Child mortality prediction

Contributors:

Bartosz Sroka 400490
Norbert Podgórki 402111

Table of Contents:

Imports

In [189]:

```
import glob
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import geopandas as gpd
import pmdarima as pm
from typing import List
import warnings
warnings.filterwarnings('ignore')

countries = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))
```

Functions

The following functions have been consolidated in one place and have been used for data processing and analysis.

In [190]:

```
def plot_global_map(df:pd.DataFrame, year:int, continents: List[str] = list(set(countrie
    year = "{}".format(year)
    title = get_dataframe_name(df) + " " + year
    df = countries[countries['continent'].isin(continents)].set_index('name').join(df.se
    plot_global_map_figure(df=df, year=year, title=title, ax=ax)

def plot_global_map_figure(df: pd.DataFrame, year: str, title: str, ax):
    if ax is None:
        fig, ax = plt.subplots(figsize=(20, 6))

    df.plot(column=year, cmap='YlOrRd', linewidth=0.8, ax=ax, edgecolor='0.8')
    ax.set_title(title)
    if ax is None:
        vmin = df[year].min()
        vmax = df[year].max()
        sm = plt.cm.ScalarMappable(cmap='YlOrRd', norm=plt.Normalize(vmin=vmin, vmax=vma
        sm.set_array([])
        plt.tight_layout()
        plt.colorbar(sm)

def plot_global_map_subplot(df_list: List[pd.DataFrame], year: int, continents: List[str
    n_plots = len(df_list)
    fig, axes = plt.subplots(nrows=n_plots, ncols=1, figsize=(n_plots*5, 20))

    for i, df in enumerate(df_list):
        plot_global_map(df=df, year=year, ax=axes[i])

    plt.tight_layout()
    plt.show()

def get_dataframe_name(df: pd.DataFrame):
    for name, obj in globals().items():
        if obj is df:
            return name
```

Problem formulation

The problem involves predicting child mortality based on factors from multiple domains of life. In this context, a dataset has been selected that contains information on various factors that potentially influence child mortality. The goal is to develop a predictive model that can estimate the risk of death in children based on these factors.

The project also has the potential to reduce the impact of factors in the future.

Data source:

The data has been obtained from the Gapminder website [https://www.gapminder.org/data/]. Gapminder is a non-profit organization that collects and provides a wide range of global development data. They offer a comprehensive database that covers various indicators related to population, health, education, economy, and more. The data provided by Gapminder is widely used for research, analysis, and visualizations to gain insights into global trends and patterns.

Loading data:

In [191]:

```
original_data = {os.path.splitext(os.path.basename(file_name))[0] : pd.read_csv(file_nam

# predicted values
child_mortality_df = original_data["child_mortality_0_5_year_olds_dying_per_1000_born"]

# explanatory data
food_supply_df = original_data["food_supply_kilocalories_per_person_and_day"]
med_beds_df = original_data["sh_med_beds_zs"]
co2_emission_df = original_data["co2_emissions_tonnes_per_person"]
gender_equality_df = original_data["gendereq_idea"]
```

- ***child_mortality_0_5_year_olds_dying_per_1000_born***
  Death of children under five years of age per 1,000 live births. The data contains information on 196 countries spanning from 1800 to 2100. It is a combination of data from three sources:

  For the period from 1800 to 1950, the data was compiled and documented by Klara Johansson and Mattias Lindgren. The primary sources used were www.mortality.org and the International Historical Statistics series by Brian R Mitchell. Historic estimates of infant mortality rates were transformed into child mortality rates using regression analysis.

  From 1950 to 2016, the data is sourced from the UNIGME (United Nations Inter-agency Group for Child Mortality Estimation) collaboration project involving UNICEF, WHO, UN Population Division, and the World Bank. The project released new estimates of child mortality on September 19, 2019, available at www.childmortality.org. This dataset includes estimates for the majority of countries, covering the years from 1970 to 2018, with some countries having data going back to 1960 and a smaller percentage reaching back to 1950.

  From 1950 to 2100, the data is obtained from the UN POP (United Nations World Population Prospects) report for 2019. The annual data on child mortality rates is found in the WPP2019_INT_F01_ANNUAL_DEMOGRAPHIC_INDICATORS.xlsx file.

- ***food_supply_kilocalories_per_person_and_day***
  Calories measures the energy content of the food. The required intake varies, but it is normally in the range of 1500-3000 kilocalories per day. The data contains information on 178 countries.
  The data comes from FAOSTAT, which collects food statistics gathered by the Food and Agriculture Organization of the United Nations (FAO). It includes information on agricultural production, food consumption, trade, prices, food stocks, and other aspects related to agriculture and food. The data is

collected from various countries worldwide and is used for analysis, monitoring trends, planning food policies, and supporting decisions related to agriculture and food at national and international levels.

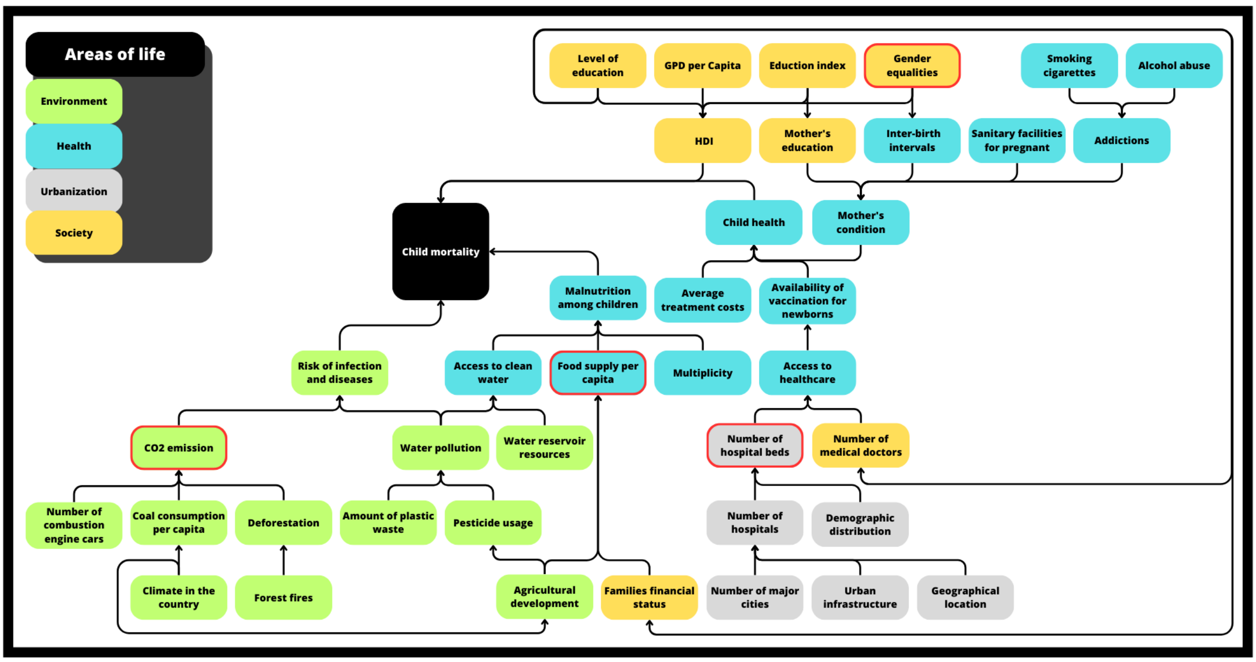
- ***sh_med_beds_zs***
  The data is sourced from the World Health Organization, supplemented by country data. The data provides information up to the year 2019 on the number of medical beds per 1000 people.

- ***co2_emissions_tonnes_per_person***
  Carbon dioxide emissions (metric tonnes of CO2 per person). The data comes from the CDIAC service, which is currently transitioning to ESS-DIVE. CDIAC has been collecting data for over 30 years until 2018. The transition process is managed by ESS-DIVE, which is part of the United States Department of Energy. ESS-DIVE is maintained by Lawrence Berkeley National Laboratory and supported by the Biological and Environmental Research program of the United States Department of Energy (BER).

- ***gendereq_idea***
  Two expert-coded indicators from V-Dem were used to operationalize gender equality–power distribution by gender and female participation in civil society organizations—as well as three observational indicators on the ratio between female and male mean years of schooling (GHDx), the proportion of lower chamber female legislators (V-Dem)and the proportion of women in ministerial-level positions (IPU). The five indicators were aggregated into the gender equality sub-component using IRT.1Power distributed by gender 2CSO women's participation 3.Female vs. male mean years of schooling 4.Lower chamber female legislators 4.Election women in the cabinet. The final indicator obtained in this way is given as a percentage.

DAG and confoundings:

The selected parameters with the least mutual correlation and the most impact on the target value have been marked with a red border.



Based on the analysis of the problem, we have selected parameters for our DAG in such a way that their interdependence is minimized while still influencing the target variable. Additionally, each parameter is derived from a different area of life. If we were to expand the graph, we might discover some connections between these parameters, but they are relatively distant, as illustrated by our DAG. None of the parameters are a result of any other analyzed parameter.

Data Preprocessing

Due to the large amount of data and variations in data collection methods, the dataset contains numerous missing values. To address this, we selected the year 2019, which had the most complete data. In cases where there were significant missing values, we chose to remove the corresponding data. However, whenever possible, we applied imputation methods, such as ARIMA, to estimate missing data based on previous measurements.

The goal was to have the resulting dataset include as many countries as possible, encompassing all the analyzed indicators. Below, we conducted parameter analysis and prepared the data for further analysis.

