Code Documentation

1. Preparation

```
import os
import re
import pandas as pd
import numpy as np
import difflib
```

```
OUTPUT_DIR = '../output_data'
```

1.A. Data Input and Output

```
JMP_INPUT_FILE = '../input_data/JMP/jmp.csv'
JMP_OUTPUT_FILE = f'{OUTPUT_DIR}/table_jmp.csv'
IFS_INPUT_DIR = '../input_data/IFs'
IFS_OUTPUT_FILE = f'{OUTPUT_DIR}/table_ifs.csv'
```

1.B. Common Functions

Common functions are a collection of functions used by both data sources (IFS and JMP).

- **merge_id**: This function merges two data tables based on a common column replaces missing values with 0, and renames the column for easier identification.
- **cleanup_semicolon**: Replaces all occurrences of semicolons (;) with an empty string, cleaning up the extra characters that have been included in the Excel format from IFS.
- **replace_bs_with_alb**: BS needs to be updated to ALB for consistency or clarification.

```
def merge_id(prev_table, keys_table, name):
    merged_df = prev_table.merge(keys_table, left_on=name, right_on=name, how='left')
    merged_df = merged_df.rename(columns={'id': f'{name}_id'})
    merged_df = merged_df.drop(columns=[name])
    merged_df[f'{name}_id'] = merged_df[f'{name}_id'].where(merged_df[f'{name}_id'].notna(), 0).astype(int)
    return merged_df
```

```
def cleanup_semicolon(source):
    with open(source, 'r') as file:
        content = file.read()
    updated_content = content.replace(';', '')
    with open(source, 'w') as file:
        file.write(updated_content)
```

```
def replace_bs_with_alb(x):
    return "ALB" if x == "BS" else x
```

1.C. Key Table Generator

This function generates a unique key table for a specified column from both IFS and JMP table, saving the keys to a CSV file. If the CSV file already exists, it appends new values to the existing file while ensuring unique IDs for each entry.

```
def create_table_key(dataframe, column):
    file_path = f'{OUTPUT_DIR}/key_{column}.csv'
    new_table = pd.DataFrame(
        dataframe[column].unique(),
        columns=[column]
    ).dropna().sort_values(column).reset_index(drop=True)

    # If the file already exists, load it
    if os.path.exists(file_path):
        existing_table = pd.read_csv(file_path)
        # Find the new values that are not in the existing table
        new_values = new_table[~new_table[column].isin(existing_table[column])]
        if not new_values.empty:
            # Assign IDs to the new values, starting after the max existing ID
            max_id = existing_table['id'].max()
            new_values['id'] = range(max_id + 1, max_id + 1 + len(new_values))
            # Append the new values to the existing table
            updated_table = pd.concat([existing_table, new_values], ignore_index=True)
        else:
            updated_table = existing_table  # No new values to add, keep existing table as is
    else:
        # If the file doesn't exist, create new IDs starting from 1
        new_table['id'] = range(1, len(new_table) + 1)
        updated_table = new_table
    updated_table[['id', column]].to_csv(file_path, index=False)
    return updated_table
```

