Objectives

Section 1: Loading and Preprocess the data

1.1: Preprocess each CSV file

In [ ]:
```
#Ignore all warnings
import warnings
warnings.filterwarnings("ignore")

#Importing the necessary libraries
import pandas as pd
from prettypandas import PrettyPandas
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

Preprocess Trade Indicators - FAOSTAT_data_en_2-22-2024.csv file

In [ ]:
```
# Load the dataset
df_trade = pd.read_csv('./Food trade indicators - FAOSTAT_data_en_2-22-20

# Select the columns and rename them
df_trade = df_trade[['Area', 'Item', 'Year', 'Element', 'Value']]

# Drop rows with any empty values
df_trade = df_trade.dropna()

# Pivot the table for 'Import Value' and 'Export Value'
df_import = df_trade[df_trade['Element'] == 'Import Value'].pivot_table(
    index=['Area', 'Item', 'Year'],
    values='Value',
    aggfunc='first'
).rename(columns={'Value': 'Import_Value'}).reset_index()

df_export = df_trade[df_trade['Element'] == 'Export Value'].pivot_table(
    index=['Area', 'Item', 'Year'],
    values='Value',
    aggfunc='first'
).rename(columns={'Value': 'Export_Value'}).reset_index()

# Merge the pivoted DataFrames on 'Area', 'Item', and 'Year'
df_trade_values = pd.merge(df_import, df_export, on=['Area', 'Item', 'Yea
```

```
# Create a new column for 'Year' 3 years ahead
df_trade_values['Year_3_Ahead'] = df_trade_values['Year'] + 3

# Merge with itself to get export value 3 years ahead
df_trade_values = pd.merge(df_trade_values, df_trade_values[['Area', 'Ite
                           left_on=['Area', 'Item', 'Year_3_Ahead'], righ
                           suffixes=('', '_3_Years_Ahead'), how='left')

# Sort by 'Area', 'Item', 'Year' to ensure chronological order for lag an
df_trade_values.sort_values(by=['Area', 'Item', 'Year'], inplace=True)

# Function to create lag and rolling window features within each group
def create_features(group):
    # Create lag features for 'Export_Value'
    group['Export_Value_Lag1'] = group['Export_Value'].shift(1)
    group['Export_Value_Lag2'] = group['Export_Value'].shift(2)
    group['Export_Value_Lag3'] = group['Export_Value'].shift(3)

    # Create a 3-year rolling mean for 'Export_Value'
    group['Export_Value_Rolling_Mean3'] = group['Export_Value'].rolling(w

    return group

# Apply the function to each group
df_trade_values = df_trade_values.groupby(['Area', 'Item']).apply(create_

# Drop rows with any NaN values created by the lag and rolling operations
df_trade_values.dropna(inplace=True)

# Drop unnecessary columns
df_trade_values.drop(['Year_3_Ahead', 'Year_3_Years_Ahead'], axis=1, inpl
```

Preprocess Pesticides use - FAOSTAT_data_en_2-27-2024.csv file

In [ ]:

```
# Load the dataset
df_pesticides = pd.read_csv('./Pesticides use - FAOSTAT_data_en_2-27-2024

# Select the columns
df_pesticides = df_pesticides[['Area', 'Item', 'Year', 'Element', 'Value'

# Drop rows with any empty values
df_pesticides = df_pesticides.dropna()

# Pivot the table
df_pesticides_pivot = df_pesticides.pivot_table(
    index=['Area', 'Year'],
    columns=['Item', 'Element'],
    values='Value',
    aggfunc='first'
).reset_index()
```

```
# Join with _
df_pesticides_pivot.columns = ['_'.join(col).strip() for col in df_pestic
df_pesticides_pivot = df_pesticides_pivot.rename(columns={'Area_': 'Area'

# Rename the columns
df_pesticides_pivot = df_pesticides_pivot.rename(columns=lambda x: f'Pest

# Fill missing values with 0 assuming no pesticides of that type were use
df_pesticides_pivot = df_pesticides_pivot.fillna(0)

PrettyPandas(df_pesticides_pivot.head())
```

Out[ ]:

	Area	Year	Pesticide_Fungicides and Bactericides_Agricultural Use	Pesticide_Fungicides – Seed treatments_Agricultural Use	Pesticide_Herb
0	Albania	2000	105.730000	0.050000	
1	Albania	2001	108.080000	0.060000	
2	Albania	2002	110.430000	0.070000	
3	Albania	2003	112.770000	0.080000	
4	Albania	2004	115.120000	0.090000	

Preprocess Land use - FAOSTAT_data_en_2-22-2024.csv file

In [ ]:

```
# Load the dataset
df_land_use = pd.read_csv('./Land use - FAOSTAT_data_en_2-22-2024.csv', l

# Select the columns
df_land_use = df_land_use[['Area', 'Year', 'Item', 'Value']]

# Drop rows with any empty values
df_land_use = df_land_use.dropna()

# Drop rows with 'Item' = 'Country are' and 'Land area'
df_land_use = df_land_use.loc[~df_land_use['Item'].isin(['Country area',

# Drop rows with any empty values
df_land_use = df_land_use.dropna()

# Pivot the table to have one row per 'Area' and 'Year' and each 'Item' a
df_land_use_pivot = df_land_use.pivot_table(
    index=['Area', 'Year'],
    columns='Item',
    values='Value'
).reset_index()

# Rename the columns
```

```
df_land_use_pivot.columns = ['Area', 'Year'] + [f'LandUse_{col}' for col

PrettyPandas(df_land_use_pivot.head())
```

Out[ ]:

	Area	Year	LandUse_Agriculture	LandUse_Agriculture area actually irrigated	LandUse_Arable land
0	Afghanistan	1980	38049.000000	nan	7910.000000
1	Afghanistan	1981	38053.000000	nan	7910.000000
2	Afghanistan	1982	38054.000000	nan	7910.000000
3	Afghanistan	1983	38054.000000	nan	7910.000000
4	Afghanistan	1984	38054.000000	nan	7910.000000

Preprocess Land temperature change - FAOSTAT_data_en_2-27-2024.csv

In [ ]:

