# Code Documentation (Part-By-Part Merging and Harmonization):

The following is the code used for mapping and merging variables across the years, part by part in a section. This is a summary of what the code is doing:

* It loads files
* Stores them in data frames
* Renames variables from other years to variable names in the reference year for merge mappings
* Mappings lists with updated names are made
* All column names are iterated across the years (so that no column name is skipped)
* Iterate through each dataframe and mapping, appending data to the corresponding column in the merged\_data dictionary. NaNs are added for missing columns to maintain consistency.
* Final renaming to standardize column names in the merged dataframe according to the provided schema.
* Saving the merged files to a CSV format
* In the odd case where there are multiple files from the same year for a single section, we concatenate the files first and then carry on from step one

# Import necessary libraries

import pandas as pd

import numpy as np

# Load the first file

df1 = pd.read\_excel(r"C:\Users\warra\Desktop\Freelance\data\data\MaleMerge\14. Section 3 Part 1 Farm Asset inventory\2013\_s3p1\_m.xlsx")

# Load the second file

df2 = pd.read\_excel(r"C:\Users\warra\Desktop\Freelance\data\data\MaleMerge\14. Section 3 Part 1 Farm Asset inventory\2013\_s3p1p2\_q10\_m.xlsx")

# Merge the two files based on a common column

merged\_df = pd.concat([df1, df2], ignore\_index=True)

# Decision: Using pd.concat to concatenate dataframes vertically (row-wise) to unify two separate files of the same year.

# Drop redundant columns

merged\_df.drop(merged\_df.columns[merged\_df.columns.str.contains('Unnamed', case=True)], axis=1, inplace=True)

merged\_df.drop(merged\_df.columns[merged\_df.columns.str.contains(' ', case=False)], axis=1, inplace=True)

# Decision: Dropping columns with 'Unnamed' and spaces to clean the dataframe, ensuring no irrelevant columns are included.

# Save the merged dataframe to a CSV file

merged\_df.to\_csv('2013\_s3p1.csv', index=True)

# Decision: Saving the cleaned and merged dataframe to a CSV file for consistent format handling in subsequent operations.

# This code block stores file paths to variables to make the code neat

import pandas as pd

import numpy as np

# Store Excel file locations to variables

agri\_2012 = r"C:\Users\warra\Desktop\Freelance\data\data\MaleMerge\14. Section 3 Part 1 Farm Asset inventory\2012\_s3p1\_m.xlsx"

agri\_2012\_5 = r"C:\Users\warra\Desktop\Freelance\data\data\MaleMerge\14. Section 3 Part 1 Farm Asset inventory\2012\_5\_s3p1.xlsx"

agri\_2013 = r"C:\Users\warra\Desktop\Freelance\data\data\MaleMerge\14. Section 3 Part 1 Farm Asset inventory\2013\_s3p1.csv"

agri\_2014 = r"C:\Users\warra\Desktop\Freelance\data\data\MaleMerge\14. Section 3 Part 1 Farm Asset inventory\2014\_s3p1\_m.xlsx"

# Read Excel files

df\_2012 = pd.read\_excel(agri\_2012)

df\_2012\_5 = pd.read\_excel(agri\_2012\_5)

df\_2013 = pd.read\_csv(agri\_2013)

df\_2014 = pd.read\_excel(agri\_2014)

# Decision: Reading data from multiple sources into pandas dataframes for uniform processing.

# This code block will be used to standardize column names across the years to avoid discrepancies during the merging process.