1.D. Country Mapping

This section compares two lists of country names—jmp_country_list from the JMP dataset and ifs_country_list from the IFS dataset—and finds the closest matches using string similarity. It also includes a mapping for countries with naming differences between the two lists.

```
data_jmp = pd.read_csv(JMP_INPUT_FILE, encoding='latin-1')
```

```
jmp_country_list = list(data_jmp["COUNTRY, AREA OR TERRITORY"].unique())
ifs_country_list = ['All countries WHHS Tool1','Congo Dem. Republic of the','Ethiopia','Ghana','Guatemala','Haiti','In
                    'Indonesia','Kenya','Liberia','Madagascar','Malawi','Mali','Mozambique','Nepal','Nigeria','Philippines',
                    'Rwanda','Senegal','Sudan South','Tanzania','Uganda','Zambia']
```

```
# Find the closest match
for country in ifs_country_list:
    probability = difflib.get_close_matches(country, jmp_country_list, n=3, cutoff=0.4)
    if probability:
        if country not in probability:
            print(f"{country} -> {list(probability)}")
    else:
        print(f"NOT FOUND: {country}")
```

```
NOT FOUND: All countries WHHS Tool1
Congo Dem. Republic of the -> ['Democratic Republic of the Congo', 'Republic of Korea', 'Iran (Islamic Republic of)'
Sudan South -> ['Sudan', 'San Marino', 'South Sudan']
Tanzania -> ['Panama', 'Canada', 'Mauritania']
```

```
country_mapping = {
    "All countries WHHS Tool1": "All Countries",
    "United Republic of Tanzania": "Tanzania",
    "Congo Dem. Republic of the": "Democratic Republic of the Congo",
    "Sudan South": "South Sudan",
}
```

```
def map_country_name(country):
    return country_mapping.get(country, country)
```

3. IFS Dataset

```
final_columns = ['indicator','year','country','unit','value_name','jmp_category','commitment','value']
```

```
files_to_keep = [
    "01. Deaths by Category of Cause - Millions (2nd Dimensions = Diarrhea).csv",
    "06. Poverty Headcount less than $2.15 per Day, Log Normal - Millions.csv",
    "08. State Failure Instability Event - IFs Index.csv",
    "11. Governance Effectiveness - WB index.csv",
```

```
    # "12. Value Added by Sector, Currency - Billion dollars.csv",
    "13. Sanitation Services, Access, percent of population (2nd Dimensions = Basic + Safely Managed).csv",
    "14. Sanitation Services, Access, Number of people, million (2nd Dimensions = Basic + Safely Managed).csv",
    "15. Sanitation Services, Expenditure, Capital, Billion $ (2nd Dimensions = Basic + Safely Managed).csv",
    "16. Sanitation Services, Expenditure, Maintenance, Billion $ (2nd Dimensions = Basic + Safely Managed).csv",
    "17. Water Services, Access, percent of population (2nd Dimension = Basic + Safely Managed).csv",
    "18. Water Services, Access, Number of people, million (2nd Dimensions = Basic + Safely Managed).csv",
    "19. Water Services, Expenditure, Capital, Billion $ (2nd Dimensions = Basic + Safely Managed).csv",
    "20. Water Services, Expenditure, Maintenance, Billion $ (2nd Dimensions = Basic + Safely Managed).csv",
    # "21. Population - Millions.csv",
    "23. GDP (PPP) - Billion dollars.csv",
    "24. Stunted children, History and Forecast - Million.csv",
    # "25. Population under 5 Years, Headcount - Millions.csv",
    "26. Malnourished Children, Headcount - Millions.csv"
]
year_filter_config = {
    "year_range": {
        "years": list(range(2019, 2050)),
        "files": [
            "13. Sanitation Services, Access, percent of population (2nd Dimensions = Basic + Safely Managed).csv",
            "17. Water Services, Access, percent of population (2nd Dimension = Basic + Safely Managed).csv"
        ]
    },
    "milestone_years": [2030, 2050]
}
```

```
files = [
    f"{IFS_INPUT_DIR}/{f}" for f in os.listdir(IFS_INPUT_DIR)
    if os.path.isfile(os.path.join(IFS_INPUT_DIR, f))
]
files = [f"{IFS_INPUT_DIR}/{file}" for file in files_to_keep]
```

3.A. IFS Functions

IFS functions are a collection of functions used only by IFS data source

- **base_jmp_category**: This function assigns or updates the JMP category based on specific conditions in the input data, particularly for records where the value is base; It converts certain base categories to simplified abbreviations ("BS" or "SM"), otherwise; retains the existing category
- **get_ifs_name**: This function extracts and cleans the name of an IFS data file by removing unwanted text and formatting, such as numbering, directory paths, and file extensions.
- **get_value_types**: This function processes a string by manipulating its structure to generate a list of values based on certain patterns. But it also replaces occurrences of '*0*' with '_0.' in the string, since '_0_5' is '0.5'.