```
# Load the dataset
df_temperature = pd.read_csv('./Land temperature change - FAOSTAT_data_en

# Select the columns
df_temperature = df_temperature[['Area', 'Months', 'Year', 'Element', 'Va

# Drop rows with any empty values
df_temperature = df_temperature.dropna()

# Filter for 'Temperature change' during the 'Meteorological year'
df_temperature_annual = df_temperature[
    (df_temperature['Element'] == 'Temperature change') &
    (df_temperature['Months'] == 'Meteorological year')
]

# Select relevant columns
df_temperature_annual = df_temperature_annual[['Area', 'Year', 'Value']]

# Rename the columns
df_temperature_annual.rename(columns={'Value': 'TempChange_Annual'}, inpl

PrettyPandas(df_temperature_annual.head())
```

Out[ ]:

	Area	Year	TempChange_Annual
184	Afghanistan	2000	0.993000
185	Afghanistan	2001	1.311000
186	Afghanistan	2002	1.365000
187	Afghanistan	2003	0.587000
188	Afghanistan	2004	1.373000

Preprocess Foreign direct investment - FAOSTAT_data_en_2-27-2024.csv

In [ ]:

```
# Load the dataset
df_fdi = pd.read_csv('./Foreign direct investment - FAOSTAT_data_en_2-27-

# Select the columns
df_fdi_relevant = df_fdi[['Area', 'Year', 'Item', 'Value']]

# Drop rows with any empty values
df_fdi_relevant = df_fdi_relevant.dropna()

# Use FDI inflows to agriculture only
df_fdi_agri = df_fdi_relevant[df_fdi_relevant['Item'].str.contains('FDI i

# Pivot the table
df_fdi_pivot = df_fdi_agri.pivot_table(
    index=['Area', 'Year'],
    columns='Item',
    values='Value'
).reset_index()

# Rename the columns
df_fdi_pivot.rename(columns=lambda x: f'{x.replace(", ", "_").replace(" "

PrettyPandas(df_fdi_pivot.head())
```

Out[ ]:

Item	Area	Year	FDI_inflows_to_Agriculture_Forestry_and_Fishing
0	Albania	2004	0.642888
1	Albania	2005	0.494601
2	Albania	2006	2.508966
3	Albania	2007	2.737334
4	Albania	2008	-79.100597

Preprocess Food security indicators - FAOSTAT_data_en_2-22-2024.csv

In [ ]:
```
# Load the dataset
df_food_security = pd.read_csv('./Food security indicators  - FAOSTAT_dat

# Select the columns
df_food_security = df_food_security[['Area', 'Year', 'Item', 'Value']]

# Filter for relevant 'Item' categories based on the focus of the analysi
irrelevant_items = [
    'Prevalence of anemia among women of reproductive age (15-49 years)',
    'Prevalence of low birthweight (percent)',
    'Per capita food production variability (constant 2014-2016 thousand
    'Percent of arable land equipped for irrigation (percent) (3-year ave
]

# Drop irrelevant items
df_food_security_relevant = df_food_security[~df_food_security['Item'].is

# Drop rows with any empty values
df_food_security_relevant = df_food_security_relevant.dropna()

# Convert 'Year' to a string to handle both single years and ranges (e.g.
df_food_security_relevant['Year'] = df_food_security_relevant['Year'].ast

# Split into yearly and 3-year average DataFrames
df_yearly = df_food_security_relevant[~df_food_security_relevant['Year'].
df_3year_avg = df_food_security_relevant[df_food_security_relevant['Year'

# Expand 3-year averages into annual values
expanded_rows = []
for _, row in df_3year_avg.iterrows():
    start_year, end_year = map(int, row['Year'].split('-'))
    for year in range(start_year, end_year + 1):
        new_row = row.copy()
        new_row['Year'] = str(year)
        expanded_rows.append(new_row)

df_expanded = pd.DataFrame(expanded_rows)

# Merge expanded 3-year data with yearly data, giving precedence to yearl
df_combined = pd.concat([df_yearly, df_expanded]).drop_duplicates(subset=

# Pivot the table
df_fsi_combined_pivot = df_combined.pivot_table(
    index=['Area', 'Year'],
    columns='Item',
    values='Value',
    aggfunc='first'
).reset_index()

# Rename the columns
df_fsi_combined_pivot.columns = ['Area', 'Year'] + [f'FSI_{c.replace(", "
```

```
PrettyPandas(df_fsi_combined_pivot.head())
```

Out[ ]:

	Area	Year	FSI_Average_dietary_energy_supply_adequacy_percent_3_year_
0	Afghanistan	2000	8ℓ
1	Afghanistan	2001	8ℓ
2	Afghanistan	2002	8ℓ
3	Afghanistan	2003	8ʕ
4	Afghanistan	2004	9ℷ

Preprocess Food balances indicators - FAOSTAT_data_en_2-22-2024.csv

In [ ]:

```
# Load the dataset
df_food_balances = pd.read_csv('./Food balances indicators - FAOSTAT_data

# Select columns
df_food_balances_relevant = df_food_balances[['Area', 'Year', 'Element',

# Drop rows with any empty values
df_food_balances_relevant = df_food_balances_relevant.dropna()

# Drop rows with Item = Meat, Eggs, Milk - Excluding Butter, Fish, Seafoo
df_food_balances_relevant = df_food_balances_relevant.loc[~df_food_balanc

# only keep elements 'export quantity'
df_food_balances_relevant = df_food_balances_relevant.loc[df_food_balance

# Create a new column 'Element_Item' combining 'Element' and 'Item'
df_food_balances_relevant['Element_Item'] = df_food_balances_relevant['El

