Code Snippets For MLP Project

Data Preparation

Customer Price Indicators Data Preparation (Code Snippet)

```python
# Drop redundant columns from the dataframe
columns_to_drop = ['Domain Code', 'Domain', 'Item Code', 'Element Code', 'Element', 'Unit', 'Flag',
                   'Flag Description', 'Note', 'Area Code (M49)', 'Year Code']
consumer_pi_cleaned = consumer_pi.drop(columns=columns_to_drop)

# Split the DataFrame based on 'Item' values
consumer_price = consumer_pi_cleaned[consumer_pi_cleaned['Item'] == 'Consumer Prices, Food Indices (2015 = 100)']
food_price = consumer_pi_cleaned[consumer_pi_cleaned['Item'] == 'Food price inflation']

# Drop 'Item' column and rename 'Values' column
consumer_price = consumer_price.drop(columns='Item')
consumer_price.rename(columns={'Value': 'Total Consumer Price ($) for the Year'}, inplace=True)

food_price = food_price.drop(columns='Item')
food_price.rename(columns={'Value': 'Avg Food price inflation(%) for the Year'}, inplace=True)

# Aggregate 'Months' column by sum for Consumer Price and by mean for Food Price
consumer_price_grouped = consumer_price.groupby(['Area', 'Year']).agg({'Total Consumer Price ($) for the Year': 'sum'}).reset_index()
food_price_grouped = food_price.groupby(['Area', 'Year']).agg({'Avg Food price inflation(%) for the Year': 'mean'}).reset_index()

# Merge datasets back on common columns
cleaned_consumer_pi = pd.merge(consumer_price_grouped, food_price_grouped,
                               on=['Area', 'Year'])

# Export the cleaned_consumer_pi DataFrame to CSV
cleaned_consumer_pi.to_csv('cleaned_consumer_pi.csv', index=False)
```

Crop Price Indicators Data Preparation (Code Snippet)

```python
# Multiply all values in the 'Value' column by 100 since unit is (100g/hectare)
crop_pi['Value'] *= 100

# Divide all values in the 'Value' column by 1,000,000 to convert to (tonne/hectare)
crop_pi['Value'] /= 1000000

# Drop redundant columns
columns_to_drop = ['Domain Code', 'Domain', 'Item Code (CPC)', 'Element Code', 'Element', 'Unit',
                   'Flag', 'Flag Description', 'Area Code (M49)', 'Year Code']
crop_pi_cleaned = crop_pi.drop(columns=columns_to_drop)

# Split the DataFrame based on unique 'Item' values
items = crop_pi_cleaned['Item'].unique()

# Create a dataframes dictionary that holds each of the split dataframes as key value pairs.
dataframes = {}
for item in items:
    item_df = crop_pi_cleaned[crop_pi_cleaned['Item'] == item]
    # Replace all punctuations to form the item names
    item_name = item.lower().replace(', ', '_').replace(' ', '_').replace('-', '_').replace('.', '') + " (tonne/hectare)"
    # Append each dataframe name and its corresponding values to our empty dictionary.
    dataframes[item_name] = item_df

# Export each dataframe to CSV and perform column renaming
for df_name, df in dataframes.items():
    df.drop(columns='Item', inplace=True)
    df.rename(columns={'Value': f'Yearly Yield for {df_name.replace("_", " ").title()}'}, inplace=True)
    # df.to_csv(f'{df_name}.csv', index=False)

# Merge all dataframes back on common columns
common_columns = ['Area', 'Year']
cleaned_crop_pi = pd.concat(dataframes.values()).groupby(common_columns).sum().reset_index()

# Export the cleaned_crop_pi DataFrame to CSV
cleaned_crop_pi.to_csv('cleaned_crop_pi.csv', index=False)
```

<u>Emissions Data Preparation</u>

```
[137]:
# Split the DataFrame into two diffrent dataframes based on 'Domain' values and drop redundant columns
organic_emissions = emissions[emissions['Domain'] == 'Emissions from Drained organic soils'].drop(columns=['Domain Code', 'Domain', 'Area Code (M49)',
                                                                                                          'Year Code'])

crop_emissions = emissions[emissions['Domain'] == 'Emissions from Crops'].drop(columns=['Domain Code', 'Domain', 'Area Code (M49)', 'Year Code'])

# Now for Crop Emissions
# Split crop_emissions into two dataframe based on Ch4 and N2O emissions and drop redundant columns
crop_ch4_emissions = crop_emissions[crop_emissions['Element'] == 'Crops total (Emissions CH4)'].drop(columns=['Element', 'Element Code', 'Item',
                                                                                                             'Item Code (CPC)', 'Source', 'Source Code',
                                                                                                             'Unit', 'Flag Description',
                                                                                                             'Note', 'Flag'])

crop_n2o_emissions = crop_emissions[crop_emissions['Element'] == 'Crops total (Emissions N2O)'].drop(columns=['Element', 'Element Code', 'Item',
                                                                                                             'Item Code (CPC)', 'Source',
                                                                                                             'Source Code', 'Unit',
                                                                                                             'Flag Description', 'Note', 'Flag'])

# Apply the `Units` column for the  crop_emissions values
crop_ch4_emissions.rename(columns={'Value': 'Crop ch4 emissions (Tonnes) for the year'}, inplace=True)
crop_n2o_emissions.rename(columns={'Value': 'Crop n2o emissions (Tonnes) for the year'}, inplace=True)
crop_ch4_emissions['Crop ch4 emissions (Tonnes) for the year'] *= 1000
crop_n2o_emissions['Crop n2o emissions (Tonnes) for the year'] *= 1000

# For co2 organic emissions
co2_organic_emissions = organic_emissions[organic_emissions['Element'] == 'Emissions (CO2)']

# Split the `co2_organic_emissions` into two dataframes
cropland_co2_organic = co2_organic_emissions[co2_organic_emissions['Item'] == 'Cropland organic soils'].drop(columns=['Element', 'Element Code',
                                                                                                                     'Item', 'Item Code (CPC)',
                                                                                                                     'Source', 'Source Code',
                                                                                                                     'Unit', 'Flag Description',
                                                                                                                     'Note', 'Flag'])

grassland_co2_organic = co2_organic_emissions[co2_organic_emissions['Item'] == 'Grassland organic soils'].drop(columns=['Element', 'Element Code',
                                                                                                                       'Item', 'Item Code (CPC)',
                                                                                                                       'Source', 'Source Code',
```