In [192]:

```
years_range = [str(i) for i in range(2000, 2020)]
```

Data imputation

The analyzed data have been examined over the years, thus it was appropriate to apply ARIMA-based imputation. By filling in the missing data in this way, we obtained a complete dataset for model training.

child_mortality_0_5_year_olds_dying_per_1000_born

In [193]:

```
child_mortality_df.columns
```

Out[193]:

```
Index(['country', '1800', '1801', '1802', '1803', '1804', '1805', '1806',
       '1807', '1808',
       ...
       '2091', '2092', '2093', '2094', '2095', '2096', '2097', '2098', '20
99',
       '2100'],
      dtype='object', length=302)
```

In [194]:

```
child_mortality_df = child_mortality_df[["country"] + years_range]
child_mortality_df
```

Out[194]:

	country	2000	2001	2002	2003	2004	2005	2006	2007	2008	...
0	Afghanistan	129.00	125.00	121.00	117.00	113.00	109.00	104.00	100.00	96.00	...
1	Angola	206.00	200.00	193.00	185.00	176.00	167.00	157.00	148.00	138.00	...
2	Albania	25.90	24.50	23.10	21.80	20.40	19.20	17.90	16.70	15.50	...
3	Andorra	6.41	6.16	5.93	5.71	5.49	5.27	5.05	4.84	4.62	...
4	United Arab Emirates	11.20	10.90	10.60	10.30	10.00	9.73	9.44	9.18	8.93	...
...	...	...	...	...	...	...	...	...	...	...	...
192	Samoa	21.10	20.40	19.90	19.50	19.20	19.00	18.90	18.90	18.90	...
193	Yemen	94.90	90.30	85.60	81.10	76.70	72.50	68.40	64.60	61.00	...
194	South Africa	73.90	75.80	77.40	79.10	79.40	78.50	76.00	71.00	64.80	...
195	Zambia	162.00	153.00	142.00	130.00	119.00	110.00	101.00	95.40	90.40	...
196	Zimbabwe	105.00	104.00	103.00	102.00	102.00	101.00	101.00	100.00	97.00	...

197 rows × 21 columns

In [195]:

```
child_mortality_df[child_mortality_df.isna().any(axis=1)]
```

Out[195]:

country	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	2010	2011	2012	201

0 rows × 21 columns

The way the data was collected ensures that the processed dataset does not contain any missing values. All values for the analyzed countries are filled in during data collection by Gapminder.

food_supply_kilocalories_per_person_and_day

In [196]:

```
food_supply_df.columns
```

Out[196]:

```
Index(['country', '1961', '1962', '1963', '1964', '1965', '1966', '1967',
       '1968', '1969', '1970', '1971', '1972', '1973', '1974', '1975', '19
76',
       '1977', '1978', '1979', '1980', '1981', '1982', '1983', '1984', '19
85',
       '1986', '1987', '1988', '1989', '1990', '1991', '1992', '1993', '19
94',
       '1995', '1996', '1997', '1998', '1999', '2000', '2001', '2002', '20
03',
       '2004', '2005', '2006', '2007', '2008', '2009', '2010', '2011', '20
12',
       '2013', '2014', '2015', '2016', '2017', '2018'],
      dtype='object')
```

In [197]:

```
food_supply_df = food_supply_df[["country"] + years_range[:-1]]
```

In [198]:

```
food_supply_df[food_supply_df.isna().any(axis=1)]
```

Out[198]:

	country	2000	2001	2002	2003	2004	2005	2006	2007	2008
3	Netherlands Antilles	3080.0	3050.0	3070.0	3060.0	3080.0	3090.0	3090.0	3070.0	3080.0
20	Bermuda	2650.0	2610.0	2580.0	2490.0	2460.0	2520.0	2590.0	2630.0	2700.0
24	Brunei	2800.0	2880.0	2930.0	2980.0	3000.0	2980.0	2970.0	2920.0	2910.0
29	Czechoslovakia	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
109	Montenegro	NaN	NaN	NaN	NaN	NaN	NaN	3280.0	3410.0	3480.0
139	Serbia and Montenegro	2650.0	2610.0	2630.0	2700.0	2700.0	2700.0	NaN	NaN	NaN
145	Serbia	NaN	NaN	NaN	NaN	NaN	NaN	2750.0	2710.0	2720.0
167	USSR	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
175	Yugoslavia	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

For Serbia and Montenegro, the values of food supply were split into two separate countries after 2005.

In [199]:

```
serbia_montenegro_df = food_supply_df[food_supply_df['country'].isin(['Montenegro', 'Ser
montenegro = serbia_montenegro_df.iloc[0].combine_first(serbia_montenegro_df.iloc[1])
serbia = serbia_montenegro_df.iloc[2].combine_first(serbia_montenegro_df.iloc[1])
```

In [200]:

```
food_supply_df = food_supply_df.drop(food_supply_df[food_supply_df['country'].isin(['Mon
```

In [201]:

```
food_supply_df = food_supply_df.append(montenegro, ignore_index=True)
food_supply_df = food_supply_df.append(serbia, ignore_index=True)
```

In [202]:

```
food_supply_df[food_supply_df.isna().any(axis=1)]
```

Out[202]:

	country	2000	2001	2002	2003	2004	2005	2006	2007	2008
3	Netherlands Antilles	3080.0	3050.0	3070.0	3060.0	3080.0	3090.0	3090.0	3070.0	3080.0
20	Bermuda	2650.0	2610.0	2580.0	2490.0	2460.0	2520.0	2590.0	2630.0	2700.0
24	Brunei	2800.0	2880.0	2930.0	2980.0	3000.0	2980.0	2970.0	2920.0	2910.0
29	Czechoslovakia	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
164	USSR	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
172	Yugoslavia	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

There is a lack of data for more than 5 years in the past, therefore, we decide not to fill them and discard them in further analysis.

In [203]:

```
food_supply_df = food_supply_df.dropna()
```

In [204]:

```
food_supply_df[food_supply_df.isna().any(axis=1)]
```

Out[204]:

country	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012

The data is up to the year 2018 and does not contain any missing values. We filled in the year 2019 using ARIMA

In [205]:

```
df = food_supply_df.drop(["country"], axis = 1)
```

In [206]:

```
pred = []
for _, row in df.iterrows():
    auto_arima=pm.auto_arima(row, start_p = 0, start_q = 0, max_p = 12, max_q = 12, m =
    prediction = pd.DataFrame(auto_arima.predict(n_periods=1))
    pred.append(prediction)
```

In [207]:

```
for i in range(len(pred)):
    pred[i] = float(int(pred[i].iat[0, 0]))
```

In [208]:

```
food_supply_df["2019"] = pred
```

sh_med_beds_zs

In [209]:

```
med_beds_df.columns
```

Out[209]:

```
Index(['country', '1960', '1961', '1962', '1963', '1964', '1965', '1966',
       '1967', '1968', '1969', '1970', '1971', '1972', '1973', '1974', '19
75',
       '1976', '1977', '1978', '1979', '1980', '1981', '1982', '1983', '19
84',
       '1985', '1986', '1987', '1988', '1989', '1990', '1991', '1992', '19
93',
       '1994', '1995', '1996', '1997', '1998', '1999', '2000', '2001', '20
02',
       '2003', '2004', '2005', '2006', '2007', '2008', '2009', '2010', '20
11',
       '2012', '2013', '2014', '2015', '2016', '2017', '2018', '2019'],
      dtype='object')
```

In [210]:

```
med_beds_df = med_beds_df [["country"] + years_range]
```

In [211]:

```
med_beds_df.isna().sum().tail(5)
```

Out[211]:

```
2015     94
2016     97
2017    103
2018    165
2019    193
dtype: int64
```

In [212]:

```
med_beds_2019 = med_beds_df[med_beds_df['2019'].notna()]
```

There are only 8 rows with a value in 2019 that are retained in the result.

In [213]:

```
med_beds_df = med_beds_df [["country"] + years_range[:-2]]
med_beds_df
```

Out[213]:

	country	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
0	Afghanistan	0.30	0.39	0.39	0.39	0.39	0.42	0.42	0.42	0.42	0.42	0.43	0.44
1	Angola	NaN	NaN	NaN	NaN	NaN	0.80	NaN	NaN	NaN	NaN	NaN	NaN
2	Albania	3.26	3.26	3.14	3.07	3.01	3.08	3.12	3.09	NaN	3.01	2.99	2.88
3	Andorra	3.20	2.59	NaN	3.30	NaN	2.70	2.60	2.60	NaN	2.50	NaN	NaN
4	United Arab Emirates	2.38	2.28	2.19	2.19	2.19	2.19	1.88	1.88	1.86	1.93	1.93	1.07
...	...	...	...	...	...	...	...	...	...	...	...	...	...
196	Samoa	3.30	NaN	1.50	2.04	NaN	1.00	NaN	1.00	NaN	NaN	NaN	NaN
197	Yemen	0.59	0.59	0.59	0.59	0.59	0.61	0.70	0.70	0.70	0.70	0.72	0.70
198	South Africa	NaN	NaN	3.10	NaN	2.87	2.80	NaN	2.41	2.39	NaN	2.30	NaN
199	Zambia	NaN	NaN	NaN	NaN	2.00	NaN	NaN	NaN	1.90	NaN	2.00	NaN
200	Zimbabwe	NaN	NaN	NaN	NaN	NaN	NaN	3.00	NaN	NaN	NaN	NaN	1.70

201 rows × 19 columns

In [214]:

```
med_beds_df = med_beds_df[med_beds_df.isna().sum(axis=1) < 9]
```

Countries that have more than 9 missing values in the analyzed range are not considered for further analysis.

