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_2-22-2024.csv')

# 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', 'Year'], how='inner')

# 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', 'Item', 'Year', 'Export_Value']],
                           left_on=['Area', 'Item', 'Year_3_Ahead'], right_on=['Area', 'Item', 'Year'],
                           suffixes=('', '_3_Years_Ahead'), how='left')

# Sort by 'Area', 'Item', 'Year' to ensure chronological order for lag and rolling calculations
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(window=3).mean()

    return group

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

# 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, inplace=True)

# One-hot encode the 'Item' columns but keep the original column
item_columns = df_trade_values['Item']
df_trade_values = pd.get_dummies(df_trade_values, columns=['Item'])
df_trade_values['Item'] = item_columns
```

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.csv')

# 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_pesticides_pivot.columns.values]
df_pesticides_pivot = df_pesticides_pivot.rename(columns={'Area_': 'Area', 'Year_': 'Year'})

# Rename the columns
df_pesticides_pivot = df_pesticides_pivot.rename(columns=lambda x: f'Pesticide_{x}' if x not in ['Area', 'Year'] else x)

PrettyPandas(df_pesticides_pivot.head())
```

Out[ ]:

	Area	Year	Pesticide_Fungicides and Bactericides_Agricultural Use	Pesticide_Fungicides – Seed treatments_Agricultural Use	Pesticide_Herbicides_Agricultural Use	Pesticide_Insecticides_Agricultural Use	Pesticide_Insecticides – Seed Treatments_Agricultural Use	Pesticide_Pesticides (total)_Agricultural Use
0	Albania	2000	105.730000	0.050000	7.990000	169.600000	9.010000	307.980000
1	Albania	2001	108.080000	0.060000	7.990000	174.520000	10.810000	319.380000
2	Albania	2002	110.430000	0.070000	7.980000	179.440000	12.610000	330.780000
3	Albania	2003	112.770000	0.080000	7.980000	184.360000	14.410000	342.170000
4	Albania	2004	115.120000	0.090000	7.980000	189.280000	16.210000	353.570000

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', low_memory=False)

# 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', 'Land area', 'Agricultural land'])]

# 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' as a column
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 in df_land_use_pivot.columns[2:]]

PrettyPandas(df_land_use_pivot.head())
```

Out[ ]:

	Area	Year	LandUse_Agriculture	LandUse_Agriculture area actually irrigated	LandUse_Arable land	LandUse_Cropland	LandUse_Cropland area actually irrigated	LandUse_Farm buildings and Farmyards	LandUse_Forestry area actually irrigated	LandUse_Land area actually irrigated	LandUse_L area equip for irrigat
0	Afghanistan	1980	38049.000000	nan	7910.000000	8049.000000	nan	nan	nan	nan	2505.000
1	Afghanistan	1981	38053.000000	nan	7910.000000	8053.000000	nan	nan	nan	nan	2520.000
2	Afghanistan	1982	38054.000000	nan	7910.000000	8054.000000	nan	nan	nan	nan	2535.000
3	Afghanistan	1983	38054.000000	nan	7910.000000	8054.000000	nan	nan	nan	nan	2550.000
4	Afghanistan	1984	38054.000000	nan	7910.000000	8054.000000	nan	nan	nan	nan	2581.000

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_2-27-2024.csv')

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

# 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'}, inplace=True)

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_2-27-2024.csv')

# 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 inflows to Agriculture, Forestry and Fishing')]

# 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(" ", "_")}' if x not in ['Area', 'Year'] else x, inplace=True)
```

```
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_data_en_2-22-2024.csv')

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

# Filter for relevant 'Item' categories based on the focus of the analysis
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 int$ per capita)',
    'Percent of arable land equipped for irrigation (percent) (3-year average)'
]

# Drop irrelevant items
df_food_security_relevant = df_food_security[~df_food_security['Item'].isin(irrelevant_items)].copy()

# 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., '2000-2002')
df_food_security_relevant['Year'] = df_food_security_relevant['Year'].astype(str)

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

# 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 yearly data
df_combined = pd.concat([df_yearly, df_expanded]).drop_duplicates(subset=['Area', 'Year', 'Item'], keep='first')

# 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(", ", "_").replace(" ", "_").replace("(", "").replace(")", "").replace("-", "_")}'
                                                    for c in df_fsi_combined_pivot.columns[2:]]

PrettyPandas(df_fsi_combined_pivot.head())
```

Out[ ]:

	Area	Year	FSI_Average_dietary_energy_supply_adequacy_percent_3_year_average	FSI_Average_protein_supply_g/cap/day_3_year_average	FSI_Cereal_import_dependency_ratio_
0	Afghanistan	2000	88.000000	51.400000	
1	Afghanistan	2001	88.000000	51.400000	
2	Afghanistan	2002	88.000000	51.400000	
3	Afghanistan	2003	89.000000	52.100000	
4	Afghanistan	2004	92.000000	54.000000	

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_en_2-22-2024.csv')

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

# 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, Seafood
df_food_balances_relevant = df_food_balances_relevant.loc[~df_food_balances_relevant['Item'].isin(['Meat', 'Eggs', 'Milk - Excluding Butter', 'Fish, Seafood'])]

# only keep elements 'export quantity'
df_food_balances_relevant = df_food_balances_relevant.loc[df_food_balances_relevant['Element'] == 'Export Quantity']

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

# 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(" ", "_").replace(",", "_").replace("-", "_").replace("/", "_")}'
                              if x not in ['Area', 'Year'] else x, inplace=True)

PrettyPandas(df_food_balances_pivot.head())
```

Out[ ]:

Element_Item	Area	Year	FoodBalance_Export_Quantity_Alcoholic_Beverages	FoodBalance_Export_Quantity_Cereals___Excluding_Beer	FoodBalance_Export_Quantity_Fruits___Excl
0	Afghanistan	2010	nan	0.000000	
1	Afghanistan	2011	nan	0.000000	
2	Afghanistan	2012	nan	0.000000	
3	Afghanistan	2013	nan	0.000000	
4	Afghanistan	2014	0.000000	2.000000	

Preprocess Fertilizers use - FAOSTAT_data_en_2-27-2024.csv

In [ ]:

```
# Load the dataset
df_fertilizers = pd.read_csv('./Fertilizers use - FAOSTAT_data_en_2-27-2024.csv')

# 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(" ", "_").replace(",", "_").replace("-", "_").replace("(", "").replace(")", "")}'
                            if x not in ['Area', 'Year'] else x, inplace=True)

PrettyPandas(df_fertilizers_pivot.head())
```

Out[ ]:

Item	Area	Year	FertilizerUse_Ammonia__anhydrous	FertilizerUse_Ammonium_nitrate_AN	FertilizerUse_Ammonium_sulphate	FertilizerUse_Calcium_ammonium_nitrate_CAN_and_
0	Afghanistan	2002	nan	nan	nan	
1	Afghanistan	2003	nan	nan	nan	
2	Afghanistan	2004	nan	nan	nan	
3	Afghanistan	2005	nan	nan	nan	
4	Afghanistan	2006	nan	nan	nan	

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_2-22-2024.csv')

# 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'])['Value'].mean().reset_index()

# Rename the column
df_yearly_exchange_rates.rename(columns={'Value': 'Average_Exchange_Rate'}, inplace=True)

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_emissions_pivot.columns.values]

# Rename columns
df_emissions_pivot.columns = ['Area', 'Year'] + [f'Emission_{c.replace(" ", "_").replace("(", "").replace(")", "").replace("-", "_")}'
                                                 for c in df_emissions_pivot.columns[2:]]

PrettyPandas(df_emissions_pivot.head())
```

