Public Health Database of Children Screened for Malnutrition
By: The Standard Deviants

Team Members:

Name	Email	UID
Bradley Walker	walkerspad@aol.com	u0483666
Monica Regulagadda	u1504835@umail.utah.edu	u1504835
Daniel Goldgar	u0414652@umail.utah.edu	u0414652
Mikayla Compton	u1538218@umail.utah.edu	u1538218
David Bean	u1526388@umail.utah.edu	u1526388

Github Repository:

https://github.com/David-Bean/DBMI-Wrangling-Group-3-Project/tree/main

Data:

Economic data will be primarily sourced from the World Bank Open Data repository at https://data.worldbank.org/

- GDP per capita
- Gini wealth coefficient
- Food production index
- Infant Mortality rate
- % Pregnant women receiving prenatal care
- Poverty headcount ratio
- Unemployment rate
- Literacy rate
- Primary school enrollment rate
- Mortality rate under-5
- Rural population
- Maternal mortality ratio
- Prevalence of Stunting
- Prevalence of Underweight

- Prevalence of wasting

Code:

For our project we used SQL, Python, and Jupyter Notebooks to create a database, combine our datasets and to perform the data cleaning.

Database(SQL):

```
--CREATE DATABASE childhood_malnutrition;
--for purposes of debugging recreate table
DROP TABLE IF EXISTS participant;
CREATE TABLE participant (
Country TEXT, --name of country where screening is taking place
Stake TEXT, --faith-based geographic name where screening is taking place
childIndex INTEGER, --each child screened is given an index ID within each "stake" or geographic
--or congregational unit where screening is taking place
stakedbname TEXT, --derived; is a combo of Country and Stake to use in data analysis
--one instead of 2 columns
screenCount INTEGER, --number of times child is screened with the highest number
--being the earliest screening and "1" being the last screening
gender TEXT, --Boy or Girl
lds TEXT, --religious self-identification
screenId TEXT, --appears to have been intended to be a primary key, but...
--where more than one screening entry took place on same day it did not increment
--so ended up NOT being a primary key; useful in identifying duplicate entries
screenDate TEXT, --self-explanatory; can be converted to just a YEAR-Month-Day format using DATE function
--Duplicates allowed/present; sometimes could be 2nd/3rd/higher height and weight obtained same day
--but probably more often is just unintentional double entry of the same screening data
weight NUMERIC, --child's weight in kg at screening
height NUMERIC, --child's height in cm at screening
age NUMERIC, --child's age in months at screening
obese TEXT, --true/false answer given based on weight per height (BMI); is derived
ha NUMERIC, --height z-score; is derived using height, weight, age, gender (DOB was removed
--from data for de-identification but can be calculated using age with some difficulty)
wa NUMERIC, --weight z-score; is derived
```

```
wh NUMERIC, --weight/height z-score also known as BMI
status TEXT, --nutritional status on date of screening as to whether needs
--treatment; is derived; does not define treatment see added column
malnutrition_classification
--for that
muac NUMERIC, --experimental to see if parents would be able to accurately
--calculate MUAC scores (they were not anywhere near competent to perform it)
PRIMARY KEY (Country, Stake, childIndex, screenCount)
);

--create additional participant columns for classification of row errors and malnutrition
status
ALTER TABLE participant ADD COLUMN row_error_type INTEGER;
ALTER TABLE participant ADD COLUMN malnutrition_classification TEXT;

--create malnutrition classification which is used to determine treatment for child
UPDATE participant SET malnutrition_classification = 'Normal';
UPDATE participant SET malnutrition_classification = 'At Risk'
WHERE wa < -1 OR ha < -1 OR wh < -1;
UPDATE participant SET malnutrition_classification = 'Stunted'
WHERE (wa < -2 OR ha < -2) AND wh < 0;
UPDATE participant SET malnutrition_classification = 'Severely Stunted'
WHERE (wa < -3 OR ha < -3) AND wh < 0;
UPDATE participant SET malnutrition_classification = 'MAM'
WHERE wh < -2;
UPDATE participant SET malnutrition_classification = 'SAM'
WHERE wh < -3;

--Identify and flag rows with nulls
SELECT * FROM participant WHERE stakedbname = ''; --No nulls
SELECT * FROM participant WHERE gender = ''; --No nulls
SELECT * FROM participant WHERE lds = '' OR lds = 'Unknown'; --7535
nulls/Unknowns
--here OK to use rows but eliminate rows if using for analysis of
--lds vs non-LDS OR in multivariate analysis
SELECT * FROM participant WHERE screenID = ''; --no nulls
SELECT * FROM participant WHERE weight = ''; --756 rows need to be flagged as
analysis not possible
SELECT * FROM participant WHERE height = ''; --757 nearly same rows as weight and
same flag applied
```

```
SELECT * FROM participant WHERE age = ''; --no nulls
SELECT * FROM participant WHERE obese = ''; --no nulls
SELECT * FROM participant WHERE ha = '' AND (row_error_type = 1 OR
row_error_type = -4);
--8 rows affected all of which are already flagged under poor Stake performace flag
--Answer for wa, wh exact same as for ha
--end of null classification SQL searches; rest derived or have nulls by design (muac)--

--set initial value for row_error_type;
UPDATE participant SET row_error_type = 1;
--classify row_error_type
--1=Normal,-1=Nulls,-2=Outliers,-3=Duplicates,-4=missing lds classification, -5=Haiti
Test data;
UPDATE participant SET row_error_type = -1 WHERE weight = '' OR height = '';
SELECT COUNT(*) FROM participant WHERE row_error_type = -1;  --758 rows
UPDATE participant SET row_error_type = -4 WHERE lds = '' OR lds = 'Unknown'
AND row_error_type = 1;
SELECT COUNT(*) FROM participant WHERE row_error_type = -4;  --2325 rows
instead of 2335
UPDATE participant SET row_error_type = -5 WHERE Stake = 'Haiti Test';
UPDATE participant SET row_error_type = -1 WHERE gender <> 'Boy' AND  gender <>
'Girl';
--See section on categorical outliers
SELECT COUNT(*) FROM participant WHERE row_error_type = -1;

--Identify incorrect or odd column values for categorical variables (or categorical
outliers)
SELECT Country, COUNT(*) FROM participant
WHERE row_error_type = 1 or row_error_type = -4 GROUP BY Country; -- no
unexpected values
SELECT Stake, COUNT(*) FROM participant
WHERE row_error_type = 1 or row_error_type = -4 GROUP BY Stake; --no unexpected
values
SELECT childIndex, COUNT(*) FROM participant
WHERE row_error_type = 1 or row_error_type = -4 GROUP BY childIndex; --no
unexpected values
SELECT stakedbname, COUNT(*) FROM participant
WHERE row_error_type = 1 or row_error_type = -4 GROUP BY stakedbname; --no
unexpected values
SELECT screenCount, COUNT(*) FROM participant
```

WHERE row_error_type = 1 or row_error_type = -4 GROUP BY screenCount; --no unexpected values
SELECT gender, COUNT(*) FROM participant
WHERE row_error_type = 1 or row_error_type = -4 GROUP BY gender; --see below to see unexpected values

--gender COUNT(*)

--Boy 90029
--Chico 3
--F 7
--Girl 88865
--Girl 1 Why is there a separate category for "Girl" with only 1 row? I don't know
--M 14 Maybe "white space"?

