Exploring the Role of Nutrition in COVID-19 Mortality: A Comprehensive Analysis of Global Dietary Patterns

By Austin Mallie, Duy-Anh Dang, and Saloni Barhate

University of San Diego, Master of Applied Data Science

ADS502 Applied Data Mining

Professor An Tran, M.S

9 December 2024

Abstract

This study examines the role of nutrition in predicting COVID-19 outcomes, focusing on the mortality and recovery rates across global populations. Using the "COVID-19 Healthy Diet Dataset," dietary metrics such as caloric, protein, and fat intake were analyzed to determine their impact on pandemic resilience, specifically COVID-related deaths. The project involved cleaning and transforming raw data, performing exploratory data analysis (EDA), and applying machine learning tools including Ordinary Least Squares Regression, Support Vector Machine and Random Forest. Results demonstrated that higher protein and fat intake exhibited significantly higher mortality risks. These findings highlight the importance of integrating nutritional considerations into public health strategies and diets to improve resilience during pandemics and other health crises.

Introduction

The COVID-19 pandemic has exposed stark disparities in health outcomes across countries, prompting efforts to understand the factors driving these differences. While healthcare infrastructure and socioeconomic conditions have been extensively studied, the role of nutrition has not been adequately explored. Nutrition significantly impacts immune function, inflammation, and recovery from illnesses, making it a potentially critical factor in explaining variations in COVID-19 outcomes (Calder et al., 2020). This study examines whether dietary metrics such as caloric, protein, and fat intake can predict COVID-19 outcomes, specifically mortality and recovery rates. By investigating these relationships, this research seeks to provide actionable insights for public health policies and strategies to enhance resilience during future pandemics. The central question guiding this analysis is whether global disparities in dietary supply influence the health outcomes of populations affected by COVID-19 and if so, how these insights can inform targeted interventions.

Data Description

The dataset used in this study, the "COVID-19 Healthy Diet Dataset," was obtained from Kaggle and includes detailed information on dietary supply metrics and COVID-19 statistics for various countries (Mariaren, 2020). The dietary metrics include caloric, protein, and fat intake per capita, while the COVID-19 data include case counts, deaths, and recoveries. The dataset is split into caloric, food, protein, and fat intake which was all merged together to be preprocessed. Additionally, demographic variables such as healthcare access and economic indicators provide contextual information. Despite its richness, the dataset required extensive cleaning and preprocessing due to challenges such as missing values, outliers, and inconsistent scaling. Missing values were particularly prevalent in the dietary metrics for some regions, where outliers

in both dietary and COVID-19 statistics threatened to distort the analyses. These issues necessitated the implementation of robust preprocessing techniques to ensure the integrity of the analysis and model results.

The distribution of deaths is right-skewed with many countries' mortality rate under 5%. The figure below illustrates the mortality of COVID-related deaths:

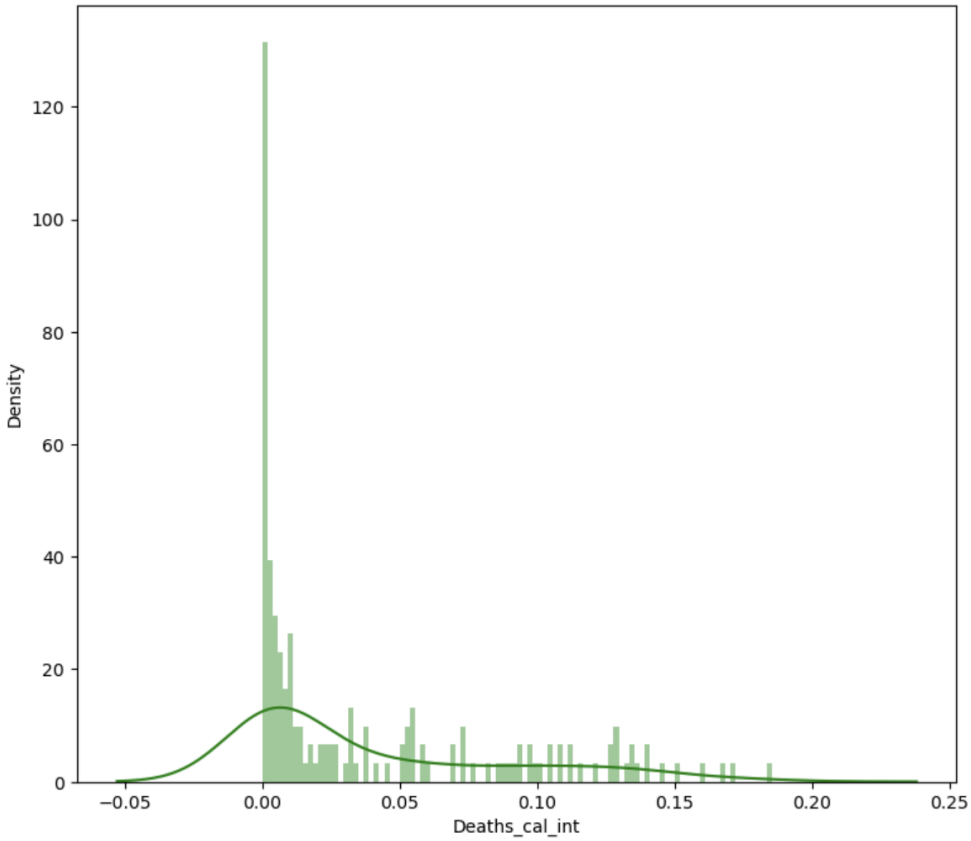


Figure 1. Density curve of mortality of COVID-related deaths

Exploratory Data Analysis

Exploratory data analysis (EDA) provided significant insights into the relationships between dietary metrics and COVID-19 outcomes. Such correlations are essential for identifying potential predictive variables and understanding their influence on target outcomes, as discussed by Tan et al. (2019). Additionally, a positive relationship was observed between fat intake and

animal product intake rates, suggesting that regions with higher fat supply experienced higher intake of animal products. Animal products and vegetal products are highly negatively correlated so the animal product column was removed to avoid multicollinearity. Below the heatmap figure explores the feature correlation in the caloric intake dataset:

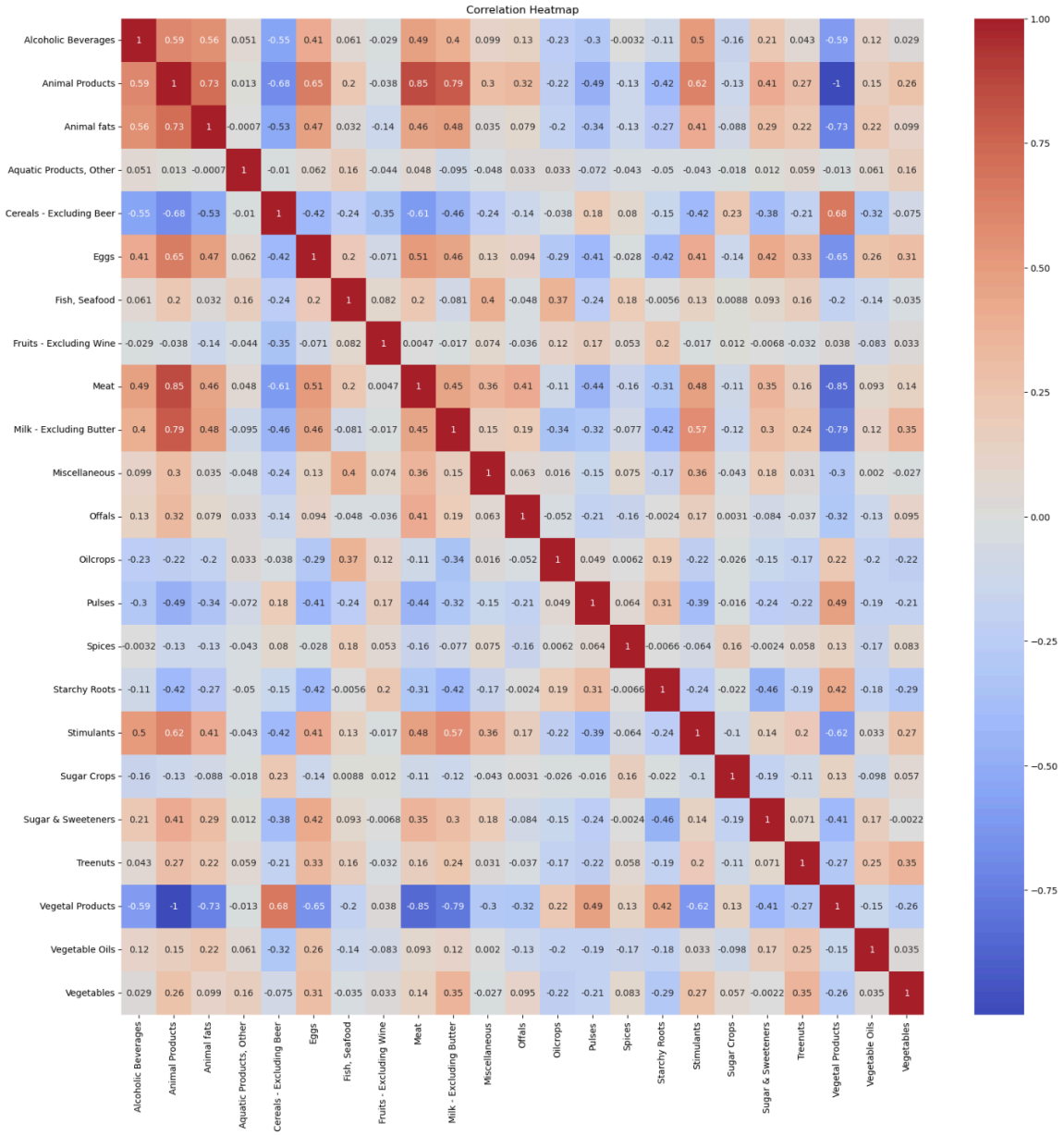
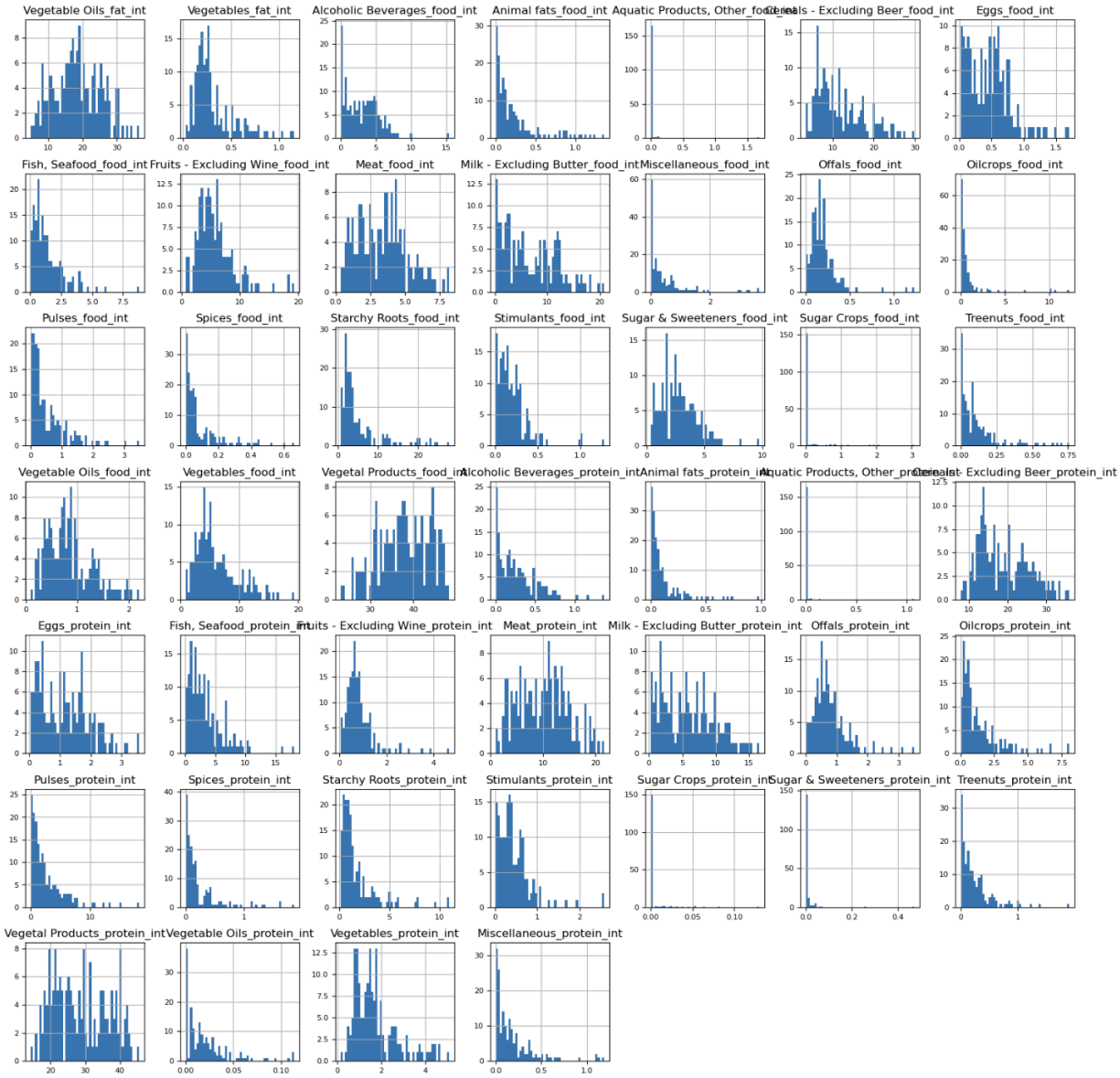


Figure 2. Heatmap of feature correlation in caloric intake dataset

In a different heatmap, alcoholic beverages, animal products (later removed), animal fats, and meat have a positive correlation with COVID deaths. These relationships underscore the importance of nutrient availability in mitigating the severity of health crises. Countries with caloric supply exceeding 2500 kcal/day consistently demonstrated higher recovery rates compared to those with lower caloric intake, highlighting the role of adequate caloric intake in supporting recovery.