Out[ ]:

	Area	Year	Emission_Crops_total_Emissions_CH4_All_Crops	Emission_Crops_total_Emissions_N2O_All_Crops	Emission_Emissions_CO2_Cropland_organic_soils	Emission_Emiss
0	Afghanistan	2000	20.847100	0.705600	0.000000	
1	Afghanistan	2001	19.260500	0.705400	0.000000	
2	Afghanistan	2002	21.255300	1.065600	0.000000	
3	Afghanistan	2003	23.701700	1.311700	0.000000	
4	Afghanistan	2004	30.308900	1.085600	0.000000	

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_2-22-2024.csv')

# 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'], columns='Item', values='Value').reset_index()

# Rename the columns
df_crops_pivoted = df_crops_pivoted.rename(columns=lambda x: 'CropYield_' + x.replace(', ', '_').replace(' ', '_').replace('(', '').replace(')', '')
                                        if x not in ['Area', 'Year'] else x)

PrettyPandas(df_crops_pivoted.head())
```

Out[ ]:

Item	Area	Year	CropYield_Cereals_primary	CropYield_Citrus_Fruit_Total	CropYield_Fibre_Crops_Fibre_Equivalent	CropYield_Fruit_Primary	CropYield_Oilcrops_Cake_Equivalent
0	Afghanistan	2000	8063.000000	71245.000000	3990.000000	76730.000000	3833.000000
1	Afghanistan	2001	10067.000000	71417.000000	3990.000000	80268.000000	3829.000000
2	Afghanistan	2002	16698.000000	71477.000000	3990.000000	80174.000000	3818.000000
3	Afghanistan	2003	14580.000000	73423.000000	3850.000000	82792.000000	3844.000000
4	Afghanistan	2004	13348.000000	78025.000000	3843.000000	79157.000000	3951.000000

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_data_en_2-22-2024.csv')

# 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 annual value
).reset_index()

# Rename the columns
df_consumer_prices_pivot.rename(columns={
    'Consumer Prices, Food Indices (2015 = 100)': 'ConsumerPrice_Food_Indices',
    '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_Inflation
0	Afghanistan	2000	26.629848	nan
1	Afghanistan	2001	29.893548	12.780692
2	Afghanistan	2002	35.344892	18.254516
3	Afghanistan	2003	40.203113	14.102244
4	Afghanistan	2004	45.840561	14.072172

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, forestry and fishing': 'Employment_Agriculture_Work_Hours_Per_Week',
    'Employment in agriculture, forestry and fishing - ILO modelled estimates': 'Employment_Agriculture_Estimates'
}, inplace=True)

PrettyPandas(df_employment_pivot.head())
```

Out[ ]:

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

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

```python
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(str)
df_food_balances_pivot['Year'] = df_food_balances_pivot['Year'].astype(str)

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'].astype(str)
df_yearly_exchange_rates['Year'] = df_yearly_exchange_rates['Year'].astype(str)

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'].astype(str)
df_consumer_prices_pivot['Year'] = df_consumer_prices_pivot['Year'].astype(str)

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'], how='left')
df_merged = pd.merge(df_merged, df_temperature_annual, on=['Area', 'Year'], how='left')
df_merged = pd.merge(df_merged, df_fdi_pivot, on=['Area', 'Year'], how='left')
df_merged = pd.merge(df_merged, df_fsi_combined_pivot, on=['Area', 'Year'], how='left')
df_merged = pd.merge(df_merged, df_food_balances_pivot, on=['Area', 'Year'], how='left')
df_merged = pd.merge(df_merged, df_fertilizers_pivot, on=['Area', 'Year'], how='left')
df_merged = pd.merge(df_merged, df_yearly_exchange_rates, on=['Area', 'Year'], how='left')
df_merged = pd.merge(df_merged, df_emissions_pivot, on=['Area', 'Year'], how='left')
df_merged = pd.merge(df_merged, df_crops_pivoted, on=['Area', 'Year'], how='left')
df_merged = pd.merge(df_merged, df_consumer_prices_pivot, on=['Area', 'Year'], how='left')
df_merged = pd.merge(df_merged, df_employment_pivot, on=['Area', 'Year'], how='left')
```

1.3: Recoding labels into classes

In [ ]:
```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder

# Calculate percentiles for each item type
def calculate_percentiles(df):
    low = np.percentile(df['Export_Value_3_Years_Ahead'], 33)
    medium = np.percentile(df['Export_Value_3_Years_Ahead'], 66)
    return {
        'low': low,
        'medium': medium,
    }

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

# 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['low']:
        return 0 # Low
    elif value <= thresholds['medium'] and value > thresholds['low']:
        return 1 # Medium
    else:
        return 2 # High

df_merged['Export_Value_3_Years_Ahead_Category'] = df_merged.apply(categorize_value, axis=1)

# 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: 3
```

In [ ]:
```python
# Dropping the columns
df_merged = df_merged.drop(columns=['Area'])
df_merged = df_merged.drop(columns=['Year'])
df_merged = df_merged.drop(columns=['Export_Value_3_Years_Ahead'])

# Dropping duplicate rows
df_merged = df_merged.drop_duplicates()
```

Section 2: Selecting training, validation, and test sets

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

def split_data(df, test_size=0.2, validation_size=0.25):
    df_train, df_temp = train_test_split(df, test_size=test_size, random_state=42)
    df_validation, df_test = train_test_split(df_temp, test_size=validation_size, random_state=42)
    return df_train, df_validation, df_test

grouped = df_merged.groupby('Item')
train_list = []
validation_list = []
test_list = []

for name, group in grouped:
    train, validation, test = split_data(group)
```

```
    train_list.append(train)
    validation_list.append(validation)
    test_list.append(test)

train_df = pd.concat(train_list)
val_df = pd.concat(validation_list)
test_df = pd.concat(test_list)

train_df = train_df.sample(frac=1, random_state=42).reset_index(drop=True)
val_df = val_df.sample(frac=1, random_state=42).reset_index(drop=True)
test_df = test_df.sample(frac=1, random_state=42).reset_index(drop=True)
```

Section 3: Imputing missing data, select features

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

# Drop Item column
train_df = train_df.drop(columns=['Item'])
val_df = val_df.drop(columns=['Item'])
test_df = test_df.drop(columns=['Item'])

# Drop columns with missing values percentage greater than 50%
missing_values = train_df.isnull().mean()
columns_to_drop = missing_values[missing_values > 0.5].index
train_df.drop(columns=columns_to_drop, inplace=True)
test_df.drop(columns=columns_to_drop, inplace=True)
val_df.drop(columns=columns_to_drop, inplace=True)
```