<u>Emissions (Part 2)</u>

```
# Rename 'Items' columns for these two dataframes
cropland_co2_organic.rename(columns={'Value': 'Cropland Organic co2 emissions (Tonnes) for the year'}, inplace=True)
grassland_co2_organic.rename(columns={'Value': 'Grassland Organic co2 emissions (Tonnes) for the year'}, inplace=True)

co2_organic_emissions = pd.merge(cropland_co2_organic, grassland_co2_organic, on=['Area', 'Year'])
co2_organic_emissions['Cropland Organic co2 emissions (Tonnes) for the year'] *= 1000
co2_organic_emissions['Grassland Organic co2 emissions (Tonnes) for the year'] *= 1000

# For n2o organic emissions
n2o_organic_emissions = organic_emissions[organic_emissions['Element'] == 'Emissions (N2O)']

# Split the `n2o_organic_emissions` into two dataframes
cropland_n2o_organic = n2o_organic_emissions[n2o_organic_emissions['Item'] == 'Cropland organic soils'].drop(columns=['Element', 'Element Code',
                                                                                                                     'Item', 'Item Code (CPC)',
                                                                                                                     'Source', 'Source Code',
                                                                                                                     'Unit', 'Flag Description',
                                                                                                                     'Note', 'Flag'])

grassland_n2o_organic = n2o_organic_emissions[n2o_organic_emissions['Item'] == 'Grassland organic soils'].drop(columns=['Element', 'Element Code',
                                                                                                                       'Item', 'Item Code (CPC)',
                                                                                                                       'Source', 'Source Code',
                                                                                                                       'Unit', 'Flag Description',
                                                                                                                       'Note', 'Flag'])

# Rename 'Items' columns for these two dataframes
cropland_n2o_organic.rename(columns={'Value': 'Cropland Organic n2o emissions (Tonnes) for the year'}, inplace=True)
grassland_n2o_organic.rename(columns={'Value': 'Grassland Organic n2o emissions (Tonnes) for the year'}, inplace=True)

n2o_organic_emissions = pd.merge(cropland_n2o_organic, grassland_n2o_organic, on=['Area', 'Year'])
n2o_organic_emissions['Cropland Organic n2o emissions (Tonnes) for the year'] *= 1000
n2o_organic_emissions['Grassland Organic n2o emissions (Tonnes) for the year'] *= 1000

# Merge crop_ch4_emissions and crop_n2o_emissions on common columns
crop_emissions_merged = pd.merge(crop_ch4_emissions, crop_n2o_emissions, on=['Area', 'Year'])

# Merge co2_organic_emissions and n2o_organic_emissions on common columns
```

Emissions Data Preparation (Part 3)

```
n2o_organic_emissions['Grassland Organic n2o emissions (Tonnes) for the year'] *= 1000

# Merge crop_ch4_emissions and crop_n2o_emissions on common columns
crop_emissions_merged = pd.merge(crop_ch4_emissions, crop_n2o_emissions, on=['Area', 'Year'])

# Merge co2_organic_emissions and n2o_organic_emissions on common columns
organic_emissions_merged = pd.merge(co2_organic_emissions, n2o_organic_emissions, on=['Area', 'Year'])

# Step 8: Merge crop_emissions_merged and organic_emissions_merged on common columns
cleaned_emissions = pd.merge(crop_emissions_merged, organic_emissions_merged, on=['Area', 'Year'])


# Export the final_emissions DataFrame to CSV
#cleaned_emissions.to_csv('cleaned_emissions.csv', index=False)
```

Employment Data Preparation

```
[135]: # Drop redundant columns
employment_cleaned = employment.drop(columns=['Domain Code', 'Domain', 'Indicator Code', 'Sex Code', 'Sex',
                                              'Element Code', 'Element', 'Source Code', 'Source',
                                              'Flag', 'Flag Description', 'Note', 'Year Code', 'Area Code (M49)'])

# Split the DataFrame based on 'Indicator' values
agric_employment = employment_cleaned[employment_cleaned['Indicator'] == 'Employment in agriculture, forestry and fishing - ILO modelled estimates'].copy(

# Drop the 'Indicator' column
agric_employment.drop(columns=['Indicator', 'Unit'], inplace=True)

# Incorporate the 'Unit' column to 'Value' for agric_employment and rename columns
agric_employment['Value'] *= 1000  # Multiply by 1000 for '1000 No' in 'Unit'
agric_employment.rename(columns={'Value': 'Employment in agriculture, forestry and fishing'}, inplace=True)
cleaned_employment = agric_employment

# Export the final_employment DataFrame to CSV
#cleaned_employment.to_csv('cleaned_employment.csv', index=False)
```

Exchange Rates Data Preparation

```
[133]: exchange_rates = pd.read_csv(r'C:\Users\Crowntech\Documents\Projects\Mlp project\Exchange rate - FAOSTAT_data_en_2-22-2024.csv')

# Drop redundant columns
exchange_rates_cleaned = exchange_rates.drop(columns=['Domain Code', 'Domain', 'ISO Currency Code (FAO)',
                                                      'Currency', 'Element Code', 'Element',
                                                      'Months Code', 'Flag', 'Flag Description',
                                                      'Unit', 'Area Code (M49)', 'Year Code'])

# Rename the 'Value' column to 'Average Exchange rate (yearly)'
exchange_rates_cleaned.rename(columns={'Value': 'Average Exchange rate (yearly)'}, inplace=True)

# Aggregate 'Average Exchange rate (yearly)' column to average yearly value
exchange_rates_cleaned['Average Exchange rate (yearly)'] = exchange_rates_cleaned.groupby(['Area', 'Year'])['Average Exchange rate (yearly)'].transform('m

# Drop the duplicate rows since the 'transform' function added duplicates
exchange_rates_cleaned.drop_duplicates(subset=['Area', 'Year'], inplace=True)

# Drop the 'Months' column
exchange_rates_cleaned.drop(columns=['Months'], inplace=True)

# Keep only required columns
cleaned_exchange_rate = exchange_rates_cleaned[['Area', 'Year', 'Average Exchange rate (yearly)']].copy()

# Display the first few rows of the final DataFrame
#exchange_rates_final.head(50)

# Export to a csv file
cleaned_exchange_rate.to_csv('cleaned_exchange_rate.csv', index=False)
```

