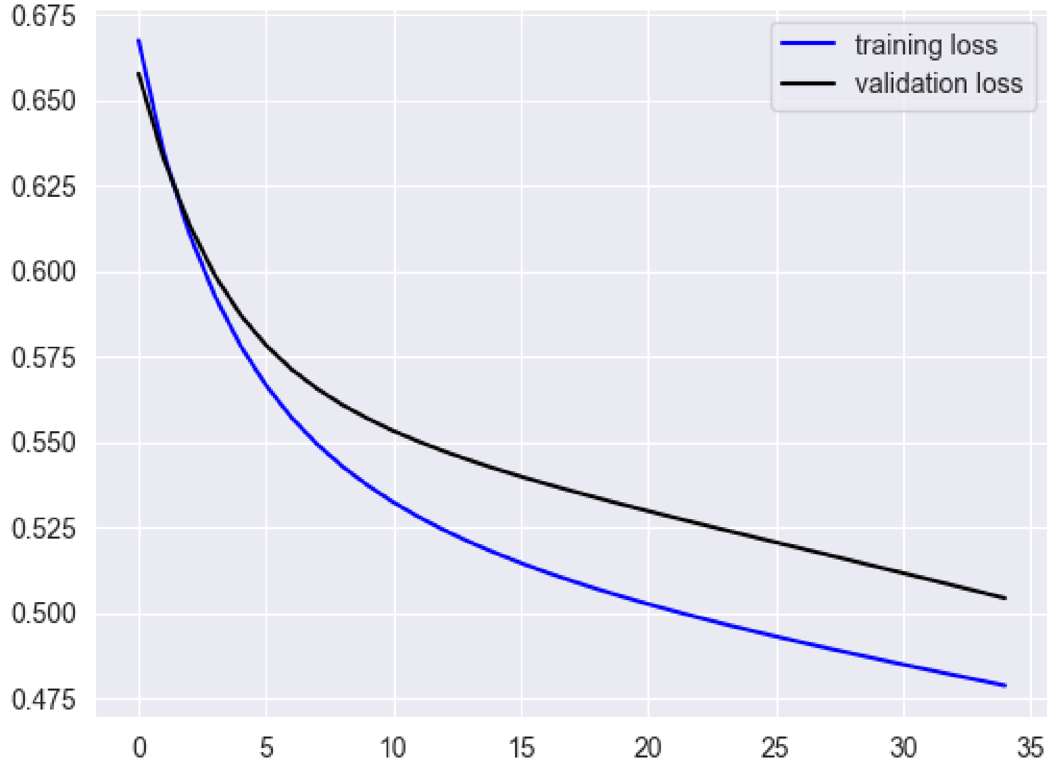
[322]:
```
# Setting to Evaluation Mode and testing our model on the test set to get␣
↪estimate of its performance.
model.eval()
for batch, (X_test, y_test) in enumerate(test_dataloader):
    test_pred = model.forward(X_test)
    test_f1_scores, test_accuracy = my_stats(y_test, test_pred,␣
↪show_confusion_matrix=True)
    print("\n test accuracy: {:2.2f}".format(test_accuracy))
    test_pred_numpy = test_pred.detach().numpy()
    print('\n The F1 scores for each of the classes are: '+str(test_f1_scores))
```

```
    print("\n Loss graph:")
    fig, ax = plt.subplots()
    losses = np.array(losses)
    ax.plot(losses[:, 0], 'b-', label='training loss')
    ax.plot(losses[:, 1], 'k-', label='validation loss')
    plt.legend(loc='upper right')
```

```
 Confusion matrix:
```




```
Prediction outputs appended to 'predictions.csv'

 test accuracy: 0.82

 The F1 scores for each of the classes are: [0.88198758 0.62      ]

 Loss graph:
```



5.11 4.10 Saving MLP Model

5.11.1 4.10.1 MLP Model - Cereal

[323]:
```
# Saving the best model as a .pkl file
model_save_path = 'cereal_mlp_model.pkl'
with open(model_save_path, 'wb') as f:
    pickle.dump(model, f)
print(f"Model saved as {model_save_path}")
```

```
Model saved as cereal_mlp_model.pkl
```

5.11.2 4.10.2 MLP Model - Tobacco

[324]:
```
# Saving the best model as a .pkl file
model_save_path = 'tobacco_mlp_model.pkl'
with open(model_save_path, 'wb') as f:
    pickle.dump(model, f)
print(f"Model saved as {model_save_path}")
```

```
Model saved as tobacco_mlp_model.pkl
```