The figures below paint the distribution of the numerical features:



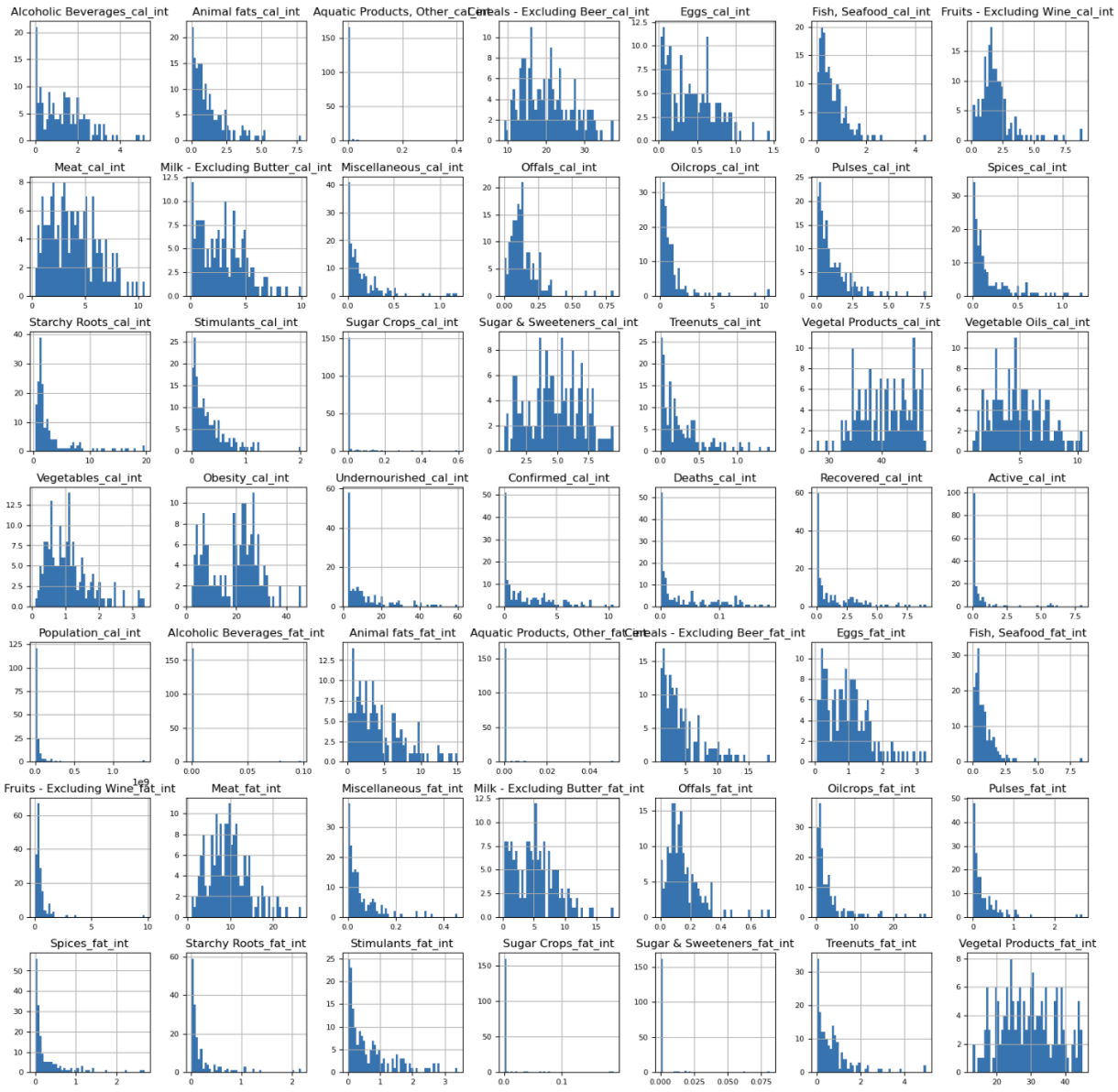


Figure 3. Distribution of the numerical features

High-income countries consistently displayed surplus dietary resources, while low-income countries exhibited critical deficits in protein and caloric supply. This visualization technique effectively highlighted the uneven global distribution of nutritional resources and its potential impact on health outcomes. The EDA findings underscored global nutritional disparities and their significant implications for public health. Higher dietary resources in affluent countries correlate with better COVID-19 outcomes, while deficits in low-income

regions exacerbate vulnerabilities. These insights provide a strong foundation for predictive modeling and policy recommendations, emphasizing the critical role of nutrition in health resilience. Kiribati and Samoa and their obesity rates suffered due to the introduction of western food products. Next to both of these (since they both have a smaller relative population of under 1 million population), the USA has the next highest obesity rate.

Feature Selection and Engineering

Dimensionality reduction and feature selection techniques were critical steps in refining the dataset to enhance predictive modeling.

Recursive Feature Elimination Cross Validation (RFECV) was used to systematically identify the most predictive variables for modeling. The reason for doing 10-fold cross-validation is particularly effective for smaller datasets due to its ability to maximize the use of available data and provide a more reliable estimate of model performance. By dividing the dataset into 10 folds, each fold is used as a validation set once, and the remaining 9 folds are used for training. This allows the model to be trained and evaluated on different subsets of the data, reducing the risk of overfitting. Additionally, with a larger number of folds, the variance in the model's performance across different folds is likely to be smaller, leading to a more stable and reliable estimate. In contrast, fewer folds, like 5-fold cross-validation, might not provide as robust an estimate of model performance, especially with smaller datasets. With fewer folds, each fold contains a larger portion of the data, which can lead to higher variance in the model's performance across different folds.

Protein supply, fat supply, and caloric intake consistently emerged as the top predictors of COVID-19 outcomes, highlighting their strong influence on recovery and mortality rates. Paired with a Random Forest model as the estimator, was employed to rank the features by importance.

The feature rankings were then validated by examining the model's performance metrics, ensuring that the selected features contributed meaningfully to the predictions. In terms of linear importance, the figure below shows the features that have a significant impact on linear regression model:

Feature: 45 Sugar Crops_fat_int, Score: 7.75556
Feature: 47 Treenuts_fat_int, Score: -4.21844
Feature: 50 Vegetables_fat_int, Score: -10.37562
Feature: 52 Animal fats_food_int, Score: -6.30815
Feature: 53 Aquatic Products, Other_food_int, Score: -5.60801
Feature: 54 Cereals - Excluding Beer_food_int, Score: -8.90798
Feature: 55 Eggs_food_int, Score: -25.23133
Feature: 61 Offals_food_int, Score: -9.87039
Feature: 67 Sugar & Sweeteners_food_int, Score: -6.35657
Feature: 68 Sugar Crops_food_int, Score: -19.00679
Feature: 72 Vegetal Products_food_int, Score: -12.99565
Feature: 74 Animal fats_protein_int, Score: -11.50380
Feature: 76 Cereals - Excluding Beer_protein_int, Score: -18.75715
Feature: 77 Eggs_protein_int, Score: -15.70640
Feature: 80 Meat_protein_int, Score: -6.01851
Feature: 87 Stimulants_protein_int, Score: -15.73155
Feature: 88 Sugar Crops_protein_int, Score: -0.05425

Figure 4. Features having a significant impact on linear regression model in terms of linear importance

In addition to these techniques, feature engineering was employed to create new composite metrics that produced deeper insights into the relationship between nutrition and health outcomes. A calorie-to-protein ratio was calculated to address dietary balance, serving as an indicator of the adequacy of protein intake relative to overall caloric consumption. This derived feature captured nuanced dietary diversity indices, which aggregated nutritional metrics to represent overall dietary variety and its potential effect on health outcomes.

These preprocessing steps ensured that the predictive models were trained on the most relevant and informative features, maximizing their power and interpretability. By reducing noise, enhancing feature significance, and incorporating engineered metrics, the modeling

process became more robust and effective. This approach not only improved the models' accuracy but also provided actionable insights into the critical role of nutrition in shaping COVID-19 outcomes, supporting the development of targeted public health interventions.

Data Preprocessing and Cleaning

The first step in preparing the dataset for analysis involved addressing missing values. Missing dietary data for certain countries were imputed using regional averages to maintain consistency while minimizing bias. According to Tan et al. (2019), imputing missing values with region-specific averages is an effective strategy for preserving the statistical integrity of datasets, particularly when regional patterns are known. For example, missing protein intake values for the Sub-Saharan African countries were replaced with the average protein intake of other nations in the region. This method ensured that the imputed values aligned with established regional dietary trends, thereby reducing the risk of introducing systemic bias into the analysis. It was also necessary to convert the string column “Undernourished_cal_int” to a float column during the preparation phase.

“Obesity_cal_int”, “Undernourished_cal_int”, “Confirmed_cal_int”, “Deaths_cal_int”, “Recovered_cal_int”, “Active_cal_int” are all the columns with missing data. To handle this, K-Nearest neighbor was used as a way to fill in missing values in the dataset. The process involves identifying the k-nearest neighbors (k = 5 neighbors) to a data point with a missing value with neighbors determined based on their distance metric Euclidean distance. Once the nearest neighbors are identified, the missing value is imputed using the mean of the corresponding values from these neighbors. One advantage of this approach is that it preserves the underlying patterns in the data and provides more accurate imputations compared to simple imputation of mean or median, improving model performance and insights. Additionally, KNN

Imputer is a non-parametric method, meaning it doesn't make assumptions about the data distribution, making it versatile and suitable for a wide range of datasets.

To standardize the scale of numerical features such as caloric supply, protein intake, and fat intake, normalization was performed using the StandardScaler function from scikit-learn. Normalization is critical in machine learning preprocessing, as it ensures that variables with larger magnitudes do not disproportionately impact the model's performance (Tan et al., 2019). These preprocessing steps transformed the raw dataset into a clean, balanced format, suitable for exploratory data analysis and predictive modeling.

Modeling and Conclusions

Machine learning models - Multiple Linear Regression, OLS Regression, and Random Forest with cross validation, KNN regression, SVM for regression, and XGBoost regressor - were applied to analyze the relationships between dietary metics and COVID-19 outcomes. Of these models, K-nearest neighbors achieved superior robustness and accuracy by aggregating predictions, reducing overfitting and capturing complex nonlinear relationships of nutrition and mortality rates. Model performance was evaluated by maximizing correlation coefficient (R^2) and reducing mean square error. On a performance standpoint, KNN has the highest R^2 (0.44) and lowest MSE. The support vector machine for regression was the next closest predictor with R^2 (0.43).

The present research, therefore, identifies that nutrition became an integral part of COVID-19 outcomes' forecast, particularly mortality and recovery rate in several parts of the world. Among other findings, the most significant evidence was regions with low fat supply had an enhanced probability of death. These all hint at the strong impact of enough nutritional resources as a propeller of resilience in pandemic health situations. The team followed through

with extensive data preprocessing, handling missing values and outliers for dependability in model performance. EDA highlighted some very critical global nutritional inequities, showing that high-income nations had more dietary resources and better health outcomes. Dimensionality reduction and feature selection methods like RFE increased the efficiency of the whole predictive modeling process by enhancing both accuracy and interpretability.

K-Nearest Neighbor was the best, with an correlation coefficient of 0.44, proving its capability to handle complex nonlinear interactions. Feature importance analysis confirmed protein supply to be the most relevant among the selected predictors of the recovery rates. Although KNN, boasts the highest R^2 of the machine learning models that trained, the SVM with a suitable kernel (e.g., RBF) is better at regression due to its scalability and robust algorithm. KNN is better at predicting local relationships with smaller datasets. In contrast, SVM has a more powerful algorithm at predicting high dimensional data and can be further improved upon in the future with the addition of more features and widely available data. It can handle complex relationships, is less prone to overfitting, and can be more efficient than KNN, especially when dealing with large datasets. Experimentation and careful tuning of hyperparameters are crucial to achieve the best results.

The findings have key implications for public health policy. Introducing dietary considerations into global health policy can enhance resilience at the population level to pandemics and other health emergencies. Dietary disparities, especially within low-income communities, become all the more imperative to redress if health inequities are to be reduced and pandemic preparedness improved. Alcoholic beverages, animal products, animal fats, and meat have a positive correlation with COVID deaths.

References

Analytics Vidhya. (2020, March 30). *Support vector regression tutorial for machine learning*. Retrieved December 3, 2024, from https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/

Analytics Vidhya. (2021, June 17). *Understanding random forest*. Retrieved December 5, 2024, from https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Baumer, B., Kaplan, D., & Horton, N. (n.d.). *Understanding regression: Simple linear regression*. Retrieved December 3, 2024, from https://datasciencebook.ca/regression1.html

Calder, P. C., Carr, A. C., Gombart, A. F., & Eggersdorfer, M. (2020). Optimal nutritional status for a well-functioning immune system is an important factor to protect against viral infections. Nutrients, 12(4), 1181. https://doi.org/10.3390/nu12041181

Larose, C., & Larose, D. T. (2019). Data science using Python and R. John Wiley & Sons.

Mariaren. (2020). COVID-19 healthy diet dataset. Kaggle. Retrieved from https://www.kaggle.com/datasets/mariaren/covid19-healthy-diet-dataset

Tan, P. N., Steinbach, M., Karpatne, A., & Kumar, V. (2019). Introduction to data mining (2nd ed.). Pearson.

University of Utah, Department of Sociology. (n.d.). *Regression analysis*. Retrieved December 9, 2024, from https://soc.utah.edu/sociology3112/regression.php

Yale University, Department of Statistics. (1997). *Linear regression and multiple regression*. Retrieved December 9, 2024, from http://www.stat.yale.edu/Courses/1997-98/101/linmult.htm

502 Final Project

December 9, 2024

1 502 Final Project

We are predicting Deaths and Recovered.

Link to Kaggle Dataset: https://www.kaggle.com/datasets/mariaren/covid19-healthy-diet-dataset

[62]:
```
import pandas as pd
import numpy as np
import seaborn as sns

from sklearn.tree import DecisionTreeClassifier
import statsmodels.api as sm
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn import linear_model
from sklearn.impute import KNNImputer
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFE
from sklearn.metrics import mean_squared_error
from sklearn.feature_selection import RFECV
import xgboost as xgb
from sklearn.svm import SVR
```

[63]:
```
fat_intake = pd.read_csv('Fat_Supply_Quantity_Data.csv')
cal_intake = pd.read_csv('Food_Supply_kcal_Data.csv')
food_intake = pd.read_csv('Food_Supply_Quantity_kg_Data.csv')
protein_intake = pd.read_csv('Protein_Supply_Quantity_Data.csv')
desc = pd.read_csv('Supply_Food_Data_Descriptions.csv')
desc
```

[63]:
```
                  Categories  \
0        Alcoholic Beverages
1                Animal fats
2            Animal Products
```

3	Aquatic Products, Other
4	Cereals - Excluding Beer
5	Eggs
6	Fish, Seafood
7	Fruits - Excluding Wine
8	Meat
9	Milk - Excluding Butter
10	Miscellaneous
11	Offals
12	Oilcrops
13	Pulses
14	Spices
15	Starchy Roots
16	Stimulants
17	Sugar & Sweeteners
18	Sugar Crops
19	Treenuts
20	Vegetable Oils
21	Vegetables
22	Vegetal Products

	Items
0	Alcohol, Non-Food; Beer; Beverages, Alcoholic;…
1	Butter, Ghee; Cream; Fats, Animals, Raw; Fish,…
2	Aquatic Animals, Others; Aquatic Plants; Bovin…
3	Aquatic Animals, Others; Aquatic Plants; Meat,…
4	Barley and products; Cereals, Other; Maize and…
5	Eggs
6	Cephalopods; Crustaceans; Demersal Fish; Fresh…
7	Apples and products; Bananas; Citrus, Other; D…
8	Bovine Meat; Meat, Other; Mutton & Goat Meat; …
9	Milk - Excluding Butter
10	Infant food; Miscellaneous
11	Offals, Edible
12	Coconuts - Incl Copra; Cottonseed; Groundnuts …
13	Beans; Peas; Pulses, Other and products
14	Cloves; Pepper; Pimento; Spices, Other
15	Cassava and products; Potatoes and products; R…
16	Cocoa Beans and products; Coffee and products;…
17	Honey; Sugar (Raw Equivalent); Sugar non-centr…
18	Sugar beet; Sugar cane
19	Nuts and products
20	Coconut Oil; Cottonseed Oil; Groundnut Oil; Ma…
21	Onions; Tomatoes and products; Vegetables, Other
22	Alcohol, Non-Food; Apples and products; Banana…

[64]:
```
fat_intake[['Country','Obesity','Population']].sort_values(by = 'Obesity',␣
↪ascending = False)
```

[64]:

	Country	Obesity	Population
80	Kiribati	45.6	125000.0
131	Samoa	45.5	200000.0
161	United States of America	37.3	329878000.0
83	Kuwait	37.0	4691000.0
133	Saudi Arabia	35.0	35041000.0
..	…	…	…
150	Timor-Leste	2.9	1318000.0
166	Vietnam	2.1	96209000.0
52	French Polynesia	NaN	280000.0
109	New Caledonia	NaN	295000.0
147	Taiwan*	NaN	23610000.0

[170 rows x 3 columns]

The USA is the largest country with the third largest obesity. Kiribati and Samoa both have under 1 million population and their obesity rates suffered due to the introduction of western food products.