--Impossible to say if the errors caused miscalculation of Z-scores; will Flag as outliers
--(25 total rows)

SELECT lds, COUNT(*) FROM participant
WHERE row_error_type = 1 or row_error_type = -4 GROUP BY lds;

--lds COUNT(*)
-- 2301
--No 82735
--Unknown 5181 --will ad Unknown to above flag for lds "missing"
--Yes 88677

SELECT screenID, malnutrition_classification, COUNT(*) FROM participant
WHERE row_error_type = 1 or row_error_type = -4 GROUP BY screenID;
--9000 duplicated values of screenID, 8983 with 2 duplicates and 17 with 4 duplicates
--these are exact row duplicates of all values including to the second on the time EXCEPT
--for the stake and stakedbname. This is only possible with a very unusual episodic
--programming error or more worrisome, with fraud. What to say?
--All these rows (not just the minimum) will be marked as duplicates
--Query to update column row_error_type
UPDATE participant SET row_error_type = -3 WHERE screenID IN (
SELECT screenID FROM participant

```
WHERE row_error_type = 1 or row_error_type = -4 GROUP BY screenID
HAVING COUNT(*) > 1
);
--18,000+ (more than 10% of rows) removed for most analyses, although almost 100%
had
--outliers in addition to duplicates; they were all flagged to not use for analyses

SELECT DATE(screenDate), COUNT(*) FROM participant
WHERE row_error_type = 1 or row_error_type = -4 GROUP BY DATE(screenDate);
--no unexpected values

--WEIGHT AND HEIGHT (weight, height) are analyzed in their Z-score form (wa, ha,
wh)
--not separately from Z-scores

--age has many inappropriate entries as it was supposed to used only between
--6--60 months but was used in some cases the month before birth and after 60 months
--here 4290 rows were flagged for being outliers using directly below query
UPDATE participant SET row_error_type = -2 WHERE age IN
(SELECT age FROM participant
WHERE (row_error_type = 1 or row_error_type = -4)
AND (age < 6 OR age > 60) GROUP BY age ORDER BY age);

SELECT obese, COUNT(*) FROM participant GROUP BY obese;
--FALSE           139487
--TRUE            25890
--undefined   14316

--I was told obesity was never used or "fixed" but a quick look at it states it was
unusable
--Here we'll leave it alone and not flag rows based on its values as it is not used
--for any calculations

--set outlier flags for z-scores wa, ha, wh
UPDATE participant SET row_error_type =  -2 WHERE
wa < -5 OR wa > 2 OR ha < -5 OR ha > 2 OR wh < -5 OR wh > 2;

--Number of rows for each Country/Stake combination needed to perform calculations
--calculation of percentage of row removed for nulls, outliers, and duplicates
DROP TABLE IF EXISTS number_rows;
```

```
CREATE TABLE number_rows(
Country text,
Stake text,
number_rows_Country_Stake NUMERIC
);
INSERT INTO number_rows SELECT Country, Stake, COUNT(*) AS
number_rows_Country_Stake
FROM participant GROUP BY Country, Stake;
SELECT * FROM number_rows;

DROP TABLE IF EXISTS number_rows1;
CREATE TABLE number_rows1(
Country text,
Stake text,
number_rows_Country_Stake1 NUMERIC
);
INSERT INTO number_rows1 SELECT Country, Stake, COUNT(*) AS
number_rows_County_Stake1
FROM participant WHERE  row_error_type = 1 or row_error_type = -4 GROUP BY
Country, Stake;
SELECT * FROM number_rows1;

--Calculate error rates in each stake geographic location and
--flag entire stake as outlier if error rates > 40% and don't use
--data from those stakes for analyses