In [ ]:
```
# Select features using RFE with a Random Forest model
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

#columns start with 'Item'
item_columns = [col for col in train_df.columns if 'Item_' in col]

# Define the model
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Define the RFE
rfe = RFE(model, n_features_to_select=20)

# Fit the RFE
X_train = train_df.drop(columns=['Export_Value_3_Years_Ahead_Category'] + item_columns)
y_train = train_df['Export_Value_3_Years_Ahead_Category']
rfe.fit(X_train, y_train)

# Get the selected features
selected_features = X_train.columns[rfe.support_].tolist()
print(f"Selected features: {selected_features}")

#apply the selected features to the datasets
train_df = train_df[selected_features + ['Export_Value_3_Years_Ahead_Category'] + item_columns]
val_df = val_df[selected_features + ['Export_Value_3_Years_Ahead_Category'] + item_columns]
test_df = test_df[selected_features + ['Export_Value_3_Years_Ahead_Category'] + item_columns]
```

```
Selected features: ['Import_Value', 'Export_Value', 'Export_Value_Lag1', 'Export_Value_Lag2', 'Export_Value_Lag3', 'Export_Value_Rolling_Mean3', 'LandUse_Agricultu
re', 'LandUse_Arable land', 'LandUse_Cropland', 'LandUse_Land area equipped for irrigation', 'LandUse_Permanent crops', 'LandUse_Permanent meadows and pastures',
'FSI_Average_protein_supply_g/cap/day_3_year_average', 'FSI_Value_of_food_imports_in_total_merchandise_exports_percent_3_year_average', 'Average_Exchange_Rate', 'E
mission_Crops_total_Emissions_N2O_All_Crops', 'CropYield_Cereals_primary', 'CropYield_Oilcrops_Cake_Equivalent', 'CropYield_Oilcrops_Oil_Equivalent', 'CropYield_Ve
getables_Primary']
```

In [ ]:
```
# Drop the target column
X_train = train_df.drop(columns=['Export_Value_3_Years_Ahead_Category'])
y_train = train_df['Export_Value_3_Years_Ahead_Category']

# Initialize the KNNImputer
imputer = KNNImputer(n_neighbors=5)

# Fit the imputer on the training data and transform it
train_df_imputed = imputer.fit_transform(train_df)
train_df = pd.DataFrame(train_df_imputed, columns=train_df.columns)

# Transform the test data with the same imputer
test_df_imputed = imputer.transform(test_df)
test_df = pd.DataFrame(test_df_imputed, columns=test_df.columns)

# Transform the validation data
val_df_imputed = imputer.transform(val_df)
val_df = pd.DataFrame(val_df_imputed, columns=val_df.columns)
```

Section 4: 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 != 'Export_Value_3_Years_Ahead_Category']

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

# Apply the scaling
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_scale)
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_Value_3_Years_Ahead_Category'].values
scaled_val_df['Export_Value_3_Years_Ahead_Category'] = val_df['Export_Value_3_Years_Ahead_Category'].values
scaled_test_df['Export_Value_3_Years_Ahead_Category'] = test_df['Export_Value_3_Years_Ahead_Category'].values

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

Out[ ]:

	Import_Value	Export_Value	Export_Value_Lag1	Export_Value_Lag2	Export_Value_Lag3	Export_Value_Rolling_Mean3	LandUse_Agriculture	LandUse_Arable land	LandUse_Cropland	Lan are fo
0	-0.268980	-0.254013	-0.250791	-0.248111	-0.244865	-0.252297	-0.379929	-0.354770	-0.370088	
1	-0.262246	-0.103120	-0.248912	-0.247546	-0.164031	-0.198819	-0.333500	-0.252903	-0.266535	
2	-0.270896	-0.254207	-0.250961	-0.249323	-0.245923	-0.252809	-0.371971	-0.336618	-0.352881	
3	-0.254280	-0.181176	-0.175844	-0.162202	-0.184523	-0.174209	-0.301782	-0.180668	-0.198914	
4	-0.119183	0.083080	0.134733	0.096710	0.062194	0.105205	-0.202547	-0.065540	0.001861	

Section 5: Building and evaluating a multilayer perceptron (MLP)

In [ ]:

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

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

n_classes = 3

model = models.Sequential([
    layers.Dense(512, activation='relu',
                 kernel_regularizer=regularizers.l2(0.002)),
    layers.Dense(256, activation='relu',
                 kernel_regularizer=regularizers.l2(0.002)),
    layers.Dense(128, activation='relu',
                 kernel_regularizer=regularizers.l2(0.002)),
    layers.Dense(64, activation='relu',
                 kernel_regularizer=regularizers.l2(0.002)),
    layers.Dense(32, activation='relu',
                 kernel_regularizer=regularizers.l2(0.002)),
    layers.Dense(16, activation='relu',
                 kernel_regularizer=regularizers.l2(0.002)),
    layers.Dense(n_classes, activation='softmax')
])

optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)

# 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, restore_best_weights=True)

# Prepare feature and target datasets for training, validation, and testing
X_train = scaled_train_df.drop(columns=['Export_Value_3_Years_Ahead_Category'])
y_train = scaled_train_df['Export_Value_3_Years_Ahead_Category']
X_val = scaled_val_df.drop(columns=[ 'Export_Value_3_Years_Ahead_Category'])
y_val = scaled_val_df['Export_Value_3_Years_Ahead_Category']
X_test = scaled_test_df.drop(columns=['Export_Value_3_Years_Ahead_Category'])
y_test = scaled_test_df['Export_Value_3_Years_Ahead_Category']


class_weights_array = class_weight.compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weights_dict = {i: weight for i, weight in enumerate(class_weights_array)}

print(f"Class Weights: {class_weights_dict}")

# Train the model
history = model.fit(
    X_train,
    y_train,
    epochs=300,
    batch_size=1024,
    validation_data=(X_val, y_val),
    verbose=1,
    callbacks=[early_stopping],
    class_weight=class_weights_dict
)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(
    X_test,
    y_test,
    verbose=0
)
print(f'Test Loss: {test_loss}, Test Accuracy: {test_accuracy}')
```