[65]:
```
fat_intake[['Country','Obesity']].sort_values(by = 'Obesity', ascending = False)
```

[65]:

	Country	Obesity
80	Kiribati	45.6
131	Samoa	45.5
161	United States of America	37.3
83	Kuwait	37.0
133	Saudi Arabia	35.0
..	…	…
150	Timor-Leste	2.9
166	Vietnam	2.1
52	French Polynesia	NaN
109	New Caledonia	NaN
147	Taiwan*	NaN

[170 rows x 2 columns]

[66]:
```
fat_intake
```

[66]:

	Country	Alcoholic Beverages	Animal Products	\
0	Afghanistan	0.0000	21.6397	
1	Albania	0.0000	32.0002	
2	Algeria	0.0000	14.4175	
3	Angola	0.0000	15.3041	
4	Antigua and Barbuda	0.0000	27.7033	
..	…	…	…	

```
165  Venezuela (Bolivarian Republic of)              0.0000          16.3261
166                             Vietnam              0.0000          33.2484
167                               Yemen              0.0000          12.5401
168                              Zambia              0.0783           9.6005
169                            Zimbabwe              0.0000          10.3796

     Animal fats  Aquatic Products, Other  Cereals - Excluding Beer    Eggs  \
0         6.2224                     0.0                    8.0353  0.6859
1         3.4172                     0.0                    2.6734  1.6448
2         0.8972                     0.0                    4.2035  1.2171
3         1.3130                     0.0                    6.5545  0.1539
4         4.6686                     0.0                    3.2153  0.3872
..           …                       …                         …     …
165       2.2673                     0.0                    2.5449  0.6555
166       3.8238                     0.0                    3.7155  0.7839
167       2.0131                     0.0                   11.5271  0.5514
168       1.6113                     0.0                   14.3225  0.6266
169       2.9543                     0.0                    9.7922  0.3682

     Fish, Seafood  Fruits - Excluding Wine     Meat  …  Vegetable Oils  \
0           0.0327                   0.4246   6.1244  …         17.0831
1           0.1445                   0.6418   8.7428  …          9.2443
2           0.2008                   0.5772   3.8961  …         27.3606
3           1.4155                   0.3488  11.0268  …         22.4638
4           1.5263                   1.2177  14.3202  …         14.4436
..             …                        …       …  …              …
165         0.5707                   0.9640   7.0949  …         29.5211
166         1.1217                   0.4079  26.4292  …          5.6211
167         0.3847                   0.2564   8.0010  …         23.6312
168         1.0070                   0.1343   4.9010  …         15.2848
169         0.2455                   0.0614   4.5674  …         26.9396

     Vegetables  Obesity  Undernourished  Confirmed    Deaths  Recovered  \
0        0.3593      4.5            29.8   0.142134  0.006186   0.123374
1        0.6503     22.3             6.2   2.967301  0.050951   1.792636
2        0.5145     26.6             3.9   0.244897  0.006558   0.167572
3        0.1231      6.8              25   0.061687  0.001461   0.056808
4        0.2469     19.1             NaN   0.293878  0.007143   0.190816
..          …        …               …         …         …          …
165      0.1851     25.2            21.2   0.452585  0.004287   0.424399
166      0.6373      2.1             9.3   0.002063  0.000036   0.001526
167      0.1667     14.1            38.9   0.007131  0.002062   0.004788
168      0.1567      6.5            46.7   0.334133  0.004564   0.290524
169      0.0789     12.3            51.3   0.232033  0.008854   0.190964

       Active  Population  Unit (all except Population)
0    0.012574  38928000.0                             %
```

```
1    1.123714   2838000.0                          %
2    0.070767  44357000.0                          %
3    0.003419  32522000.0                          %
4    0.095918     98000.0                          %
..        …         …                        …
165  0.023899  28645000.0                          %
166  0.000501  96209000.0                          %
167  0.000282  29826000.0                          %
168  0.039045  18384000.0                          %
169  0.032214  14863000.0                          %

[170 rows x 32 columns]
```

[67]:
```
cal_intake
```

[67]:
```
                              Country  Alcoholic Beverages  Animal Products  \
0                         Afghanistan               0.0000           4.7774
1                             Albania               0.9120          16.0930
2                             Algeria               0.0896           6.0326
3                              Angola               1.9388           4.6927
4                  Antigua and Barbuda               2.3041          15.3672
..                                 …                    …               …
165  Venezuela (Bolivarian Republic of)               0.8454           7.2303
166                            Vietnam               0.7150          10.9806
167                              Yemen               0.0000           3.4667
168                             Zambia               1.1925           3.3043
169                           Zimbabwe               1.4269           3.9356

     Animal fats  Aquatic Products, Other  Cereals - Excluding Beer    Eggs  \
0         0.8504                     0.0                   37.1186  0.1501
1         1.0591                     0.0                   16.2107  0.8091
2         0.1941                     0.0                   25.0112  0.4181
3         0.2644                     0.0                   18.3521  0.0441
4         1.5429                     0.0                   13.7215  0.2057
..           …                       …                         …     …
165       0.6007                     0.0                   21.3126  0.2892
166       0.9363                     0.0                   26.9833  0.2894
167       0.3394                     0.0                   32.0727  0.1455
168       0.3230                     0.0                   31.5528  0.1988
169       0.6904                     0.0                   29.8044  0.1381

     Fish, Seafood  Fruits - Excluding Wine    Meat  …  Vegetable Oils  \
0           0.0000                  1.4757  1.2006  …          2.3012
1           0.1471                  3.8982  3.8688  …          2.8244
2           0.1195                  3.1805  1.2543  …          5.7638
3           0.8372                  2.3133  2.9302  …          4.2741
4           1.7280                  3.6824  7.0356  …          4.6904
```

```
..              …                       …     …  …              …
165         0.4449                  2.3804  3.1368  …          7.5417
166         1.0385                  1.8046  7.8311  …          1.3279
167         0.1697                  1.1879  2.0121  …          3.9515
168         0.5714                  0.2236  1.5155  …          3.0062
169         0.1611                  0.4373  1.7491  …          6.2601

     Vegetables  Obesity  Undernourished  Confirmed    Deaths  Recovered  \
0        0.7504      4.5            29.8   0.142134  0.006186   0.123374
1        2.7508     22.3             6.2   2.967301  0.050951   1.792636
2        2.0457     26.6             3.9   0.244897  0.006558   0.167572
3        0.3525      6.8              25   0.061687  0.001461   0.056808
4        1.2960     19.1             NaN   0.293878  0.007143   0.190816
..          …      …               …        …       …        …
165      0.6674     25.2            21.2   0.452585  0.004287   0.424399
166      1.9578      2.1             9.3   0.002063  0.000036   0.001526
167      0.3636     14.1            38.9   0.007131  0.002062   0.004788
168      0.4472      6.5            46.7   0.334133  0.004564   0.290524
169      0.2532     12.3            51.3   0.232033  0.008854   0.190964

       Active  Population  Unit (all except Population)
0    0.012574  38928000.0                             %
1    1.123714   2838000.0                             %
2    0.070767  44357000.0                             %
3    0.003419  32522000.0                             %
4    0.095918     98000.0                             %
..        …         …                          …
165  0.023899  28645000.0                             %
166  0.000501  96209000.0                             %
167  0.000282  29826000.0                             %
168  0.039045  18384000.0                             %
169  0.032214  14863000.0                             %

[170 rows x 32 columns]
```

[68]:
```
food_intake
```

[68]:
```
                              Country  Alcoholic Beverages  Animal fats  \
0                         Afghanistan                0.0014       0.1973
1                             Albania                1.6719       0.1357
2                             Algeria                0.2711       0.0282
3                              Angola                5.8087       0.0560
4                  Antigua and Barbuda                3.5764       0.0087
..                                 …                    …          …
165  Venezuela (Bolivarian Republic of)                2.5952       0.0403
166                            Vietnam                1.4591       0.1640
167                              Yemen                0.0364       0.0446
```

```
168                          Zambia              5.7360       0.0829
169                        Zimbabwe              4.0552       0.0755

     Animal Products  Aquatic Products, Other  Cereals - Excluding Beer  \
0             9.4341                  0.0000                    24.8097
1            18.7684                  0.0000                     5.7817
2             9.6334                  0.0000                    13.6816
3             4.9278                  0.0000                     9.1085
4            16.6613                  0.0000                     5.9960
..               …                       …                         …
165          14.7565                  0.0000                    12.9253
166           8.5765                  0.0042                    16.8740
167           5.7874                  0.0000                    27.2077
168           6.0197                  0.0000                    21.1938
169           8.1489                  0.0000                    22.6240

       Eggs  Fish, Seafood  Fruits - Excluding Wine    Meat  …  Vegetables  \
0    0.2099         0.0350                   5.3495  1.2020  …      6.7642
1    0.5815         0.2126                   6.7861  1.8845  …     11.7753
2    0.5277         0.2416                   6.3801  1.1305  …     11.6484
3    0.0587         1.7707                   6.0005  2.0571  …      2.3041
4    0.2274         4.1489                  10.7451  5.6888  …      5.4495
..      …              …                        …       …  …         …
165  0.3389         0.9456                   7.6460  3.8328  …      4.1474
166  0.3077         2.6392                   5.9029  4.4382  …     11.9508
167  0.2579         0.5240                   5.1344  2.7871  …      3.2135
168  0.3399         1.6924                   1.0183  1.8427  …      3.4649
169  0.2678         0.5518                   2.2000  2.6142  …      2.3213

     Vegetal Products  Obesity  Undernourished  Confirmed    Deaths  \
0             40.5645      4.5            29.8   0.142134  0.006186
1             31.2304     22.3             6.2   2.967301  0.050951
2             40.3651     26.6             3.9   0.244897  0.006558
3             45.0722      6.8              25   0.061687  0.001461
4             33.3233     19.1             NaN   0.293878  0.007143
..               …        …               …        …         …
165           35.2416     25.2            21.2   0.452585  0.004287
166           41.4232      2.1             9.3   0.002063  0.000036
167           44.2126     14.1            38.9   0.007131  0.002062
168           43.9789      6.5            46.7   0.334133  0.004564
169           41.8526     12.3            51.3   0.232033  0.008854

     Recovered    Active  Population  Unit (all except Population)
0     0.123374  0.012574  38928000.0                             %
1     1.792636  1.123714   2838000.0                             %
2     0.167572  0.070767  44357000.0                             %
3     0.056808  0.003419  32522000.0                             %
```

```
4     0.190816  0.095918     98000.0                          %
..         …       …          …                    …
165   0.424399  0.023899  28645000.0                          %
166   0.001526  0.000501  96209000.0                          %
167   0.004788  0.000282  29826000.0                          %
168   0.290524  0.039045  18384000.0                          %
169   0.190964  0.032214  14863000.0                          %

[170 rows x 32 columns]
```

[69]:
```
protein_intake
```

[69]:
```
                              Country  Alcoholic Beverages  Animal Products  \
0                         Afghanistan               0.0000           9.7523
1                             Albania               0.1840          27.7469
2                             Algeria               0.0323          13.8360
3                              Angola               0.6285          15.2311
4                 Antigua and Barbuda               0.1535          33.1901
..                                 …                    …                …
165  Venezuela (Bolivarian Republic of)               0.1955          22.5411
166                           Vietnam               0.1555          20.4466
167                             Yemen               0.0000          10.0122
168                            Zambia               0.4824           9.8925
169                          Zimbabwe               0.2929          11.3443

     Animal fats  Aquatic Products, Other  Cereals - Excluding Beer    Eggs  \
0         0.0277                   0.0000                   35.9771  0.4067
1         0.0711                   0.0000                   14.2331  1.8069
2         0.0054                   0.0000                   26.5633  1.2916
3         0.0277                   0.0000                   20.3882  0.1756
4         0.1289                   0.0000                   10.5108  0.4850
..           …                        …                         …       …
165       0.1244                   0.0000                   21.6526  0.8707
166       0.1555                   0.0056                   18.5247  0.7665
167       0.0188                   0.0000                   35.1179  0.4320
168       0.0338                   0.0000                   28.5182  0.5839
169       0.0391                   0.0000                   33.1934  0.5077

     Fish, Seafood  Fruits - Excluding Wine     Meat  …  Vegetables  \
0           0.0647                  0.5824   3.1337  …      1.1370
1           0.6274                  1.2757   7.6582  …      3.2456
2           0.6350                  1.1624   3.5088  …      3.1267
3           5.4436                  1.2754   7.6248  …      0.8133
4           8.2146                  1.2586  16.0670  …      1.6024
..             …                       …       …  …         …
165         2.6477                  1.0662  11.8347  …      1.0129
166         5.7435                  0.7165  11.0426  …      3.7216
```