DROP TABLE IF EXISTS temp;
CREATE TABLE IF NOT EXISTS temp AS SELECT participant.Country,
participant.Stake,
number_rows_Country_Stake,number_rows_Country_Stake1,
(number_rows_Country_Stake1 * 1.0)/(number_rows_Country_Stake * 1.0)
AS non_error_rate_stake FROM participant, number_rows, number_rows1
WHERE participant.Country = number_rows.Country
AND participant.Stake = number_rows.Stake
AND participant.Country = number_rows1.Country
AND participant.Stake = number_rows1.Stake
GROUP BY participant.Country, participant.Stake;


SET row_error_type = -2 FROM participant WHERE Stake IN
(SELECT Stake FROM temp WHERE non_error_rate < 0.60);
```

```
SELECT * FROM temp; --data not copied here from the query b/c 243 rows;
--could see by executing code

--1=Normal,-1=Nulls,-2=Outliers,-3=Duplicates,-4=missing lds classification, -5=Haiti
Test data;
SELECT row_error_type, COUNT(*) FROM participant GROUP BY row_error_type
ORDER BY row_error_type DESC;
--row_error_type      COUNT(*)
--        1           108055
--       -1           3
--       -2           64330
--       -3           1972
--       -4           5327
--       -5           6
--Number of usable rows = 108,055 + 5327 = 113,382
--Total number of rows = 179,693
--Percent of rows usable = 113,382/179,693 X 100 = 63.1%
--Of outliers(-2), 23,410 were too "high", and of those 23,410,
--8212 rows also had a value too "low" (or "double wrong")
--The biggest issue was obtaining accurate heights and weights by far
--and that issue should be am emphasis in the future;
--having height/weights obtained by 2 people is that start of fixing this issue
--Just as important would be flagging the row during data capture and forcing
--re-measurements rather than allowing entry of outliers
--This advice is also important for nulls and duplicates (duplicates within same Stake)
--The rows should also list who input the data and re-train or disallow individuals
--who do not demonstrate technical proficiency in capture and data entry of
height/weight
--It's better to prevent entry of incorrect data than fix it
--in this case it can't be fixed post-hoc and I assume that's often the case
--The methodology used to analyze the dataset without data wrangling was to simply
select
--the data points where values were outliers.  This was quite similar to what the
wrangling
--assignment produced, so the results are quite similar to the original results.
--As for the initial question of whether this dataset needed wrangling,
--it should be obvious that the answer was yes.

--Identification and flagging of row with nulls/outliers/duplicates is finished
```

```
--EDA will continue 1st creating percentiles on Z-scores/means on Z-scores wa, ha, wh
--Then examine the initial status of the children and where available response to
treatment

DROP TABLE IF EXISTS temp;
CREATE TABLE temp(
percentile text,
wa numeric
);
INSERT INTO temp SELECT 'placeholder', wa FROM participant WHERE
row_error_type = 1 or row_error_type = -4 ORDER BY wa;

DROP TABLE IF EXISTS temp1;
CREATE TABLE temp1(
percentile text,
wa numeric
);

--Calculate percentiles and averages for Z-scores wa, ha, wh(bmi)
INSERT INTO temp1 SELECT '1st percentile or MIN', MIN(wa) FROM temp;
INSERT INTO temp1 SELECT '20th percentile', wa FROM temp WHERE rowid =
round(113382*0.2);
INSERT INTO temp1 SELECT '40th percentile', wa FROM temp WHERE rowid =
round(113382*0.4);