- **cleanup_data**: This function cleans a DataFrame by removing unnecessary parts of the text in the "unit" column and ensuring consistency in the "value" column. Specifically, it unifies the unit formatting by removing "2017" from units like "Billion 2017" and handles space and empty value issues in the "value" column.
- **filter_dataframe_by_year**: This function filters a DataFrame based on a year configuration, depending on the config in previous section. It checks the configuration to determine whether to filter by a specific year range or milestone years.
- **remove_unmatches_jmp_category**: This function identifies rows in a dataset where the JMP category ("jmp_category") does not match the base category ("category_base"), according to specific rules. It returns True for rows where the mismatch occurs, indicating that the row should be removed.
- **remove_unmatch_commitment**: This function identifies rows in a dataset where the "commitment" year does not match the actual "year" of the data. It returns True for rows where the mismatch occurs, indicating that the row should be removed.

```
def base_jmp_category(x):
    if x["value_name"] == "Base":
        if x["category_base"] == "Basic":
            return "BS"
        if x["category_base"] == "SafelyManaged":
            return "SM"
        return np.nan
    return x["jmp_category"]
```

```
def get_ifs_name(source):
    source = re.sub(r"\s*\(2nd Dimension.*?\)", "", source)
    return re.sub(r'^\d+\. ', '', source.replace(f"{IFS_INPUT_DIR}/", "")).replace(".csv", "")
```

```
def get_value_types(lst):
    lst = lst.split('.')[0]
    lst = lst.replace('_0_','_0.').split("_")
    return lst
```

```
def cleanup_data(dataframe):
    dataframe['unit'] = dataframe['unit'].apply(lambda x: x.replace("2017","") if x else None)
    dataframe['value'] = dataframe['value'].apply(lambda x: x.replace(' ','') if ' ' in str(x) else x)
    dataframe['value'] = dataframe['value'].apply(lambda x: x if len(str(x)) > 0 else np.nan)
```

```
def filter_dataframe_by_year(dataframe, filename):
    filename = filename.split("/")[3]
    if filename in year_filter_config["year_range"]["files"]: # Filter using the year_range
```

```
        filtered_df = dataframe[dataframe['year'].isin(year_filter_config["year_range"]["years"])]
    else: # Filter using milestone_years
        filtered_df = dataframe[dataframe['year'].isin(year_filter_config["milestone_years"])]
    return filtered_df.reset_index(drop=True)
```

```
def remove_unmatches_jmp_category(x):
    if x["value_name"] != "Base":
        if x["category_base"] == "Basic" and x["jmp_category"] == "SM":
            return True
        if x["category_base"] == "SafelyManaged" and x["jmp_category"] == "BS":
            return True
    return False
```

```
def remove_unmatch_commitment(x):
    if str(x['commitment']).strip() == "2030":
        if str(x["year"]) != "2030":
            return True
    if str(x['commitment']).strip() == "2050":
        if str(x["year"]) != "2050":
            return True
    return False
```