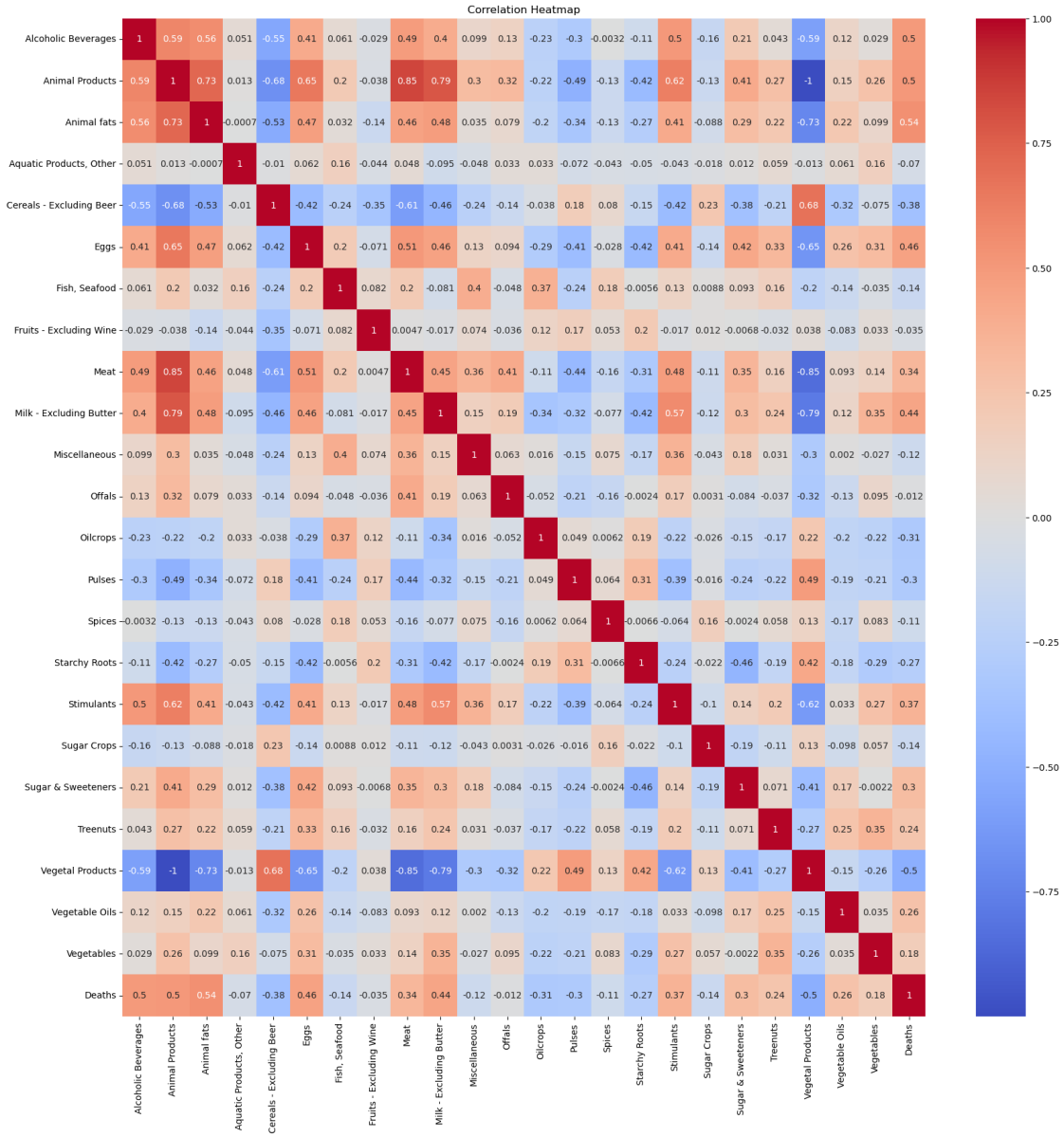
```
167         0.9392                   0.4884   5.9453  …      0.5448
168         3.0126                   0.0931   4.3158  …      0.8039
169         1.0837                   0.2636   6.6582  …      0.5955

     Miscellaneous  Obesity  Undernourished  Confirmed    Deaths  Recovered  \
0           0.0462      4.5            29.8   0.142134  0.006186   0.123374
1           0.0544     22.3             6.2   2.967301  0.050951   1.792636
2           0.1399     26.6             3.9   0.244897  0.006558   0.167572
3           0.0924      6.8              25   0.061687  0.001461   0.056808
4           0.2947     19.1             NaN   0.293878  0.007143   0.190816
..             …        …               …          …         …          …
165         0.0267     25.2            21.2   0.452585  0.004287   0.424399
166         0.0389      2.1             9.3   0.002063  0.000036   0.001526
167         0.0564     14.1            38.9   0.007131  0.002062   0.004788
168         0.0592      6.5            46.7   0.334133  0.004564   0.290524
169         0.0586     12.3            51.3   0.232033  0.008854   0.190964

       Active  Population  Unit (all except Population)
0    0.012574  38928000.0                             %
1    1.123714   2838000.0                             %
2    0.070767  44357000.0                             %
3    0.003419  32522000.0                             %
4    0.095918     98000.0                             %
..        …         …                              …
165  0.023899  28645000.0                             %
166  0.000501  96209000.0                             %
167  0.000282  29826000.0                             %
168  0.039045  18384000.0                             %
169  0.032214  14863000.0                             %

[170 rows x 32 columns]
```

[70]:
```
desc['Categories']
```

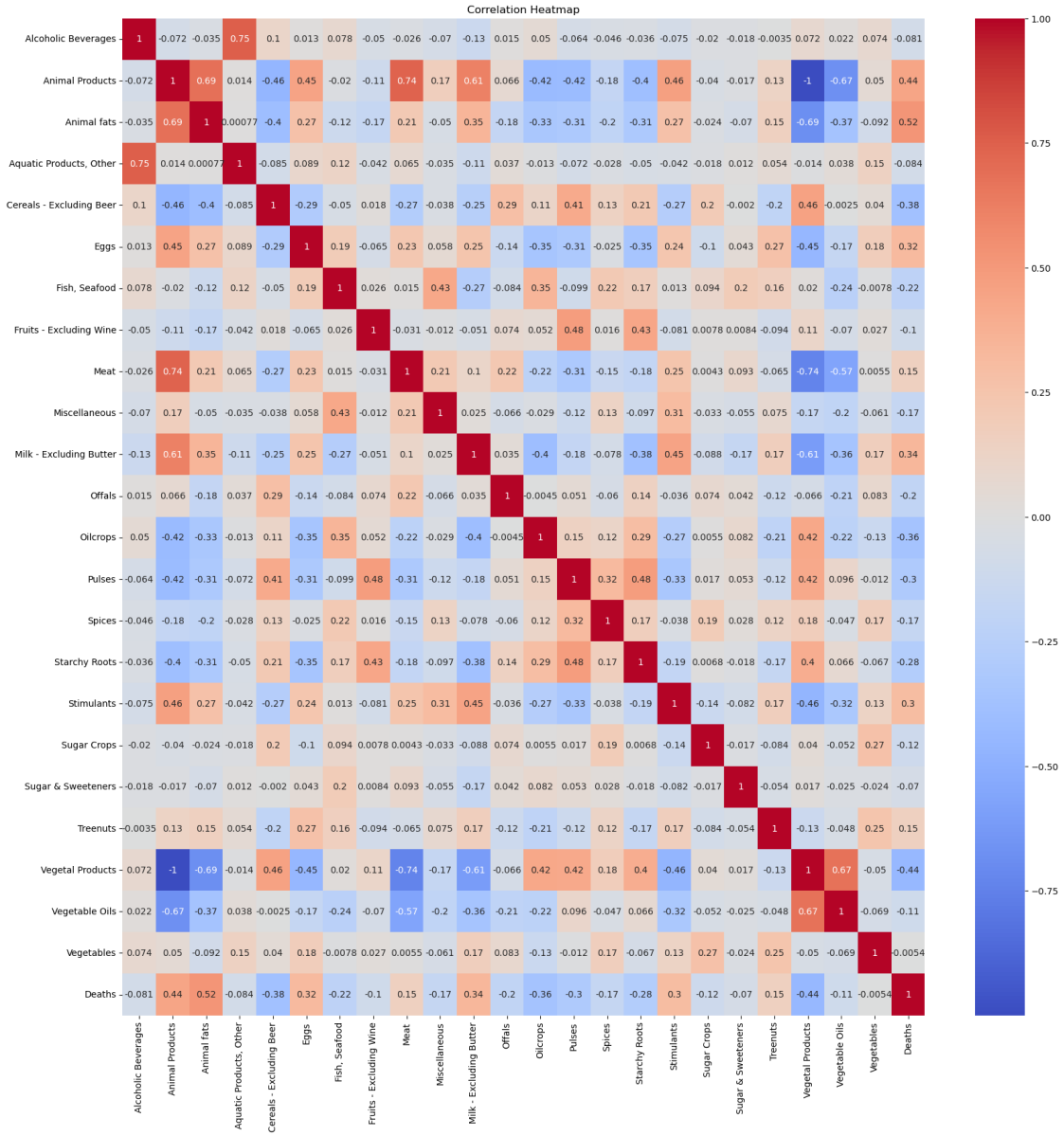
[70]:
```
0          Alcoholic Beverages
1                 Animal fats
2              Animal Products
3      Aquatic Products, Other
4     Cereals - Excluding Beer
5                         Eggs
6                Fish, Seafood
7      Fruits - Excluding Wine
8                         Meat
9      Milk - Excluding Butter
10               Miscellaneous
11                       Offals
12                    Oilcrops
```

```
13                      Pulses
14                      Spices
15                Starchy Roots
16                   Stimulants
17          Sugar & Sweeteners
18                 Sugar Crops
19                    Treenuts
20               Vegetable Oils
21                   Vegetables
22             Vegetal Products
Name: Categories, dtype: object
```

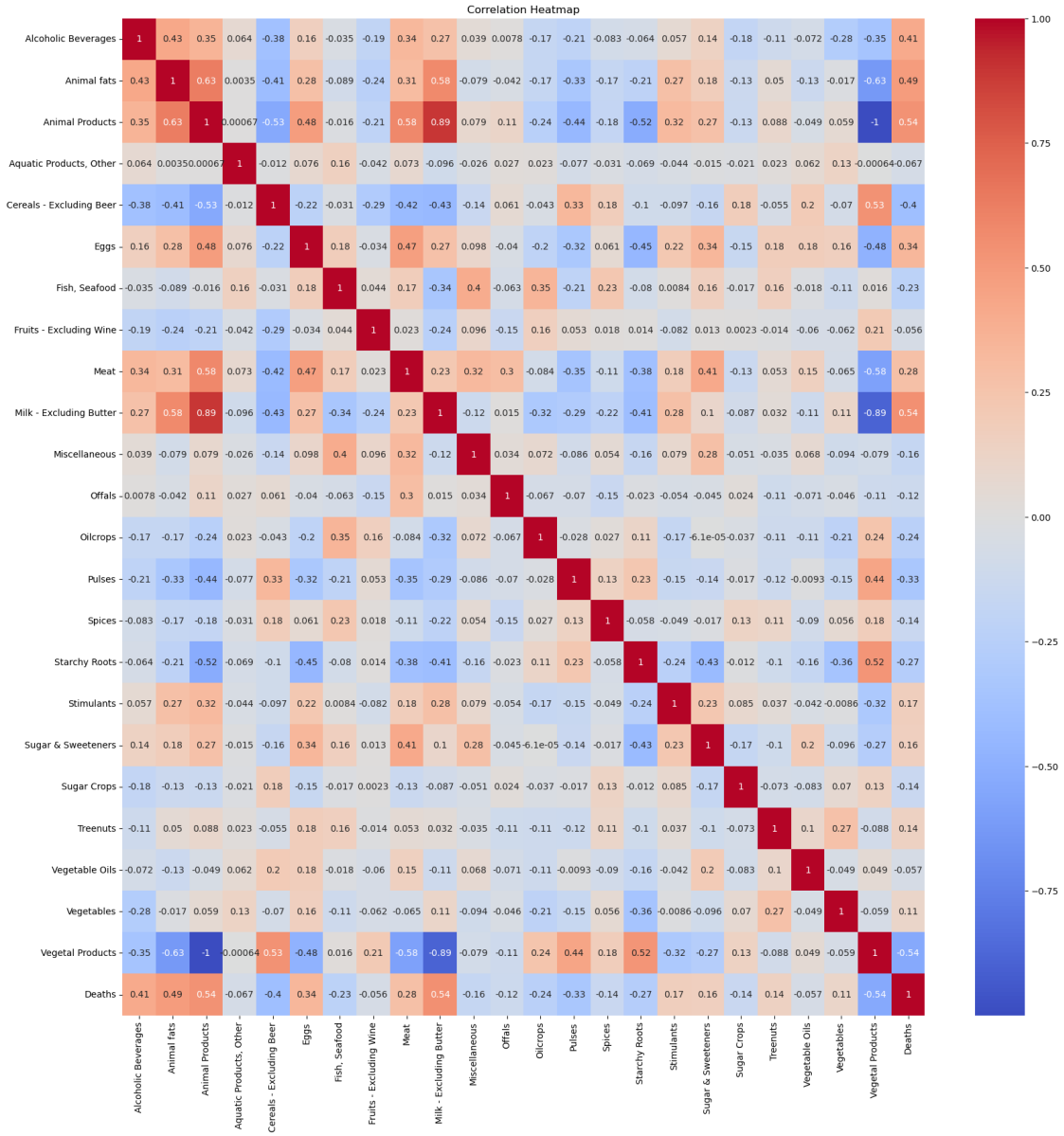
[71]:
```
for x in [cal_intake, fat_intake, food_intake, protein_intake]:

    copied = x.copy()
    correlation_matrix = copied.drop(['Obesity', 'Undernourished', 'Confirmed',␣
 ↪'Recovered', 'Active', 'Population', 'Unit (all except Population)'], axis =␣
 ↪1).corr()

    # Heatmap
    plt.figure(figsize=(20,20))
    sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
    plt.title('Correlation Heatmap')
    plt.show()

    # Correlation coefficient between 'deaths' and 'recovered'
    correlation_coefficient = copied['Deaths'].corr(copied['Recovered'])
    print("Correlation Coefficient between Deaths and Recovered:",␣
 ↪correlation_coefficient)
plt.show()
```