INSERT INTO temp1 SELECT '50th percentile or MEDIAN', wa FROM temp WHERE
rowid = round(113382*0.5);
INSERT INTO temp1 SELECT 'Average Weight', AVG(wa) FROM temp;
INSERT INTO temp1 SELECT '60th percentile', wa FROM temp WHERE rowid =
round(113382*0.6);
INSERT INTO temp1 SELECT '80th percentile', wa FROM temp WHERE rowid =
round(113382*0.8);
INSERT INTO temp1 SELECT '99th percentile or MAX', MAX(wa) FROM temp;
SELECT * FROM temp1;

--Percentile                  Weight/Age   Height/Age   Weight/Height(BMI)
--1st percentile or MIN              -4.99        -4.99        -4.99
--20th percentile             -2.59        -3.12        -2.49
--40th percentile             -1.94        -2.33        -0.57
--50th percentile/MEDIAN      -1.66        -1.98        -1.27
--Average Weight              -1.66        -1.89        -0.88
```

```
--60th percentile            -1.38        -1.62        +0.59
--80th percentile            -0.72        -0.71        +1.71
--99th percentile or MAX     +1.99        +1.99        +1.99

DROP TABLE IF EXISTS temp;
CREATE TABLE temp(
percentile text,
ha numeric
);
INSERT INTO temp SELECT 'placeholder', ha FROM participant WHERE
row_error_type = 1 or row_error_type = -4 ORDER BY ha;

DROP TABLE IF EXISTS temp1;
CREATE TABLE temp1(
percentile text,
ha numeric
);

--Calculate percentiles and averages for Z-scores wa, ha, wh(bmi)
INSERT INTO temp1 SELECT '1st percentile or MIN', MIN(ha) FROM temp;
INSERT INTO temp1 SELECT '20th percentile', ha FROM temp WHERE rowid =
round(113382*0.2);
INSERT INTO temp1 SELECT '40th percentile', ha FROM temp WHERE rowid =
round(113382*0.4);
INSERT INTO temp1 SELECT '50th percentile or MEDIAN', ha FROM temp WHERE
rowid = round(113382*0.5);
INSERT INTO temp1 SELECT 'Average Weight', AVG(ha) FROM temp;
INSERT INTO temp1 SELECT '60th percentile', ha FROM temp WHERE rowid =
round(113382*0.6);
INSERT INTO temp1 SELECT '80th percentile', ha FROM temp WHERE rowid =
round(113382*0.8);
INSERT INTO temp1 SELECT '99th percentile or MAX', MAX(ha) FROM temp;
SELECT * FROM temp1;

DROP TABLE IF EXISTS temp;
CREATE TABLE temp(
percentile text,
wh numeric
);
```

```
INSERT INTO temp SELECT 'placeholder', wh FROM participant WHERE
row_error_type = 1 or row_error_type = -4 ORDER BY wa;

DROP TABLE IF EXISTS temp1;
CREATE TABLE temp1(
percentile text,
wh numeric
);

--Calculate percentiles and averages for Z-scores wa, ha, wh(bmi)
INSERT INTO temp1 SELECT '1st percentile or MIN', MIN(wh) FROM temp;
INSERT INTO temp1 SELECT '20th percentile', wh FROM temp WHERE rowid =
round(113382*0.2);
INSERT INTO temp1 SELECT '40th percentile', wh FROM temp WHERE rowid =
round(113382*0.4);
INSERT INTO temp1 SELECT '50th percentile or MEDIAN', wh FROM temp WHERE
rowid = round(113382*0.5);
INSERT INTO temp1 SELECT 'Average Weight', AVG(wh) FROM temp;
INSERT INTO temp1 SELECT '60th percentile', wh FROM temp WHERE rowid =
round(113382*0.6);
INSERT INTO temp1 SELECT '80th percentile', wh FROM temp WHERE rowid =
round(113382*0.8);
INSERT INTO temp1 SELECT '99th percentile or MAX', MAX(wh) FROM temp;
SELECT * FROM temp1;

--Calculate frequencies of different malnutrition classifications in screened population
SELECT malnutrition_classification, COUNT(*),
(COUNT(*) * 1.0)/(113382*1.0) AS malnutrition_class_frequency
FROM participant WHERE row_error_type = 1 OR row_error_type = -4
GROUP BY malnutrition_classification;
```