```
Class Weights: {0: 1.0127976826459624, 1: 1.007201718967675, 2: 0.9805977011494252}
Epoch 1/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.3609 - loss: 2.5027 - val_accuracy: 0.5490 - val_loss: 2.3360
Epoch 2/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5856 - loss: 2.2640 - val_accuracy: 0.6457 - val_loss: 2.0487
Epoch 3/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6456 - loss: 2.0090 - val_accuracy: 0.6805 - val_loss: 1.8847
Epoch 4/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6778 - loss: 1.8592 - val_accuracy: 0.7045 - val_loss: 1.7596
Epoch 5/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7032 - loss: 1.7365 - val_accuracy: 0.7245 - val_loss: 1.6503
Epoch 6/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7273 - loss: 1.6311 - val_accuracy: 0.7410 - val_loss: 1.5588
Epoch 7/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7430 - loss: 1.5433 - val_accuracy: 0.7471 - val_loss: 1.4833
Epoch 8/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7538 - loss: 1.4701 - val_accuracy: 0.7552 - val_loss: 1.4209
Epoch 9/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7637 - loss: 1.4087 - val_accuracy: 0.7639 - val_loss: 1.3665
Epoch 10/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7700 - loss: 1.3556 - val_accuracy: 0.7699 - val_loss: 1.3202
Epoch 11/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7767 - loss: 1.3094 - val_accuracy: 0.7747 - val_loss: 1.2796
Epoch 12/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7818 - loss: 1.2687 - val_accuracy: 0.7793 - val_loss: 1.2435
Epoch 13/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7868 - loss: 1.2321 - val_accuracy: 0.7824 - val_loss: 1.2114
Epoch 14/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7915 - loss: 1.1993 - val_accuracy: 0.7853 - val_loss: 1.1825
Epoch 15/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.7956 - loss: 1.1696 - val_accuracy: 0.7885 - val_loss: 1.1556
Epoch 16/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7993 - loss: 1.1423 - val_accuracy: 0.7906 - val_loss: 1.1313
Epoch 17/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8020 - loss: 1.1174 - val_accuracy: 0.7947 - val_loss: 1.1088
Epoch 18/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8055 - loss: 1.0945 - val_accuracy: 0.7958 - val_loss: 1.0880
Epoch 19/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8082 - loss: 1.0735 - val_accuracy: 0.7979 - val_loss: 1.0691
Epoch 20/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8095 - loss: 1.0543 - val_accuracy: 0.8004 - val_loss: 1.0510
Epoch 21/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8128 - loss: 1.0361 - val_accuracy: 0.8020 - val_loss: 1.0344
Epoch 22/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8150 - loss: 1.0192 - val_accuracy: 0.8061 - val_loss: 1.0183
Epoch 23/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8183 - loss: 1.0034 - val_accuracy: 0.8084 - val_loss: 1.0029
Epoch 24/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8215 - loss: 0.9886 - val_accuracy: 0.8117 - val_loss: 0.9890
Epoch 25/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8227 - loss: 0.9750 - val_accuracy: 0.8140 - val_loss: 0.9750
Epoch 26/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8249 - loss: 0.9618 - val_accuracy: 0.8166 - val_loss: 0.9625
Epoch 27/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8265 - loss: 0.9498 - val_accuracy: 0.8177 - val_loss: 0.9498
Epoch 28/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8287 - loss: 0.9378 - val_accuracy: 0.8214 - val_loss: 0.9383
Epoch 29/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8302 - loss: 0.9269 - val_accuracy: 0.8245 - val_loss: 0.9263
Epoch 30/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8324 - loss: 0.9160 - val_accuracy: 0.8271 - val_loss: 0.9158
Epoch 31/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8353 - loss: 0.9056 - val_accuracy: 0.8300 - val_loss: 0.9061
Epoch 32/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8370 - loss: 0.8955 - val_accuracy: 0.8315 - val_loss: 0.8967
Epoch 33/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8373 - loss: 0.8867 - val_accuracy: 0.8324 - val_loss: 0.8883
Epoch 34/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8391 - loss: 0.8769 - val_accuracy: 0.8341 - val_loss: 0.8801
Epoch 35/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8405 - loss: 0.8681 - val_accuracy: 0.8355 - val_loss: 0.8723
Epoch 36/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8419 - loss: 0.8605 - val_accuracy: 0.8376 - val_loss: 0.8649
Epoch 37/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8428 - loss: 0.8524 - val_accuracy: 0.8370 - val_loss: 0.8576
Epoch 38/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8441 - loss: 0.8450 - val_accuracy: 0.8390 - val_loss: 0.8508
Epoch 39/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8444 - loss: 0.8373 - val_accuracy: 0.8388 - val_loss: 0.8445
Epoch 40/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8454 - loss: 0.8298 - val_accuracy: 0.8405 - val_loss: 0.8379
Epoch 41/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8462 - loss: 0.8233 - val_accuracy: 0.8407 - val_loss: 0.8317
Epoch 42/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8464 - loss: 0.8165 - val_accuracy: 0.8424 - val_loss: 0.8263
Epoch 43/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8484 - loss: 0.8108 - val_accuracy: 0.8428 - val_loss: 0.8205
Epoch 44/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8490 - loss: 0.8044 - val_accuracy: 0.8436 - val_loss: 0.8151
Epoch 45/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8501 - loss: 0.7974 - val_accuracy: 0.8424 - val_loss: 0.8112
Epoch 46/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8505 - loss: 0.7926 - val_accuracy: 0.8428 - val_loss: 0.8064
Epoch 47/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8519 - loss: 0.7867 - val_accuracy: 0.8444 - val_loss: 0.7998
Epoch 48/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8522 - loss: 0.7813 - val_accuracy: 0.8446 - val_loss: 0.7952
Epoch 49/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8522 - loss: 0.7758 - val_accuracy: 0.8449 - val_loss: 0.7906
Epoch 50/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8526 - loss: 0.7703 - val_accuracy: 0.8446 - val_loss: 0.7864
Epoch 51/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8533 - loss: 0.7657 - val_accuracy: 0.8459 - val_loss: 0.7816
Epoch 52/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8538 - loss: 0.7610 - val_accuracy: 0.8455 - val_loss: 0.7781
Epoch 53/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8545 - loss: 0.7567 - val_accuracy: 0.8465 - val_loss: 0.7747
Epoch 54/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8549 - loss: 0.7529 - val_accuracy: 0.8456 - val_loss: 0.7698
Epoch 55/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8567 - loss: 0.7475 - val_accuracy: 0.8471 - val_loss: 0.7668
Epoch 56/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8575 - loss: 0.7436 - val_accuracy: 0.8472 - val_loss: 0.7633
Epoch 57/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8572 - loss: 0.7398 - val_accuracy: 0.8494 - val_loss: 0.7581
Epoch 58/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8581 - loss: 0.7354 - val_accuracy: 0.8475 - val_loss: 0.7554
Epoch 59/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8583 - loss: 0.7318 - val_accuracy: 0.8503 - val_loss: 0.7514
Epoch 60/300
```