# Rename columns in df

df\_2012.rename(columns={

'S3P1\_Assets': 's3p1\_sr',

'S3P1Q1': 's3p1\_q2',

'S3P1Q2': 's3p1\_q3',

'S3P1Q4': 's3p1\_q4',

'S3P1Q3': 's3p1\_q5',

'PROVINCE\_ID': 'P\_ID',

'DISTRICT\_ID': 'D\_ID',

'TEHSIL\_ID': 'T\_ID',

'UC\_ID': 'UC\_ID',

'MAUZA\_ID': 'M\_ID'

}, inplace=True)

df\_2012\_5.rename(columns={

'Round': 'round',

'S3P1\_CODE': 's3p1\_sr',

'S3P1\_NAME': 's3p1\_name',

'S3P1Q1': 's3p1\_q3',

'S3P1Q4': 's3p1\_q4',

'S3P1Q3': 's3p1\_q5',

'C\_PROVINCE': 'P\_ID',

'C\_DISTRICT': 'D\_ID',

'C\_TEHSIL': 'T\_ID',

'C\_UC': 'UC\_ID',

'C\_MOUZA': 'M\_ID',

'S3P1Q2': 'Ass\_Y\_OldestU',

'S3P1Q5\_CAPACITY': 'Ass\_EP\_Cap',

'S3P1Q5\_UNIT': 'Ass\_EP\_U'

}, inplace=True)

df\_2013.rename(columns={

'hid': 'hid',

'round': 'round',

's3p1\_sr': 's3p1\_sr',

's3p1\_asset': 's3p1\_name',

's3p1\_q2': 's3p1\_q2',

's3p1\_q4': 's3p1\_q3',

's3p1\_q5': 's3p1\_q4',

's3p1\_q10': 's3p1\_q5',

's3p1\_q6': 's3p1\_q6',

's3p1\_q7': 's3p1\_q7',

's3p1\_q8': 's3p1\_q8',

's3p1\_q9': 's3p1\_q9',

's3p1\_q3': 'Ass\_ALL\_Part'

}, inplace=True)

# Decision: Standardizing column names across dataframes to ensure uniformity during the merge process. Different datasets have varying column names, hence renaming is crucial to avoid mismatches.

# Mapping lists for the given datasets:

mapping\_2012 = [

"hid", "round", "s3p1\_sr", "None", "None", "s3p1\_q2", "s3p1\_q3", "s3p1\_q4",

"s3p1\_q5", "None", "None", "None", "None", "None", "P\_ID", "D\_ID", "T\_ID",

"UC\_ID", "M\_ID", "None", "None", "None", "None"

]

mapping\_2012\_5 = [

"hid", "round", "s3p1\_sr", "s3p1\_name", "None", "None", "s3p1\_q3", "s3p1\_q4",

"s3p1\_q5", "None", "None", "None", "None", "None", "P\_ID", "D\_ID", "T\_ID", "UC\_ID",

"M\_ID", "C\_HH\_NUM", "Ass\_Y\_OldestU", "Ass\_EP\_Cap", "Ass\_EP\_U"

]

mapping\_2013 = [

"hid", "round", "s3p1\_sr", "s3p1\_name", "None", "s3p1\_q2", "s3p1\_q3", "s3p1\_q4",

"s3p1\_q5", "s3p1\_q6", "s3p1\_q7", "s3p1\_q8", "s3p1\_q9", "Ass\_ALL\_Part", "None",

"None", "None", "None", "None", "None", "None", "None", "None"

]

mapping\_2014 = [

"hid", "round", "s3p1\_sr", "s3p1\_name", "s3p1\_q1", "s3p1\_q2", "s3p1\_q3", "s3p1\_q4",

"s3p1\_q5", "s3p1\_q6", "s3p1\_q7", "s3p1\_q8", "s3p1\_q9", "None", "None", "None",

"None", "None", "None", "None", "None", "None", "None"

]

# Decision: Mapping each year's columns to a standardized set of column names. 'None' is used to represent missing columns for certain datasets.

# Create a list of all possible columns in the correct order

all\_columns = []

for col in mapping\_2012:

if col and col not in all\_columns:

all\_columns.append(col)

for col in mapping\_2012\_5:

if col and col not in all\_columns:

all\_columns.append(col)

for col in mapping\_2013:

if col and col not in all\_columns:

all\_columns.append(col)

for col in mapping\_2014:

if col and col not in all\_columns:

all\_columns.append(col)

# Decision: Creating a comprehensive list of all unique columns from each dataset to ensure the merged dataframe captures all relevant data.