Fertilizer Used Data Preparation (Part 1)

```
[118]: # Drop redundant columns
fertilizers_used_cleaned = fertilizers_used.drop(columns=['Domain Code', 'Domain', 'Item Code',
                                                          'Year Code', 'Element Code', 'Element',
                                                          'Item Code', 'Flag', 'Flag Description',
                                                          'Unit'])
# Change the `Item` column into `Fertilizer Type Used`
fertilizers_used_cleaned.rename(columns={'Item': 'Fertilizer Type Used'}, inplace=True)

# Use the `Item` column to split the dataset into 24 different datasets on
ammonia_anhydrous = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Ammonia, anhydrous', :]
                     .rename(columns={'Value': 'Amount of Ammonia Anhydrous used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code (M49)'])

ammonium_nitrate = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Ammonium nitrate (AN)', :]
                     .rename(columns={'Value': 'Amount of Ammonium Nitrate (AN) Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code (M49)'


ammonium_sulphate = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Ammonium sulphate', :]
                     .rename(columns={'Value': 'Amount of Ammonium Sulphate Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code (M49)'])

calcium_nitrate = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Calcium ammonium nitrate (CAN) and other mixtures wi
                     .rename(columns={'Value': 'Amount of Calcium Nitrate/Calcium Carbonate Fertilizers Used (Yearly)'})).drop(columns=['Fertilizer Type 

diammonium_phosphate = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Diammonium phosphate (DAP)', :]
                     .rename(columns={'Value': 'Amount of Diammonium phosphate (DAP) Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code (

fertilizers_nec = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Fertilizers n.e.c.', :]
                     .rename(columns={'Value': 'Amount of Fertilizers n.e.c. Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code (M49)'])

monoammonium_phosphate = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Monoammonium phosphate (MAP)', :]
                     .rename(columns={'Value': 'Amount of Monoammonium phosphate (MAP) Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code

other_nitro_fertilizers = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Other nitrogenous fertilizers, n.e.c.', :]
```

Fertilizer Used Data Preparation (Part 2)

```
other_nitro_fertilizers = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Other nitrogenous fertilizers, n.e.c.', :]
                     .rename(columns={'Value': 'Amount of Other Nitrogenous Fertilizers Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code

other_nk_compounds = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Other NK compounds', :]
                     .rename(columns={'Value': 'Amount of Other NK Compounds Fertilizer Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code

other_phosphatic_fertilizers = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Other nitrogenous fertilizers, n.e.c.',
                     .rename(columns={'Value': 'Amount of Other Phosphatic Fertilizers Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code 

other_potassic_fertilizers = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Other potassic fertilizers, n.e.c.', :]
                     .rename(columns={'Value': 'Amount of Other Potassic Fertilizers Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code (M

phosphate_rock_fertilizers = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Phosphate rock', :]
                     .rename(columns={'Value': 'Amount of Phosphate Rock Fertilizers Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code (M

pk_compound_fertilizers = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'PK compounds', :]
                     .rename(columns={'Value': 'Amount of PK Compounds Fertilizers Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code (M49

potassium_chloride = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Potassium chloride (muriate of potash) (MOP)', :]
                     .rename(columns={'Value': 'Amount of Potassium Chloride (MOP) Fertilizer Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Are

potassium_nitrate = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Potassium nitrate', :]
                     .rename(columns={'Value': 'Amount of Potassium nitrate Fertilizer Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code 

potassium_sulphate = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Potassium sulphate (sulphate of potash) (SOP)', :
                     .rename(columns={'Value': 'Amount of Potassium Sulphate (SOP) Fertilizer Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Are

sodium_nitrate = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Sodium nitrate', :]
                     .rename(columns={'Value': 'Amount of Sodium Nitrate Fertilizer Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code (M4

superphosphates_35 = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Superphosphates above 35%', :]
                     .rename(columns={'Value': 'Amount of Superphosphates (above 35%) Fertilizer Used (Yearly)'})).drop(columns=['Fertilizer Type Used', '
```

Fertilizer Used Data Preparation (Part 3)

```python
                    .rename(columns={'Value': 'Amount of Potassium Sulphate (SOP) Fertilizer Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'A

sodium_nitrate = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Sodium nitrate', :]
                    .rename(columns={'Value': 'Amount of Sodium Nitrate Fertilizer Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code (

superphosphates_35 = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Superphosphates above 35%', :]
                    .rename(columns={'Value': 'Amount of Superphosphates (above 35%) Fertilizer Used (Yearly)'})).drop(columns=['Fertilizer Type Used',

superphosphates_other = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Superphosphates, other', :]
                    .rename(columns={'Value': 'Amount of Superphosphates other Fertilizer Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area

urea = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Urea', :]
                    .rename(columns={'Value': 'Amount of Urea Fertilizer Used (Yearly)'})).drop(columns=['Fertilizer Type Used', 'Area Code (M49)'])

urea_uan = (fertilizers_used_cleaned.loc[fertilizers_used_cleaned['Fertilizer Type Used'] == 'Urea and ammonium nitrate solutions (UAN)', :]
                    .rename(columns={'Value': 'Amount of Urea and Ammonium Nitrate solutions (UAN) Fertilizer Used (Yearly)'})).drop(columns=['Fertiliz

# Merge all the dataframes into one
df_list = [ammonia_anhydrous, ammonium_nitrate, ammonium_sulphate, calcium_nitrate, diammonium_phosphate, fertilizers_nec, monoammonium_phosphate,
           other_nitro_fertilizers, other_nk_compounds, other_phosphatic_fertilizers, other_potassic_fertilizers, phosphate_rock_fertilizers,
           pk_compound_fertilizers, potassium_chloride, potassium_nitrate, potassium_sulphate, sodium_nitrate, superphosphates_35,
           superphosphates_other, urea, urea_uan]

# Merge all DataFrames on 'Year' and 'Area' columns using a left join
cleaned_fertilizers = df_list[0]
for df in df_list[1:]:
    cleaned_fertilizers = pd.merge(cleaned_fertilizers, df, on=['Year', 'Area'], how='left')
cleaned_fertilizers.head(50)
# Export to csv
#cleaned_fertilizers.to_csv('cleaned_fertilizers.csv', index=False)
```