# Pivot the table
df_food_balances_pivot = df_food_balances_relevant.pivot_table(
    index=['Area', 'Year'],
    columns='Element_Item',
    values='Value',
    aggfunc='first'
).reset_index()

# Rename the columns
df_food_balances_pivot.rename(columns=lambda x: f'FoodBalance_{x.replace(

# Fill missing values with 0 assuming no food of that type was produced
df_food_balances_pivot = df_food_balances_pivot.fillna(0)

PrettyPandas(df_food_balances_pivot.head())
```

Out[ ]:

Element_Item	Area	Year	FoodBalance_Export_Quantity_Alcoholic_Beverages
0	Afghanistan	2010	0.000000
1	Afghanistan	2011	0.000000
2	Afghanistan	2012	0.000000
3	Afghanistan	2013	0.000000
4	Afghanistan	2014	0.000000

Preprocess Fertilizers use - FAOSTAT_data_en_2-27-2024.csv

In [ ]:

```
# Load the dataset
df_fertilizers = pd.read_csv('./Fertilizers use - FAOSTAT_data_en_2-27-20

# Select the columns
df_fertilizers_relevant = df_fertilizers[['Area', 'Year', 'Item', 'Value'

# Drop rows with any empty values
df_fertilizers_relevant = df_fertilizers_relevant.dropna()

# Pivot the table
df_fertilizers_pivot = df_fertilizers_relevant.pivot_table(
    index=['Area', 'Year'],
    columns='Item',
    values='Value',
    aggfunc='first'
).reset_index()

# Rename the columns
df_fertilizers_pivot.rename(columns=lambda x: f'FertilizerUse_{x.replace(

# Fill missing values with 0 assuming no fertilizers of that type were us
df_fertilizers_pivot = df_fertilizers_pivot.fillna(0)

PrettyPandas(df_fertilizers_pivot.head())
```

Out[ ]:

Item	Area	Year	FertilizerUse_Ammonia__anhydrous	FertilizerUse_Ammoniu
0	Afghanistan	2002	0.000000	
1	Afghanistan	2003	0.000000	
2	Afghanistan	2004	0.000000	
3	Afghanistan	2005	0.000000	
4	Afghanistan	2006	0.000000	

Preprocess Exchange rate - FAOSTAT_data_en_2-22-

2024.csv

In [ ]:
```
# Load the dataset
df_exchange_rates = pd.read_csv('./Exchange rate - FAOSTAT_data_en_2-22-2

# Select the columns
df_exchange_rates = df_exchange_rates[['Area', 'Year', 'Value']]

# Drop rows with any empty values
df_exchange_rates = df_exchange_rates.dropna()

# Group by 'Area' and 'Year' and calculate the mean 'Value'
df_yearly_exchange_rates = df_exchange_rates.groupby(['Area', 'Year'])['V

# Rename the column
df_yearly_exchange_rates.rename(columns={'Value': 'Average_Exchange_Rate'

PrettyPandas(df_yearly_exchange_rates.head())
```

Out[ ]:

	Area	Year	Average_Exchange_Rate
0	Afghanistan	1980	44.129167
1	Afghanistan	1981	49.479902
2	Afghanistan	1982	50.599608
3	Afghanistan	1983	50.599608
4	Afghanistan	1984	50.599606

Preprocess Emissions - FAOSTAT_data_en_2-27-2024.csv

In [ ]:
```
# Load the dataset
df_emissions = pd.read_csv('./Emissions - FAOSTAT_data_en_2-27-2024.csv')

# Select the columns
df_emissions = df_emissions[['Area', 'Year', 'Element', 'Item', 'Value']]

# Drop rows with any empty values
df_emissions = df_emissions.dropna()

# Pivot the table
df_emissions_pivot = df_emissions.pivot_table(
    index=['Area', 'Year'],
    columns=['Element', 'Item'],
    values='Value',
    aggfunc='first'
).reset_index()

# Join with '_'
df_emissions_pivot.columns = ['_'.join(col).strip() for col in df_emissio
```

```
# Rename columns
df_emissions_pivot.columns = ['Area', 'Year'] + [f'Emission_{c.replace("

PrettyPandas(df_emissions_pivot.head())
```

Out[ ]:

	Area	Year	Emission_Crops_total_Emissions_CH4_All_Crops	Emission_Cr
0	Afghanistan	2000	20.847100	
1	Afghanistan	2001	19.260500	
2	Afghanistan	2002	21.255300	
3	Afghanistan	2003	23.701700	
4	Afghanistan	2004	30.308900	

Preprocess Crops production indicators - FAOSTAT_data_en_2-22-2024.csv

In [ ]:

```
# Load the dataset
df_crops = pd.read_csv('./Crops production indicators - FAOSTAT_data_en_2

# Select the columns
df_crops_filtered = df_crops[['Area', 'Year', 'Element', 'Item', 'Value']

# Drop rows with any empty values
df_crops_filtered = df_crops_filtered.dropna()

# Pivot the table
df_crops_pivoted = df_crops_filtered.pivot_table(index=['Area', 'Year'],

# Rename the columns
df_crops_pivoted = df_crops_pivoted.rename(columns=lambda x: 'CropYield_'

# Fill missing values with 0 assuming no crops of that type were produced
df_crops_pivoted = df_crops_pivoted.fillna(0)

PrettyPandas(df_crops_pivoted.head())
```

Out[ ]:

Item	Area	Year	CropYield_Cereals_primary	CropYield_Citrus_Fruit_Total	C
0	Afghanistan	2000	8063.000000	71245.000000	
1	Afghanistan	2001	10067.000000	71417.000000	
2	Afghanistan	2002	16698.000000	71477.000000	
3	Afghanistan	2003	14580.000000	73423.000000	
4	Afghanistan	2004	13348.000000	78025.000000	

Preprocess Consumer prices indicators -

FAOSTAT_data_en_2-22-2024.csv

In [ ]:
```
# Load the dataset
df_consumer_prices = pd.read_csv('./Consumer prices indicators - FAOSTAT_