# Function to standardize and merge DataFrames

def standardize\_and\_merge(dfs, mappings, all\_columns):

merged\_data = {col: [] for col in all\_columns}

for df, mapping in zip(dfs, mappings):

print(f"Processing DataFrame with columns: {df.columns.tolist()}")

for i, col in enumerate(mapping):

if col:

ref\_col = col

if ref\_col not in merged\_data:

merged\_data[ref\_col] = []

if col in df.columns:

print(f"Appending data for column {col}")

merged\_data[ref\_col].extend(df[col].tolist())

else:

print(f"Column {col} not found in DataFrame. Adding NaNs.")

merged\_data[ref\_col].extend([np.nan] \* len(df))

# Decision: Iterate through each dataframe and mapping, appending data to the corresponding column in the merged\_data dictionary. NaNs are added for missing columns to maintain consistency.

max\_len = max(len(v) for v in merged\_data.values())

for key in merged\_data:

col\_len = len(merged\_data[key])

if col\_len < max\_len:

merged\_data[key].extend([np.nan] \* (max\_len - col\_len))

# Decision: Ensure all columns in merged\_data have equal length by appending NaNs where necessary.

merged\_df = pd.DataFrame.from\_dict(merged\_data)

return merged\_df

# List of dataframes and their respective mappings

dfs = [df\_2012, df\_2012\_5, df\_2013, df\_2014]

mappings = [mapping\_2012, mapping\_2012\_5, mapping\_2013, mapping\_2014]

# Standardize and merge the dataframes

merged\_df = standardize\_and\_merge(dfs, mappings, all\_columns)

# Rename columns for the merged file (if needed)

rename\_mapping = {

'hid': 'HID',

'round': 'Survey\_Round',

's3p1\_sr': 'Ass\_C',

's3p1\_name': 'Ass\_N',

's3p1\_q1': 'Ass\_PR',

's3p1\_q2': 'Ass\_CR',

's3p1\_q3': 'Ass\_HH\_no',

's3p1\_q4': 'Ass\_Val\_T',

's3p1\_q5': 'Ass\_Rep',

's3p1\_q6': 'Ass\_S',

's3p1\_q7': 'Ass\_S\_Val',

's3p1\_q8': 'Ass\_P',

's3p1\_q9': 'Ass\_P\_Val'

}

# Decision: Final renaming to standardize column names in the merged dataframe according to the provided schema.

merged\_df.rename(columns=rename\_mapping, inplace=True)

# Save the merged dataframe to a CSV file

merged\_df.to\_csv('merged\_Section\_3\_part\_1.csv', index=False)

# Decision: Saving the final standardized and merged dataframe to a CSV file for further analysis or use.

### Key Decision Points

1. **Reading Files:** Using pd.read\_excel and pd.read\_csv to load data into dataframes, facilitating easy manipulation and merging.
2. **Concatenating Data:** pd.concat is used to combine data from two files of the same year, ensuring all relevant data is included.
3. **Dropping Redundant Columns:** Removing columns with 'Unnamed' and spaces to clean the data, avoiding irrelevant columns in the final dataset.
4. **Standardizing Column Names:** Renaming columns in each dataframe to a common schema to ensure consistency during merging.
5. **Mapping Columns:** Creating mappings for each year's dataset to standardize column names and handle missing columns.
6. **Building a Comprehensive Column List:** Ensuring all potential columns across datasets are included in the final merged dataframe.
7. **Merging Dataframes:** Using a function to append data to a dictionary based on standardized column names, adding NaNs for missing columns to maintain consistency.
8. **Renaming Final Columns:** Applying a final renaming to the merged dataframe for uniform column names as per the provided schema.
9. **Saving the Final Dataset:** Exporting the cleaned and standardized dataframe to a CSV file for further use.

# Code Documentation (Section-By-Section Merging and Harmonization):

The following sections outline the process of merging multiple CSV files based on specified keys. Each section handles the merge at different levels of detail (e.g., household, plot, season).