In [215]:

```
med_beds_df = med_beds_df.T
med_beds_df = med_beds_df.fillna(method='bfill')
med_beds_df = med_beds_df.fillna(method='ffill')
med_beds_df = med_beds_df.T
med_beds_df
```

Out[215]:

	country	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
0	Afghanistan	0.3	0.39	0.39	0.39	0.39	0.42	0.42	0.42	0.42	0.42	0.43	0.44
2	Albania	3.26	3.26	3.14	3.07	3.01	3.08	3.12	3.09	3.01	3.01	2.99	2.88
4	United Arab Emirates	2.38	2.28	2.19	2.19	2.19	2.19	1.88	1.88	1.86	1.93	1.93	1.07
5	Argentina	4.1	4.0	4.0	4.0	4.0	4.0	4.5	4.5	4.5	4.5	4.5	4.39
6	Armenia	6.44	5.03	4.35	4.42	4.44	4.46	4.42	4.07	3.82	3.72	3.73	3.74
...	...	...	...	...	...	...	...	...	...	...	...	...	...
189	United States	3.49	3.47	3.39	3.33	3.26	3.2	3.18	3.14	3.13	3.08	3.05	2.97
190	Uzbekistan	5.33	5.34	5.54	5.48	5.26	5.19	5.12	4.83	4.67	4.58	4.44	4.32
191	St. Vincent and the Grenadines	4.7	4.7	4.5	4.5	4.5	4.5	3.0	3.0	2.6	2.6	2.6	2.52
194	Vietnam	2.34	2.4	1.4	2.8	2.8	2.34	2.66	2.9	2.9	3.1	2.91	2.5
197	Yemen	0.59	0.59	0.59	0.59	0.59	0.61	0.7	0.7	0.7	0.7	0.72	0.7

115 rows × 19 columns

In [216]:

```
df = med_beds_df.drop(["country"], axis = 1)
```

In [217]:

```
pred = []
for _, row in df.iterrows():
    auto_arima=pm.auto_arima(row, start_p = 0, start_q = 0, max_p = 12, max_q = 12, m =
    prediction = pd.DataFrame(auto_arima.predict(n_periods=2))
    pred.append(prediction)
```

In [218]:

```
pred_2018 = []
pred_2019 = []
for i in range(len(pred)):
    pred_2018.append(round(pred[i].iat[0, 0], 2))
    pred_2019.append(round(pred[i].iat[1, 0], 2))
```

In [219]:

```
med_beds_df["2018"] = pred_2018
med_beds_df["2019"] = pred_2019
```

Replace a predicted values in 2019 for 8 countries by oryginal data.

In [220]:

```
common_countries = list(set(med_beds_df['country']).intersection(set(med_beds_2019['coun
med_beds_df.loc[med_beds_df['country'].isin(common_countries), '2019'] = med_beds_2019.l
```

co2_emissions_tonnes_per_person

In [221]:

```
co2_emission_df = co2_emission_df[["country"] + years_range[:-1]]
```

In [222]:

```
co2_emission_df[co2_emission_df.isna().any(axis=1)]
```

Out[222]:

	country	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
172	Timor-Leste	NaN	NaN	0.175	0.17	0.181	0.177	0.177	0.177	0.191	0.212	0.215	0.221

In [223]:

```
co2_emission_df = co2_emission_df.T
co2_emission_df = co2_emission_df.fillna(method='bfill')
co2_emission_df = co2_emission_df.T
co2_emission_df[co2_emission_df.isna().any(axis=1)]
```

Out[223]:

country	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012

In [224]:

```
df = co2_emission_df.drop(["country"], axis = 1)
```

In [225]:

```
pred = []
for _, row in df.iterrows():
    auto_arima=pm.auto_arima(row, start_p = 0, start_q = 0, max_p = 12, max_q = 12, m =
    prediction = pd.DataFrame(auto_arima.predict(n_periods=1))
    pred.append(prediction)
```

In [226]:

```
for i in range(len(pred)):
    pred[i] = float(pred[i].iat[0, 0])
```

In [227]:

```
co2_emission_df["2019"] = pred
```

gendereq_idea

In [228]:

```
gender_equality_df = gender_equality_df[["country"] + years_range]
```

In [229]:

```
gender_equality_df[gender_equality_df.isna().any(axis=1)]
```

Out[229]:

	country	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	2010	2011	20
103	Montenegro	NaN	NaN	NaN	NaN	NaN	NaN	46.8	49.3	49.7	...	50.8	50.9	51
142	South Sudan	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	31.1	31
154	Timor-Leste	NaN	NaN	44.1	47.1	45.4	45.7	46.5	46.6	46.6	...	45.4	45.4	47

3 rows × 21 columns

The column for the year 2019 does not have any missing data, therefore there is no need for imputing the remaining columns.

Summary

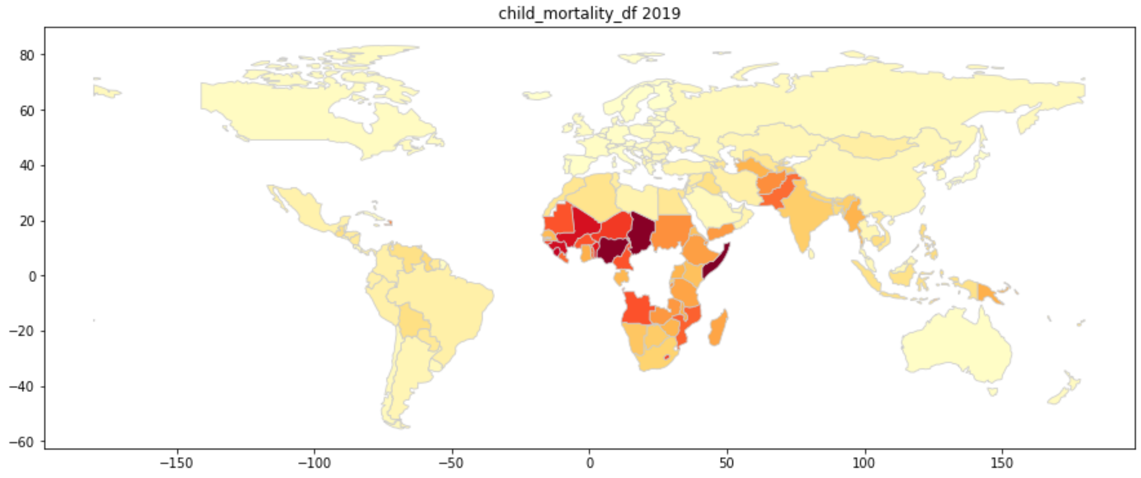
In [230]:

```
child_mortality_df.to_csv('./analysis_data/child_mortality.csv')
plot_global_map(child_mortality_df, year='2019')
child_mortality_df
```

Out[230]:

	country	2000	2001	2002	2003	2004	2005	2006	2007	2008	...
0	Afghanistan	129.00	125.00	121.00	117.00	113.00	109.00	104.00	100.00	96.00	...
1	Angola	206.00	200.00	193.00	185.00	176.00	167.00	157.00	148.00	138.00	...
2	Albania	25.90	24.50	23.10	21.80	20.40	19.20	17.90	16.70	15.50	...
3	Andorra	6.41	6.16	5.93	5.71	5.49	5.27	5.05	4.84	4.62	...
4	United Arab Emirates	11.20	10.90	10.60	10.30	10.00	9.73	9.44	9.18	8.93	...
...	...	...	...	...	...	...	...	...	...	...	...
192	Samoa	21.10	20.40	19.90	19.50	19.20	19.00	18.90	18.90	18.90	...
193	Yemen	94.90	90.30	85.60	81.10	76.70	72.50	68.40	64.60	61.00	...
194	South Africa	73.90	75.80	77.40	79.10	79.40	78.50	76.00	71.00	64.80	...
195	Zambia	162.00	153.00	142.00	130.00	119.00	110.00	101.00	95.40	90.40	...
196	Zimbabwe	105.00	104.00	103.00	102.00	102.00	101.00	101.00	100.00	97.00	...