```
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8597 - loss: 0.7274 - val_accuracy: 0.8487 - val_loss: 0.7489
Epoch 61/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8592 - loss: 0.7245 - val_accuracy: 0.8494 - val_loss: 0.7461
Epoch 62/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8598 - loss: 0.7220 - val_accuracy: 0.8503 - val_loss: 0.7416
Epoch 63/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8609 - loss: 0.7172 - val_accuracy: 0.8499 - val_loss: 0.7402
Epoch 64/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8604 - loss: 0.7148 - val_accuracy: 0.8515 - val_loss: 0.7359
Epoch 65/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8614 - loss: 0.7107 - val_accuracy: 0.8516 - val_loss: 0.7336
Epoch 66/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8612 - loss: 0.7086 - val_accuracy: 0.8518 - val_loss: 0.7306
Epoch 67/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8615 - loss: 0.7055 - val_accuracy: 0.8525 - val_loss: 0.7282
Epoch 68/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8621 - loss: 0.7023 - val_accuracy: 0.8528 - val_loss: 0.7255
Epoch 69/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8627 - loss: 0.6994 - val_accuracy: 0.8530 - val_loss: 0.7231
Epoch 70/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8630 - loss: 0.6965 - val_accuracy: 0.8526 - val_loss: 0.7206
Epoch 71/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8632 - loss: 0.6937 - val_accuracy: 0.8529 - val_loss: 0.7182
Epoch 72/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8635 - loss: 0.6910 - val_accuracy: 0.8531 - val_loss: 0.7159
Epoch 73/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8639 - loss: 0.6883 - val_accuracy: 0.8530 - val_loss: 0.7137
Epoch 74/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8646 - loss: 0.6857 - val_accuracy: 0.8533 - val_loss: 0.7115
Epoch 75/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8649 - loss: 0.6831 - val_accuracy: 0.8537 - val_loss: 0.7094
Epoch 76/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8652 - loss: 0.6807 - val_accuracy: 0.8540 - val_loss: 0.7072
Epoch 77/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8657 - loss: 0.6782 - val_accuracy: 0.8541 - val_loss: 0.7052
Epoch 78/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8662 - loss: 0.6758 - val_accuracy: 0.8543 - val_loss: 0.7032
Epoch 79/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8663 - loss: 0.6735 - val_accuracy: 0.8547 - val_loss: 0.7012
Epoch 80/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8665 - loss: 0.6712 - val_accuracy: 0.8547 - val_loss: 0.6993
Epoch 81/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8666 - loss: 0.6690 - val_accuracy: 0.8554 - val_loss: 0.6974
Epoch 82/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8672 - loss: 0.6668 - val_accuracy: 0.8556 - val_loss: 0.6955
Epoch 83/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8674 - loss: 0.6647 - val_accuracy: 0.8561 - val_loss: 0.6937
Epoch 84/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8678 - loss: 0.6625 - val_accuracy: 0.8562 - val_loss: 0.6918
Epoch 85/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8680 - loss: 0.6604 - val_accuracy: 0.8562 - val_loss: 0.6900
Epoch 86/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8681 - loss: 0.6584 - val_accuracy: 0.8561 - val_loss: 0.6883
Epoch 87/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8682 - loss: 0.6564 - val_accuracy: 0.8562 - val_loss: 0.6866
Epoch 88/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8687 - loss: 0.6544 - val_accuracy: 0.8566 - val_loss: 0.6850
Epoch 89/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8690 - loss: 0.6525 - val_accuracy: 0.8566 - val_loss: 0.6833
Epoch 90/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8690 - loss: 0.6506 - val_accuracy: 0.8574 - val_loss: 0.6817
Epoch 91/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8691 - loss: 0.6487 - val_accuracy: 0.8574 - val_loss: 0.6802
Epoch 92/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8692 - loss: 0.6469 - val_accuracy: 0.8576 - val_loss: 0.6787
Epoch 93/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8693 - loss: 0.6451 - val_accuracy: 0.8580 - val_loss: 0.6772
Epoch 94/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8695 - loss: 0.6434 - val_accuracy: 0.8583 - val_loss: 0.6757
Epoch 95/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8698 - loss: 0.6416 - val_accuracy: 0.8586 - val_loss: 0.6743
Epoch 96/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8703 - loss: 0.6399 - val_accuracy: 0.8590 - val_loss: 0.6728
Epoch 97/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8705 - loss: 0.6383 - val_accuracy: 0.8593 - val_loss: 0.6714
Epoch 98/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8708 - loss: 0.6366 - val_accuracy: 0.8596 - val_loss: 0.6700
Epoch 99/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8712 - loss: 0.6349 - val_accuracy: 0.8599 - val_loss: 0.6687
Epoch 100/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8717 - loss: 0.6334 - val_accuracy: 0.8599 - val_loss: 0.6673
Epoch 101/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8720 - loss: 0.6318 - val_accuracy: 0.8597 - val_loss: 0.6660
Epoch 102/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8725 - loss: 0.6302 - val_accuracy: 0.8596 - val_loss: 0.6648
Epoch 103/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8725 - loss: 0.6286 - val_accuracy: 0.8596 - val_loss: 0.6635
Epoch 104/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8727 - loss: 0.6271 - val_accuracy: 0.8595 - val_loss: 0.6623
Epoch 105/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8728 - loss: 0.6256 - val_accuracy: 0.8593 - val_loss: 0.6610
Epoch 106/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8731 - loss: 0.6242 - val_accuracy: 0.8595 - val_loss: 0.6598
Epoch 107/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8732 - loss: 0.6227 - val_accuracy: 0.8595 - val_loss: 0.6586
Epoch 108/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8734 - loss: 0.6214 - val_accuracy: 0.8597 - val_loss: 0.6574
Epoch 109/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8736 - loss: 0.6199 - val_accuracy: 0.8601 - val_loss: 0.6562
Epoch 110/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8737 - loss: 0.6185 - val_accuracy: 0.8602 - val_loss: 0.6550
Epoch 111/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8740 - loss: 0.6171 - val_accuracy: 0.8605 - val_loss: 0.6539
Epoch 112/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8743 - loss: 0.6158 - val_accuracy: 0.8608 - val_loss: 0.6528
Epoch 113/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8746 - loss: 0.6144 - val_accuracy: 0.8609 - val_loss: 0.6517
Epoch 114/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8748 - loss: 0.6131 - val_accuracy: 0.8612 - val_loss: 0.6505
Epoch 115/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8750 - loss: 0.6118 - val_accuracy: 0.8612 - val_loss: 0.6494
Epoch 116/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8753 - loss: 0.6105 - val_accuracy: 0.8615 - val_loss: 0.6483
Epoch 117/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8754 - loss: 0.6092 - val_accuracy: 0.8618 - val_loss: 0.6472
Epoch 118/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8755 - loss: 0.6080 - val_accuracy: 0.8619 - val_loss: 0.6462
Epoch 119/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.8758 - loss: 0.6066 - val_accuracy: 0.8619 - val_loss: 0.6451
Epoch 120/300
```

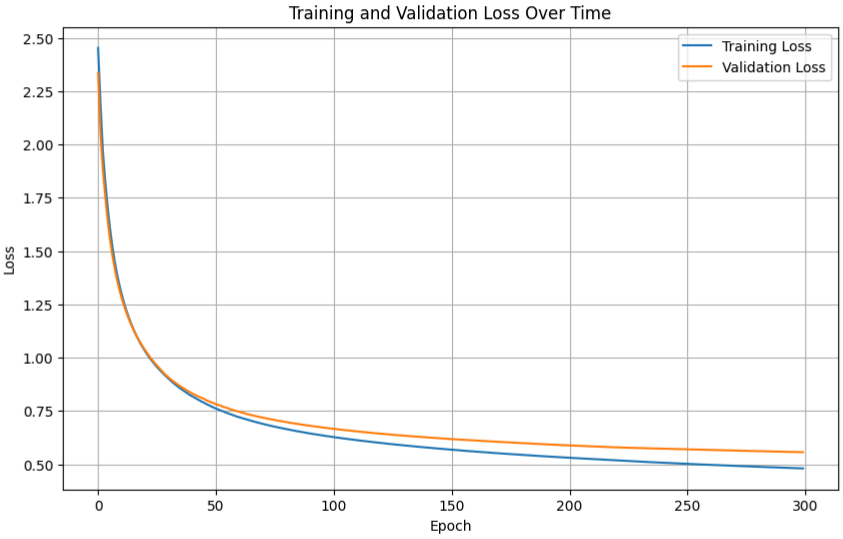
```
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8758 - loss: 0.6054 - val_accuracy: 0.8619 - val_loss: 0.6441
Epoch 121/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8758 - loss: 0.6042 - val_accuracy: 0.8621 - val_loss: 0.6431
Epoch 122/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8759 - loss: 0.6029 - val_accuracy: 0.8622 - val_loss: 0.6422
Epoch 123/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8762 - loss: 0.6017 - val_accuracy: 0.8619 - val_loss: 0.6412
Epoch 124/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8764 - loss: 0.6005 - val_accuracy: 0.8618 - val_loss: 0.6402
Epoch 125/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8764 - loss: 0.5994 - val_accuracy: 0.8618 - val_loss: 0.6393
Epoch 126/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8766 - loss: 0.5982 - val_accuracy: 0.8619 - val_loss: 0.6384
Epoch 127/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8768 - loss: 0.5971 - val_accuracy: 0.8618 - val_loss: 0.6375
Epoch 128/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8769 - loss: 0.5960 - val_accuracy: 0.8616 - val_loss: 0.6365
Epoch 129/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8771 - loss: 0.5948 - val_accuracy: 0.8620 - val_loss: 0.6356
Epoch 130/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8772 - loss: 0.5937 - val_accuracy: 0.8620 - val_loss: 0.6347
Epoch 131/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8772 - loss: 0.5926 - val_accuracy: 0.8620 - val_loss: 0.6338
Epoch 132/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8773 - loss: 0.5915 - val_accuracy: 0.8622 - val_loss: 0.6329
Epoch 133/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8776 - loss: 0.5905 - val_accuracy: 0.8624 - val_loss: 0.6320
Epoch 134/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8777 - loss: 0.5895 - val_accuracy: 0.8621 - val_loss: 0.6311
Epoch 135/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8777 - loss: 0.5884 - val_accuracy: 0.8624 - val_loss: 0.6304
Epoch 136/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8778 - loss: 0.5874 - val_accuracy: 0.8620 - val_loss: 0.6295
Epoch 137/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8778 - loss: 0.5863 - val_accuracy: 0.8624 - val_loss: 0.6287
Epoch 138/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8779 - loss: 0.5853 - val_accuracy: 0.8622 - val_loss: 0.6279
Epoch 139/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8779 - loss: 0.5842 - val_accuracy: 0.8621 - val_loss: 0.6270
Epoch 140/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8780 - loss: 0.5832 - val_accuracy: 0.8616 - val_loss: 0.6262
Epoch 141/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8783 - loss: 0.5822 - val_accuracy: 0.8618 - val_loss: 0.6255
Epoch 142/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8783 - loss: 0.5813 - val_accuracy: 0.8619 - val_loss: 0.6247
Epoch 143/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8784 - loss: 0.5803 - val_accuracy: 0.8621 - val_loss: 0.6238
Epoch 144/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8785 - loss: 0.5793 - val_accuracy: 0.8622 - val_loss: 0.6230
Epoch 145/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8788 - loss: 0.5784 - val_accuracy: 0.8622 - val_loss: 0.6223
Epoch 146/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8789 - loss: 0.5774 - val_accuracy: 0.8622 - val_loss: 0.6216
Epoch 147/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8790 - loss: 0.5765 - val_accuracy: 0.8620 - val_loss: 0.6208
Epoch 148/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8792 - loss: 0.5755 - val_accuracy: 0.8619 - val_loss: 0.6201
Epoch 149/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8797 - loss: 0.5745 - val_accuracy: 0.8622 - val_loss: 0.6194
Epoch 150/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8796 - loss: 0.5736 - val_accuracy: 0.8624 - val_loss: 0.6186
Epoch 151/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8799 - loss: 0.5727 - val_accuracy: 0.8624 - val_loss: 0.6179
Epoch 152/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8799 - loss: 0.5718 - val_accuracy: 0.8625 - val_loss: 0.6172
Epoch 153/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8800 - loss: 0.5708 - val_accuracy: 0.8626 - val_loss: 0.6165
Epoch 154/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8803 - loss: 0.5699 - val_accuracy: 0.8631 - val_loss: 0.6158
Epoch 155/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8801 - loss: 0.5690 - val_accuracy: 0.8627 - val_loss: 0.6152
Epoch 156/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8801 - loss: 0.5681 - val_accuracy: 0.8634 - val_loss: 0.6146
Epoch 157/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8802 - loss: 0.5672 - val_accuracy: 0.8631 - val_loss: 0.6139
Epoch 158/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8801 - loss: 0.5664 - val_accuracy: 0.8634 - val_loss: 0.6133
Epoch 159/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8802 - loss: 0.5655 - val_accuracy: 0.8631 - val_loss: 0.6127
Epoch 160/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8802 - loss: 0.5647 - val_accuracy: 0.8634 - val_loss: 0.6120
Epoch 161/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8802 - loss: 0.5639 - val_accuracy: 0.8634 - val_loss: 0.6114
Epoch 162/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8801 - loss: 0.5630 - val_accuracy: 0.8635 - val_loss: 0.6107
Epoch 163/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8803 - loss: 0.5622 - val_accuracy: 0.8641 - val_loss: 0.6101
Epoch 164/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8804 - loss: 0.5614 - val_accuracy: 0.8641 - val_loss: 0.6095
Epoch 165/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8806 - loss: 0.5606 - val_accuracy: 0.8639 - val_loss: 0.6089
Epoch 166/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8806 - loss: 0.5598 - val_accuracy: 0.8640 - val_loss: 0.6082
Epoch 167/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8808 - loss: 0.5590 - val_accuracy: 0.8637 - val_loss: 0.6076
Epoch 168/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8809 - loss: 0.5582 - val_accuracy: 0.8640 - val_loss: 0.6070
Epoch 169/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8809 - loss: 0.5574 - val_accuracy: 0.8637 - val_loss: 0.6063
Epoch 170/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8808 - loss: 0.5566 - val_accuracy: 0.8639 - val_loss: 0.6057
Epoch 171/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8808 - loss: 0.5559 - val_accuracy: 0.8639 - val_loss: 0.6051
Epoch 172/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8811 - loss: 0.5551 - val_accuracy: 0.8639 - val_loss: 0.6044
Epoch 173/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8813 - loss: 0.5543 - val_accuracy: 0.8639 - val_loss: 0.6038
Epoch 174/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8814 - loss: 0.5536 - val_accuracy: 0.8639 - val_loss: 0.6032
Epoch 175/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8814 - loss: 0.5528 - val_accuracy: 0.8641 - val_loss: 0.6026
Epoch 176/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8815 - loss: 0.5520 - val_accuracy: 0.8646 - val_loss: 0.6019
Epoch 177/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8816 - loss: 0.5513 - val_accuracy: 0.8643 - val_loss: 0.6014
Epoch 178/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8817 - loss: 0.5505 - val_accuracy: 0.8644 - val_loss: 0.6008
Epoch 179/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8819 - loss: 0.5498 - val_accuracy: 0.8645 - val_loss: 0.6002
Epoch 180/300
```