Food Security Indicator Data Preparation

```python
[144]: fsi = pd.read_csv(r'C:\Users\Crowntech\Documents\Projects\Mlp project\Food security indicators  - FAOSTAT_data_en_2-22-2024.csv')

# Drop redundant columns
fsi.drop(columns=['Domain Code', 'Domain', 'Area Code (M49)', 'Element', 'Element Code', 'Item Code', 'Flag', 'Flag Description', 'Note', 'Year Code'],
         inplace=True)

# Split DataFrame based on 'Item' column values
food_prod_variability = fsi[fsi['Item'] == 'Per capita food production variability (constant 2014-2016 thousand int$ per capita)'].drop(columns=['Item',
                                                                                                                                                 'Unit'])

food_prod_variability['Value'] *= 1000
food_prod_variability.rename(columns={'Value': 'Per Capita Food Production Value (International Dollar)'}, inplace=True)

food_supply_variability = fsi[fsi['Item'] == 'Per capita food supply variability (kcal/cap/day)'].drop(columns=['Item', 'Unit'])
food_supply_variability['Value'] *= 1000
food_supply_variability.rename(columns={'Value': 'Per Capita Food Supply Value (Calories per Capita Per Day)'}, inplace=True)

political_stability = fsi[fsi['Item'] == 'Political stability and absence of violence/terrorism (index)'].drop(columns=['Item', 'Unit'])
political_stability.rename(columns={'Value': 'Political stability and absence of violence/terrorism (index)'}, inplace=True)

anemia_prevalence = fsi[fsi['Item'] == 'Prevalence of anemia among women of reproductive age (15-49 years)'].drop(columns=['Item', 'Unit'])
anemia_prevalence.rename(columns={'Value': 'Percentage Prevalence of anemia among women of reproductive age (15-49 years)'}, inplace=True)

# Merge the DataFrames back on common columns 'Year' and 'Area'
fsi_final_merge = pd.merge(food_prod_variability, food_supply_variability, on=['Year', 'Area'])
fsi_final_merge = pd.merge(fsi_final_merge, political_stability, on=['Year', 'Area'])
cleaned_fsi = pd.merge(fsi_final_merge, anemia_prevalence, on=['Year', 'Area'])

cleaned_fsi.head(50)

# Export to CSV
#cleaned_fsi.to_csv('cleaned_fsi.csv', index=False)
```

Food Trade Indicators Data Preparation (Part 1)

```
[141]: fti = pd.read_csv(r'C:\Users\Crowntech\Documents\Projects\Mlp project\Food trade indicators - FAOSTAT_data_en_2-22-2024.csv')

# Drop specified columns
fti.drop(columns=['Domain Code', 'Domain', 'Area Code (M49)', 'Year Code', 'Item Code (CPC)',
                  'Element Code', 'Unit', 'Flag', 'Flag Description', 'Note'], inplace=True)

# Multiply all values in the 'Value' column by 1000
fti['Value'] *= 1000

# Split DataFrame based on 'Element' column values
export_value = fti[fti['Element'] == 'Export Value'].drop(columns=['Element'])
import_value = fti[fti['Element'] == 'Import Value'].drop(columns=['Element'])

# Split 'export_value' DataFrame based on 'Item' column values
fruits_and_veggies = export_value[export_value['Item'] == 'Fruit and Vegetables'].drop(columns=['Item']).rename(columns={'Value': 'Export Value of Fruits
non_food = export_value[export_value['Item'] == 'Non-food'].drop(columns=['Item']).rename(columns={'Value': 'Export Value of Non-food Items (USD)'})
other_food = export_value[export_value['Item'] == 'Other food'].drop(columns=['Item']).rename(columns={'Value': 'Export Value of Other food Items (USD)'})
sugar_and_honey = export_value[export_value['Item'] == 'Sugar and Honey'].drop(columns=['Item']).rename(columns={'Value': 'Export Value of Sugar and Honey
tobacco = export_value[export_value['Item'] == 'Tobacco'].drop(columns=['Item']).rename(columns={'Value': 'Export Value of Tobacco (USD)'})

# Merge the 'export_value' DataFrames back on 'Year' and 'Area' columns
export_merged = pd.merge(fruits_and_veggies, non_food, on=['Year', 'Area'])
export_merged = pd.merge(export_merged, other_food, on=['Year', 'Area'])
export_merged = pd.merge(export_merged, sugar_and_honey, on=['Year', 'Area'])
export_merged = pd.merge(export_merged, tobacco, on=['Year', 'Area'])

# Split 'import_value' DataFrame based on 'Item' column values
import_alcoholic_beverages = import_value[import_value['Item'] == 'Alcoholic Beverages'].drop(columns=['Item']).rename(columns={'Value': 'Import Value of
import_cereals_and_prep = import_value[import_value['Item'] == 'Cereals and Preparations'].drop(columns=['Item']).rename(columns={'Value': 'Import Value o
import_dairy = import_value[import_value['Item'] == 'Dairy Products and Eggs'].drop(columns=['Item']).rename(columns={'Value': 'Import Value of Dairy Prod
import_fats_and_oils = import_value[import_value['Item'] == 'Fats and Oils (excluding Butter)'].drop(columns=['Item']).rename(columns={'Value': 'Import Va
import_fruits_and_veggies = import_value[import_value['Item'] == 'Fruit and Vegetables'].drop(columns=['Item']).rename(columns={'Value': 'Import Value of
```

Food Trade Indicators Data Preparation (Part 2)

```
+ Code                                                                JupyterLab   Python 3 (ipykernel)
import_dairy = import_value[import_value['Item'] == 'Dairy Products and Eggs'].drop(columns=['Item']).rename(columns={'Value': 'Import Value of Dairy Prod
import_fats_and_oils = import_value[import_value['Item'] == 'Fats and Oils (excluding Butter)'].drop(columns=['Item']).rename(columns={'Value': 'Import Va
import_fruits_and_veggies = import_value[import_value['Item'] == 'Fruit and Vegetables'].drop(columns=['Item']).rename(columns={'Value': 'Import Value of
import_meats = import_value[import_value['Item'] == 'Meat and Meat Preparations'].drop(columns=['Item']).rename(columns={'Value': 'Import Value of Meat ar
import_non_alcohols = import_value[import_value['Item'] == 'Non-alcoholic Beverages'].drop(columns=['Item']).rename(columns={'Value': 'Import Value of Nor
import_non_edible_fats = import_value[import_value['Item'] == 'Non-edible Fats and Oils'].drop(columns=['Item']).rename(columns={'Value': 'Import Value of
import_non_food = import_value[import_value['Item'] == 'Non-food'].drop(columns=['Item']).rename(columns={'Value': 'Import Value of Non-food (USD)'})
import_other_food = import_value[import_value['Item'] == 'Other food'].drop(columns=['Item']).rename(columns={'Value': 'Import Value of Other food (USD)'}
import_sugar_and_honey = import_value[import_value['Item'] == 'Sugar and Honey'].drop(columns=['Item']).rename(columns={'Value': 'Import Value of Sugar ar
import_tobacco = import_value[import_value['Item'] == 'Tobacco'].drop(columns=['Item']).rename(columns={'Value': 'Import Value of Tobacco (USD)'})

# Merge the 'import_value' DataFrames back on 'Year' and 'Area' columns
import_merged = pd.merge(import_alcoholic_beverages, import_cereals_and_prep, on=['Year', 'Area'])
import_merged = pd.merge(import_merged, import_dairy, on=['Year', 'Area'])
import_merged = pd.merge(import_merged, import_fats_and_oils, on=['Year', 'Area'])
import_merged = pd.merge(import_merged, import_fruits_and_veggies, on=['Year', 'Area'])
import_merged = pd.merge(import_merged, import_meats, on=['Year', 'Area'])
import_merged = pd.merge(import_merged, import_non_alcohols, on=['Year', 'Area'])
import_merged = pd.merge(import_merged, import_non_edible_fats, on=['Year', 'Area'])
import_merged = pd.merge(import_merged, import_non_food, on=['Year', 'Area'])
import_merged = pd.merge(import_merged, import_other_food, on=['Year', 'Area'])
import_merged = pd.merge(import_merged, import_sugar_and_honey, on=['Year', 'Area'])
import_merged = pd.merge(import_merged, import_tobacco, on=['Year', 'Area'])

# Merge the 'export_merged' and 'import_merged' DataFrames back on 'Year' and 'Area' columns
cleaned_fti = pd.merge(export_merged, import_merged, on=['Year', 'Area'])

# Sort the DataFrame by the 'Area' and `Year` column
cleaned_fti = cleaned_fti.sort_values(by=['Area', 'Year'])
cleaned_fti.head(50)
# Export to CSV
#sorted_fti.to_csv('cleaned_fti.csv', index=False)
```