malnutrition_classification	COUNT(*)	malnutrition_class_frequency
At Risk	26000	0.23
MAM	16858	0.15
Normal	11522	0.10
SAM	8784	0.08
Severely Stunted	21974	0.19
Stunted	28244	0.25

```
DROP TABLE IF EXISTS temp;
```

```
CREATE TABLE IF NOT EXISTS temp AS SELECT Country, Stake, childIndex,
COUNT(*) AS screenCount_number_total_times
FROM participant WHERE  row_error_type = 1 or row_error_type = -4
GROUP BY Country, Stake, ChildIndex
HAVING COUNT(*) > 1 ORDER BY Country, Stake, childIndex;

ALTER TABLE temp ADD COLUMN last_wa_measurement;
ALTER TABLE temp ADD COLUMN first_wa_measurement;
ALTER TABLE temp ADD COLUMN last_ha_measurement;
ALTER TABLE temp ADD COLUMN first_ha_measurement;
ALTER TABLE temp ADD COLUMN last_wh_measurement;
ALTER TABLE temp ADD COLUMN first_wh_measurement;

--Calculate average change in Z-scores wa,ha, wh
UPDATE temp SET last_wa_measurement = wa FROM participant
WHERE temp.Country = participant.Country
AND temp.Stake = participant.Stake
AND temp.childIndex = participant.childIndex
AND screenCount = 1;

UPDATE temp SET first_wa_measurement = wa FROM participant
WHERE temp.Country = participant.Country
AND temp.Stake = participant.Stake
AND temp.childIndex = participant.childIndex
AND screenCount = screenCount_number_total_times;

UPDATE temp SET last_ha_measurement = ha FROM participant
WHERE temp.Country = participant.Country
AND temp.Stake = participant.Stake
AND temp.childIndex = participant.childIndex
AND screenCount = 1;

UPDATE temp SET first_ha_measurement = ha FROM participant
WHERE temp.Country = participant.Country
AND temp.Stake = participant.Stake
AND temp.childIndex = participant.childIndex
AND screenCount = screenCount_number_total_times;

UPDATE temp SET last_wh_measurement = wh FROM participant
WHERE temp.Country = participant.Country
```

```
AND temp.Stake = participant.Stake
AND temp.childIndex = participant.childIndex
AND screenCount = 1;

UPDATE temp SET first_wh_measurement = wh FROM participant
WHERE temp.Country = participant.Country
AND temp.Stake = participant.Stake
AND temp.childIndex = participant.childIndex
AND screenCount = screenCount_number_total_times;

SELECT AVG(last_wa_measurement - first_wa_measurement) AS
average_overall_change_wa,
AVG(last_ha_measurement - first_ha_measurement) AS average_overall_change_ha,
AVG(last_wh_measurement - first_wh_measurement) AS average_overall_change_wh
FROM temp;
```

```
average_overall_change_wa    average_overall_change_ha
average_overall_change_wh
0.79089648737574    0.22028964982151    0.34460649441802
```