Level 1: Household - Household Member Level

# Define the folder path where the data files are located

data\_folder = r"C:\Users\warra\Desktop\Freelance\data\data\MaleMerge\2. ALL MERGED CSV Parts"

# List of all the file names to be merged

file\_names = ['1. merged\_roster.csv']

Decision: Specify the directory containing the data files and list the files to be processed for merging.

import pandas as pd

import os

Decision: Import necessary libraries for data manipulation and file handling.

# Initialize an empty DataFrame to hold the merged data

merged\_df = pd.DataFrame()

Decision: Create an empty DataFrame to store the merged data.

# Iterate over each file in the list, read it, and concatenate it to the merged DataFrame

for file\_name in file\_names:

file\_path = os.path.join(data\_folder, file\_name) # Create the full file path

print(f"Looking for file: {file\_path}") # Debug print

if os.path.exists(file\_path):

print(f"File found: {file\_path}") # Debug print

df = pd.read\_csv(file\_path, dtype=str) # Read the CSV file with all columns as strings to handle mixed data types

merged\_df = pd.concat([merged\_df, df], ignore\_index=True) # Concatenate the current file's data to the merged DataFrame

else:

print(f"File not found: {file\_path}") # Debug print

Decision: Loop through the list of files, check if each file exists, read the file if found, and concatenate it to the merged DataFrame. Reading columns as strings ensures consistent data types.

# Proceed only if merged\_df is not empty

if not merged\_df.empty:

# Define the merge keys

merge\_keys = ['HID', 'PID']

# Filter out merge keys that are not in the DataFrame

available\_merge\_keys = [key for key in merge\_keys if key in merged\_df.columns]

# Reorder columns to have available merge keys first and then the rest alphabetically

remaining\_columns = [col for col in merged\_df.columns if col not in available\_merge\_keys] # Get columns excluding available merge keys

# Sort the remaining columns alphabetically; put columns starting with digits at the end

sorted\_columns = sorted(remaining\_columns, key=lambda x: (x[0].isdigit(), x))

# Define the final column order with available merge keys first

column\_order = available\_merge\_keys + sorted\_columns

# Reorder the DataFrame columns according to the defined order

merged\_df = merged\_df[column\_order]

# Remove columns that are entirely empty

merged\_df.dropna(axis=1, how='all', inplace=True)

# Remove rows that are entirely empty

merged\_df.dropna(axis=0, how='all', inplace=True)

# Save the merged and processed data to a new CSV file

merged\_df.to\_csv('1. Roaster\_Household-HouseholdMember\_Level.csv', index=False)

print("Merged data saved successfully!")

else:

print("No data to merge and save.")

Decision: Identify and keep only the columns defined by merge keys. Alphabetically sort other columns, giving precedence to columns starting with letters over those starting with digits. Drop entirely empty columns and rows to clean up the DataFrame before saving it to a new CSV file.

Keys used (HID, PID) to handle merging at a detailed level. This ensures that the merged data wrt Household ID and Person ID.

**Level 2: Household-Household Member-Plot-Crop-Season Level**

# Initialize an empty DataFrame to hold the merged data

merged\_df = pd.DataFrame()

# Iterate over each file in the list, read it, and concatenate it to the merged DataFrame

for file\_name in file\_names:

file\_path = os.path.join(data\_folder, file\_name) # Create the full file path

print(f"Looking for file: {file\_path}") # Debug print

if os.path.exists(file\_path):

print(f"File found: {file\_path}") # Debug print

df = pd.read\_csv(file\_path, dtype=str) # Read the CSV file with all columns as strings to handle mixed data types

merged\_df = pd.concat([merged\_df, df], ignore\_index=True) # Concatenate the current file's data to the merged DataFrame

else:

print(f"File not found: {file\_path}") # Debug print

# Proceed only if merged\_df is not empty

if not merged\_df.empty:

# Define the merge keys

merge\_keys = ['HID', 'PID', 'Plt\_ID', 'Crop\_ID', 'Season']