Foreign Direct Investment Data Preparation

```
[142]: fdi = pd.read_csv(r'C:\Users\Crowntech\Documents\Projects\Mlp project\Foreign direct investment - FAOSTAT_data_en_2-27-2024.csv')

# Drop specified columns
fdi.drop(columns=['Domain', 'Domain Code', 'Area Code (M49)', 'Element Code', 'Element',
                  'Item Code', 'Year Code', 'Unit', 'Flag', 'Flag Description', 'Note'], inplace=True)

# Multiply 'Value' column by 1000000
fdi['Value'] *= 1000000

# Filter rows based on 'Item' column values
total_fdi_inflows = fdi[fdi['Item'] == 'Total FDI inflows'].drop(columns=['Item']).rename(columns={'Value': 'Total FDI Inflows (Dollars)'})
total_fdi_outflows = fdi[fdi['Item'] == 'Total FDI outflows'].drop(columns=['Item']).rename(columns={'Value': 'Total FDI Outflows (Dollars)

# Merge the 'total_fdi_inflows' and 'total_fdi_outflows' DataFrames on 'Area' and 'Year' columns
cleaned_fdi = pd.merge(total_fdi_inflows, total_fdi_outflows, on=['Area', 'Year'])

#fdi_merged.head(50)

# Export to CSV
#fdi_merged.to_csv('cleaned_fdi.csv', index=False)
```

`Land Temperature Change` **data preparation**

Land Temperature Change Data Preparation

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

# Split DataFrame based on 'Element' column values
standard_dev = land_temp_change[land_temp_change['Element'] == 'Standard Deviation']
temp_change = land_temp_change[land_temp_change['Element'] == 'Temperature change']

# Split 'standard_dev' DataFrame based on 'Months' column values
dec_jan_feb = standard_dev[standard_dev['Months'] == 'Dec-Jan-Feb'].drop(columns=['Months', 'Element']).rename(columns={'Value': 'Standard Deviation of Ch
jun_jul_aug = standard_dev[standard_dev['Months'] == 'Jun-Jul-Aug'].drop(columns=['Months', 'Element']).rename(columns={'Value': 'Standard Deviation of Ch
mar_apr_may = standard_dev[standard_dev['Months'] == 'Mar-Apr-May'].drop(columns=['Months', 'Element']).rename(columns={'Value': 'Standard Deviation of Ch
metereological_year = standard_dev[standard_dev['Months'] == 'Meteorological year'].drop(columns=['Months', 'Element']).rename(columns={'Value': 'Standard
sep_oct_nov = standard_dev[standard_dev['Months'] == 'Sep-Oct-Nov'].drop(columns=['Months', 'Element']).rename(columns={'Value': 'Standard Deviation of Ch

# Merge the 'standard_dev' DataFrames back on 'Area' and 'Year' columns
sd_merged = pd.concat([dec_jan_feb, jun_jul_aug, mar_apr_may, metereological_year, sep_oct_nov])
sd_merged = sd_merged.groupby(['Area', 'Year']).sum().reset_index()

# Split 'temp_change' DataFrame based on 'Months' column values
temp_dec_jan_feb = temp_change[temp_change['Months'] == 'Dec-Jan-Feb'].drop(columns=['Months', 'Element']).rename(columns={'Value': 'Change In Temperature
temp_jun_jul_aug = temp_change[temp_change['Months'] == 'Jun-Jul-Aug'].drop(columns=['Months', 'Element']).rename(columns={'Value': 'Change In Temperature
temp_mar_apr_may = temp_change[temp_change['Months'] == 'Mar-Apr-May'].drop(columns=['Months', 'Element']).rename(columns={'Value': 'Change In Temperature
temp_metereological_year = temp_change[temp_change['Months'] == 'Meteorological year'].drop(columns=['Months', 'Element']).rename(columns={'Value': 'Chang
temp_sep_oct_nov = temp_change[temp_change['Months'] == 'Sep-Oct-Nov'].drop(columns=['Months', 'Element']).rename(columns={'Value': 'Change In Temperature

# Merge the 'temp_change' DataFrames back on 'Area' and 'Year' columns
temp_merged = pd.concat([temp_dec_jan_feb, temp_jun_jul_aug, temp_mar_apr_may, temp_metereological_year, temp_sep_oct_nov])
temp_merged = temp_merged.groupby(['Area', 'Year']).sum().reset_index()

temp_merged = temp_merged.groupby(['Area', 'Year']).sum().reset_index()

# Merge the 'sd_merged' and 'temp_merged' DataFrames on 'Year' and 'Area' columns
cleaned_land_temp_change = pd.merge(sd_merged, temp_merged, on=['Area', 'Year'])
```

Land Use Data Preparation

```python
land_use.drop(columns=['Domain Code', 'Domain', 'Area Code (M49)', 'Element Code', 'Element',
                       'Item Code', 'Year Code', 'Flag', 'Flag Description', 'Note', 'Unit'],
              inplace=True)