3.B. IFS Data Processing

3.B.1. Combine, Filter and Remap IFS Values

This section describes the process of transforming and processing IFS data files into a unified DataFrame (combined_df). The transformation involves cleaning, reshaping, and filtering the data to prepare it for analysis.

```
combined_df = pd.DataFrame(columns=final_columns)
for file in files:
    print(f"Processing {file}")
    # test only 1 file
    # if file != "../input_data/IFs/14. Sanitation Services, Access, Number of people, million (2nd Dimensions =
Basic + Safely Managed).csv":
    #   continue
    cleanup_semicolon(file)
```

```
    data = pd.read_csv(file, header=[1,2,4,5], sep=',')
    new_columns = list(data.columns)
    for i, col in enumerate(new_columns):
        if col == ('Unnamed: 0_level_0', 'Unnamed: 0_level_1', 'Unnamed: 0_level_2', 'Unnamed: 0_level_3'):
            new_columns[i] = 'Year'
    data.columns = new_columns
    df = pd.DataFrame(data.to_dict('records'))
    df_melted = df.melt(id_vars=['Year'], var_name='variable', value_name='value')
    new_data = []
    for value_list in df_melted.to_dict('records'):
        value_type = get_value_types(value_list["variable"][3])
        new_data.append({
            "year": int(value_list["Year"]),
            "country": map_country_name(value_list["variable"][0]),
            "category_base": value_list["variable"][1],
            "unit": value_list["variable"][2],
            "value_type": list(filter(lambda v:v,value_type)),
            "value": value_list["value"]
        })
    df = pd.DataFrame(new_data)
    df = filter_dataframe_by_year(df, file)
    df_split = pd.DataFrame(df['value_type'].tolist(), index=df.index)
    df_split.columns = ['value_name', 'jmp_category', 'commitment']
    df_final = pd.concat([df, df_split], axis=1)
    df_final['indicator'] = get_ifs_name(file)
    df_final['jmp_category'] = df_final.apply(base_jmp_category, axis=1)
    df_final['jmp_category'] = df_final['jmp_category'].apply(replace_bs_with_alb)
    df_final['remove'] = df_final.apply(remove_unmatches_jmp_category, axis=1)
    df_final = df_final[df_final['remove'] == False].reset_index(drop=True)
    df_final = df_final[final_columns]
    combined_df = pd.concat([combined_df.dropna(axis=1, how='all'), df_final], ignore_index=True)
```

```
Processing ../input_data/IFs/01. Deaths by Category of Cause - Millions (2nd Dimensions = Diarrhea).csv
Processing ../input_data/IFs/06. Poverty Headcount less than $2.15 per Day, Log Normal - Millions.csv
Processing ../input_data/IFs/08. State Failure Instability Event - IFs Index.csv
Processing ../input_data/IFs/11. Governance Effectiveness - WB index.csv
Processing ../input_data/IFs/13. Sanitation Services, Access, percent of population (2nd Dimensions = Basic + Safe
Processing ../input_data/IFs/14. Sanitation Services, Access, Number of people, million (2nd Dimensions = Basic +
Processing ../input_data/IFs/15. Sanitation Services, Expenditure, Capital, Billion $ (2nd Dimensions = Basic + Safe
Processing ../input_data/IFs/16. Sanitation Services, Expenditure, Maintenance, Billion $ (2nd Dimensions = Basic -
```

```
Processing ../input_data/IFs/17. Water Services, Access, percent of population (2nd Dimension = Basic + Safely Ma
Processing ../input_data/IFs/18. Water Services, Access, Number of people, million (2nd Dimensions = Basic + Safe
Processing ../input_data/IFs/19. Water Services, Expenditure, Capital, Billion $ (2nd Dimensions = Basic + Safely M
Processing ../input_data/IFs/20. Water Services, Expenditure, Maintenance, Billion $ (2nd Dimensions = Basic + Sa
Processing ../input_data/IFs/23. GDP (PPP) - Billion dollars.csv
Processing ../input_data/IFs/24. Stunted children, History and Forecast - Million.csv
Processing ../input_data/IFs/26. Malnourished Children, Headcount - Millions.csv
```

```
# Test for Congo
# combined_df[
#    (combined_df["indicator"] == "Sanitation Services, Access, Number of people, million") &
#    (combined_df["country"] == "Democratic Republic of the Congo") &
#    (combined_df["value_name"] == "Base")
#]
```

```
combined_df['remove'] = combined_df.apply(lambda x: remove_unmatch_commitment(x), axis=1)
combined_df = combined_df[combined_df['remove'] == False].reset_index(drop=True)
combined_df = combined_df.drop(columns=['remove'])
```

```
#combined_df[
#(combined_df['country'] == "Democratic Republic of the Congo")
#& (combined_df['indicator'] == "GDP (PPP) - Billion dollars")
#& (combined_df['commitment'] == "2030")
#]
```