# Filter out merge keys that are not in the DataFrame

available\_merge\_keys = [key for key in merge\_keys if key in merged\_df.columns]

# Reorder columns to have available merge keys first and then the rest alphabetically

remaining\_columns = [col for col in merged\_df.columns if col not in available\_merge\_keys] # Get columns excluding available merge keys

# Sort the remaining columns alphabetically; put columns starting with digits at the end

sorted\_columns = sorted(remaining\_columns, key=lambda x: (x[0].isdigit(), x))

# Define the final column order with available merge keys first

column\_order = available\_merge\_keys + sorted\_columns

# Reorder the DataFrame columns according to the defined order

merged\_df = merged\_df[column\_order]

# Remove columns that are entirely empty

merged\_df.dropna(axis=1, how='all', inplace=True)

# Remove rows that are entirely empty

merged\_df.dropna(axis=0, how='all', inplace=True)

# Save the merged and processed data to a new CSV file

merged\_df.to\_csv('8.1. EconomicEventsAndShocks\_Household-HouseholdMember-PlotID-CropID-Season\_Level.csv', index=False)

print("Merged data saved successfully!")

else:

print("No data to merge and save.")

Decision: Perform similar steps as before but include additional keys (Plt\_ID, Crop\_ID, Season) to handle merging at a more detailed level. This ensures that the merged data respects the granularity of plots, crops, and seasons.

**Level 3: Household – Season – Shock Type Level**

# Initialize an empty DataFrame to hold the merged data

merged\_df = pd.DataFrame()

# Iterate over each file in the list, read it, and concatenate it to the merged DataFrame

for file\_name in file\_names:

file\_path = os.path.join(data\_folder, file\_name) # Create the full file path

print(f"Looking for file: {file\_path}") # Debug print

if os.path.exists(file\_path):

print(f"File found: {file\_path}") # Debug print

df = pd.read\_csv(file\_path, dtype=str) # Read the CSV file with all columns as strings to handle mixed data types

merged\_df = pd.concat([merged\_df, df], ignore\_index=True) # Concatenate the current file's data to the merged DataFrame

else:

print(f"File not found: {file\_path}") # Debug print

# Proceed only if merged\_df is not empty

if not merged\_df.empty:

# Define the merge keys

merge\_keys = ['HID', 'Season', 'Shock\_ID']

# Filter out merge keys that are not in the DataFrame

available\_merge\_keys = [key for key in merge\_keys if key in merged\_df.columns]

# Reorder columns to have available merge keys first and then the rest alphabetically

remaining\_columns = [col for col in merged\_df.columns if col not in available\_merge\_keys] # Get columns excluding available merge keys

# Sort the remaining columns alphabetically; put columns starting with digits at the end

sorted\_columns = sorted(remaining\_columns, key=lambda x: (x[0].isdigit(), x))

# Define the final column order with available merge keys first

column\_order = available\_merge\_keys + sorted\_columns

# Reorder the DataFrame columns according to the defined order

merged\_df = merged\_df[column\_order]

# Remove columns that are entirely empty

merged\_df.dropna(axis=1, how='all', inplace=True)

# Remove rows that are entirely empty

merged\_df.dropna(axis=0, how='all', inplace=True)

# Save the merged and processed data to a new CSV file

merged\_df.to\_csv('8.2. EconomicEventsAndShocks\_Household-Season-ShockType\_Level.csv', index=False)

print("Merged data saved successfully!")

else:

print("No data to merge and save.")

Decision: Apply the same logic but with a different set of merge keys (HID, Season, Shock\_ID) to capture data at the household, season, and shock type level. This ensures that all relevant details are preserved in the merged data.

## Conclusion

* **Key Decisions Across Sections**:
  + Consistently read files and concatenate data to ensure no information is lost.
  + Clean and standardize data by removing empty columns and rows.
  + Sort and reorder columns to maintain a logical structure.
  + Save the processed data to CSV files for further use.
* **Goal**: Ensure that the merged datasets are comprehensive, clean, and ready for analysis while maintaining the integrity and granularity of the original data.