197 rows × 21 columns



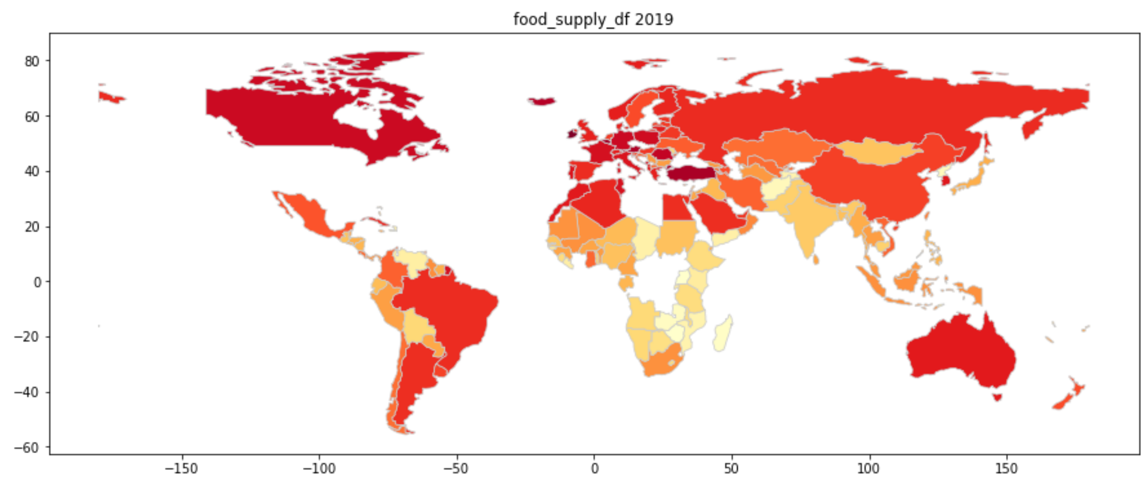
In [231]:

```
food_supply_df.to_csv('./analysis_data/food_supply.csv')
plot_global_map(food_supply_df, year='2019')
food_supply_df
```

Out[231]:

	country	2000	2001	2002	2003	2004	2005	2006	2007	2008	...
0	Afghanistan	1790.0	1740.0	1830.0	1890.0	1970.0	1950.0	1970.0	2050.0	2040.0	...
1	Angola	1790.0	1830.0	1920.0	1980.0	2030.0	2080.0	2120.0	2170.0	2250.0	...
2	Albania	2730.0	2800.0	2860.0	2770.0	2790.0	2870.0	2860.0	2860.0	2950.0	...
4	United Arab Emirates	3300.0	3320.0	3360.0	3340.0	3290.0	3210.0	3200.0	3190.0	3150.0	...
5	Argentina	3260.0	3210.0	2980.0	3010.0	3030.0	3110.0	3110.0	3150.0	3160.0	...
...	...	...	...	...	...	...	...	...	...	...	...
173	South Africa	2890.0	2910.0	2910.0	2930.0	2940.0	2950.0	2930.0	2920.0	2920.0	...
174	Zambia	1870.0	1850.0	1850.0	1900.0	1870.0	1870.0	1840.0	1780.0	1800.0	...
175	Zimbabwe	1980.0	2030.0	2020.0	2010.0	2040.0	2030.0	2120.0	2110.0	2090.0	...
176	Montenegro	2650.0	2610.0	2630.0	2700.0	2700.0	2700.0	3280.0	3410.0	3480.0	...
177	Serbia	2650.0	2610.0	2630.0	2700.0	2700.0	2700.0	2750.0	2710.0	2720.0	...

172 rows × 21 columns



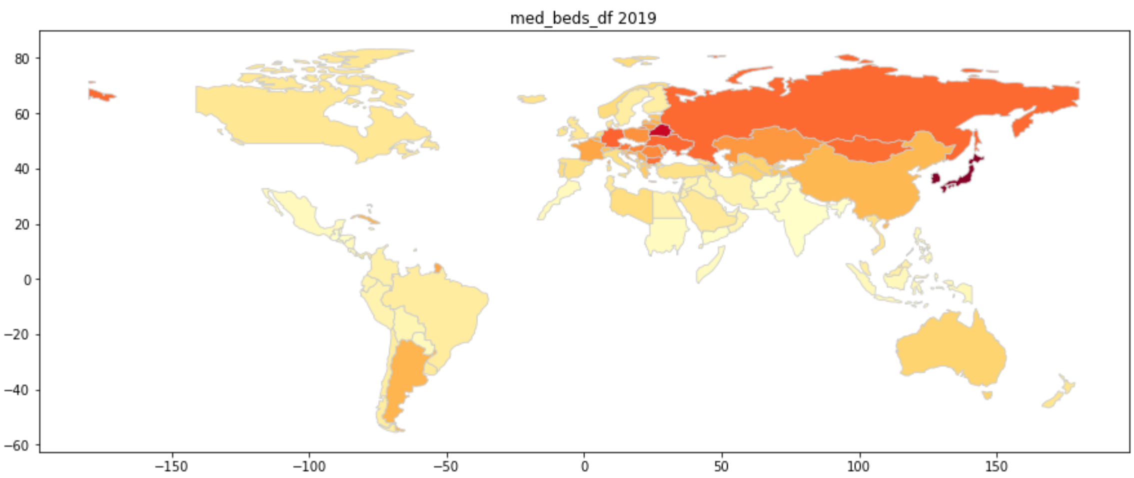
In [232]:

```
med_beds_df.to_csv('./analysis_data/med_beds.csv')
plot_global_map(med_beds_df, year='2019')
med_beds_df
```

Out[232]:

	country	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	2010	2011	20
0	Afghanistan	0.3	0.39	0.39	0.39	0.39	0.42	0.42	0.42	0.42	...	0.43	0.44	0.
2	Albania	3.26	3.26	3.14	3.07	3.01	3.08	3.12	3.09	3.01	...	2.99	2.88	2.
4	United Arab Emirates	2.38	2.28	2.19	2.19	2.19	2.19	1.88	1.88	1.86	...	1.93	1.07	1.
5	Argentina	4.1	4.0	4.0	4.0	4.0	4.0	4.5	4.5	4.5	...	4.5	4.39	4.
6	Armenia	6.44	5.03	4.35	4.42	4.44	4.46	4.42	4.07	3.82	...	3.73	3.74	4.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
189	United States	3.49	3.47	3.39	3.33	3.26	3.2	3.18	3.14	3.13	...	3.05	2.97	2.
190	Uzbekistan	5.33	5.34	5.54	5.48	5.26	5.19	5.12	4.83	4.67	...	4.44	4.32	4.
191	St. Vincent and the Grenadines	4.7	4.7	4.5	4.5	4.5	4.5	3.0	3.0	2.6	...	2.6	2.52	2.
194	Vietnam	2.34	2.4	1.4	2.8	2.8	2.34	2.66	2.9	2.9	...	2.91	2.5	2
197	Yemen	0.59	0.59	0.59	0.59	0.59	0.61	0.7	0.7	0.7	...	0.72	0.7	0.

115 rows × 21 columns



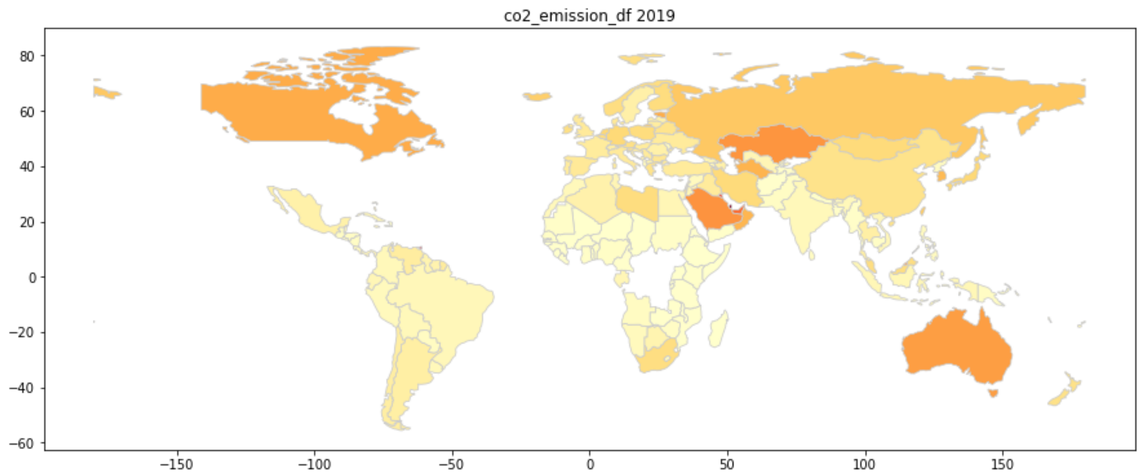
In [233]:

```
co2_emission_df.to_csv('./analysis_data/co2_emission.csv')
plot_global_map(co2_emission_df, year='2019')
co2_emission_df
```

Out[233]:

	country	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	20
0	Afghanistan	0.037	0.0376	0.0471	0.0509	0.0368	0.0515	0.0622	0.0838	0.152	...	0.
1	Angola	0.581	0.571	0.72	0.496	0.998	0.979	1.1	1.2	1.18	...	1.
2	Albania	0.966	1.03	1.2	1.38	1.34	1.38	1.27	1.29	1.46	...	1.
3	Andorra	8.02	7.79	7.59	7.32	7.36	7.3	6.75	6.52	6.43	...	6.
4	United Arab Emirates	35.7	30.5	24.1	28.5	27.5	25.0	23.0	21.6	21.7	...	18
...	...	...	...	...	...	...	...	...	...	...	...	
189	Samoa	0.82	0.836	0.831	0.868	0.842	0.898	0.912	0.947	0.98	...	1.
190	Yemen	0.832	0.895	0.844	0.901	0.955	0.985	1.02	0.974	1.01	...	[illegible]
191	South Africa	8.42	8.16	7.73	8.66	9.5	8.69	9.22	9.47	9.94	...	9.
192	Zambia	0.172	0.176	0.179	0.185	0.182	0.189	0.183	0.149	0.164	...	0.1
193	Zimbabwe	1.16	1.05	0.996	0.886	0.785	0.887	0.853	0.803	0.624	...	0.6