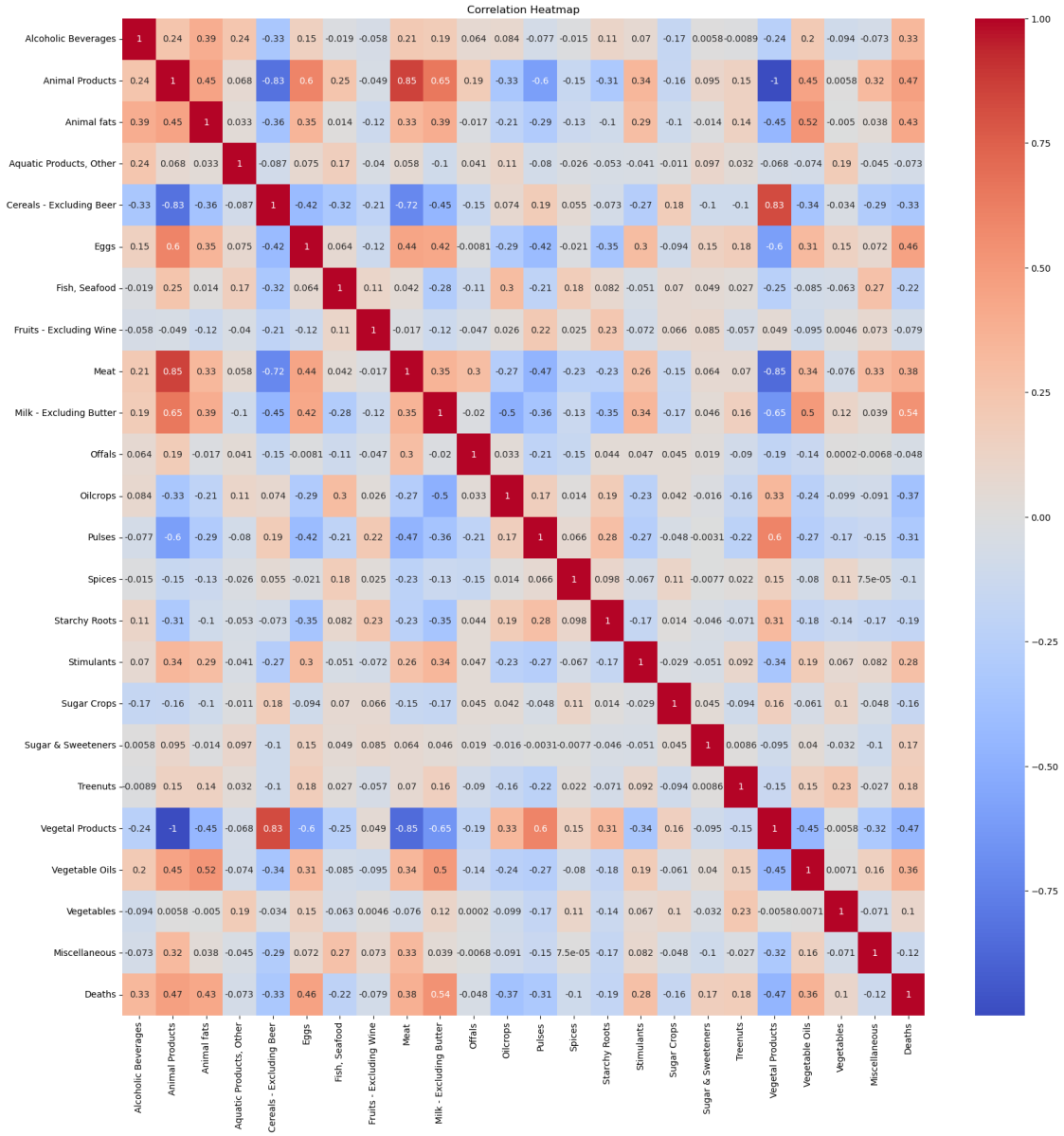
Correlation Coefficient between Deaths and Recovered: 0.6718005807074969



Correlation Coefficient between Deaths and Recovered: 0.6718005807074969



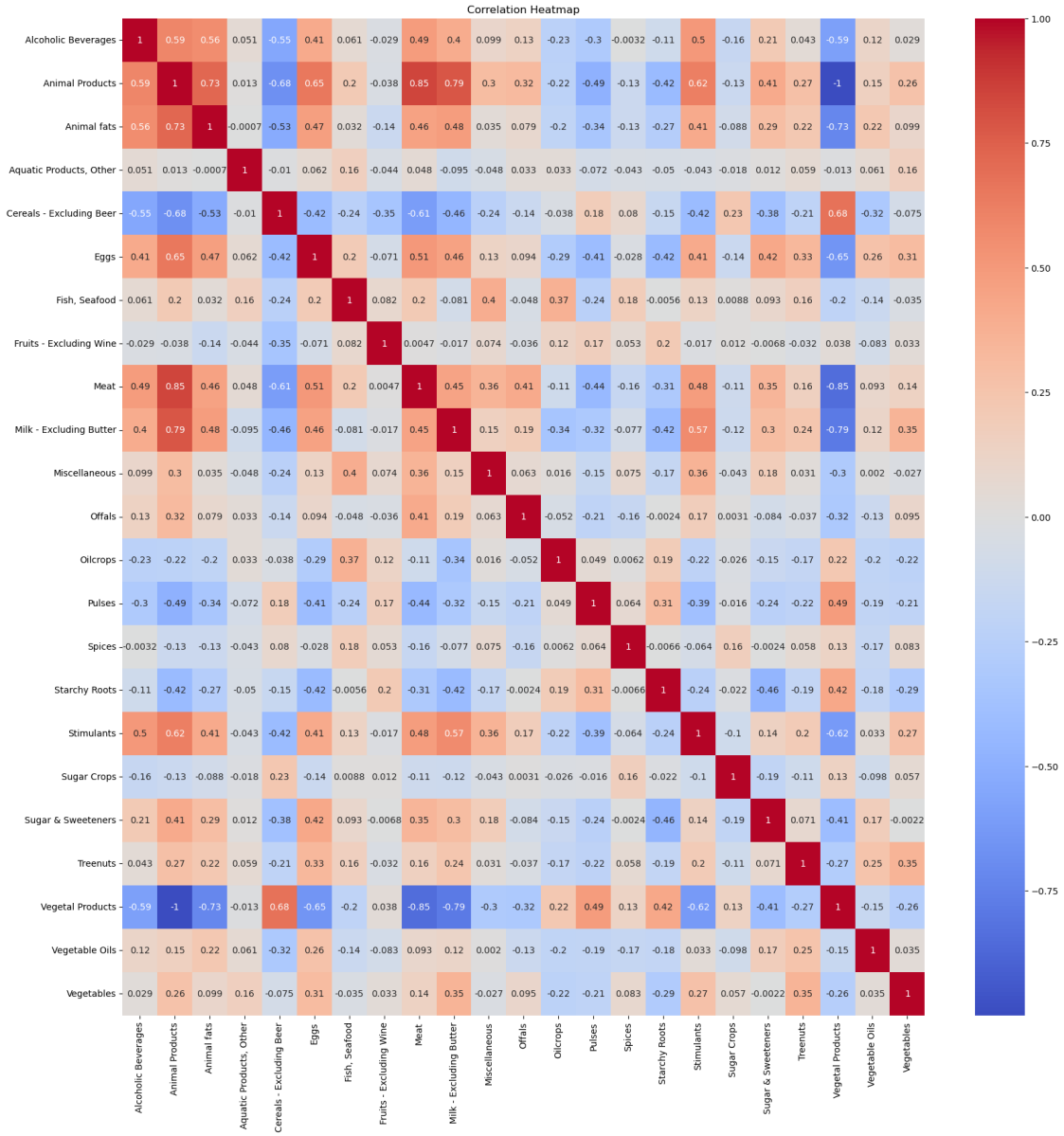
Correlation Coefficient between Deaths and Recovered: 0.6718005807074969



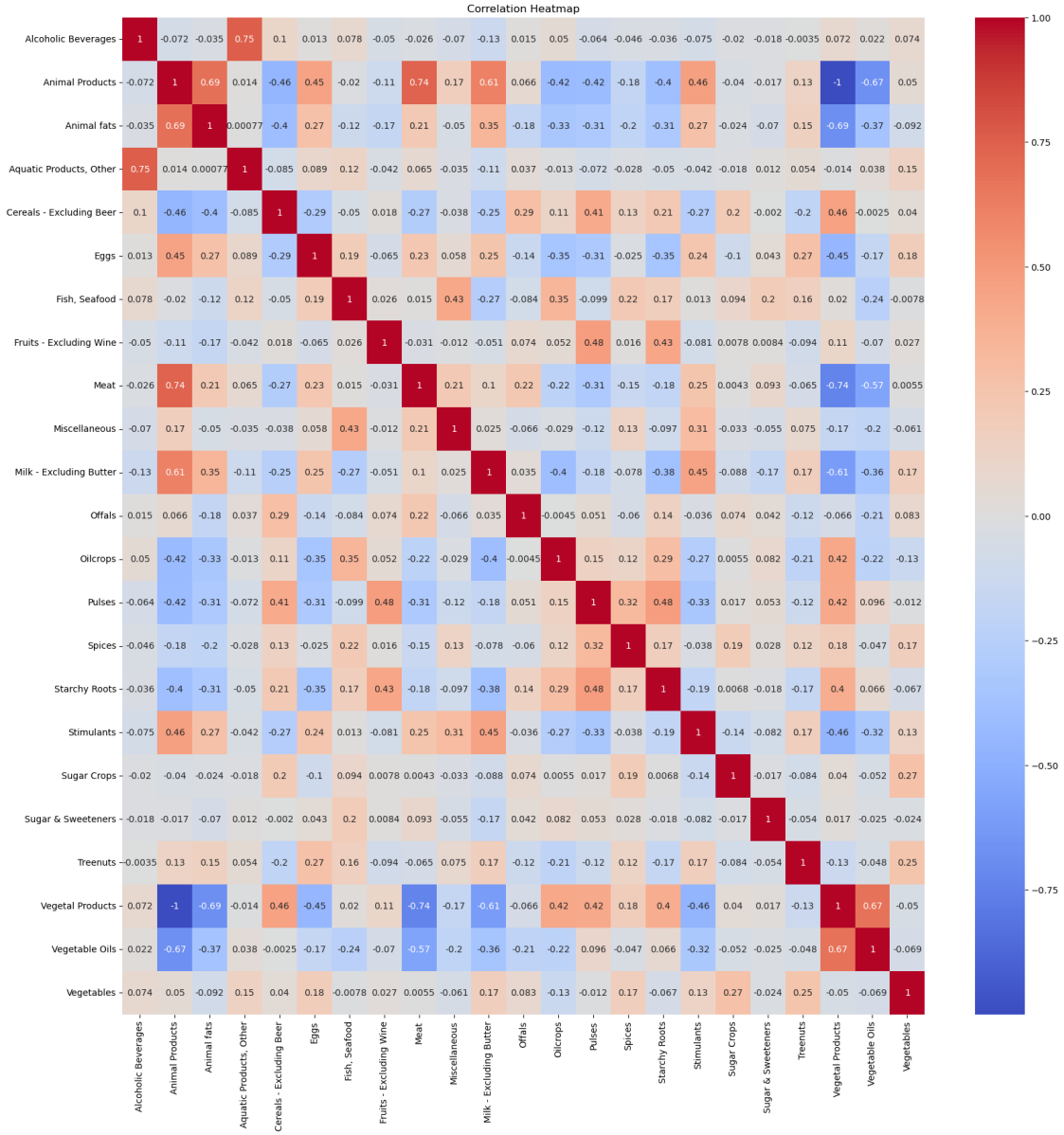
Correlation Coefficient between Deaths and Recovered: 0.6718005807074969

[72]:
```
for x in [cal_intake, fat_intake, food_intake, protein_intake]:
    # Correlation matrix with ['Obesity', 'Undernourished', 'Confirmed',␣
↪'Deaths', 'Recovered', 'Active', 'Population', 'Unit (all except␣
↪Population)'] removed
    copied = x.copy()
    correlation_matrix = copied.drop(['Obesity', 'Undernourished', 'Confirmed',␣
↪'Deaths', 'Recovered', 'Active', 'Population', 'Unit (all except␣
↪Population)'], axis = 1).corr()
```

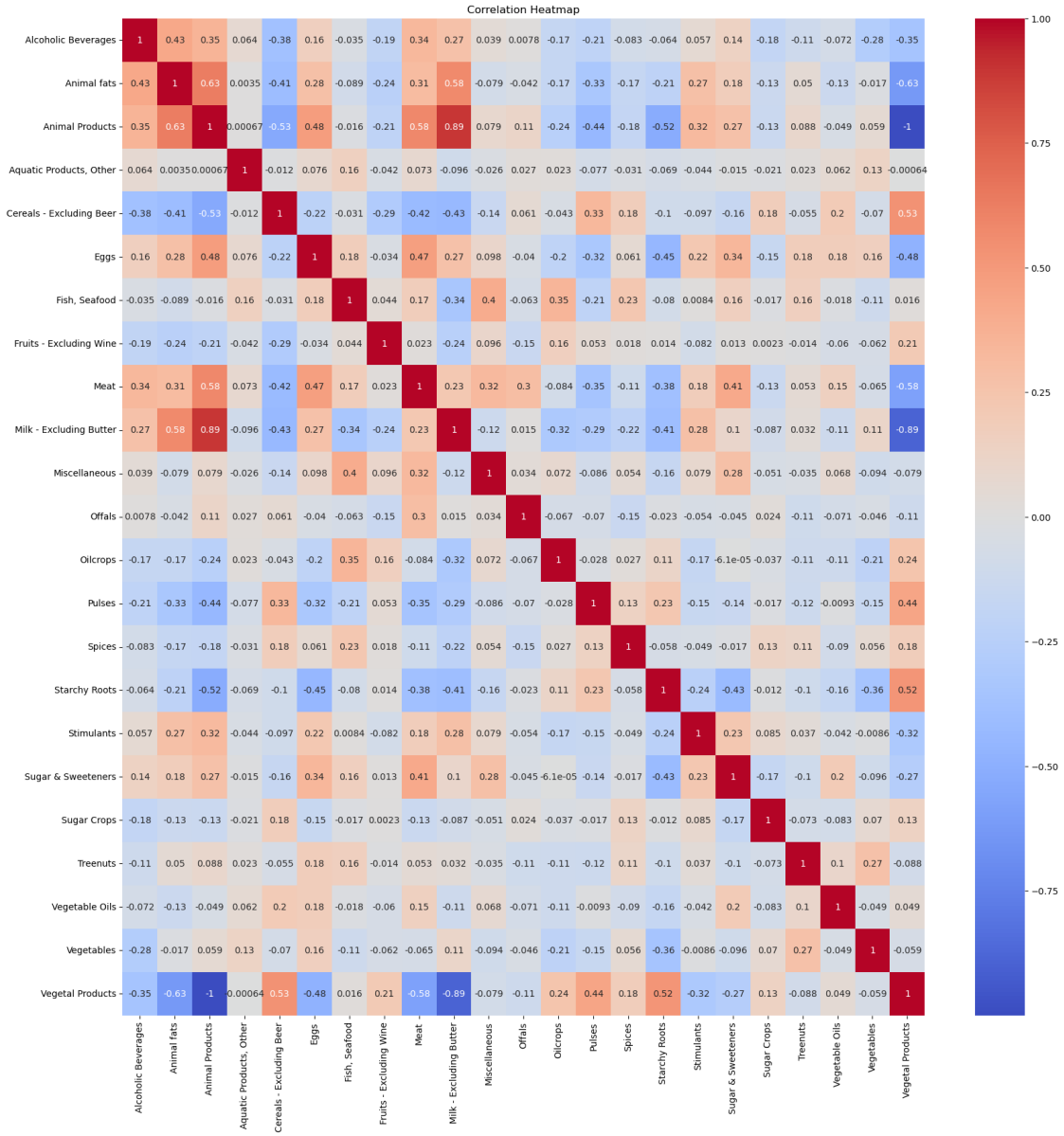
```
    # Heatmap
    plt.figure(figsize=(20,20))
    sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
    plt.title('Correlation Heatmap')
    plt.show()

    # Correlation coefficient between 'deaths' and 'recovered'
    correlation_coefficient = copied['Deaths'].corr(copied['Recovered'])
    print("Correlation Coefficient between Deaths and Recovered:",␣
 ↪correlation_coefficient)
plt.show()
```