Data Harmonization(Python):

```
# Import libraries
import pandas as pd
import numpy as np
import os
from os import listdir
import pathlib

## This file processes world bank data from a directory and outputs a .csv file
containing only data for countries and years of interest. Output is a dataframe where
each entry is a dictionary with years as keys and economic data as values.

# Create list of years containing the full date range for clinical data (may need to be
adjusted)
years_list = ['2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','2020']

# Create list of country codes for each nation in the clinical study
data_nations =
['BOL','COL','ECU','GHA','GTM','HND','HTI','KHM','KIR','LBR','MDG','MNG',
```

```
            'NGA','NIC','PER','PHL','PRY','SLV','VEN','SLE','ZWE']

# Specify directory
file_path_1 = 'Economic Indicators/All_Indicators'

# Check files in directory
files = [f for f in pathlib.Path(file_path_1).iterdir() if f.is_file()]
print(files)

# Function to process data from dataframe and return dictionary of key:value pairs for
each year
def process_edata(input_df):
   input_df = input_df.loc[:, years_list]
   df_dict = pd.Series(input_df.to_dict(orient='records'))
   nations_list = input_df.index.to_list()
   df_series = df_dict.set_axis(nations_list)
   df_series = df_series.loc[data_nations]
   return df_series

# Function to process data from all files in directory and combine in single dataframe
def process_files(file_path):
   add_df = pd.DataFrame(index=data_nations)
   files_lst = [f for f in pathlib.Path(file_path).iterdir() if f.is_file()]
   for f in files_lst:
      file_data = pd.read_csv(f, index_col=1, skiprows=4)
      file_name = os.path.basename(f)
      file_name = file_name[:-4]
      print(file_name)
      indicator_col = process_edata(file_data)
      add_df[file_name] = indicator_col
   return add_df

# Process combined dataframe
combined_df = process_files(file_path_1)

# Rename country indices to match clinical data
combined_df.rename(index={'SLE':'WAL','LBR':'LIB'}, inplace=True)

# Write to .csv file
combined_df.to_csv('all_econ_data.csv')
```

Data Cleaning(Jupyter Notebook):