194 rows × 21 columns



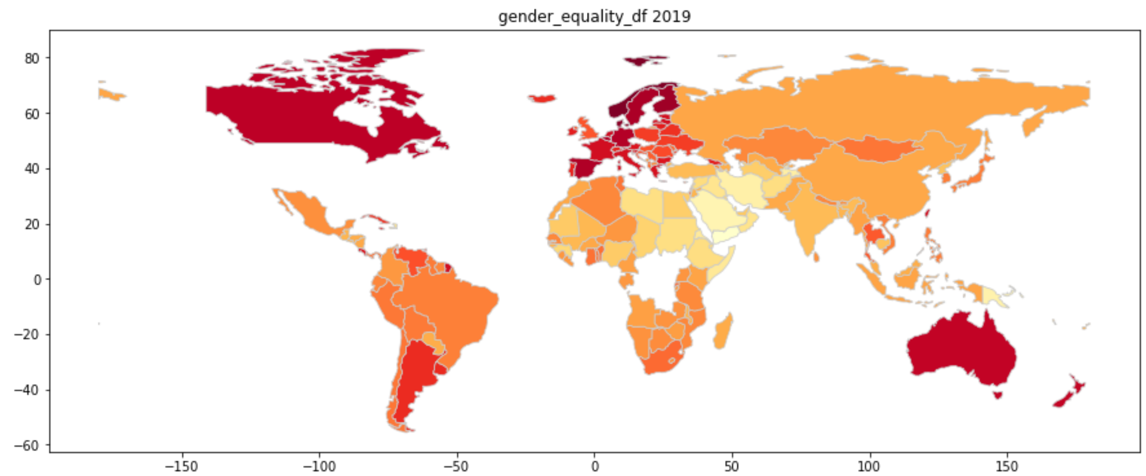
In [234]:

```
gender_equality_df.to_csv('./analysis_data/gender_equality.csv')
plot_global_map(gender_equality_df, year='2019')
gender_equality_df
```

Out[234]:

	country	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	2010	2011	20
0	Afghanistan	0.0	3.25	26.2	30.3	32.0	34.0	34.0	34.6	33.4	...	34.0	34.0	34
1	Angola	37.6	37.60	40.7	45.0	45.2	45.2	45.2	45.2	44.4	...	46.9	49.0	48
2	Albania	50.6	53.00	53.8	55.5	53.8	56.5	56.5	58.1	55.5	...	57.8	57.8	58
3	United Arab Emirates	27.6	31.20	28.9	28.9	30.2	34.7	32.8	37.1	40.0	...	41.6	41.3	4
4	Argentina	68.8	71.30	71.3	73.5	75.6	75.6	73.9	77.6	76.0	...	77.6	77.6	77
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
167	Vanuatu	51.0	48.80	48.9	50.9	49.2	49.4	49.4	48.3	48.3	...	48.5	48.5	48
168	Yemen	14.8	16.60	14.8	14.8	16.7	17.4	17.4	17.4	15.6	...	15.6	16.2	16
169	South Africa	67.0	67.10	67.1	62.5	63.4	65.3	65.3	65.3	65.3	...	65.7	65.7	65
170	Zambia	48.5	48.60	48.6	48.6	48.6	47.5	47.9	48.8	48.8	...	49.8	48.3	49
171	Zimbabwe	49.6	49.60	49.6	49.8	49.8	50.4	50.4	50.4	50.4	...	51.7	52.6	52

172 rows × 21 columns



We are checking how many countries exist for which data is available for all 5 analyzed indicators

In [235]:

```
countries_names = set(child_mortality_df['country']).intersection(food_supply_df['countr
print("Number of data intersection countries: ", len(countries_names))
print("\n", countries_names)
```

```
Number of data intersection countries:  100

 {'Kazakhstan', 'Moldova', 'Afghanistan', 'United Arab Emirates', 'Croati
a', 'Russia', 'Cyprus', 'Paraguay', 'Sweden', 'Cuba', 'Finland', 'Lebano
n', 'Djibouti', 'China', 'Yemen', 'Luxembourg', 'Serbia', 'Honduras', 'Mex
ico', 'Canada', 'Romania', 'Kyrgyz Republic', 'Pakistan', 'Australia', 'Ba
rbados', 'Malaysia', 'Malta', 'Belgium', 'Turkmenistan', 'Tajikistan', 'Ar
gentina', 'Albania', 'Saudi Arabia', 'Egypt', 'Norway', 'Israel', 'Latvi
a', 'Estonia', 'Switzerland', 'India', 'Costa Rica', 'Denmark', 'Belarus',
'France', 'Armenia', 'Tunisia', 'Iceland', 'New Zealand', 'Oman', 'Montene
gro', 'Germany', 'Colombia', 'Kuwait', 'Mongolia', 'Jordan', 'Hungary', 'E
cuador', 'Portugal', 'Trinidad and Tobago', 'Chile', 'Bolivia', 'Bulgari
a', 'Uruguay', 'Azerbaijan', 'United Kingdom', 'Lithuania', 'Greece', 'Geo
rgia', 'Austria', 'Panama', 'Uzbekistan', 'Iraq', 'Guatemala', 'Spain', 'P
eru', 'Bosnia and Herzegovina', 'Vietnam', 'El Salvador', 'Nicaragua', 'Ph
ilippines', 'Slovenia', 'Jamaica', 'Dominican Republic', 'Iran', 'Ukrain
e', 'Italy', 'Slovak Republic', 'Turkey', 'South Korea', 'Morocco', 'North
Macedonia', 'Indonesia', 'Brazil', 'Netherlands', 'Poland', 'United State
s', 'Czech Republic', 'Sudan', 'Japan', 'Ireland'}
```

Data correlation analysis

Checking correlation is not the best indicator for analyzing relationships between parameters. In our project, we use correlation analysis to see how the indicators relate to the overall knowledge presented in the DAG.

In [236]:

```
dataframes = {
    'child_mortality': child_mortality_df,
    'food_supply': food_supply_df,
    'med_beds': med_beds_df,
    'co2_emission': co2_emission_df,
    'gender_equality': gender_equality_df
}

for name, df in dataframes.items():
    df.rename(columns={'2019': name}, inplace=True)
merged_df = child_mortality_df[['country']].copy()

for name, df in dataframes.items():
    merged_df = merged_df.merge(df[['country', name]], on='country')
```

In [237]:

```
merged_df.to_csv('./analysis_data/analysis_data.csv')
```

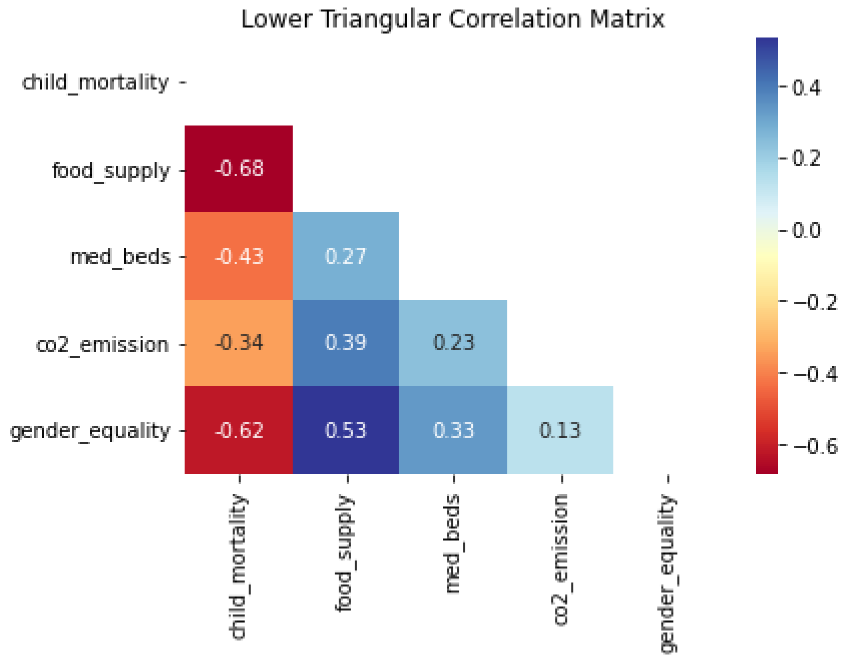
In [238]:

```
correlations = merged_df.iloc[:, 1:].corrwith(merged_df['child_mortality'])
print(correlations)
```

```
child_mortality    1.000000
food_supply       -0.680848
med_beds          -0.434256
co2_emission      -0.343271
gender_equality   -0.618480
dtype: float64
```

In [239]:

```
correlation_matrix = merged_df.corr()
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
sns.heatmap(data=correlation_matrix, mask=mask, annot=True, cmap='RdYlBu')
plt.title('Lower Triangular Correlation Matrix')
plt.show()
```



An inverse correlation has been observed between all of the KPIs and the "child_mortality" indicator. This means that when KPI values decrease, the child mortality rate increases. Correlations are not the best measure because, in the case of co2_emission, an inverse correlation was calculated contrary to what should be expected in reality. However, the key point is that the values are more correlated with the explained variable than with each other.

For an optimal model, the explanatory variables should have weak correlations among themselves but a strong correlation with the variable we want to predict.

Data standardization

Data standardization is used to transform variables in a way that allows for comparison on a uniform scale. This enables better interpretation of results and analysis of patterns, as the variables have a similar range of values and are more comparable. When comparing the number of medical beds and per capita calorie intake, comparing them without standardization would be difficult due to the significant difference in scale between these variables. Therefore, we apply standardization to enable a fair comparison.

However, we leave the child_mortality parameter unchanged to preserve its original meaning and facilitate result interpretation.