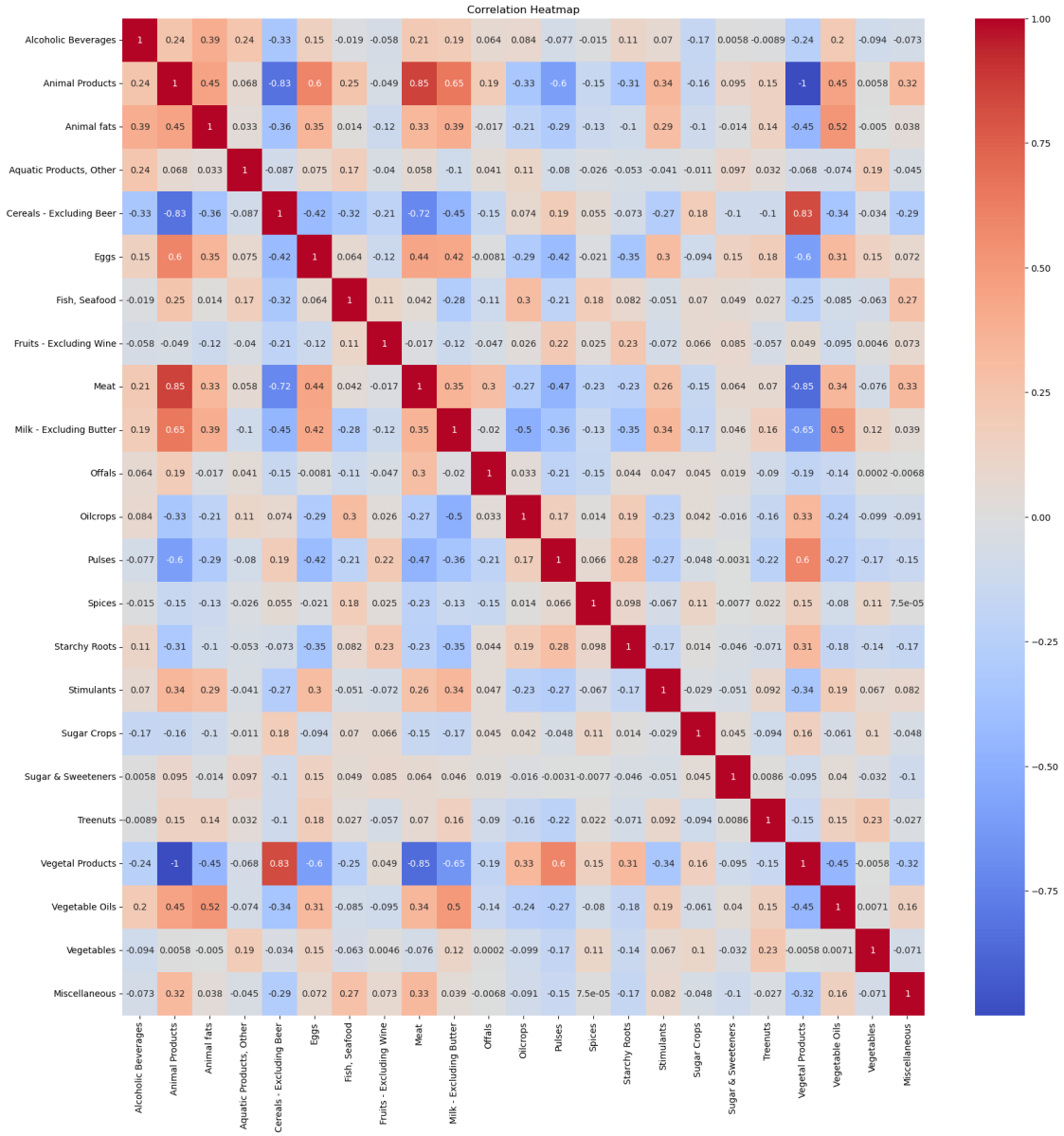
Correlation Coefficient between Deaths and Recovered: 0.6718005807074969



Correlation Coefficient between Deaths and Recovered: 0.6718005807074969



Correlation Coefficient between Deaths and Recovered: 0.6718005807074969




```
Correlation Coefficient between Deaths and Recovered: 0.6718005807074969
```

Animal products and vegetal products are highly negatively correlated so we should remove one (we chose animal products) to avoid mulitcolinearity.

2 Merge all dataframes together

[73]:
```
#Removed all similar columns ['Obesity', 'Undernourished', 'Confirmed',␣
↪'Deaths', 'Recovered', 'Active', 'Population', 'Unit (all except␣
↪Population)']
```

```
cal_copy = cal_intake.copy().drop(['Animal Products', 'Unit (all except␣
 ↪Population)'], axis = 1)
fat_copy = fat_intake.copy().drop(['Animal Products', 'Obesity',␣
 ↪'Undernourished', 'Confirmed', 'Deaths', 'Recovered', 'Active',␣
 ↪'Population', 'Unit (all except Population)'], axis = 1)
food_copy = food_intake.copy().drop(['Animal Products', 'Obesity',␣
 ↪'Undernourished', 'Confirmed', 'Deaths', 'Recovered', 'Active',␣
 ↪'Population', 'Unit (all except Population)'], axis = 1)
protein_copy = protein_intake.copy().drop(['Animal Products', 'Obesity',␣
 ↪'Undernourished', 'Confirmed', 'Deaths', 'Recovered', 'Active',␣
 ↪'Population', 'Unit (all except Population)'], axis = 1)

#rename all columns with suffix to differentiate
cal_copy.columns = cal_copy.columns + '_cal_int'
fat_copy.columns = fat_copy.columns + '_fat_int'
food_copy.columns = food_copy.columns + '_food_int'
protein_copy.columns = protein_copy.columns + '_protein_int'

#merge all dataframe together
merged = pd.merge(cal_copy, fat_copy, left_on = 'Country_cal_int', right_on =␣
 ↪'Country_fat_int', how ='inner').merge(food_copy, left_on =␣
 ↪'Country_cal_int', right_on = 'Country_food_int', how = 'inner').
 ↪merge(protein_copy, left_on = 'Country_cal_int', right_on =␣
 ↪'Country_protein_int', how = 'inner')
merged = merged.
 ↪drop(['Country_fat_int','Country_food_int','Country_protein_int'], axis = 1)
merged
```

[73]:

	Country_cal_int	Alcoholic Beverages_cal_int	\
0	Afghanistan	0.0000	
1	Albania	0.9120	
2	Algeria	0.0896	
3	Angola	1.9388	
4	Antigua and Barbuda	2.3041	
..	…	…	
165	Venezuela (Bolivarian Republic of)	0.8454	
166	Vietnam	0.7150	
167	Yemen	0.0000	
168	Zambia	1.1925	
169	Zimbabwe	1.4269	

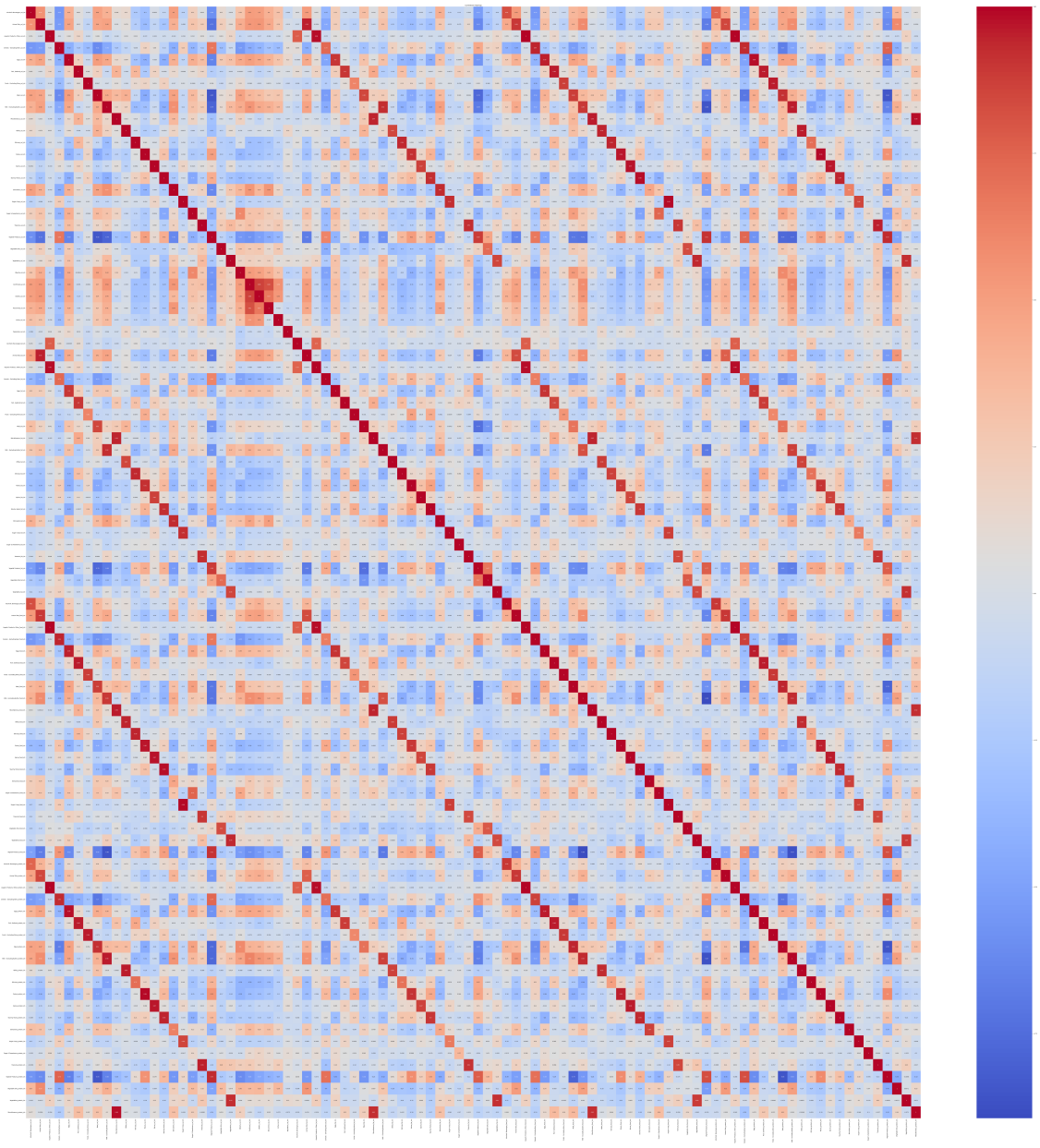
	Animal fats_cal_int	Aquatic Products, Other_cal_int	\
0	0.8504	0.0	
1	1.0591	0.0	
2	0.1941	0.0	
3	0.2644	0.0	
4	1.5429	0.0	

```
..                    …                                  …
165               0.6007                               0.0
166               0.9363                               0.0
167               0.3394                               0.0
168               0.3230                               0.0
169               0.6904                               0.0

     Cereals - Excluding Beer_cal_int  Eggs_cal_int  Fish, Seafood_cal_int  \
0                              37.1186        0.1501                 0.0000
1                              16.2107        0.8091                 0.1471
2                              25.0112        0.4181                 0.1195
3                              18.3521        0.0441                 0.8372
4                              13.7215        0.2057                 1.7280
..                                 …             …                      …
165                            21.3126        0.2892                 0.4449
166                            26.9833        0.2894                 1.0385
167                            32.0727        0.1455                 0.1697
168                            31.5528        0.1988                 0.5714
169                            29.8044        0.1381                 0.1611

     Fruits - Excluding Wine_cal_int  Meat_cal_int  \
0                             1.4757        1.2006
1                             3.8982        3.8688
2                             3.1805        1.2543
3                             2.3133        2.9302
4                             3.6824        7.0356
..                               …             …
165                           2.3804        3.1368
166                           1.8046        7.8311
167                           1.1879        2.0121
168                           0.2236        1.5155
169                           0.4373        1.7491

     Milk - Excluding Butter_cal_int  …  Spices_protein_int  \
0                             2.4512  …              0.1664
1                             9.9441  …              0.0000
2                             3.9869  …              0.1776
3                             0.5067  …              0.0000
4                             4.6904  …              0.3438
..                               …  …                   …
165                           2.6474  …              0.0000
166                           0.5618  …              0.5221
167                           0.6545  …              0.0657
168                           0.5217  …              0.0846
169                           1.0817  …              0.0488

     Starchy Roots_protein_int  Stimulants_protein_int  \
```

```
0                      0.1941                   0.5546
1                      0.8867                   0.2635
2                      1.4638                   0.4628
3                      5.1941                   0.1017
4                      0.4666                   0.4113
..                        …                        …
165                    1.1195                   0.3287
166                    0.2333                   0.5444
167                    0.2066                   0.3193
168                    1.2863                   0.0762
169                    0.5467                   0.2636

     Sugar Crops_protein_int  Sugar & Sweeteners_protein_int  \
0                     0.0000                          0.0000
1                     0.0000                          0.0042
2                     0.0000                          0.0000
3                     0.0000                          0.0092
4                     0.0000                          0.0000
..                       …                               …
165                   0.0000                          0.0178
166                   0.0167                          0.0056
167                   0.0000                          0.0000
168                   0.0000                          0.0000
169                   0.0000                          0.0000

     Treenuts_protein_int  Vegetal Products_protein_int  \
0                  0.1387                       40.2477
1                  0.2677                       22.2552
2                  0.2745                       36.1694
3                  0.0092                       34.7782
4                  0.0737                       16.8161
..                    …                            …
165                0.0000                       27.4545
166                0.3277                       29.5617
167                0.0188                       39.9831
168                0.0000                       40.1117
169                0.1367                       38.6508

     Vegetable Oils_protein_int  Vegetables_protein_int  \
0                        0.0000                  1.1370
1                        0.0084                  3.2456
2                        0.0269                  3.1267
3                        0.0092                  0.8133
4                        0.0430                  1.6024
..                          …                       …
165                      0.0533                  1.0129
166                      0.0000                  3.7216
```

```
167                      0.0000                  0.5448
168                      0.0000                  0.8039
169                      0.0293                  0.5955

     Miscellaneous_protein_int
0                       0.0462
1                       0.0544
2                       0.1399
3                       0.0924
4                       0.2947
..                         …
165                     0.0267
166                     0.0389
167                     0.0564
168                     0.0592
169                     0.0586

[170 rows x 96 columns]
```

[74]:
```
# Correlation matrix
correlation_matrix = merged.corr()

# Heatmap
plt.figure(figsize=(150, 150))
ax = sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.title('Correlation Heatmap')
plt.show()

# Correlation coefficient between 'deaths' and 'recovered'
correlation_coefficient = merged['Deaths_cal_int'].
 ↪corr(merged['Recovered_cal_int'])
print("Correlation Coefficient between Deaths and Recovered:",␣
 ↪correlation_coefficient)

ax.set_xticklabels(ax.get_xticklabels(), fontsize= 50)
ax.set_yticklabels(ax.get_yticklabels(), fontsize= 50)

plt.show()
```



```
Correlation Coefficient between Deaths and Recovered: 0.6718005807074969
```