3.B.2. IFS Data Cleanup

```
cleanup_data(combined_df)
combined_df.head()
```

	indicator	year	country	unit	value_name	jmp_category	commitment	value
0	Deaths by Category of Cause - Millions	2030	All Countries	Mil People	Base	NaN	None	1.237
1	Deaths by Category of Cause - Millions	2050	All Countries	Mil People	Base	NaN	None	1.143

	indicator	year	country	unit	value_name	jmp_category	commitment	value
2	Deaths by Category of Cause - Millions	2030	All Countries	Mil People	FS	ALB	2030	1.075
3	Deaths by Category of Cause - Millions	2050	All Countries	Mil People	FS	ALB	2050	1.068
4	Deaths by Category of Cause - Millions	2030	All Countries	Mil People	FS	SM	2030	0.955

3.C. IFS Table of Keys

3.C.1. Indicators

```
indicator_table = create_table_key(combined_df, 'indicator')
indicator_table
```

	id	indicator
0	1	Deaths by Category of Cause - Millions
1	2	GDP (PPP) - Billion dollars
2	3	Governance Effectiveness - WB index
3	4	Malnourished Children, Headcount - Millions
4	5	Poverty Headcount less than $2.15 per Day, Log...
5	6	Sanitation Services, Access, Number of people,...
6	7	Sanitation Services, Access, percent of popula...
7	8	Sanitation Services, Expenditure, Capital, Bil...

	id	indicator
8	9	Sanitation Services, Expenditure, Maintenance,...
9	10	State Failure Instability Event - IFs Index
10	11	Stunted children, History and Forecast - Million
11	12	Water Services, Access, Number of people, million
12	13	Water Services, Access, percent of population
13	14	Water Services, Expenditure, Capital, Billion $
14	15	Water Services, Expenditure, Maintenance, Bill...

3.C.2. Units

```
units_table = create_table_key(combined_df, 'unit')
units_table
```

	id	unit
0	1	Billion $
1	2	Index
2	3	Index 0-5
3	4	Mil People
4	5	Million
5	6	Percent
6	7	Trillion $

3.C.3. Value Names

```
value_names_table = create_table_key(combined_df, 'value_name')
value_names_table
```

	id	value_name
0	1	Base
1	2	FS
2	3	FW
3	4	FWS
4	5	SI
5	6	WI
6	7	WSI

3.C.4. JMP Categories

```
jmp_categories_table = create_table_key(combined_df, 'jmp_category')
jmp_categories_table
```

	id	jmp_category
0	1	ALB
1	2	SM

3.C.5. JMP Names Table (Custom)

```
jmp_names_table = pd.DataFrame([
    {"id": 1,"jmp_name": "Water"},
    {"id": 2,"jmp_name": "Sanitation"},
    {"id": 3,"jmp_name": "Water and Sanitation"}
])
jmp_names_table.to_csv(f'{OUTPUT_DIR}/key_jmp_name.csv',index=False)
```

3.C.6. Commitments

```
commitments_table = create_table_key(combined_df, 'commitment')
commitments_table
```

	id	commitment

0	1	0.5x
1	2	2030
2	3	2050
3	4	2x
4	5	4x
5	6	6x

3.C.7. Country

```
countries_table = create_table_key(combined_df, 'country')
countries_table
```

	id	country
0	1	All Countries
1	2	Democratic Republic of the Congo
2	3	Ethiopia
3	4	Ghana
4	5	Guatemala
5	6	Haiti
6	7	India
7	8	Indonesia
8	9	Kenya
9	10	Liberia
10	11	Madagascar
11	12	Malawi
12	13	Mali
13	14	Mozambique
14	15	Nepal
15	16	Nigeria
16	17	Philippines
17	18	Rwanda

	id	country
18	19	Senegal
19	20	South Sudan
20	21	Tanzania
21	22	Uganda
22	23	Zambia