# Multiply 'Value' column by 1000
land_use['Value'] *= 1000

# Split DataFrame based on 'Item' column values
agric_land = land_use[land_use['Item'] == 'Agricultural land'].rename(columns={'Value': 'Country Agricultural Land Mass (Hectares)'})
agric = land_use[land_use['Item'] == 'Agriculture'].rename(columns={'Value': 'Country Land Mass for Agriculture Only (Hectares)'})
arable_land = land_use[land_use['Item'] == 'Arable land'].rename(columns={'Value': 'Country Arable Land Mass (Hectares)'})
country_area = land_use[land_use['Item'] == 'Country Area'].rename(columns={'Value': 'Country Area Land Mass (Hectares)'})
cropland = land_use[land_use['Item'] == 'Cropland'].rename(columns={'Value': 'Country Cropland Mass (Hectares)'})
land_area = land_use[land_use['Item'] == 'Land area'].rename(columns={'Value': 'Country Land Area (Hectares)'})
equipped_irrigated_land_area = land_use[land_use['Item'] == 'Land area equipped for irrigation'].rename(columns={'Value': 'Country Land Area Equip
permanent_crops = land_use[land_use['Item'] == 'Permanent crops'].rename(columns={'Value': 'Country Permanent Crops Land Area (Hectares)'})
permanent_meadows = land_use[land_use['Item'] == 'Permanent meadows and pastures'].rename(columns={'Value': 'Country Meadows and Pastures Total La
temp_crops = land_use[land_use['Item'] == 'Temporary crops'].rename(columns={'Value': 'Country Total Land Area for Temporary Crops (Hectares)'})
temp_fallow = land_use[land_use['Item'] == 'Temporary fallow'].rename(columns={'Value': 'Country Total Land Area for Temporary Fallow (Hectares)'}
temp_meadows = land_use[land_use['Item'] == 'Temporary meadows and pastures'].rename(columns={'Value': 'Country Total Land Area for Temporary Mead

# Merge the 20 dataframes on 'Year' and 'Area' columns
dfs_to_merge = [agric_land, agric, arable_land, country_area, cropland, land_area, equipped_irrigated_land_area,
                permanent_crops, permanent_meadows, temp_crops,temp_fallow, temp_meadows]

land_use_final_merged = pd.concat(dfs_to_merge).groupby(['Year', 'Area']).sum().reset_index()

# Drop the `Item` column
land_use_final_merged.drop(columns=['Item'], inplace=True)

# Sort the DataFrame by the 'Area' and `Year` columns
cleaned_land_use = land_use_final_merged.sort_values(by=['Area', 'Year'])
```

Pesticides Use Data Preparation (Part 1)

```python
# Load the pesticides DataFrame
pesticides = pd.read_csv(r'C:\Users\Crowntech\Documents\Projects\Mlp project\Pesticides use - FAOSTAT_data_en_2-27-2024.csv')

# Drop redundant columns
pesticides_cleaned = pesticides.drop(columns=['Domain Code', 'Domain', 'Area Code (M49)', 'Element Code',
                                              'Item Code', 'Year Code', 'Flag', 'Flag Description', 'Note'])

# Split the DataFrame based on 'Element' values
agricultural_use = pesticides_cleaned[pesticides_cleaned['Element'] == 'Agricultural Use']
per_area_cropland = pesticides_cleaned[pesticides_cleaned['Element'] == 'Use per area of cropland']
per_agric_value = pesticides_cleaned[pesticides_cleaned['Element'] == 'Use per value of agricultural production']

# Split the 'agricultural_use' DataFrame based on 'Item' values
agricultural_use_items = ['Pesticides (Total)', 'Insecticides', 'Herbicides', 'Fungicides and Bactericides',
                          'Fungicides - Seed treatments', 'Insecticides - Seed Treatments', 'Rodenticides']

agricultural_use_dataframes = {}
for item in agricultural_use_items:
    df_name = item.lower().replace(" ", "_") + "_used"
    agricultural_use_dataframes[df_name] = agricultural_use[agricultural_use['Item'] == item].copy()
    agricultural_use_dataframes[df_name].rename(columns={'Value': f'Total Amount of {item} Used (Tonnes) for Agriculture'}, inplace=True)
    agricultural_use_dataframes[df_name].drop(columns=['Item', 'Element', 'Unit'], inplace=True)

# Merge the 7 dataframes from 'agricultural_use_dataframes' on common columns 'Year' and 'Area'
agricultural_use_merged = pd.concat(agricultural_use_dataframes.values()).groupby(['Year', 'Area']).sum().reset_index()

# Process 'per_area_cropland' DataFrame
per_area_cropland.drop(columns=['Item', 'Element', 'Unit'], inplace=True)
per_area_cropland['Value'] /= 1000  # Convert kg/hectare to tonne/hectare
per_area_cropland.rename(columns={'Value': 'Total Amount of Pesticides(Tonnes) Used per Hectare of Cropland'}, inplace=True)

# Process 'per_agric_value' DataFrame
per_agric_value.drop(columns=['Item', 'Element', 'Unit'], inplace=True)
```

Pesticides Use Data Preparation (Part 2)

```python
                           'Fungicides – Seed treatments', 'Insecticides – Seed Treatments', 'Rodenticides']

agricultural_use_dataframes = {}
for item in agricultural_use_items:
    df_name = item.lower().replace(" ", "_") + "_used"
    agricultural_use_dataframes[df_name] = agricultural_use[agricultural_use['Item'] == item].copy()
    agricultural_use_dataframes[df_name].rename(columns={'Value': f'Total Amount of {item} Used (Tonnes) for Agriculture'}, inplace=True)
    agricultural_use_dataframes[df_name].drop(columns=['Item', 'Element', 'Unit'], inplace=True)

# Merge the 7 dataframes from 'agricultural_use_dataframes' on common columns 'Year' and 'Area'
agricultural_use_merged = pd.concat(agricultural_use_dataframes.values()).groupby(['Year', 'Area']).sum().reset_index()

# Process 'per_area_cropland' DataFrame
per_area_cropland.drop(columns=['Item', 'Element', 'Unit'], inplace=True)
per_area_cropland['Value'] /= 1000  # Convert kg/hectare to tonne/hectare
per_area_cropland.rename(columns={'Value': 'Total Amount of Pesticides(Tonnes) Used per Hectare of Cropland'}, inplace=True)