3 Visualize the data

[75]:
```
# summarize the number of rows with missing values for each column
missing_col = []
for x in merged.columns:
    n_miss = merged[x].isnull().sum()
    perc = n_miss / merged.shape[0] * 100
```

```
    print(f'> {x}, Missing: %d (%.1f%%)' % (n_miss, perc))
    if perc > 0:
        missing_col.append(x)
print(f"\n {missing_col} are all the columns with over 0% missing data")
```

```
> Country_cal_int, Missing: 0 (0.0%)
> Alcoholic Beverages_cal_int, Missing: 0 (0.0%)
> Animal fats_cal_int, Missing: 0 (0.0%)
> Aquatic Products, Other_cal_int, Missing: 0 (0.0%)
> Cereals - Excluding Beer_cal_int, Missing: 0 (0.0%)
> Eggs_cal_int, Missing: 0 (0.0%)
> Fish, Seafood_cal_int, Missing: 0 (0.0%)
> Fruits - Excluding Wine_cal_int, Missing: 0 (0.0%)
> Meat_cal_int, Missing: 0 (0.0%)
> Milk - Excluding Butter_cal_int, Missing: 0 (0.0%)
> Miscellaneous_cal_int, Missing: 0 (0.0%)
> Offals_cal_int, Missing: 0 (0.0%)
> Oilcrops_cal_int, Missing: 0 (0.0%)
> Pulses_cal_int, Missing: 0 (0.0%)
> Spices_cal_int, Missing: 0 (0.0%)
> Starchy Roots_cal_int, Missing: 0 (0.0%)
> Stimulants_cal_int, Missing: 0 (0.0%)
> Sugar Crops_cal_int, Missing: 0 (0.0%)
> Sugar & Sweeteners_cal_int, Missing: 0 (0.0%)
> Treenuts_cal_int, Missing: 0 (0.0%)
> Vegetal Products_cal_int, Missing: 0 (0.0%)
> Vegetable Oils_cal_int, Missing: 0 (0.0%)
> Vegetables_cal_int, Missing: 0 (0.0%)
> Obesity_cal_int, Missing: 3 (1.8%)
> Undernourished_cal_int, Missing: 7 (4.1%)
> Confirmed_cal_int, Missing: 6 (3.5%)
> Deaths_cal_int, Missing: 6 (3.5%)
> Recovered_cal_int, Missing: 6 (3.5%)
> Active_cal_int, Missing: 8 (4.7%)
> Population_cal_int, Missing: 0 (0.0%)
> Alcoholic Beverages_fat_int, Missing: 0 (0.0%)
> Animal fats_fat_int, Missing: 0 (0.0%)
> Aquatic Products, Other_fat_int, Missing: 0 (0.0%)
> Cereals - Excluding Beer_fat_int, Missing: 0 (0.0%)
> Eggs_fat_int, Missing: 0 (0.0%)
> Fish, Seafood_fat_int, Missing: 0 (0.0%)
> Fruits - Excluding Wine_fat_int, Missing: 0 (0.0%)
> Meat_fat_int, Missing: 0 (0.0%)
> Miscellaneous_fat_int, Missing: 0 (0.0%)
> Milk - Excluding Butter_fat_int, Missing: 0 (0.0%)
> Offals_fat_int, Missing: 0 (0.0%)
> Oilcrops_fat_int, Missing: 0 (0.0%)
> Pulses_fat_int, Missing: 0 (0.0%)
```

```
> Spices_fat_int, Missing: 0 (0.0%)
> Starchy Roots_fat_int, Missing: 0 (0.0%)
> Stimulants_fat_int, Missing: 0 (0.0%)
> Sugar Crops_fat_int, Missing: 0 (0.0%)
> Sugar & Sweeteners_fat_int, Missing: 0 (0.0%)
> Treenuts_fat_int, Missing: 0 (0.0%)
> Vegetal Products_fat_int, Missing: 0 (0.0%)
> Vegetable Oils_fat_int, Missing: 0 (0.0%)
> Vegetables_fat_int, Missing: 0 (0.0%)
> Alcoholic Beverages_food_int, Missing: 0 (0.0%)
> Animal fats_food_int, Missing: 0 (0.0%)
> Aquatic Products, Other_food_int, Missing: 0 (0.0%)
> Cereals - Excluding Beer_food_int, Missing: 0 (0.0%)
> Eggs_food_int, Missing: 0 (0.0%)
> Fish, Seafood_food_int, Missing: 0 (0.0%)
> Fruits - Excluding Wine_food_int, Missing: 0 (0.0%)
> Meat_food_int, Missing: 0 (0.0%)
> Milk - Excluding Butter_food_int, Missing: 0 (0.0%)
> Miscellaneous_food_int, Missing: 0 (0.0%)
> Offals_food_int, Missing: 0 (0.0%)
> Oilcrops_food_int, Missing: 0 (0.0%)
> Pulses_food_int, Missing: 0 (0.0%)
> Spices_food_int, Missing: 0 (0.0%)
> Starchy Roots_food_int, Missing: 0 (0.0%)
> Stimulants_food_int, Missing: 0 (0.0%)
> Sugar & Sweeteners_food_int, Missing: 0 (0.0%)
> Sugar Crops_food_int, Missing: 0 (0.0%)
> Treenuts_food_int, Missing: 0 (0.0%)
> Vegetable Oils_food_int, Missing: 0 (0.0%)
> Vegetables_food_int, Missing: 0 (0.0%)
> Vegetal Products_food_int, Missing: 0 (0.0%)
> Alcoholic Beverages_protein_int, Missing: 0 (0.0%)
> Animal fats_protein_int, Missing: 0 (0.0%)
> Aquatic Products, Other_protein_int, Missing: 0 (0.0%)
> Cereals - Excluding Beer_protein_int, Missing: 0 (0.0%)
> Eggs_protein_int, Missing: 0 (0.0%)
> Fish, Seafood_protein_int, Missing: 0 (0.0%)
> Fruits - Excluding Wine_protein_int, Missing: 0 (0.0%)
> Meat_protein_int, Missing: 0 (0.0%)
> Milk - Excluding Butter_protein_int, Missing: 0 (0.0%)
> Offals_protein_int, Missing: 0 (0.0%)
> Oilcrops_protein_int, Missing: 0 (0.0%)
> Pulses_protein_int, Missing: 0 (0.0%)
> Spices_protein_int, Missing: 0 (0.0%)
> Starchy Roots_protein_int, Missing: 0 (0.0%)
> Stimulants_protein_int, Missing: 0 (0.0%)
> Sugar Crops_protein_int, Missing: 0 (0.0%)
> Sugar & Sweeteners_protein_int, Missing: 0 (0.0%)
```

```
> Treenuts_protein_int, Missing: 0 (0.0%)
> Vegetal Products_protein_int, Missing: 0 (0.0%)
> Vegetable Oils_protein_int, Missing: 0 (0.0%)
> Vegetables_protein_int, Missing: 0 (0.0%)
> Miscellaneous_protein_int, Missing: 0 (0.0%)

 ['Obesity_cal_int', 'Undernourished_cal_int', 'Confirmed_cal_int',
'Deaths_cal_int', 'Recovered_cal_int', 'Active_cal_int'] are all the columns
with over 0% missing data
```

4 Prepare the Data

[76]:
```python
#Check if any columns are categorical and need to be label encoded
labels = []

for x in merged.columns:
    print(type(merged[x][0]))
    if merged[x].dtype == 'object':
        labels.append(x)

print(str(len(labels)) + ' columns are categorical:')
print(labels)
```

```
<class 'str'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

```
<class 'str'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

```
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
2 columns are categorical:
['Country_cal_int', 'Undernourished_cal_int']
```

Replace '<2.5' with a float.

[77]:

```
# must replace <2.5 in undernourished
merged['Undernourished_cal_int'] = merged['Undernourished_cal_int'].replace('<2.
 ↪5', '2.5').astype(float)
merged_copy = merged.copy()
merged['Undernourished_cal_int'].unique()
```

[77]:

```
array([29.8,  6.2,  3.9, 25. ,  nan,  4.6,  4.3,  2.5, 14.7,  7.5, 10.1,
       17.1, 26.4,  3.6, 20. , 12.6, 16.4,  9.9, 59.6, 37.5,  2.7,  8.5,
        4.8, 40.3, 19. ,  5.6, 18.9,  9.5,  7.9,  4.5,  9. ,  2.9, 20.6,
        3.7,  4.2, 10.5, 10.2,  5.5, 15.2, 16.5, 28. ,  8.1, 49.3, 12.9,
       14.5,  8.3,  4.9, 29. ,  8. , 12.2, 29.4, 47.8,  2.8,  7.1, 11. ,
       13.1, 37.2, 44.4, 17.5, 10.3,  6.3, 10.4,  6.5, 13.4,  3.4, 27.9,
       10.6, 27.3,  8.7, 17. ,  3.2,  6.8, 20.3, 10. , 10.7,  9.7, 13.3,
       36.8,  5.7,  7. , 11.3, 25.6,  8.9, 20.1,  3.5,  7.8, 24.9, 16.1,
        5.4, 41. ,  2.6, 30.7,  7.2, 21.2,  9.3, 38.9, 46.7, 51.3])
```

[78]:

```
#Check if any columns are categorical and need to be label encoded
labels = []
```

```python
for x in merged.columns:
    print(type(merged[x][0]))
    if merged[x].dtype == 'object':
        labels.append(x)

print(str(len(labels)) + ' columns are categorical:')
print(labels)
```

```
<class 'str'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

```
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

```
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
1 columns are categorical:
['Country_cal_int']
```

5 KNN Imputer

[79]:
```
# Create an imputer object with a specific number of neighbors
imputer = KNNImputer(n_neighbors = 5)

# Fit and transform the data
# Remove Indentifier (Country)
pre_X = merged.drop(['Country_cal_int'], axis = 1)
```

[80]:
```
filled_data = imputer.fit_transform(pre_X)
filled_df = pd.DataFrame(filled_data, columns = pre_X.columns)

# Convert back to DataFrame
filled_df
```

[80]:

	Alcoholic Beverages_cal_int	Animal fats_cal_int	\
0	0.0000	0.8504	
1	0.9120	1.0591	
2	0.0896	0.1941	
3	1.9388	0.2644	
4	2.3041	1.5429	
..	…	…	
165	0.8454	0.6007	
166	0.7150	0.9363	
167	0.0000	0.3394	
168	1.1925	0.3230	
169	1.4269	0.6904	

	Aquatic Products, Other_cal_int	Cereals - Excluding Beer_cal_int	\
0	0.0	37.1186	
1	0.0	16.2107	
2	0.0	25.0112	
3	0.0	18.3521	
4	0.0	13.7215	
..	…	…	
165	0.0	21.3126	

```
166                          0.0                          26.9833
167                          0.0                          32.0727
168                          0.0                          31.5528
169                          0.0                          29.8044

     Eggs_cal_int  Fish, Seafood_cal_int  Fruits - Excluding Wine_cal_int  \
0          0.1501                 0.0000                           1.4757
1          0.8091                 0.1471                           3.8982
2          0.4181                 0.1195                           3.1805
3          0.0441                 0.8372                           2.3133
4          0.2057                 1.7280                           3.6824
..            …                      …                                …
165        0.2892                 0.4449                           2.3804
166        0.2894                 1.0385                           1.8046
167        0.1455                 0.1697                           1.1879
168        0.1988                 0.5714                           0.2236
169        0.1381                 0.1611                           0.4373

     Meat_cal_int  Milk - Excluding Butter_cal_int  Miscellaneous_cal_int  \
0          1.2006                           2.4512                 0.0250
1          3.8688                           9.9441                 0.0588
2          1.2543                           3.9869                 0.1045
3          2.9302                           0.5067                 0.0661
4          7.0356                           4.6904                 0.3086
..            …                              …                      …
165        3.1368                           2.6474                 0.0222
166        7.8311                           0.5618                 0.0340
167        2.0121                           0.6545                 0.0485
168        1.5155                           0.5217                 0.0497
169        1.7491                           1.0817                 0.0460

     …  Spices_protein_int  Starchy Roots_protein_int  \
0    …               0.1664                     0.1941
1    …               0.0000                     0.8867
2    …               0.1776                     1.4638
3    …               0.0000                     5.1941
4    …               0.3438                     0.4666
..   …                  …                        …
165  …               0.0000                     1.1195
166  …               0.5221                     0.2333
167  …               0.0657                     0.2066
168  …               0.0846                     1.2863
169  …               0.0488                     0.5467

     Stimulants_protein_int  Sugar Crops_protein_int  \
0                    0.5546                   0.0000
1                    0.2635                   0.0000
```

```
2                    0.4628                    0.0000
3                    0.1017                    0.0000
4                    0.4113                    0.0000
..                      …                       …
165                  0.3287                    0.0000
166                  0.5444                    0.0167
167                  0.3193                    0.0000
168                  0.0762                    0.0000
169                  0.2636                    0.0000

     Sugar & Sweeteners_protein_int  Treenuts_protein_int  \
0                            0.0000                0.1387
1                            0.0042                0.2677
2                            0.0000                0.2745
3                            0.0092                0.0092
4                            0.0000                0.0737
..                              …                     …
165                          0.0178                0.0000
166                          0.0056                0.3277
167                          0.0000                0.0188
168                          0.0000                0.0000
169                          0.0000                0.1367

     Vegetal Products_protein_int  Vegetable Oils_protein_int  \
0                         40.2477                      0.0000
1                         22.2552                      0.0084
2                         36.1694                      0.0269
3                         34.7782                      0.0092
4                         16.8161                      0.0430
..                            …                         …
165                       27.4545                      0.0533
166                       29.5617                      0.0000
167                       39.9831                      0.0000
168                       40.1117                      0.0000
169                       38.6508                      0.0293

     Vegetables_protein_int  Miscellaneous_protein_int
0                    1.1370                     0.0462
1                    3.2456                     0.0544
2                    3.1267                     0.1399
3                    0.8133                     0.0924
4                    1.6024                     0.2947
..                      …                        …
165                  1.0129                     0.0267
166                  3.7216                     0.0389
167                  0.5448                     0.0564
168                  0.8039                     0.0592
```

```
169                  0.5955                      0.0586

[170 rows x 95 columns]
```