Econ Data:

```
import pandas as pd
import matplotlib.pyplot as plt

all_econ = pd.read_csv("../Data/all_econ_data.csv")
all_econ.head()

import ast

all_econ = pd.read_csv("../Data/all_econ_data.csv")
countries = all_econ["Unnamed: 0"]

def unstack_data(country):
    new_df = pd.DataFrame({})
    row = all_econ.loc[all_econ["Unnamed: 0"] == country]
    for i in all_econ.columns[1::]:
        dict_string = row[i].values[0]
        dict_string = dict_string.replace("nan","'NONE'")
        dict_string = ast.literal_eval(dict_string)
        new_df[i] = dict_string
    new_df["Country"] = country
    new_df = new_df.reset_index().rename(columns={"index":"year"})
    return new_df

unstacked_data = unstack_data(countries[0])

for country in countries[1::]:
    unstacked_data = pd.concat([unstacked_data,unstack_data(country)])

# unstacked_data = unstacked_data.drop(c)

unstacked_data.to_csv("../Data/all_econ_unstacked.csv",index=False)
unstacked_data.columns

import numpy as np
import seaborn as sns
```

```
plt.rcParams["figure.figsize"] = (5,5)

all_econ = pd.read_csv("../Data/all_econ_unstacked.csv")
for i in all_econ.columns:
    all_econ[i] = all_econ[i].replace("NONE",np.nan)

ax = plt.axes()
sns.heatmap(all_econ.isna(), cbar=False, ax=ax, cmap="viridis")
plt.title("Missing Values", fontsize=12)
plt.xlabel("Columns", fontsize = 10)
plt.ylabel("Missing", fontsize = 10)
plt.savefig("../Images/MissingMap_A.png")
plt.show()

import missingno as msno
msno.matrix(all_econ,figsize=(5,7),fontsize=8)
plt.savefig("../Images/MissingMap_B.png")

msno.heatmap(all_econ,figsize=(7,5),fontsize=8)
plt.title("Missingness Correlation Map")
# This shows the correlation between missingness. If poverty is missing, Gini will
be missing as well.
plt.savefig("../Images/MissingMap_Correlation.png")

import numpy as np
import seaborn as sns
plt.rcParams["figure.figsize"] = (5,5)

all_econ = pd.read_csv("../Data/all_econ_unstacked.csv")
for i in all_econ.columns:
    all_econ[i] = all_econ[i].replace("NONE",np.nan)

plt.rcParams["figure.figsize"] = (10,10)

fig, axs = plt.subplots(5,4)
for row in range(0,5):
    for column in range(0,4):
        country_indx = row + column
        data = all_econ.loc[all_econ["Country"] ==
countries[country_indx]].drop(columns=["Country"]).isna()
```

```
        sns.heatmap(data = data, ax=axs[row, column],cbar=False,fmt = ".2f")
        axs[row,column].tick_params(left=False,bottom=False)

plt.savefig("../Images/MissingMap_By_Country.png")

miss_country = pd.DataFrame({"Country":countries})
miss_country["NaN_Count"] =
[all_econ.loc[all_econ["Country"]==c].isnull().sum().sum() for c in countries]
miss_country =
miss_country.sort_values("NaN_Count",ascending=False).reset_index(drop=Tru
e)
miss_country["% NaN"] = miss_country["NaN_Count"]/len(all_econ)

miss_var= pd.DataFrame({"Variable":all_econ.columns})
miss_var["NaN_Count"] = [all_econ[v].isnull().sum() for v in all_econ.columns]
miss_var =
miss_var.sort_values("NaN_Count",ascending=False).reset_index(drop=True)
miss_var["% NaN"] = miss_var["NaN_Count"]/len(all_econ)

plt.rcParams["figure.figsize"] = (5,5)
sns.barplot(data=miss_country,x="% NaN",y="Country",orient="h")
plt.title("Total Missingness by Country")
plt.xlim(0,1)
plt.savefig("../Images/Missingness_By_Country.png")

plt.rcParams["figure.figsize"] = (5,5)
sns.barplot(data=miss_var,x="% NaN",y="Variable",orient="h")
plt.title("Total Missingness by Variable")
plt.xlim(0,1)
plt.savefig("../Images/Missingness_By_Variable.png")

all_econ
```