# Process 'per_agric_value' DataFrame
per_agric_value.drop(columns=['Item', 'Element', 'Unit'], inplace=True)
per_agric_value['Value'] /= 1000000  # Convert gram/dollars to Tonne/dollars
per_agric_value.rename(columns={'Value': 'Total Amount of Pesticides(Tonnes) Used per Value of Agricultural Production'}, inplace=True)

# Merge the three dataframes on common columns 'Year' and 'Area' to get 'pesticides_merged'
pesticides_merged = pd.merge(agricultural_use_merged, per_area_cropland, on=['Year', 'Area'])
pesticides_merged = pd.merge(pesticides_merged, per_agric_value, on=['Year', 'Area'])

# Sort according to `Area`
cleaned_pesticides = pesticides_merged.sort_values(by='Area')
cleaned_pesticides.head(50)

# Export to 'pesticides_cleaned.csv' file
#pesticides_merged.to_csv('cleaned_pesticides.csv', index=False)
```

Final Merge

Code   JupyterLab

on common columns `Area` and `Year`.

[288]:
```python
# Convert 'Year' column to the same data type in all DataFrames
cleaned_land_temp_change['Year'] = cleaned_land_temp_change['Year'].astype(int)
cleaned_exchange_rate['Year'] = cleaned_exchange_rate['Year'].astype(int)
cleaned_employment['Year'] = cleaned_employment['Year'].astype(int)
cleaned_emissions['Year'] = cleaned_emissions['Year'].astype(int)
cleaned_crop_pi['Year'] = cleaned_crop_pi['Year'].astype(int)
cleaned_consumer_pi['Year'] = cleaned_consumer_pi['Year'].astype(int)
cleaned_fsi['Year'] = cleaned_fsi['Year'].astype(int)
cleaned_fti['Year'] = cleaned_fti['Year'].astype(int)
cleaned_fdi['Year'] = cleaned_fdi['Year'].astype(int)
cleaned_land_use['Year'] = cleaned_land_use['Year'].astype(int)
cleaned_pesticides['Year'] = cleaned_pesticides['Year'].astype(int)

# Merge DataFrames one by one
merged_data = pd.merge(cleaned_land_temp_change, cleaned_exchange_rate, on=['Area', 'Year'], how='outer')
merged_data = pd.merge(merged_data, cleaned_employment, on=['Area', 'Year'], how='outer')
merged_data = pd.merge(merged_data, cleaned_emissions, on=['Area', 'Year'], how='outer')
merged_data = pd.merge(merged_data, cleaned_crop_pi, on=['Area', 'Year'], how='outer')
merged_data = pd.merge(merged_data, cleaned_consumer_pi, on=['Area', 'Year'], how='outer')
merged_data = pd.merge(merged_data, cleaned_fsi, on=['Area', 'Year'], how='outer')
merged_data = pd.merge(merged_data, cleaned_fti, on=['Area', 'Year'], how='outer')
merged_data = pd.merge(merged_data, cleaned_fdi, on=['Area', 'Year'], how='outer')
merged_data = pd.merge(merged_data, cleaned_land_use, on=['Area', 'Year'], how='outer')
merged_data = pd.merge(merged_data, cleaned_pesticides, on=['Area', 'Year'], how='outer')

merged_data.head()
# Export the merged DataFrame to a CSV file
# merged_data.to_csv('merged_data.csv', index=False)
```

Filling Missing Values (With Time Series Interpolation)

```
dtype: int64
```

```
[293]: #Perform time Series interpolation on missing values
columns_to_interpolate = merged_data.columns.difference(['Area'])

# Convert 'Year' column to datetime index
merged_data['Year'] = pd.to_datetime(merged_data['Year'], format='%Y')
merged_data.set_index('Year', inplace=True)

# Apply time series interpolation to the entire DataFrame
merged_data_interpolated = merged_data.interpolate(method='time', axis=0, limit_direction='both')

# Reset index of the interpolated DataFrame
merged_data_interpolated.reset_index(inplace=True)

# Set the display options to show all rows
pd.set_option('display.max_columns', None)
# Check the resulting DataFrame
merged_data_interpolated.isna().sum()
```

Converting Categorical Column to Numerical

```
[299]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# from tensorflow.keras.models import Sequential
# from tensorflow.keras.layers import Dense



# Convert categorical columns to numerical columns
final_merged_numeric = pd.get_dummies(final_merge)

final_merged_numeric.head(50)
```

```
5000.0   4420000.0        0.0   6.149700e+07   5.389000e+06   1.204500e+07   1.300000e+04   0.000000e+0
```

Feature Scaling

```
[324]: # Convert the NumPy array back to a DataFrame
scaled_features_df = pd.DataFrame(features, columns=features.columns)

# Separate numeric columns (excluding boolean columns) for scaling
numeric_columns = scaled_features_df.select_dtypes(include=['number']).columns

# Scale numeric columns only
scaler = StandardScaler()
scaled_values = scaler.fit_transform(scaled_features_df[numeric_columns])

# Convert scaled numeric values back to a DataFrame
scaled_numeric_df = pd.DataFrame(scaled_values, columns=numeric_columns)

# Combine scaled numeric columns with non-numeric (boolean) columns
scaled_features_df[numeric_columns] = scaled_numeric_df

# Add the 'Year' column back to the scaled features DataFrame
scaled_features_df['Year'] = features['Year']

scaled_features_df.head()

[324]:
```

Splitting Into Training and Test

```
[338]: # Define the year ranges for training and test data
train_years = range(2000, 2017)
test_years = range(2017, 2020)

# Filter the data based on the 'Year' column
train_features = scaled_features_df[scaled_features_df['Year'].dt.year.isin(train_years)]
train_labels = labels[labels['Year'].dt.year.isin(train_years)].drop(columns='Year')

test_features = scaled_features_df[scaled_features_df['Year'].dt.year.isin(test_years)]
test_labels = labels[labels['Year'].dt.year.isin(test_years)].drop(columns='Year')

# Print the shapes of training and test data to verify
print("Training Features Shape:", train_features.shape)
print("Training Labels Shape:", train_labels.shape)
print("Test Features Shape:", test_features.shape)
print("Test Labels Shape:", test_labels.shape)

train_features.to_csv('train_features.csv', index=False)
train_labels.to_csv('train_labels.csv', index=False)
test_features.to_csv('test_features.csv', index=False)
test_labels.to_csv('test_labels.csv', index=False)
```