# Select the columns
df_consumer_prices = df_consumer_prices[['Area', 'Year', 'Item', 'Value']

# Drop rows with any empty values
df_consumer_prices = df_consumer_prices.dropna()

# Pivot the table
df_consumer_prices_pivot = df_consumer_prices.pivot_table(
    index=['Area', 'Year'],
    columns='Item',
    values='Value',
    aggfunc='mean'  # Use mean to aggregate monthly data into a single an
).reset_index()

# Rename the columns
df_consumer_prices_pivot.rename(columns={
    'Consumer Prices, Food Indices (2015 = 100)': 'ConsumerPrice_Food_Ind
    'Food price inflation': 'ConsumerPrice_Food_Price_Inflation'
}, inplace=True)

PrettyPandas(df_consumer_prices_pivot.head())
```

Out[ ]:

Item	Area	Year	ConsumerPrice_Food_Indices	ConsumerPrice_Food_Price_I
0	Afghanistan	2000	26.629848	
1	Afghanistan	2001	29.893548	12
2	Afghanistan	2002	35.344892	18
3	Afghanistan	2003	40.203113	14
4	Afghanistan	2004	45.840561	14

Preprocess Employment - FAOSTAT_data_en_2-27-2024.csv

In [ ]:
```
# Load the dataset
df_employment = pd.read_csv('./Employment - FAOSTAT_data_en_2-27-2024.csv

# Select the columns
df_employment = df_employment[['Area', 'Year', 'Indicator', 'Value']]

# Drop rows with any empty values
df_employment = df_employment.dropna()

# Pivot the table
```

```
df_employment_pivot = df_employment.pivot_table(
    index=['Area', 'Year'],
    columns='Indicator',
    values='Value',
    aggfunc='first'
).reset_index()

# Rename columns
df_employment_pivot.rename(columns={
    'Mean weekly hours actually worked per employed person in agriculture
    'Employment in agriculture, forestry and fishing - ILO modelled estim
}, inplace=True)

# Fill missing values with mean of that specific area
df_employment_pivot = df_employment_pivot.fillna(df_employment_pivot.grou

# If there are still missing values, fill them with the mean of the entir
df_employment_pivot['Employment_Agriculture_Work_Hours_Per_Week'] = df_em

PrettyPandas(df_employment_pivot.head())
```

Out[ ]:

Indicator	Area	Year	Employment_Agriculture_Estimates	Employment_Agric
0	Afghanistan	2000	2765.950000	
1	Afghanistan	2001	2805.540000	
2	Afghanistan	2002	2897.510000	
3	Afghanistan	2003	3093.270000	
4	Afghanistan	2004	3212.460000	

1.2: Perform Merging of DataFrames

In [ ]:

```
# Ensure 'Area' and 'Year' are not part of the index
df_trade_values = df_trade_values.reset_index(drop=True)

# Ensure 'Area' is string type and 'Year' is string type in all DataFrame
df_trade_values['Area'] = df_trade_values['Area'].astype(str)
df_trade_values['Year'] = df_trade_values['Year'].astype(str)

df_pesticides_pivot['Area'] = df_pesticides_pivot['Area'].astype(str)
df_pesticides_pivot['Year'] = df_pesticides_pivot['Year'].astype(str)

df_land_use_pivot['Area'] = df_land_use_pivot['Area'].astype(str)
df_land_use_pivot['Year'] = df_land_use_pivot['Year'].astype(str)

df_temperature_annual['Area'] = df_temperature_annual['Area'].astype(str)
df_temperature_annual['Year'] = df_temperature_annual['Year'].astype(str)

df_fdi_pivot['Area'] = df_fdi_pivot['Area'].astype(str)
df_fdi_pivot['Year'] = df_fdi_pivot['Year'].astype(str)
```

```
df_fsi_combined_pivot['Area'] = df_fsi_combined_pivot['Area'].astype(str)
df_fsi_combined_pivot['Year'] = df_fsi_combined_pivot['Year'].astype(str)

df_food_balances_pivot['Area'] = df_food_balances_pivot['Area'].astype(st
df_food_balances_pivot['Year'] = df_food_balances_pivot['Year'].astype(st

df_fertilizers_pivot['Area'] = df_fertilizers_pivot['Area'].astype(str)
df_fertilizers_pivot['Year'] = df_fertilizers_pivot['Year'].astype(str)

df_yearly_exchange_rates['Area'] = df_yearly_exchange_rates['Area'].astyp
df_yearly_exchange_rates['Year'] = df_yearly_exchange_rates['Year'].astyp

df_emissions_pivot['Area'] = df_emissions_pivot['Area'].astype(str)
df_emissions_pivot['Year'] = df_emissions_pivot['Year'].astype(str)

df_crops_pivoted['Area'] = df_crops_pivoted['Area'].astype(str)
df_crops_pivoted['Year'] = df_crops_pivoted['Year'].astype(str)

df_consumer_prices_pivot['Area'] = df_consumer_prices_pivot['Area'].astyp
df_consumer_prices_pivot['Year'] = df_consumer_prices_pivot['Year'].astyp

df_employment_pivot['Area'] = df_employment_pivot['Area'].astype(str)
df_employment_pivot['Year'] = df_employment_pivot['Year'].astype(str)