```
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8820 - loss: 0.5491 - val_accuracy: 0.8644 - val_loss: 0.5996
Epoch 181/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8820 - loss: 0.5483 - val_accuracy: 0.8645 - val_loss: 0.5990
Epoch 182/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8821 - loss: 0.5476 - val_accuracy: 0.8646 - val_loss: 0.5984
Epoch 183/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8821 - loss: 0.5469 - val_accuracy: 0.8646 - val_loss: 0.5979
Epoch 184/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8823 - loss: 0.5462 - val_accuracy: 0.8644 - val_loss: 0.5973
Epoch 185/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8826 - loss: 0.5454 - val_accuracy: 0.8643 - val_loss: 0.5968
Epoch 186/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8827 - loss: 0.5447 - val_accuracy: 0.8644 - val_loss: 0.5962
Epoch 187/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.8827 - loss: 0.5440 - val_accuracy: 0.8643 - val_loss: 0.5956
Epoch 188/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8827 - loss: 0.5433 - val_accuracy: 0.8644 - val_loss: 0.5951
Epoch 189/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8828 - loss: 0.5426 - val_accuracy: 0.8645 - val_loss: 0.5945
Epoch 190/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8828 - loss: 0.5419 - val_accuracy: 0.8645 - val_loss: 0.5940
Epoch 191/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8829 - loss: 0.5413 - val_accuracy: 0.8646 - val_loss: 0.5935
Epoch 192/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8830 - loss: 0.5406 - val_accuracy: 0.8644 - val_loss: 0.5929
Epoch 193/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8832 - loss: 0.5399 - val_accuracy: 0.8644 - val_loss: 0.5923
Epoch 194/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8830 - loss: 0.5393 - val_accuracy: 0.8644 - val_loss: 0.5918
Epoch 195/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8831 - loss: 0.5386 - val_accuracy: 0.8644 - val_loss: 0.5912
Epoch 196/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8835 - loss: 0.5380 - val_accuracy: 0.8645 - val_loss: 0.5907
Epoch 197/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8835 - loss: 0.5373 - val_accuracy: 0.8644 - val_loss: 0.5902
Epoch 198/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8837 - loss: 0.5367 - val_accuracy: 0.8643 - val_loss: 0.5897
Epoch 199/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8839 - loss: 0.5360 - val_accuracy: 0.8643 - val_loss: 0.5892
Epoch 200/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8839 - loss: 0.5354 - val_accuracy: 0.8643 - val_loss: 0.5887
Epoch 201/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8840 - loss: 0.5348 - val_accuracy: 0.8644 - val_loss: 0.5882
Epoch 202/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8840 - loss: 0.5341 - val_accuracy: 0.8640 - val_loss: 0.5877
Epoch 203/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8844 - loss: 0.5335 - val_accuracy: 0.8644 - val_loss: 0.5872
Epoch 204/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8846 - loss: 0.5329 - val_accuracy: 0.8641 - val_loss: 0.5868
Epoch 205/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8846 - loss: 0.5323 - val_accuracy: 0.8644 - val_loss: 0.5862
Epoch 206/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8846 - loss: 0.5316 - val_accuracy: 0.8643 - val_loss: 0.5858
Epoch 207/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8846 - loss: 0.5310 - val_accuracy: 0.8643 - val_loss: 0.5853
Epoch 208/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8848 - loss: 0.5304 - val_accuracy: 0.8643 - val_loss: 0.5848
Epoch 209/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8847 - loss: 0.5297 - val_accuracy: 0.8643 - val_loss: 0.5843
Epoch 210/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8847 - loss: 0.5291 - val_accuracy: 0.8643 - val_loss: 0.5839
Epoch 211/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8848 - loss: 0.5285 - val_accuracy: 0.8644 - val_loss: 0.5834
Epoch 212/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8850 - loss: 0.5279 - val_accuracy: 0.8641 - val_loss: 0.5829
Epoch 213/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8851 - loss: 0.5273 - val_accuracy: 0.8640 - val_loss: 0.5825
Epoch 214/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8854 - loss: 0.5267 - val_accuracy: 0.8641 - val_loss: 0.5820
Epoch 215/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8852 - loss: 0.5261 - val_accuracy: 0.8637 - val_loss: 0.5816
Epoch 216/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8857 - loss: 0.5255 - val_accuracy: 0.8637 - val_loss: 0.5811
Epoch 217/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8855 - loss: 0.5249 - val_accuracy: 0.8637 - val_loss: 0.5807
Epoch 218/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8856 - loss: 0.5242 - val_accuracy: 0.8637 - val_loss: 0.5803
Epoch 219/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8857 - loss: 0.5237 - val_accuracy: 0.8639 - val_loss: 0.5800
Epoch 220/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8858 - loss: 0.5231 - val_accuracy: 0.8637 - val_loss: 0.5795
Epoch 221/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8858 - loss: 0.5224 - val_accuracy: 0.8636 - val_loss: 0.5791
Epoch 222/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8860 - loss: 0.5218 - val_accuracy: 0.8639 - val_loss: 0.5787
Epoch 223/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8862 - loss: 0.5212 - val_accuracy: 0.8640 - val_loss: 0.5784
Epoch 224/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8862 - loss: 0.5206 - val_accuracy: 0.8640 - val_loss: 0.5781
Epoch 225/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8864 - loss: 0.5201 - val_accuracy: 0.8637 - val_loss: 0.5777
Epoch 226/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8865 - loss: 0.5195 - val_accuracy: 0.8641 - val_loss: 0.5773
Epoch 227/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8866 - loss: 0.5189 - val_accuracy: 0.8641 - val_loss: 0.5770
Epoch 228/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8867 - loss: 0.5183 - val_accuracy: 0.8639 - val_loss: 0.5767
Epoch 229/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8868 - loss: 0.5177 - val_accuracy: 0.8636 - val_loss: 0.5764
Epoch 230/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8869 - loss: 0.5171 - val_accuracy: 0.8635 - val_loss: 0.5761
Epoch 231/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8868 - loss: 0.5165 - val_accuracy: 0.8634 - val_loss: 0.5758
Epoch 232/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8869 - loss: 0.5160 - val_accuracy: 0.8631 - val_loss: 0.5755
Epoch 233/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8871 - loss: 0.5154 - val_accuracy: 0.8629 - val_loss: 0.5752
Epoch 234/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8872 - loss: 0.5149 - val_accuracy: 0.8629 - val_loss: 0.5749
Epoch 235/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8875 - loss: 0.5143 - val_accuracy: 0.8626 - val_loss: 0.5746
Epoch 236/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8872 - loss: 0.5138 - val_accuracy: 0.8622 - val_loss: 0.5745
Epoch 237/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8871 - loss: 0.5133 - val_accuracy: 0.8621 - val_loss: 0.5741
Epoch 238/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8870 - loss: 0.5127 - val_accuracy: 0.8621 - val_loss: 0.5738
Epoch 239/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8870 - loss: 0.5122 - val_accuracy: 0.8619 - val_loss: 0.5736
Epoch 240/300
```