3.D. IFS Table Results

3.D.1. Custom Table Mapping (JMP Name)

- FS = Full Sanitation Access
- FW = Full Water Access
- FWS = Full Water and Sanitation Access
- SI = Sanitation Increased
- WI = Water Increased
- WSI = Water and Sanitation Increased

```
jmp_dict = dict(zip(jmp_names_table['jmp_name'], jmp_names_table['id']))
```

```
def map_jmp_id(x):
    value_name = x["value_name"]
    # For the Base data
    if value_name == "Base":
        if x["indicator"].startswith("Water"):
            return jmp_dict['Water']
        if x["indicator"].startswith("Sanitation"):
            return jmp_dict['Sanitation']
    if 'W' in value_name and 'S' in value_name: # Water and Sanitation is indicated by 'WS' combined
        return jmp_dict['Water and Sanitation']
    if 'W' in value_name:  # Water is indicated by 'W'
        return jmp_dict['Water']
    if 'S' in value_name:  # Sanitation is indicated by 'S'
        return jmp_dict['Sanitation']
    return 0
```

```
combined_df['jmp_name_id'] = combined_df.apply(map_jmp_id, axis=1)
combined_df.tail(2)
```

	indicator	year	country	unit	value_name	jmp_category	commitment	value	jmp_name_id
34774	Malnourished Children, Headcount - Millions	2030	Zambia	Mil People	WSI	SM	6x	0.208	3
34775	Malnourished Children, Headcount - Millions	2050	Zambia	Mil People	WSI	SM	6x	0.143	3

3.D.2. IFS Key Table Mapping

```
table_with_id = merge_id(combined_df, indicator_table, 'indicator')
table_with_id = merge_id(table_with_id, units_table, 'unit')
table_with_id = merge_id(table_with_id, value_names_table, 'value_name')
table_with_id = merge_id(table_with_id, jmp_categories_table, 'jmp_category')
table_with_id = merge_id(table_with_id, commitments_table, 'commitment')
table_with_id = merge_id(table_with_id, countries_table, 'country')
```

3.D.3. IFS Final Result

```
table_with_id = table_with_id[table_with_id['value'].notna()].reset_index(drop=True)
table_with_id = table_with_id.sort_values(by='year').reset_index(drop=True)
table_with_id.reset_index(drop=True).tail()
```

	year	value	jmp_name_id	indicator_id	unit_id	value_name_id	jmp_category_id	commitment_id	country_id
34771	2050	1.406	1	14	1	3	1	3	23
34772	2050	1.539	3	5	4	7	1	5	18
34773	2050	2.348	1	14	1	3	2	3	23
34774	2050	1.097	1	5	4	6	2	6	18
34775	2050	0.143	3	4	4	7	2	6	23

3.D.2. Save IFS Table

```
table_with_id.to_csv(IFS_OUTPUT_FILE, index=False)
```

2. JMP Dataset

```
data = pd.read_csv(JMP_INPUT_FILE, encoding='latin-1')
data.head()
```

	COUNTRY, AREA OR TERRITORY	Year	Type	TOTAL - At least basic	TOTAL - Annual rate of change in \nat least basic	TOTAL - Safely managed	TOTAL - Annual rate of change in safely managed	TOTAL - Annual rate of change SM, manual calculation	TOTAL - Annual rate of change ALB, manual calculation
0	Afghanistan	2000	Water	27.4	2.5	11.1	0.9	-99.0	-99.0
1	Afghanistan	2001	Water	27.5	2.5	11.1	0.9	0.0	0.0
2	Afghanistan	2002	Water	29.7	2.5	12.0	0.9	0.9	2.2
3	Afghanistan	2003	Water	31.9	2.5	12.9	0.9	0.9	2.2
4	Afghanistan	2004	Water	34.1	2.5	13.8	0.9	0.9	2.2