# Merge the DataFrames
df_merged = df_trade_values
df_merged = pd.merge(df_merged, df_land_use_pivot, on=['Area', 'Year'], h
df_merged = pd.merge(df_merged, df_temperature_annual, on=['Area', 'Year'
df_merged = pd.merge(df_merged, df_fdi_pivot, on=['Area', 'Year'], how='l
df_merged = pd.merge(df_merged, df_fsi_combined_pivot, on=['Area', 'Year'
df_merged = pd.merge(df_merged, df_food_balances_pivot, on=['Area', 'Year
df_merged = pd.merge(df_merged, df_fertilizers_pivot, on=['Area', 'Year']
df_merged = pd.merge(df_merged, df_yearly_exchange_rates, on=['Area', 'Ye
df_merged = pd.merge(df_merged, df_emissions_pivot, on=['Area', 'Year'],
df_merged = pd.merge(df_merged, df_crops_pivoted, on=['Area', 'Year'], ho
df_merged = pd.merge(df_merged, df_consumer_prices_pivot, on=['Area', 'Ye
df_merged = pd.merge(df_merged, df_employment_pivot, on=['Area', 'Year'],

PrettyPandas(df_merged.head())
```

Out[ ]:

	Area	Item	Year	Import_Value	Export_Value	Export_Value_3_Ye
0	Afghanistan	Alcoholic Beverages	2018	5908.790000	30.940000	
1	Afghanistan	Cereals and Preparations	2012	372176.000000	0.000000	
2	Afghanistan	Cereals and Preparations	2013	419030.000000	0.000000	:
3	Afghanistan	Cereals and Preparations	2014	815313.480000	1074.450000	1(
4	Afghanistan	Cereals and Preparations	2015	768582.620000	173.080000	

1.3: Recoding labels into classes

In [ ]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder

# Calculate percentiles for each item type
def calculate_percentiles(df):
    very_low = np.percentile(df['Export_Value_3_Years_Ahead'], 20)
    low = np.percentile(df['Export_Value_3_Years_Ahead'], 40)
    medium = np.percentile(df['Export_Value_3_Years_Ahead'], 60)
    high = np.percentile(df['Export_Value_3_Years_Ahead'], 80)
    return {
        'very_low': very_low,
        'low': low,
        'medium': medium,
        'high': high
    }

# Apply calculate_percentiles to each group
percentiles = df_merged.groupby('Item').apply(lambda df: calculate_percen

# Convert the results to a dictionary
if not isinstance(percentiles, dict):
    percentiles = percentiles.to_dict()

# Categorize based on the percentiles
def categorize_value(row):
    thresholds = percentiles[row['Item']]
    value = row['Export_Value_3_Years_Ahead']
    if value <= thresholds['very_low']:
        return 'Very Low'
    elif value <= thresholds['low']:
```

```
        return 'Low'
    elif value <= thresholds['medium']:
        return 'Medium'
    elif value <= thresholds['high']:
        return 'High'
    else:
        return 'Very High'

df_merged['Export_Value_3_Years_Ahead_Category'] = df_merged.apply(catego

# Encode the labels numerically
encoder = LabelEncoder()
df_merged['Export_Value_3_Years_Ahead_Category'] = encoder.fit_transform(

# Count the unique number of labels
num_labels = df_merged['Export_Value_3_Years_Ahead_Category'].nunique()
print(f"Number of unique labels: {num_labels}")
```

```
Number of unique labels: 5
```

1.4: Preprocess merged data one more time

In [ ]:
```
# Dropping the columns
df_merged = df_merged.drop(columns=['Area'])
df_merged = df_merged.drop(columns=['Year'])
```

In [ ]:
```
# One hot encoding for column 'Item'
df_merged = pd.get_dummies(df_merged)
```

In [ ]:
```
from sklearn.impute import KNNImputer

# Check for missing values
missing_values = df_merged.isnull().sum()

missing_values_summary = missing_values[missing_values > 0]

# Dropping columns with a high percentage of missing values (50%)
high_missing_cols = missing_values_summary.index[missing_values_summary >
df_merged = df_merged.drop(columns=high_missing_cols)

# Fill the missing values using KNNImputer
imputer = KNNImputer(n_neighbors=5)
df_merged_imputed = imputer.fit_transform(df_merged)
df_merged = pd.DataFrame(df_merged_imputed, columns=df_merged.columns)
```

In [ ]:
```
# Dropping duplicate rows
df_merged = df_merged.drop_duplicates()
```

In [ ]:
```
#reset index
df_merged = df_merged.reset_index(drop=True)
```

```
# Check the final dataset
PrettyPandas(df_merged.head())
```

Out[ ]:

	Import_Value	Export_Value	Export_Value_3_Years_Ahead	Export_Value_Lag1
0	5908.790000	30.940000	60.150000	8.250000
1	372176.000000	0.000000	173.080000	0.000000
2	419030.000000	0.000000	346.570000	0.000000
3	815313.480000	1074.450000	1628.890000	0.000000
4	768582.620000	173.080000	591.290000	1074.450000

Section 2: Selecting training, validation, and test sets

In [ ]:
```
from sklearn.model_selection import train_test_split

# divide the data into training and testing sets (60% train, 20% test, 20
train_df, test_df = train_test_split(df_merged, test_size=0.4, random_sta
test_df, val_df = train_test_split(test_df, test_size=0.5, random_state=4
```

Section 3: Scaling/normalization

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

scaler = StandardScaler()

# Select features to scale
features_to_scale = [feature for feature in train_df.columns if feature !

# Fit the scaler on the training data
scaler.fit(train_df[features_to_scale])

# Apply the scaling transformation to the features of the training, valid
scaled_train_data = scaler.transform(train_df[features_to_scale])
scaled_val_data = scaler.transform(val_df[features_to_scale])
scaled_test_data = scaler.transform(test_df[features_to_scale])

# Add the target variable 'Export_Value_3_Years_Ahead' back
scaled_train_df = pd.DataFrame(scaled_train_data, columns=features_to_sca
scaled_val_df = pd.DataFrame(scaled_val_data, columns=features_to_scale)
scaled_test_df = pd.DataFrame(scaled_test_data, columns=features_to_scale
```