```
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8871 - loss: 0.5117 - val_accuracy: 0.8620 - val_loss: 0.5734
Epoch 241/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8869 - loss: 0.5112 - val_accuracy: 0.8621 - val_loss: 0.5731
Epoch 242/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8868 - loss: 0.5106 - val_accuracy: 0.8622 - val_loss: 0.5728
Epoch 243/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8867 - loss: 0.5101 - val_accuracy: 0.8624 - val_loss: 0.5724
Epoch 244/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8868 - loss: 0.5096 - val_accuracy: 0.8624 - val_loss: 0.5722
Epoch 245/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8871 - loss: 0.5091 - val_accuracy: 0.8625 - val_loss: 0.5718
Epoch 246/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8870 - loss: 0.5086 - val_accuracy: 0.8627 - val_loss: 0.5715
Epoch 247/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8871 - loss: 0.5081 - val_accuracy: 0.8627 - val_loss: 0.5712
Epoch 248/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8871 - loss: 0.5076 - val_accuracy: 0.8627 - val_loss: 0.5708
Epoch 249/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8874 - loss: 0.5071 - val_accuracy: 0.8625 - val_loss: 0.5706
Epoch 250/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8874 - loss: 0.5066 - val_accuracy: 0.8626 - val_loss: 0.5703
Epoch 251/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8874 - loss: 0.5061 - val_accuracy: 0.8622 - val_loss: 0.5700
Epoch 252/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8877 - loss: 0.5056 - val_accuracy: 0.8626 - val_loss: 0.5697
Epoch 253/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8877 - loss: 0.5050 - val_accuracy: 0.8626 - val_loss: 0.5693
Epoch 254/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8877 - loss: 0.5046 - val_accuracy: 0.8626 - val_loss: 0.5691
Epoch 255/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8878 - loss: 0.5041 - val_accuracy: 0.8625 - val_loss: 0.5688
Epoch 256/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8878 - loss: 0.5036 - val_accuracy: 0.8625 - val_loss: 0.5685
Epoch 257/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8879 - loss: 0.5031 - val_accuracy: 0.8622 - val_loss: 0.5683
Epoch 258/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8878 - loss: 0.5026 - val_accuracy: 0.8622 - val_loss: 0.5679
Epoch 259/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8880 - loss: 0.5021 - val_accuracy: 0.8619 - val_loss: 0.5677
Epoch 260/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8881 - loss: 0.5017 - val_accuracy: 0.8618 - val_loss: 0.5675
Epoch 261/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8880 - loss: 0.5012 - val_accuracy: 0.8619 - val_loss: 0.5672
Epoch 262/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8881 - loss: 0.5008 - val_accuracy: 0.8621 - val_loss: 0.5668
Epoch 263/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8880 - loss: 0.5003 - val_accuracy: 0.8620 - val_loss: 0.5665
Epoch 264/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8880 - loss: 0.4998 - val_accuracy: 0.8619 - val_loss: 0.5661
Epoch 265/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8881 - loss: 0.4993 - val_accuracy: 0.8619 - val_loss: 0.5658
Epoch 266/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8883 - loss: 0.4989 - val_accuracy: 0.8619 - val_loss: 0.5655
Epoch 267/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8884 - loss: 0.4984 - val_accuracy: 0.8616 - val_loss: 0.5653
Epoch 268/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8885 - loss: 0.4980 - val_accuracy: 0.8621 - val_loss: 0.5649
Epoch 269/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8884 - loss: 0.4975 - val_accuracy: 0.8620 - val_loss: 0.5648
Epoch 270/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8884 - loss: 0.4971 - val_accuracy: 0.8621 - val_loss: 0.5647
Epoch 271/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8884 - loss: 0.4967 - val_accuracy: 0.8622 - val_loss: 0.5643
Epoch 272/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8884 - loss: 0.4962 - val_accuracy: 0.8625 - val_loss: 0.5641
Epoch 273/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8885 - loss: 0.4958 - val_accuracy: 0.8624 - val_loss: 0.5637
Epoch 274/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8886 - loss: 0.4953 - val_accuracy: 0.8625 - val_loss: 0.5634
Epoch 275/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8885 - loss: 0.4949 - val_accuracy: 0.8622 - val_loss: 0.5631
Epoch 276/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8886 - loss: 0.4944 - val_accuracy: 0.8618 - val_loss: 0.5629
Epoch 277/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8885 - loss: 0.4940 - val_accuracy: 0.8619 - val_loss: 0.5626
Epoch 278/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8886 - loss: 0.4936 - val_accuracy: 0.8621 - val_loss: 0.5623
Epoch 279/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8886 - loss: 0.4931 - val_accuracy: 0.8622 - val_loss: 0.5620
Epoch 280/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8887 - loss: 0.4927 - val_accuracy: 0.8624 - val_loss: 0.5617
Epoch 281/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8888 - loss: 0.4922 - val_accuracy: 0.8619 - val_loss: 0.5615
Epoch 282/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8887 - loss: 0.4918 - val_accuracy: 0.8616 - val_loss: 0.5612
Epoch 283/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8887 - loss: 0.4913 - val_accuracy: 0.8619 - val_loss: 0.5611
Epoch 284/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8887 - loss: 0.4909 - val_accuracy: 0.8614 - val_loss: 0.5608
Epoch 285/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8887 - loss: 0.4905 - val_accuracy: 0.8612 - val_loss: 0.5606
Epoch 286/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8885 - loss: 0.4901 - val_accuracy: 0.8610 - val_loss: 0.5604
Epoch 287/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8886 - loss: 0.4898 - val_accuracy: 0.8608 - val_loss: 0.5600
Epoch 288/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8888 - loss: 0.4893 - val_accuracy: 0.8610 - val_loss: 0.5597
Epoch 289/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8886 - loss: 0.4889 - val_accuracy: 0.8610 - val_loss: 0.5594
Epoch 290/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8887 - loss: 0.4885 - val_accuracy: 0.8608 - val_loss: 0.5592
Epoch 291/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8889 - loss: 0.4881 - val_accuracy: 0.8609 - val_loss: 0.5589
Epoch 292/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8889 - loss: 0.4877 - val_accuracy: 0.8608 - val_loss: 0.5585
Epoch 293/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8889 - loss: 0.4873 - val_accuracy: 0.8605 - val_loss: 0.5584
Epoch 294/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8891 - loss: 0.4869 - val_accuracy: 0.8606 - val_loss: 0.5581
Epoch 295/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8890 - loss: 0.4865 - val_accuracy: 0.8604 - val_loss: 0.5578
Epoch 296/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8890 - loss: 0.4861 - val_accuracy: 0.8606 - val_loss: 0.5575
Epoch 297/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8890 - loss: 0.4857 - val_accuracy: 0.8605 - val_loss: 0.5573
Epoch 298/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8891 - loss: 0.4853 - val_accuracy: 0.8609 - val_loss: 0.5570
Epoch 299/300
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8892 - loss: 0.4849 - val_accuracy: 0.8608 - val_loss: 0.5568
Epoch 300/300
```