2.A. JMP Data Processing

2.A.1. Rename the columns

```
data.columns = [
    'country',
    'year',
    'jmp_name',
    'total_ALB',
```

```
    'annual_rate_change_ALB',
    'total_SM',
    'annual_rate_change_SM',
    'manual_rate_change_SM',
    'manual_rate_change_ALB'
]
data.head()
```

	country	year	jmp_name	total_ALB	annual_rate_change_ALB	total_SM	annual_rate_change_SM	manual_rate_change_SM	manual_rate_change_ALB
0	Afghanistan	2000	Water	27.4	2.5	11.1	0.9	-99.0	-99.0
1	Afghanistan	2001	Water	27.5	2.5	11.1	0.9	0.0	0.0
2	Afghanistan	2002	Water	29.7	2.5	12.0	0.9	0.9	2.2
3	Afghanistan	2003	Water	31.9	2.5	12.9	0.9	0.9	2.2
4	Afghanistan	2004	Water	34.1	2.5	13.8	0.9	0.9	2.2

2.A.2. Categorize the Values

```
data_melted = pd.melt(
    data,
    id_vars=['country', 'year', 'jmp_name'],  # columns to keep
    var_name='variable',  # melted
    value_name='value' # values
)
data_melted['value_type'] = data_melted['variable'].apply(lambda x: 'total' if 'total' in x else 'annual_rate_change')
data_melted['jmp_category'] = data_melted['variable'].apply(lambda x: 'ALB' if 'ALB' in x else 'SM')
data_melted['jmp_category'] = data_melted['jmp_category'].apply(replace_bs_with_alb)
data_melted['country'] = data_melted['country'].apply(map_country_name)
data_melted = data_melted.drop(columns=['variable'])
data_melted.head()
```

	country	year	jmp_name	value	value_type	jmp_category
0	Afghanistan	2000	Water	27.4	total	ALB

	country	year	jmp_name	value	value_type	jmp_category
1	Afghanistan	2001	Water	27.5	total	ALB
2	Afghanistan	2002	Water	29.7	total	ALB
3	Afghanistan	2003	Water	31.9	total	ALB
4	Afghanistan	2004	Water	34.1	total	ALB

2.B. JMP Table Keys

2.B.1. JMP Categories (Retry)

```
jmp_categories_table = create_table_key(data_melted, 'jmp_category')
jmp_categories_table
```

	id	jmp_category
0	1	ALB
1	2	SM

2.B.2. JMP Value Types

```
value_types_table = create_table_key(data_melted, 'value_type')
value_types_table
```

	id	value_type
0	1	annual_rate_change
1	2	total

2.C. JMP Table Results

2.C.1. JMP Key Table Mapping

```
table_with_id = merge_id(data_melted, value_types_table, 'value_type')
table_with_id = merge_id(table_with_id, countries_table, 'country')
table_with_id = merge_id(table_with_id, jmp_names_table, 'jmp_name')
table_with_id = merge_id(table_with_id, jmp_categories_table, 'jmp_category')
```

2.C.2. JMP Data Cleanup

- Remove Nullable Country

```
table_with_id = table_with_id[table_with_id['country_id'] != 0].reset_index(drop=True)
```

2.C.3. JMP Final Result

```
table_with_id.head()
```

	year	value	value_type_id	country_id	jmp_name_id	jmp_category_id
0	2000	37.6	2	2	1	1
1	2001	37.5	2	2	1	1
2	2002	37.5	2	2	1	1
3	2003	37.4	2	2	1	1
4	2004	37.4	2	2	1	1

2.C.3. Save JMP Table

```
table_with_id.to_csv(JMP_OUTPUT_FILE, index=False)
```

Revision #4
Created 2 October 2024 07:02:38 by Deden Bangkit
Updated 2 October 2024 07:16:34 by Deden Bangkit