JME Data:

```
#data cleaning for JME_Country_Estimates
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import missingno as msno
import matplotlib.pyplot as plt
```

```
import seaborn as sns
#  loading the CSV file with a different encoding
df = pd.read_csv('JME_Country_Estimates_May_2023 (2).csv',
encoding='ISO-8859-1')

print(df.head())

# Visualize missing data with a heatmap
plt.figure(figsize=(12, 6))
msno.heatmap(df)
plt.title('Missing Values Heatmap')
plt.show()

# Show missing value heatmap
msno.matrix(df)
plt.title('Missing Value Matrix')
plt.show()

# Or bar chart
msno.bar(df)
plt.title('Missing Values Per Column')
plt.show()

# Set the 'Country and areas' column as the index
df.set_index('Country and areas', inplace=True)

# Create a missing values matrix (True/False)
missing_values_matrix = df.isnull()

# Plot the missing values heatmap
plt.figure(figsize=(14, 8))
sns.heatmap(missing_values_matrix, cmap='viridis', cbar=False,
xticklabels=True, yticklabels=True)
plt.title('Missing Values Heatmap by Country and Feature')
plt.xlabel('Features')
plt.ylabel('Country')
plt.tight_layout()
plt.show()

#  Identify categorical and numerical columns
```

```
categorical_columns = df.select_dtypes(include=['object']).columns
numerical_columns = df.select_dtypes(exclude=['object']).columns

print("Categorical columns:", categorical_columns)
print("Numerical columns:", numerical_columns)

#  Handle missing values by Filling numerical columns with mean and categorical
columns with mode
df[numerical_columns] =
df[numerical_columns].fillna(df[numerical_columns].mean())
df[categorical_columns] =
df[categorical_columns].fillna(df[categorical_columns].mode().iloc[0])

# Clean numerical columns that have footnotes or text embedded
def clean_numeric_column(column):
    # Remove non-numeric characters and convert to float
    return pd.to_numeric(column.replace(r'[^\d.]', '', regex=True), errors='coerce')

# Columns with potential footnotes
columns_to_clean = ['Severe Wasting', 'Wasting', 'Overweight', 'Stunting',
'Underweight']
for col in columns_to_clean:
    df[col] = clean_numeric_column(df[col])

#  Encode categorical columns if needed (for heatmap visualization)
# Label encoding for categorical columns (if needed for analysis)
label_encoder = LabelEncoder()
for col in categorical_columns:
    df[col] = label_encoder.fit_transform(df[col])

#  Check for duplicate rows
print(f"Number of duplicate rows: {df.duplicated().sum()}")
df = df.drop_duplicates()

#  Generate the correlation matrix for numerical columns
correlation_matrix = df[numerical_columns].corr()

#  Create the Heatmap for the correlation matrix
plt.figure(figsize=(12, 10))  # Set figure size
```

```
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)

# Add title and labels
plt.title("Correlation Heatmap of Dataset", fontsize=16)
plt.show()

# Show missing value heatmap
msno.matrix(df)
plt.title('Missing Value Matrix')
plt.show()

# Or bar chart
msno.bar(df)
plt.title('Missing Values Per Column')
plt.show()

# Calculate missing values by country
missing_by_country = df.isnull().sum(axis=1)  # Sum missing values across rows
(per country)
df['missing_count'] = missing_by_country

# Group by country and calculate total missing values for each
missing_by_country_total = df.groupby('Country and
areas')['missing_count'].sum().sort_values(ascending=False)

# Plot missing values by country
plt.figure(figsize=(14, 8))
sns.barplot(x=missing_by_country_total.values,
y=missing_by_country_total.index, palette="viridis")
plt.title('Missing Values by Country')
plt.xlabel('Number of Missing Values')
plt.ylabel('Country')
plt.tight_layout()
plt.show()
```