```
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8892 - loss: 0.4845 - val_accuracy: 0.8605 - val_loss: 0.5566
Test Loss: 0.5308677554130554, Test Accuracy: 0.8662921190261841
```

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()
```

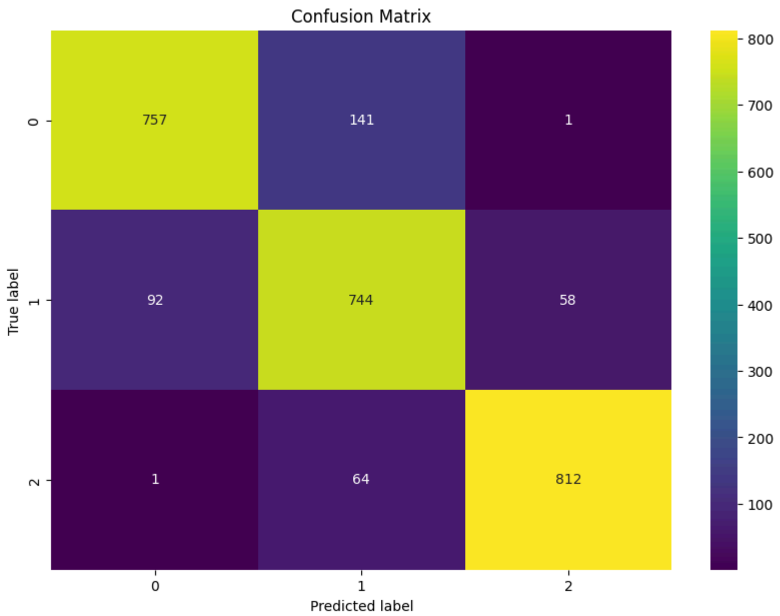


Section 6: Performance of the model

In [ ]:
```python
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report

# Generate predictions on the test data
predictions = model.predict(scaled_test_df.drop(columns=['Export_Value_3_Years_Ahead_Category']))
predicted_classes = np.argmax(predictions, axis=1)

true_labels = scaled_test_df['Export_Value_3_Years_Ahead_Category'].values

# Calculate accuracy
accuracy = round(accuracy_score(true_labels, predicted_classes), 2)
class_report = classification_report(true_labels, predicted_classes)

print(f'Accuracy: {accuracy}')
print('Classification Report:\n', class_report)
```

```
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Accuracy: 0.87
Classification Report:
               precision    recall  f1-score   support

         0.0       0.89      0.84      0.87       899
         1.0       0.78      0.83      0.81       894
         2.0       0.93      0.93      0.93       877

    accuracy                           0.87      2670
   macro avg       0.87      0.87      0.87      2670
weighted avg       0.87      0.87      0.87      2670
```

In [ ]:
```python
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
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,
})

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