```
Training Features Shape: (4238, 287)
Training Labels Shape: (4238, 5)
Test Features Shape: (750, 287)
Test Labels Shape: (750, 5)
```

MLP Model Architecture

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping


# Define the TensorFlow model architecture
def create_model(input_shape):
    model = tf.keras.Sequential([
    tf.keras.layers.Dense(512, activation='relu', input_shape=input_shape),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1)
])
    return model

# Define the input shape based on the number of features
input_shape = (X_train.shape[1],)


# Train each network separately for each label
models = []
for label_index in range(y_train.shape[1]):
    model = create_model(input_shape)
    model.compile(optimizer='adam', loss='mean_squared_error')
    # Define early stopping callback.
    early_stopping = EarlyStopping(monitor='loss', patience=4, restore_best_weights=True)
```

MLP Model Evaluation (Before Tuning)

+ Code + Text

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Convert predictions_df to NumPy array
predictions_array = predictions_df.to_numpy()

# Calculate MAE, MSE, RMSE, and R2 for each label
mae_scores = []
mse_scores = []
rmse_scores = []
r2_scores = []

for label_index in range(y_test.shape[1]):
    actual_values = y_test[:, label_index]
    predicted_values = predictions_array[:, label_index]

    mae = mean_absolute_error(actual_values, predicted_values)
    mse = mean_squared_error(actual_values, predicted_values)
    rmse = np.sqrt(mse)
    r2 = r2_score(actual_values, predicted_values)

    mae_scores.append(mae)
    mse_scores.append(mse)
    rmse_scores.append(rmse)
    r2_scores.append(r2)

# Aggregate the metrics
mean_mae = np.mean(mae_scores)
mean_mse = np.mean(mse_scores)
mean_rmse = np.mean(rmse_scores)
mean_r2 = np.mean(r2_scores)

# Display or use the aggregated evaluation metrics as needed
```

Evaluation Metrics Before Tuninig

```python
# Display or use the aggregated evaluation metrics as needed
print("Mean Absolute Error (MAE):", mean_mae)
print("Mean Squared Error (MSE):", mean_mse)
print("Root Mean Squared Error (RMSE):", mean_rmse)
print("Mean R-squared (R2):", mean_r2)
```

```
Mean Absolute Error (MAE): 1101055500.0
Mean Squared Error (MSE): 5.3172976e+18
Root Mean Squared Error (RMSE): 1962897400.0
Mean R-squared (R2): 0.2479964644786182
```

Hyperparameter Tuning (Part 1)

```python
[7] from sklearn.model_selection import GridSearchCV
    from tensorflow.keras.callbacks import EarlyStopping
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense


    from sklearn.model_selection import GridSearchCV
    from tensorflow.keras.wrappers.scikit_learn import KerasRegressor


    # Define the function to create the model (required for KerasRegressor)
    def create_model(input_shape, optimizer='adam'):
        model = Sequential([
            Dense(512, activation='relu', input_shape=input_shape),
            Dense(256, activation='relu'),
            Dense(256, activation='relu'),
            Dense(128, activation='relu'),
            Dense(128, activation='relu'),
            Dense(64, activation='relu'),
            Dense(64, activation='relu'),
            Dense(32, activation='relu'),
            Dense(32, activation='relu'),
            Dense(1)
        ])
        model.compile(optimizer=optimizer, loss='mean_squared_error')
        return model

    # Define the input shape based on the number of features
    input_shape = (X_train.shape[1],)


    # Define the hyperparameters grid for GridSearchCV
```

Hyperparameter Tuning (Part 2)

```python
param_grid = {
    'optimizer': ['adam', 'rmsprop'],
    'batch_size': [32],
    'epochs': [40, 50]
}

# Initialize a list to store the best models and their predictions
best_models = []
y_preds = []

# Train and tune each model separately
for label_index in range(y_train.shape[1]):
    # Create a KerasRegressor model
    keras_model = KerasRegressor(build_fn=create_model, input_shape=input_shape, verbose=1)

    # Define early stopping callback.
    early_stopping = EarlyStopping(monitor='loss', patience=3, restore_best_weights=True)

    # Create a GridSearchCV instance for the current model
    grid_search = GridSearchCV(estimator=keras_model, param_grid=param_grid, cv=2, verbose=1)

    # Fit the GridSearchCV instance on the training data
    grid_search.fit(X_train, y_train[:, label_index], callbacks=[early_stopping])

    # Get the best parameters and best model from GridSearchCV
    best_params = grid_search.best_params_
    best_model = grid_search.best_estimator_

    # Store the best model for evaluation
    best_models.append(best_model)

    # Make predictions using the best model
    y_pred = best_model.predict(X_test)
    y_preds.append(y_pred)
```

Evaluation After Tuning

```python
    print(f"Median: {metric_values['Median']}")
    print()
```

```
MAE:
Mean: 919240192.0
Median: 1171806976.0

MSE:
Mean: 4.631849962169172e+18
Median: 4.711112655903916e+18

RMSE:
Mean: 1852330240.0
Median: 2170509824.0

R-squared (R2):
Mean: 0.3089024441527243
Median: 0.3179316052006145
```

Feature Importance

\+ Code + Text

```python
feature_importances = []

# Train and tune each model separately, and extract feature importances
for label_index in range(y_train.shape[1]):
    # Create a RandomForestRegressor model
    rf_model = RandomForestRegressor(n_estimators=100)
    rf_model.fit(X_train, y_train[:, label_index])

    # Get feature importances and store them
    feature_importances.append(rf_model.feature_importances_)

# Aggregate feature importances across all models
aggregate_importances = np.mean(feature_importances, axis=0)

# Identify top N most important features
top_n = 10
top_feature_indices = np.argsort(aggregate_importances)[::-1][:top_n]
top_feature_names = [feature_names[i] for i in top_feature_indices]

print("Top", top_n, "most important features:")
for feature, importance in zip(top_feature_names, aggregate_importances[top_feature_indices]):
    print(feature, ":", importance)
```

```
Top 10 most important features:
Yearly Yield for Fruit Primary (Tonne/Hectare) : 0.2381941883116358
Area_Brazil : 0.0936149172853029
Year_numeric : 0.07893632557883848
Country Total Land Area for Temporary Meadows and Pastures (Hectares) : 0.05548532566622507
Area_Germany : 0.04279847719126638
Area_Netherlands (Kingdom of the) : 0.03808562061683943
Country Land Area Equipped for Irrigation (Hectares) : 0.0322493113389427
Yearly Yield for Treenuts Total (Tonne/Hectare) : 0.03220953467538942
Yearly Yield for Roots And Tubers Total (Tonne/Hectare) : 0.02909572766027766
Yearly Yield for Vegetables Primary (Tonne/Hectare) : 0.02750756765651057
```

✓ 2m 21s completed at 1:55 AM