```
scaled_train_df['Export_Value_3_Years_Ahead_Category'] = train_df['Export
scaled_val_df['Export_Value_3_Years_Ahead_Category'] = val_df['Export_Val
scaled_test_df['Export_Value_3_Years_Ahead_Category'] = test_df['Export_V

# Check the scaled training dataset
PrettyPandas(scaled_train_df.head())
```

Out[ ]:

	Import_Value	Export_Value	Export_Value_3_Years_Ahead	Export_Value_Lag1	E
0	0.168530	-0.255411	-0.257396	-0.253677	
1	-0.275834	-0.254973	-0.258235	-0.253422	
2	-0.264068	-0.252864	-0.256881	-0.251860	
3	-0.274330	-0.255118	-0.258301	-0.253669	
4	-0.185138	-0.092654	-0.091762	-0.086248	

Section 4: Building and evaluating a multilayer perceptron (MLP)

In [ ]:

```
from tensorflow.keras import layers, models, regularizers, callbacks
import tensorflow as tf
import numpy as np

np.random.seed(42)
tf.random.set_seed(42)

n_classes = 5

model = models.Sequential([
    layers.Dense(256, activation='relu',
                 input_shape=(scaled_train_df.shape[1]-1,),
                 kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(128, activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(n_classes, activation='softmax')
])

optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

# Compile the model
model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
# Add Early Stopping to prevent overfitting
early_stopping = callbacks.EarlyStopping(monitor='val_loss', patience=10,

# Train the model
history = model.fit(
    scaled_train_df.drop(columns=['Export_Value_3_Years_Ahead']),
    scaled_train_df['Export_Value_3_Years_Ahead_Category'],
    epochs=100,
    batch_size=64,
    validation_data=(scaled_val_df.drop(columns=['Export_Value_3_Years_Ah
                     scaled_val_df['Export_Value_3_Years_Ahead_Category']
    verbose=1,
    callbacks=[early_stopping]
)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(
    scaled_test_df.drop(columns=['Export_Value_3_Years_Ahead']),
    scaled_test_df['Export_Value_3_Years_Ahead_Category'],
    verbose=0
)
print(f'Test Loss: {test_loss}, Test Accuracy: {test_accuracy}')
```

```
Epoch 1/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.6659 - loss: 1.0905
- val_accuracy: 0.9874 - val_loss: 0.2720
Epoch 2/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9644 - loss: 0.3212
- val_accuracy: 0.9930 - val_loss: 0.1781
Epoch 3/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9829 - loss: 0.2072
- val_accuracy: 0.9965 - val_loss: 0.1259
Epoch 4/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9901 - loss: 0.1450
- val_accuracy: 0.9968 - val_loss: 0.0973
Epoch 5/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9915 - loss: 0.1132
- val_accuracy: 0.9983 - val_loss: 0.0768
Epoch 6/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9931 - loss: 0.0941
- val_accuracy: 0.9986 - val_loss: 0.0638
Epoch 7/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9952 - loss: 0.0775
- val_accuracy: 0.9991 - val_loss: 0.0543
Epoch 8/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9964 - loss: 0.0692
- val_accuracy: 0.9988 - val_loss: 0.0552
Epoch 9/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9965 - loss: 0.0641
- val_accuracy: 0.9988 - val_loss: 0.0488
Epoch 10/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9958 - loss: 0.0636
- val_accuracy: 0.9992 - val_loss: 0.0475
Epoch 11/100
```

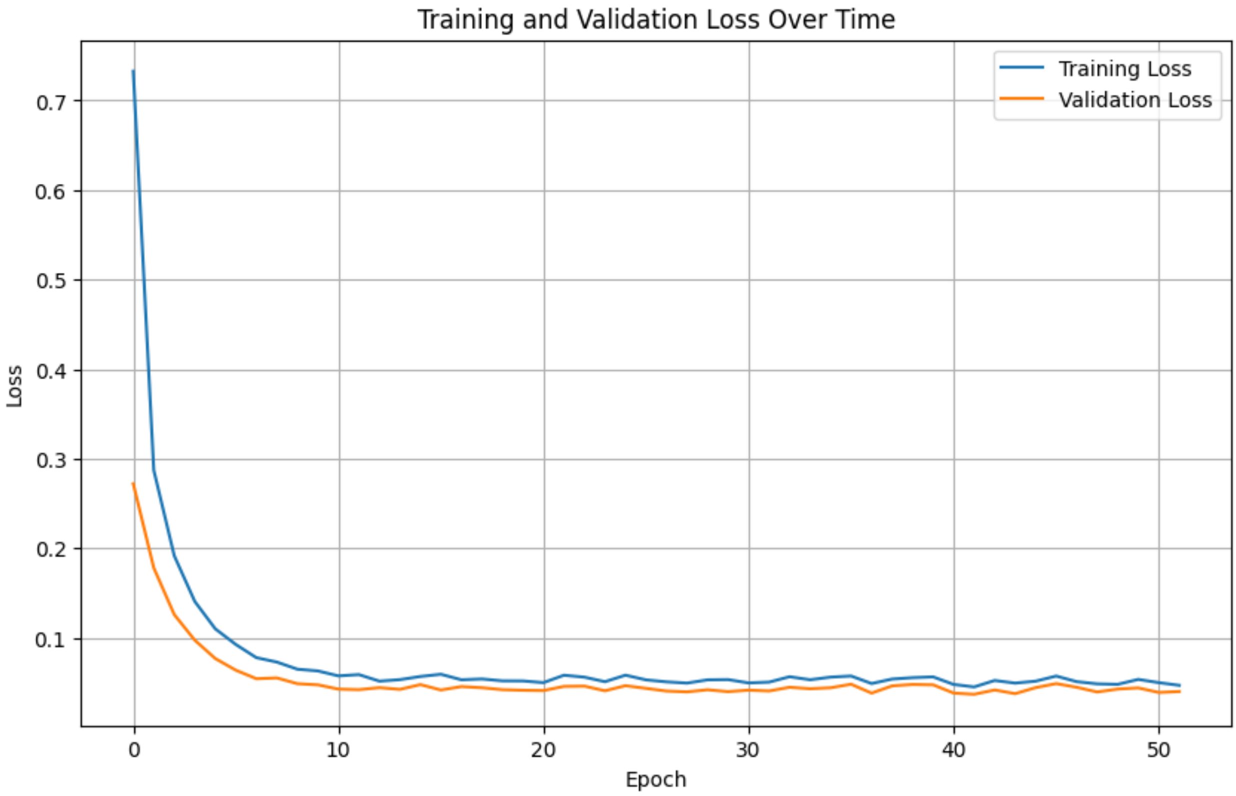
```
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9970 - loss: 0.0573
- val_accuracy: 0.9989 - val_loss: 0.0427
Epoch 12/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9961 - loss: 0.0594
- val_accuracy: 0.9996 - val_loss: 0.0421
Epoch 13/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9974 - loss: 0.0503
- val_accuracy: 0.9992 - val_loss: 0.0443
Epoch 14/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9964 - loss: 0.0518
- val_accuracy: 0.9994 - val_loss: 0.0424
Epoch 15/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9968 - loss: 0.0543
- val_accuracy: 0.9994 - val_loss: 0.0477
Epoch 16/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9958 - loss: 0.0584
- val_accuracy: 0.9994 - val_loss: 0.0417
Epoch 17/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9973 - loss: 0.0492
- val_accuracy: 0.9989 - val_loss: 0.0455
Epoch 18/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9961 - loss: 0.0558
- val_accuracy: 0.9998 - val_loss: 0.0443
Epoch 19/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9963 - loss: 0.0521
- val_accuracy: 0.9997 - val_loss: 0.0421
Epoch 20/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9965 - loss: 0.0498
- val_accuracy: 0.9998 - val_loss: 0.0414
Epoch 21/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9969 - loss: 0.0500
- val_accuracy: 0.9993 - val_loss: 0.0411
Epoch 22/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9959 - loss: 0.0561
- val_accuracy: 0.9995 - val_loss: 0.0457
Epoch 23/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9968 - loss: 0.0533
- val_accuracy: 0.9995 - val_loss: 0.0460
Epoch 24/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9973 - loss: 0.0506
- val_accuracy: 0.9992 - val_loss: 0.0409
Epoch 25/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9963 - loss: 0.0531
- val_accuracy: 0.9987 - val_loss: 0.0465
Epoch 26/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9971 - loss: 0.0534
- val_accuracy: 0.9988 - val_loss: 0.0435
Epoch 27/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9977 - loss: 0.0477
- val_accuracy: 0.9997 - val_loss: 0.0405
Epoch 28/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9972 - loss: 0.0487
- val_accuracy: 0.9992 - val_loss: 0.0396
```