In [273]:

```
norm_merged_df = pd.DataFrame(merged_df)
norm_merged_df[norm_merged_df.columns[2:]] = norm_merged_df[norm_merged_df.columns[2:]].
norm_merged_df['child_mortality'] = norm_merged_df['child_mortality'].astype('uint64')
norm_merged_df
```

Out[273]:

	country	child_mortality	food_supply	med_beds	co2_emission	gender_equality
0	Afghanistan	60	-2.772153	-1.261336	-1.065791	-1.569690
1	Albania	8	0.756283	-0.296478	-0.826050	-0.050531
2	United Arab Emirates	7	0.469842	-0.873040	2.728793	-1.176432
3	Argentina	9	0.534942	0.582092	-0.320010	0.665952
4	Armenia	12	-0.162933	0.291850	-0.772216	-0.292950
...	...	...	...	...	...	...
95	Uruguay	7	0.303185	-0.551420	-0.757207	0.687500
96	United States	6	1.740600	-0.316089	1.820592	0.601307
97	Uzbekistan	20	-0.155121	0.091818	-0.628061	-0.729304
98	Vietnam	20	-0.082209	-0.386688	-0.708762	-0.163660
99	Yemen	54	-2.506544	-1.135826	-1.037424	-2.560914

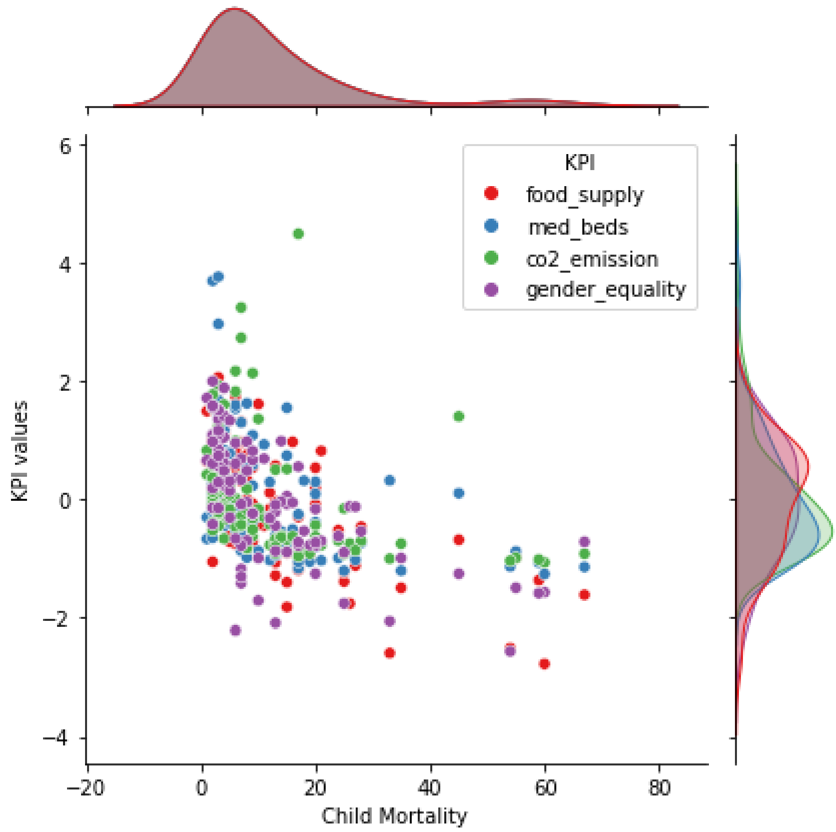
100 rows × 6 columns

Data visualization

In [275]:

```
merged_df_melted = pd.melt(norm_merged_df, id_vars='child_mortality', value_vars=['food_

g = sns.jointplot(data=merged_df_melted, x='child_mortality', y='value', hue='KPI', pale
g.set_axis_labels('Child Mortality', 'KPI values')
plt.tight_layout()
plt.show()
```



In [276]:

```
norm_merged_df.to_csv('./analysis_data/analysis_data_normalized.csv')
```

Child mortality Models

Imports

In [277]:

```
from cmdstanpy import CmdStanModel


import arviz as az
import numpy as np
import scipy.stats as stats

import matplotlib.pyplot as plt
import pandas as pd

import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [278]:

```
data = pd.read_csv("analysis_data/analysis_data_normalized.csv", index_col = [0])
data
```

Out[278]:

	country	child_mortality	food_supply	med_beds	co2_emission	gender_equality
0	Afghanistan	60	-2.772153	-1.261336	-1.065791	-1.569690
1	Albania	8	0.756283	-0.296478	-0.826050	-0.050531
2	United Arab Emirates	7	0.469842	-0.873040	2.728793	-1.176432
3	Argentina	9	0.534942	0.582092	-0.320010	0.665952
4	Armenia	12	-0.162933	0.291850	-0.772216	-0.292950
...	...	...	...	...	...	...
95	Uruguay	7	0.303185	-0.551420	-0.757207	0.687500
96	United States	6	1.740600	-0.316089	1.820592	0.601307
97	Uzbekistan	20	-0.155121	0.091818	-0.628061	-0.729304
98	Vietnam	20	-0.082209	-0.386688	-0.708762	-0.163660
99	Yemen	54	-2.506544	-1.135826	-1.037424	-2.560914

100 rows × 6 columns

Models description

Specified models use the following:

Data:

- `N` : number of observations (size of next vectors)
- `child_mortality` , `food_supply` , `med_beds` , `gender_equality` : previously presented data

Parameters:

- `alpha` : intercept parameter of the distribution
- `co2_emission_coef` , `real food_supply_coef` , `real med_beds_coef` , `real gender_equality_coef` : individual coefficients for each data

Distributions:

- `normal` : used to represent coefficient values and predictive data
- `poisson` : used to represent the predicted data

Prior

For the coefficients priors have form: `normal_rng(0.5, 0.1)` . These are values that do not distinguish any of them and give a reasonable range of influence of these parameters and their variability.

For the predictive data priors have form: `normal_rng(0, 1)` . This prior expresses a lack of strong prior knowledge or preference for any particular parameter value, allowing the data to drive the estimation process.

For the predicted data prior have form `poisson_rng(lambda)` , where:
`labda = exp(alpha + co2_emission_coef * co2_emission + food_supply_coef * food_supply + med_beds_coef * med_beds + gender_equality_coef * gender_equality)`

The Poisson distribution is commonly used to model data, where the outcome represents the number of occurrences of a specific event within a fixed unit of time or space.
This distribution is defined for non-negative integer values (hence the use of the exponential function) - it provides a probabilistic model that assigns higher probabilities to smaller values and decays as the values increase. This property makes it appropriate for situations where the outcome variable can only take on non-negative integer values.

In [305]:

```
%%writefile prior.stan

generated quantities {
  real alpha;
  real lambda;
  real child_mortality;
  real food_supply_coef;
  real co2_emission_coef;
  real gender_equality_coef;
  real med_beds_coef;
  real food_supply;
  real co2_emission;
  real med_beds;
  real gender_equality;

  food_supply_coef = normal_rng(0.5, 0.1);
  med_beds_coef = normal_rng(0.5, 0.1);
  co2_emission_coef = normal_rng(0.5, 0.1);
  gender_equality_coef = normal_rng(0.5, 0.1);

  food_supply = normal_rng(0, 1);
  med_beds = normal_rng(0, 1);
  co2_emission = normal_rng(0, 1);
  gender_equality = normal_rng(0, 1);

  alpha = normal_rng(2, 1);
  lambda = exp(alpha + co2_emission_coef * co2_emission + food_supply_coef * food_supply
  child_mortality = poisson_rng(lambda);
}
```

```
Overwriting prior.stan
```

In [306]:

```
model_prior=CmdStanModel(stan_file='prior.stan')

sim=model_prior.sample(data={}, fixed_param=True, iter_sampling=1000, iter_warmup=0, cha
```

```
INFO:cmdstanpy:compiling stan file /home/Child-Mortality-Prediction-Model/
prior.stan to exe file /home/Child-Mortality-Prediction-Model/prior
INFO:cmdstanpy:compiled model executable: /home/Child-Mortality-Prediction
-Model/prior
INFO:cmdstanpy:CmdStan start processing
chain 1 |██████████| 00:00 Sampling completed



INFO:cmdstanpy:CmdStan done processing.
```

In [307]:

```
df_prior = sim.draws_pd()
df_prior
```

Out[307]:

	lp__	accept_stat__	alpha	lambda	child_mortality	food_supply_coef	co2_emissic
0	0.0	0.0	1.538130	1.71171	3.0	0.322579	0
1	0.0	0.0	2.282170	2.22200	3.0	0.427283	0
2	0.0	0.0	1.945020	2.54679	3.0	0.515238	0
3	0.0	0.0	0.488156	1.45028	4.0	0.535180	0
4	0.0	0.0	2.172830	11.70910	11.0	0.551879	0
...	...	...	...	...	...	...	
995	0.0	0.0	2.420420	233.99600	211.0	0.665544	0
996	0.0	0.0	3.370610	53.10210	54.0	0.674072	0
997	0.0	0.0	1.794360	2.56117	2.0	0.510640	0
998	0.0	0.0	0.102117	2.31338	1.0	0.343322	0
999	0.0	0.0	1.020060	1.15545	1.0	0.505394	0

1000 rows × 13 columns

In [333]:

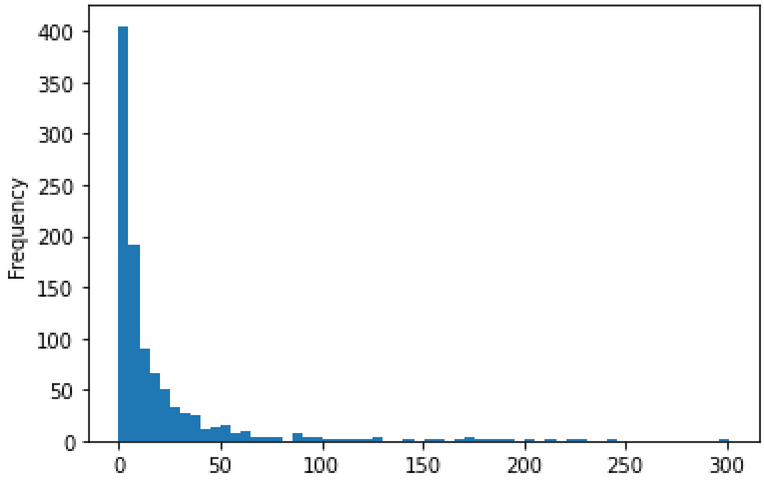
```
az.summary(sim ,var_names=['alpha', 'food_supply', 'co2_emission', 'med_beds', 'gender_e
```

Out[333]:

	mean	sd	hdi_3%	hdi_97%
alpha	1.94	1.02	-0.05	3.67
food_supply	-0.01	0.99	-1.66	1.98
co2_emission	0.03	1.00	-1.66	1.97
med_beds	0.01	1.00	-1.88	1.87
gender_equality	0.08	0.99	-1.88	1.71
lambda	18.81	32.51	0.08	64.24
child_mortality	18.78	32.55	0.00	65.00

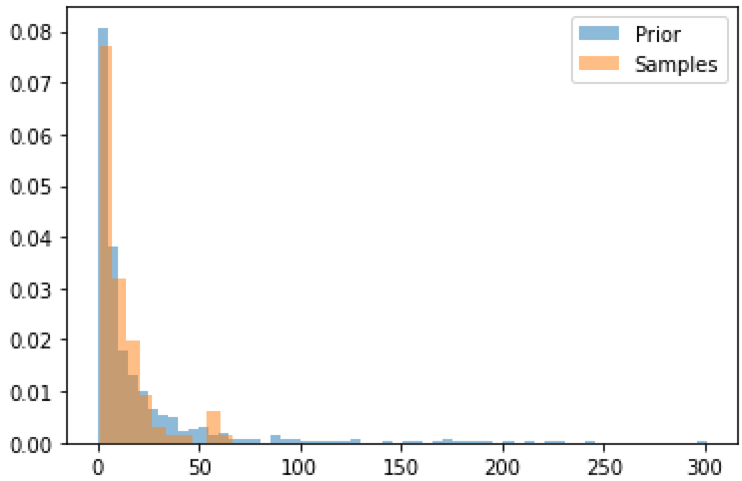
In [309]:

```
df_prior['child_mortality'].plot.hist(bins=60)
plt.show()
```



In [310]:

```
fig, ax = plt.subplots()
ax.hist(df_prior['child_mortality'], bins=60, alpha=0.5, density=True, label='Prior')
ax.hist(data["child_mortality"], bins=10, alpha=0.5, density=True, label='Samples')
ax.legend()
plt.show()
```



In [311]:

```
abs(df_prior['child_mortality'].mean() - data['child_mortality'].mean())
```

Out[311]:

```
6.370999999999999
```

Although the visual analysis in the form of graphs and the average error are not ideal indicators of the quality of the obtained model, it can be concluded that it reflects the characteristics of the problem quite well.

Posterior - first model

Posterior models use the same parameters already described in the previous section, but instead of representing the data as distributions, real data is used to fit the models.

There were no sampling issues with the model so below presented is usage of the model and the analysis of the obtained samples.

In [341]:

```
%%writefile posterior_1.stan

data {
  int N;
  int child_mortality[N];
  real co2_emission[N];
  real food_supply[N];
  real med_beds[N];
  real gender_equality[N];
}

parameters {
  real alpha;
  real co2_emission_coef;
  real food_supply_coef;
  real med_beds_coef;
  real gender_equality_coef;
}

transformed parameters {
    real lambda[N];
    for (i in 1:N){
        lambda[i] = exp(alpha + co2_emission_coef * co2_emission[i] + food_supply_coef *
    }
}

model {
  alpha ~ normal(2, 1);
  food_supply_coef ~ normal(0.5, 0.1);
  med_beds_coef ~ normal(0.5, 0.1);
  co2_emission_coef ~ normal(0.5, 0.1);
  gender_equality_coef ~ normal(0.5, 0.1);

  for (i in 1:N){
      child_mortality[i] ~ poisson(lambda[i]);
  }
}

generated quantities {
  vector [N] log_lik;
  real predicted_child_mortality[N];
  for (i in 1:N) {
    log_lik[i] = poisson_lpmf(child_mortality[i] | lambda[i]);
    predicted_child_mortality[i] = poisson_rng(lambda[i]);
  }
}
```

```
Overwriting posterior_1.stan
```

In [342]:

```
model_1=CmdStanModel(stan_file='posterior_1.stan')
```

In [343]:

```
fit_1=model_1.sample(data=dict(N=len(data), child_mortality=data.child_mortality.values,
```

```
INFO:cmdstanpy:CmdStan start processing
chain 1 |          | 00:00 Status

chain 1 |▊         | 00:00 Status

chain 1 |██▊       | 00:00 Iteration:  400 / 2000 [ 20%]  (Warmup)

chain 1 |█████▌    | 00:00 Iteration: 1001 / 2000 [ 50%]  (Sampling)

chain 1 |███████   | 00:00 Iteration: 1500 / 2000 [ 75%]  (Sampling)

chain 1 |██████████| 00:00 Sampling completed
chain 2 |██████████| 00:00 Sampling completed
chain 3 |██████████| 00:00 Sampling completed
chain 4 |██████████| 00:00 Sampling completed


INFO:cmdstanpy:CmdStan done processing.
```

In [344]:

```
df_fit_1 = fit_1.draws_pd()
df_fit_1
```

Out[344]:

	lp__	accept_stat__	stepsize__	treedepth__	n_leapfrog__	divergent__	energy__
0	2176.90	0.976479	0.401767	3.0	15.0	0.0	-2175.38
1	2178.74	0.990571	0.401767	3.0	7.0	0.0	-2176.08
2	2178.71	0.904787	0.401767	3.0	15.0	0.0	-2177.71
3	2177.71	0.909115	0.401767	4.0	15.0	0.0	-2176.59
4	2175.82	0.427766	0.401767	2.0	3.0	0.0	-2172.58
...	...	...	...	...	...	...	...
3995	2178.86	0.989782	0.370182	3.0	15.0	0.0	-2174.66
3996	2178.13	0.950394	0.370182	3.0	7.0	0.0	-2177.85
3997	2179.10	0.982329	0.370182	2.0	7.0	0.0	-2177.05
3998	2178.44	0.962058	0.370182	2.0	7.0	0.0	-2177.40
3999	2177.42	0.991105	0.370182	3.0	7.0	0.0	-2177.07

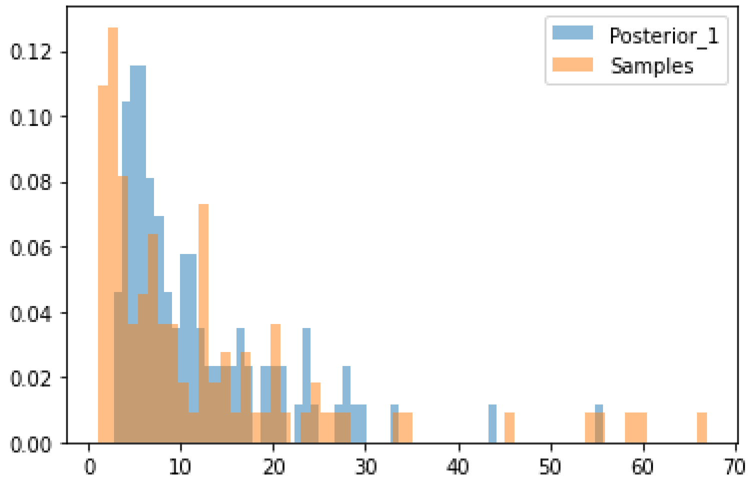
4000 rows × 312 columns

In [345]:

```
means = []
for i in range(1, 99):
    means.append(df_fit_1["predicted_child_mortality[" + str(i) + "]"].mean())
```

In [346]:

```
fig, ax = plt.subplots()
ax.hist(means, bins=60, alpha=0.5, density=True, label='Posterior_1')
ax.hist(data["child_mortality"], bins=60, alpha=0.5, density=True, label='Samples')
ax.legend()
plt.show()
```



In [347]:

```
abs(np.array(means).mean() - data['child_mortality'].mean())
```

Out[347]:

```
0.5411403061224487
```

The average error value has decreased, but more importantly, when analyzing the histograms, we can see that the model fits the real data much better - for example, there are no values greater than 70 as it was in the prior model.

Still, it's not a perfect representation and it's possible to get better results.

Posterior - second model

In the second model we specified, the single parameter alpha was replaced by a country-specific parameter alpha[i]. It was dane, by declaring alpha as an array of appropriate dimension. This change was made because when using a single value for the parameter, the posterior distribution has difficulty in accurately reflecting the observed data. However, by entering individual values for each country, the model significantly improves its ability to fit the data.

There were no sampling issues with the model so below presented is usage of the model and the analysis of the obtained samples.

In [348]:

```
%%writefile posterior_2.stan

data {
  int N;
  int child_mortality[N];
  real co2_emission[N];
  real food_supply[N];
  real med_beds[N];
  real gender_equality[N];
}

parameters {
  real alpha[N];
  real co2_emission_coef;
  real food_supply_coef;
  real med_beds_coef;
  real gender_equality_coef;
}

transformed parameters {
    real lambda[N];
    for (i in 1:N){
        lambda[i] = exp(alpha[i] + co2_emission_coef * co2_emission[i] + food_supply_coe
    }
}

model {
  alpha ~ normal(2, 1);
  food_supply_coef ~ normal(0.5, 0.1);
  med_beds_coef ~ normal(0.5, 0.1);
  co2_emission_coef ~ normal(0.5, 0.1);
  gender_equality_coef ~ normal(0.5, 0.1);

  for (i in 1:N){
      child_mortality[i] ~ poisson(lambda[i]);
  }
}

generated quantities {
  vector [N] log_lik;
  real predicted_child_mortality[N];
  for (i in 1:N) {
    log_lik[i] = poisson_lpmf(child_mortality[i] | lambda[i]);
    predicted_child_mortality[i] = poisson_rng(lambda[i]);
  }
}
```

Overwriting posterior_2.stan

In [349]:

```
model_2=CmdStanModel(stan_file='posterior_2.stan')
```

In [350]:

```
fit_2=model_2.sample(data=dict(N=len(data), child_mortality=data.child_mortality.values,
```

```
INFO:cmdstanpy:CmdStan start processing
chain 1 |          | 00:00 Status

chain 1 |          | 00:00 Status
chain 1 |          | 00:00 Iteration:    1 / 2000 [  0%]  (Warmup)

chain 1 |          | 00:00 Iteration:  200 / 2000 [ 10%]  (Warmup)

chain 1 |          | 00:00 Iteration:  500 / 2000 [ 25%]  (Warmup)

chain 1 |          | 00:00 Iteration:  800 / 2000 [ 40%]  (Warmup)

chain 1 |          | 00:00 Iteration: 1001 / 2000 [ 50%]  (Sampling)

chain 1 |          | 00:01 Iteration: 1300 / 2000 [ 65%]  (Sampling)

chain 1 |          | 00:01 Iteration: 1500 / 2000 [ 75%]  (Sampling)

chain 1 |          | 00:01 Iteration: 1700 / 2000 [ 85%]  (Sampling)
chain 1 |          | 00:01 Sampling completed
chain 2 |          | 00:01 Sampling completed
chain 3 |          | 00:01 Sampling completed
chain 4 |          | 00:01 Sampling completed

INFO:cmdstanpy:CmdStan done processing.
```

In [351]:

```
df_fit_2 = fit_2.draws_pd()
df_fit_2
```

Out[351]:

	lp__	accept_stat__	stepsize__	treedepth__	n_leapfrog__	divergent__	energy__
0	2314.90	0.947910	0.184879	4.0	15.0	0.0	-2259.18
1	2318.39	0.883142	0.184879	5.0	31.0	0.0	-2257.84
2	2319.48	0.948323	0.184879	5.0	31.0	0.0	-2270.00
3	2309.53	0.717567	0.184879	4.0	15.0	0.0	-2264.47
4	2313.28	0.976049	0.184879	4.0	31.0	0.0	-2265.81
...	...	...	...	...	...	...	...
3995	2308.62	0.888379	0.194994	4.0	15.0	0.0	-2254.71
3996	2299.33	0.989450	0.194994	4.0	15.0	0.0	-2258.38
3997	2294.11	0.946718	0.194994	5.0	31.0	0.0	-2244.33
3998	2285.18	0.930853	0.194994	4.0	15.0	0.0	-2225.32
3999	2305.92	0.997392	0.194994	5.0	31.0	0.0	-2244.25

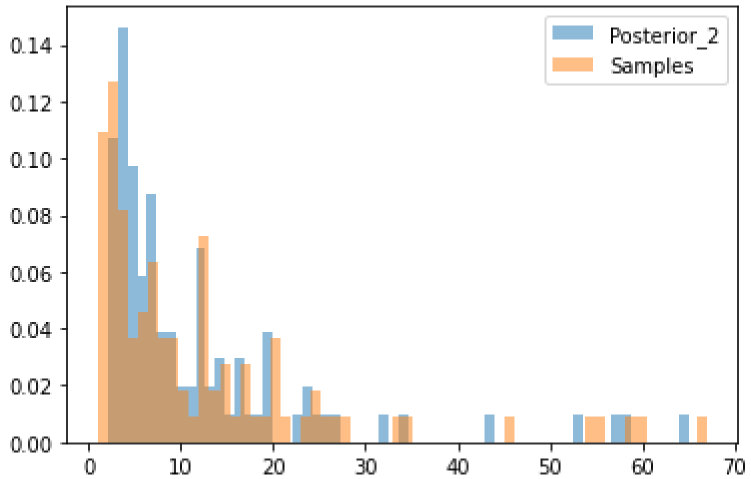
4000 rows × 411 columns

In [352]:

```
means = []
for i in range(1, 99):
    means.append(df_fit_2["predicted_child_mortality[" + str(i) + "]"].mean())
```

In [353]:

```
fig, ax = plt.subplots()
ax.hist(means, bins=60, alpha=0.5, density=True, label='Posterior_2')
ax.hist(data["child_mortality"], bins=60, alpha=0.5, density=True, label='Samples')
ax.legend()
plt.show()
```



In [354]:

```
abs(np.array(means).mean() - data['child_mortality'].mean())
```

Out[354]:

```
0.5417499999999968
```

```
<div style="text-align: justify;">
  The histogram of this model is very similar to the previous one, while the average error value has even increased. However, as it was written earlier, these are not the best metrics for comparing predictive models, so the next chapter contains a more in-depth comparison.
</div><br>
```

Model comparison

The following information criteria were used to compare the models:

- ***WAIC:***
  Evaluates the trade-off between model complexity and goodness of fit. It takes into account both the model's ability to explain the observed data (likelihood) and its complexity (number of parameters). The lower the WAIC value, the better the model's predictive performance

- ***PSIS-LOO:***
  Estimates how well a model generalizes to unseen data by evaluating its performance on leave-one-out cross-validation

In [355]:

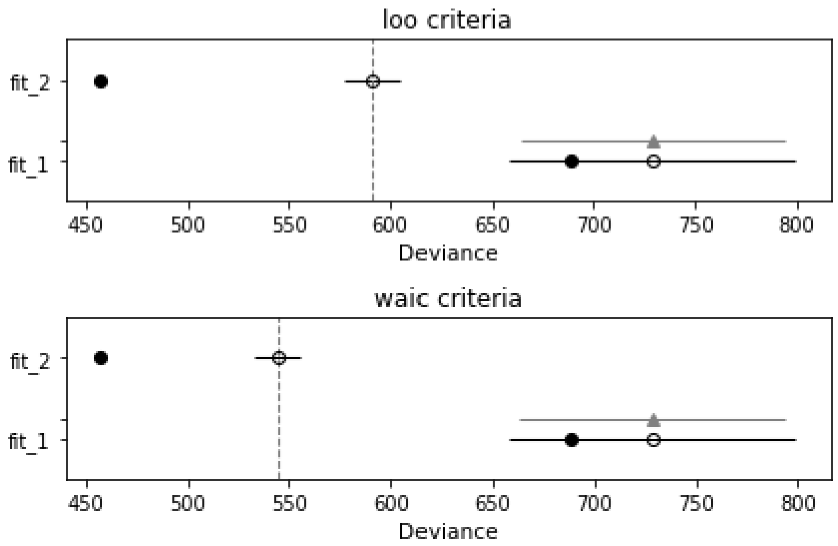
```
compare_model_loo = az.compare(
    {
        "fit_1": az.from_cmdstanpy(fit_1),
        "fit_2": az.from_cmdstanpy(fit_2)
    },
    ic="loo",
    scale="deviance"
)
```

In [356]:

```
compare_model_waic = az.compare(
    {
        "fit_1": az.from_cmdstanpy(fit_1),
        "fit_2": az.from_cmdstanpy(fit_2)
    },
    ic="waic",
    scale="deviance"
)
```

In [357]:

```
_, ax = plt.subplots(nrows=2, ncols=1)
az.plot_compare(compare_model_loo, insample_dev=True, ax=ax[0])
ax[0].set_title("loo criteria")
az.plot_compare(compare_model_waic, insample_dev=True, ax=ax[1])
ax[1].set_title("waic criteria")
plt.tight_layout()
plt.show()
```



In [358]:

```
print(compare_model_loo)
```

```
       rank         loo      p_loo       d_loo   weight         se  
dse  \
fit_2     0  591.090049  66.924732    0.000000  0.57452  13.392184   0.000
000
fit_1     1  729.013346  20.179382  137.923298  0.42548  70.421852  64.905
895

       warning loo_scale
fit_2     True  deviance
fit_1     True  deviance
```

In [359]:

```
print(compare_model_waic)
```

```
       rank        waic     p_waic      d_waic    weight         se  \
fit_2     0  544.251903  43.505659    0.000000  0.890861  11.428456
fit_1     1  728.588177  19.966797  184.336273  0.109139  70.317066

             dse  warning waic_scale
fit_2   0.000000     True   deviance
fit_1  65.406032     True   deviance
```

Explanation and analysis of the obtained results:

- ***loo / waic:***
  represents the estimated expected log predictive density for each model, lower values indicate better predictive performance - the second model obtained a better result for both criteria

- ***p_loo / p_waic:***
  measures the number of parameters that contribute effectively to the model's ability to fit the data, lower values indicate a simpler model - the results indicate that the second model is more advanced

- ***d_loo / d_waic:***
  relative difference to the best model - best is second model

- ***weight:***
  higher weights indicate better model - second model won in this criterion

- ***se:***
  represents the standard error of estimate for each model, larger standard errors indicate higher uncertainty in the estimates - here also the second model turns out to be better