```
Epoch 29/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9971 - loss: 0.0481
- val_accuracy: 0.9997 - val_loss: 0.0420
Epoch 30/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9965 - loss: 0.0543
- val_accuracy: 0.9993 - val_loss: 0.0399
Epoch 31/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9978 - loss: 0.0458
- val_accuracy: 0.9993 - val_loss: 0.0416
Epoch 32/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9979 - loss: 0.0462
- val_accuracy: 0.9995 - val_loss: 0.0406
Epoch 33/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9970 - loss: 0.0524
- val_accuracy: 0.9987 - val_loss: 0.0448
Epoch 34/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9967 - loss: 0.0560
- val_accuracy: 0.9993 - val_loss: 0.0431
Epoch 35/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9962 - loss: 0.0534
- val_accuracy: 0.9994 - val_loss: 0.0442
Epoch 36/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9961 - loss: 0.0590
- val_accuracy: 0.9981 - val_loss: 0.0481
Epoch 37/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9976 - loss: 0.0506
- val_accuracy: 0.9996 - val_loss: 0.0382
Epoch 38/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9968 - loss: 0.0486
- val_accuracy: 0.9989 - val_loss: 0.0463
Epoch 39/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9971 - loss: 0.0528
- val_accuracy: 0.9994 - val_loss: 0.0479
Epoch 40/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9965 - loss: 0.0536
- val_accuracy: 0.9991 - val_loss: 0.0475
Epoch 41/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9975 - loss: 0.0476
- val_accuracy: 0.9996 - val_loss: 0.0383
Epoch 42/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9984 - loss: 0.0411
- val_accuracy: 0.9998 - val_loss: 0.0369
Epoch 43/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9961 - loss: 0.0500
- val_accuracy: 0.9987 - val_loss: 0.0417
Epoch 44/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9962 - loss: 0.0509
- val_accuracy: 0.9997 - val_loss: 0.0376
Epoch 45/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9962 - loss: 0.0501
- val_accuracy: 0.9976 - val_loss: 0.0444
Epoch 46/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9958 - loss: 0.0524
```

```
- val_accuracy: 0.9997 - val_loss: 0.0488
Epoch 47/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9972 - loss: 0.0510
- val_accuracy: 0.9976 - val_loss: 0.0447
Epoch 48/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9967 - loss: 0.0486
- val_accuracy: 0.9997 - val_loss: 0.0395
Epoch 49/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9975 - loss: 0.0457
- val_accuracy: 0.9996 - val_loss: 0.0428
Epoch 50/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9971 - loss: 0.0480
- val_accuracy: 0.9997 - val_loss: 0.0440
Epoch 51/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9967 - loss: 0.0508
- val_accuracy: 0.9994 - val_loss: 0.0390
Epoch 52/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9973 - loss: 0.0458
- val_accuracy: 0.9984 - val_loss: 0.0400
Test Loss: 0.04100680723786354, Test Accuracy: 0.9995312094688416
```

In [ ]:

```python
plt.figure(figsize=(10, 6))

# Plot training and validation loss values
plt.plot(history.history['loss'], label='Training Loss')

if 'val_loss' in history.history:
    plt.plot(history.history['val_loss'], label='Validation Loss')

plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss Over Time')
plt.legend()
plt.grid(True)
plt.show()
```



Section 5: Performance of the model

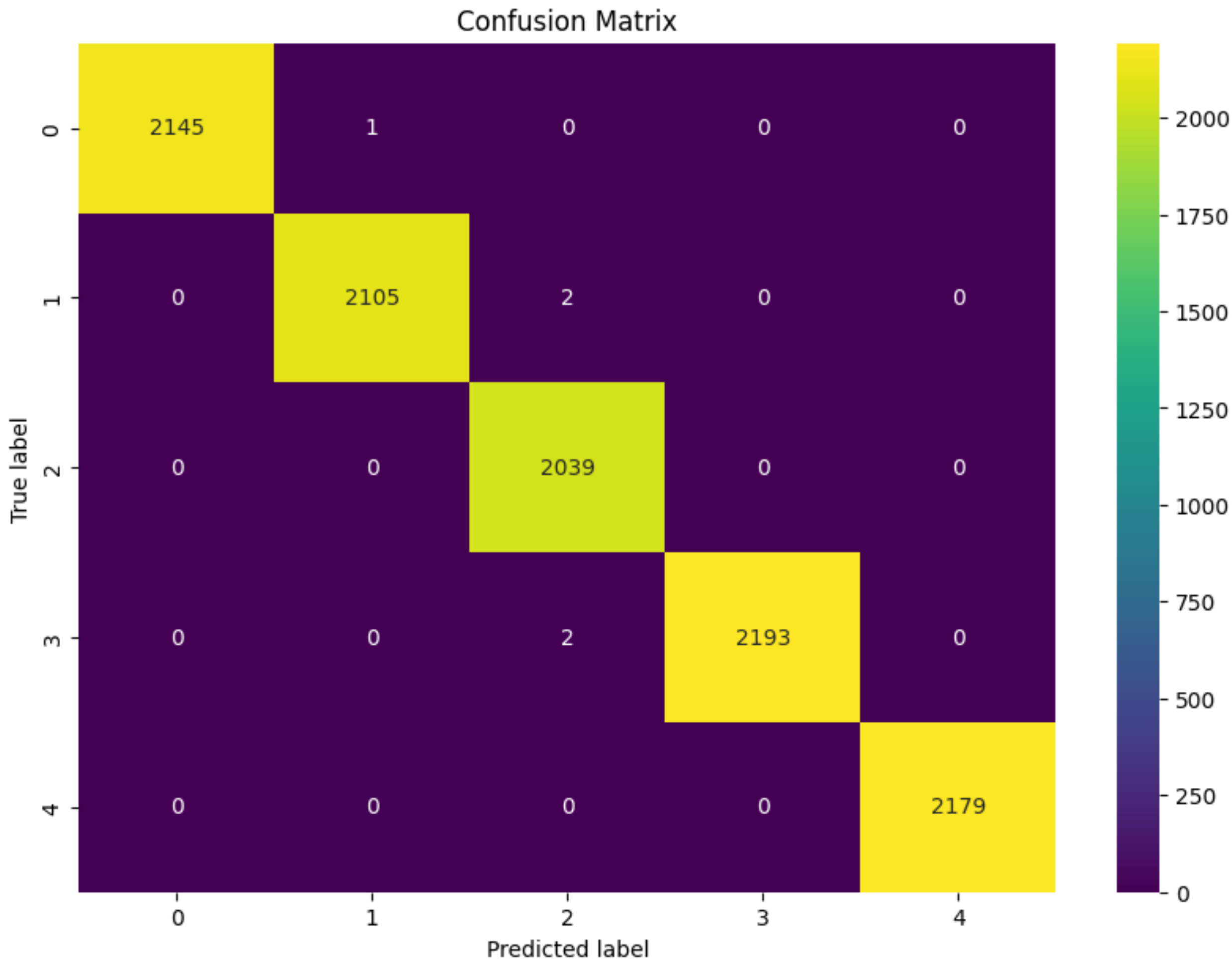
In [ ]:

```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report, confus

# Generate predictions on the test data
predictions = model.predict(scaled_test_df.drop(columns=['Export_Value_3_
predicted_classes = np.argmax(predictions, axis=1)

true_labels = scaled_test_df['Export_Value_3_Years_Ahead_Category'].value

# Calculate accuracy
accuracy = accuracy_score(true_labels, predicted_classes)
conf_matrix = confusion_matrix(true_labels, predicted_classes)
class_report = classification_report(true_labels, predicted_classes)


# Print the classification metrics
print(f'Accuracy: {accuracy}')
print('Classification Report:\n', class_report)
```

```
334/334 ━━━━━━━━━━━━━━━━━━━━ 0s 699us/step
Accuracy: 0.9995312207012939
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      2146
         1.0       1.00      1.00      1.00      2107
         2.0       1.00      1.00      1.00      2039
         3.0       1.00      1.00      1.00      2195
         4.0       1.00      1.00      1.00      2179

    accuracy                           1.00     10666
   macro avg       1.00      1.00      1.00     10666
weighted avg       1.00      1.00      1.00     10666
```

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Calculate the confusion matrix
cm = confusion_matrix(true_labels, predicted_classes)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='g', cmap='viridis')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix')
plt.show()
```



In [ ]:

```
# Save the results to a CSV file
results_df = pd.DataFrame({
    'Id': range(1, len(predicted_classes) + 1),
    'True_Label': true_labels,
    'Predicted_Class': predicted_classes,
    'Export_Value_3_Years_Ahead': test_df['Export_Value_3_Years_Ahead'].v
})

results_df.to_csv('classification_results.csv', index=False)
```