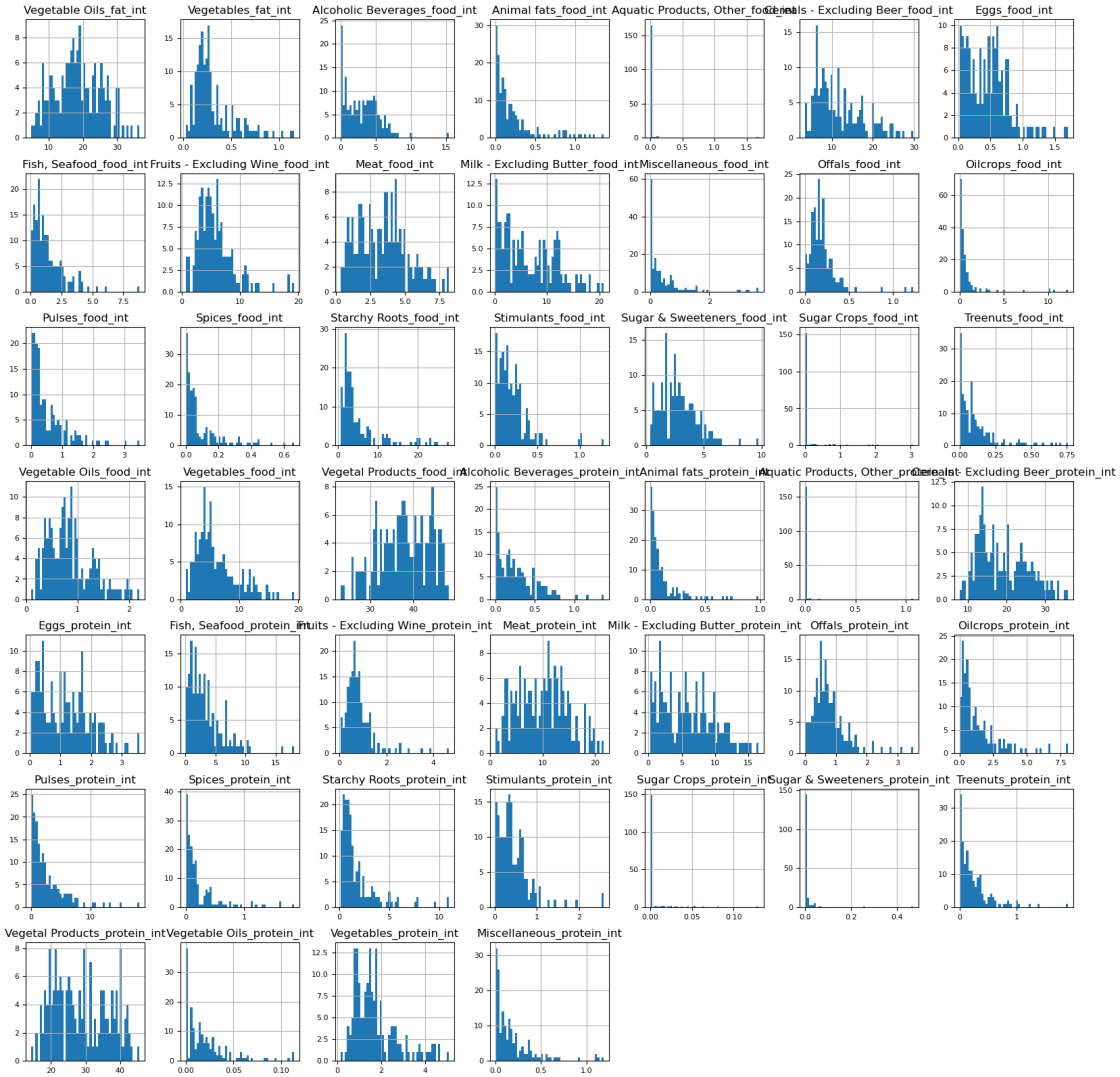
[81]:
```python
# summarize the number of rows with missing values for each column
missing_col = []
for x in filled_df.columns:
    n_miss = filled_df[x].isnull().sum()
    perc = n_miss / filled_df.shape[0] * 100
    print(f'> {x}, Missing: %d (%.1f%%)' % (n_miss, perc))
    if perc > 0:
        missing_col.append(x)
print(f"\n {missing_col} are all the columns with over 0% missing data")
```

```
> Alcoholic Beverages_cal_int, Missing: 0 (0.0%)
> Animal fats_cal_int, Missing: 0 (0.0%)
> Aquatic Products, Other_cal_int, Missing: 0 (0.0%)
> Cereals - Excluding Beer_cal_int, Missing: 0 (0.0%)
> Eggs_cal_int, Missing: 0 (0.0%)
> Fish, Seafood_cal_int, Missing: 0 (0.0%)
> Fruits - Excluding Wine_cal_int, Missing: 0 (0.0%)
> Meat_cal_int, Missing: 0 (0.0%)
> Milk - Excluding Butter_cal_int, Missing: 0 (0.0%)
> Miscellaneous_cal_int, Missing: 0 (0.0%)
> Offals_cal_int, Missing: 0 (0.0%)
> Oilcrops_cal_int, Missing: 0 (0.0%)
> Pulses_cal_int, Missing: 0 (0.0%)
> Spices_cal_int, Missing: 0 (0.0%)
> Starchy Roots_cal_int, Missing: 0 (0.0%)
> Stimulants_cal_int, Missing: 0 (0.0%)
> Sugar Crops_cal_int, Missing: 0 (0.0%)
> Sugar & Sweeteners_cal_int, Missing: 0 (0.0%)
> Treenuts_cal_int, Missing: 0 (0.0%)
> Vegetal Products_cal_int, Missing: 0 (0.0%)
> Vegetable Oils_cal_int, Missing: 0 (0.0%)
> Vegetables_cal_int, Missing: 0 (0.0%)
> Obesity_cal_int, Missing: 0 (0.0%)
> Undernourished_cal_int, Missing: 0 (0.0%)
> Confirmed_cal_int, Missing: 0 (0.0%)
> Deaths_cal_int, Missing: 0 (0.0%)
> Recovered_cal_int, Missing: 0 (0.0%)
> Active_cal_int, Missing: 0 (0.0%)
> Population_cal_int, Missing: 0 (0.0%)
> Alcoholic Beverages_fat_int, Missing: 0 (0.0%)
> Animal fats_fat_int, Missing: 0 (0.0%)
> Aquatic Products, Other_fat_int, Missing: 0 (0.0%)
> Cereals - Excluding Beer_fat_int, Missing: 0 (0.0%)
> Eggs_fat_int, Missing: 0 (0.0%)
```

> Fish, Seafood_fat_int, Missing: 0 (0.0%)
> Fruits - Excluding Wine_fat_int, Missing: 0 (0.0%)
> Meat_fat_int, Missing: 0 (0.0%)
> Miscellaneous_fat_int, Missing: 0 (0.0%)
> Milk - Excluding Butter_fat_int, Missing: 0 (0.0%)
> Offals_fat_int, Missing: 0 (0.0%)
> Oilcrops_fat_int, Missing: 0 (0.0%)
> Pulses_fat_int, Missing: 0 (0.0%)
> Spices_fat_int, Missing: 0 (0.0%)
> Starchy Roots_fat_int, Missing: 0 (0.0%)
> Stimulants_fat_int, Missing: 0 (0.0%)
> Sugar Crops_fat_int, Missing: 0 (0.0%)
> Sugar & Sweeteners_fat_int, Missing: 0 (0.0%)
> Treenuts_fat_int, Missing: 0 (0.0%)
> Vegetal Products_fat_int, Missing: 0 (0.0%)
> Vegetable Oils_fat_int, Missing: 0 (0.0%)
> Vegetables_fat_int, Missing: 0 (0.0%)
> Alcoholic Beverages_food_int, Missing: 0 (0.0%)
> Animal fats_food_int, Missing: 0 (0.0%)
> Aquatic Products, Other_food_int, Missing: 0 (0.0%)
> Cereals - Excluding Beer_food_int, Missing: 0 (0.0%)
> Eggs_food_int, Missing: 0 (0.0%)
> Fish, Seafood_food_int, Missing: 0 (0.0%)
> Fruits - Excluding Wine_food_int, Missing: 0 (0.0%)
> Meat_food_int, Missing: 0 (0.0%)
> Milk - Excluding Butter_food_int, Missing: 0 (0.0%)
> Miscellaneous_food_int, Missing: 0 (0.0%)
> Offals_food_int, Missing: 0 (0.0%)
> Oilcrops_food_int, Missing: 0 (0.0%)
> Pulses_food_int, Missing: 0 (0.0%)
> Spices_food_int, Missing: 0 (0.0%)
> Starchy Roots_food_int, Missing: 0 (0.0%)
> Stimulants_food_int, Missing: 0 (0.0%)
> Sugar & Sweeteners_food_int, Missing: 0 (0.0%)
> Sugar Crops_food_int, Missing: 0 (0.0%)
> Treenuts_food_int, Missing: 0 (0.0%)
> Vegetable Oils_food_int, Missing: 0 (0.0%)
> Vegetables_food_int, Missing: 0 (0.0%)
> Vegetal Products_food_int, Missing: 0 (0.0%)
> Alcoholic Beverages_protein_int, Missing: 0 (0.0%)
> Animal fats_protein_int, Missing: 0 (0.0%)
> Aquatic Products, Other_protein_int, Missing: 0 (0.0%)
> Cereals - Excluding Beer_protein_int, Missing: 0 (0.0%)
> Eggs_protein_int, Missing: 0 (0.0%)
> Fish, Seafood_protein_int, Missing: 0 (0.0%)
> Fruits - Excluding Wine_protein_int, Missing: 0 (0.0%)
> Meat_protein_int, Missing: 0 (0.0%)
> Milk - Excluding Butter_protein_int, Missing: 0 (0.0%)

```
> Offals_protein_int, Missing: 0 (0.0%)
> Oilcrops_protein_int, Missing: 0 (0.0%)
> Pulses_protein_int, Missing: 0 (0.0%)
> Spices_protein_int, Missing: 0 (0.0%)
> Starchy Roots_protein_int, Missing: 0 (0.0%)
> Stimulants_protein_int, Missing: 0 (0.0%)
> Sugar Crops_protein_int, Missing: 0 (0.0%)
> Sugar & Sweeteners_protein_int, Missing: 0 (0.0%)
> Treenuts_protein_int, Missing: 0 (0.0%)
> Vegetal Products_protein_int, Missing: 0 (0.0%)
> Vegetable Oils_protein_int, Missing: 0 (0.0%)
> Vegetables_protein_int, Missing: 0 (0.0%)
> Miscellaneous_protein_int, Missing: 0 (0.0%)

 [] are all the columns with over 0% missing data
```

[82]:
```
df_num = merged.iloc[:, 50:100].select_dtypes(include = ['float64', 'int64'])
df_num.hist(figsize=(20, 20), bins=50, xlabelsize=8, ylabelsize=8);
```

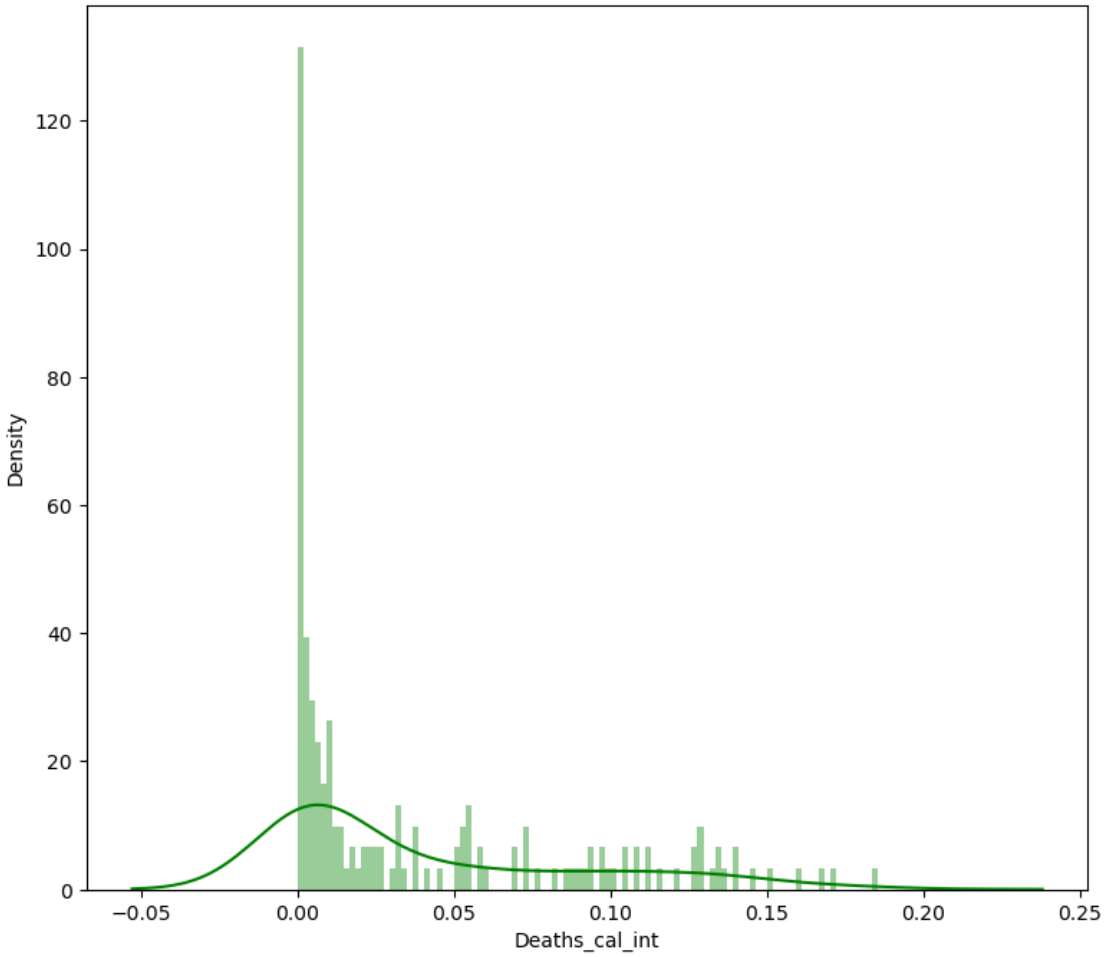


[83]:
```
print(merged['Deaths_cal_int'].describe())
plt.figure(figsize=(9, 8))
sns.distplot(merged['Deaths_cal_int'], color='g', bins=100, hist_kws={'alpha':␣
  ↪0.4})
plt.show()
```

```
count    164.000000
mean       0.039370
std        0.048718
min        0.000000
25%        0.002013
50%        0.011998
75%        0.069503
max        0.185428
```

```
Name: Deaths_cal_int, dtype: float64
```

```
/Users/duy-anh/opt/anaconda3/lib/python3.9/site-
packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



6 Normalize the data after filling in missing values

[84]:
```
X = filled_df.drop(['Confirmed_cal_int', 'Deaths_cal_int', 'Recovered_cal_int',␣
↪'Active_cal_int'], axis = 1)
Y = filled_df['Deaths_cal_int']
```

[85]:
```
scaler = StandardScaler()
normalized_data = scaler.fit_transform(X)
```

[86]:
```
# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(normalized_data, Y,␣
↪test_size = 0.2, random_state = 42)
```

7 OLS Regression

[87]:
```
ols = sm.OLS(y_train, X_train).fit()
```

[88]:
```
y_pred = ols.predict(X_test)
y_pred
```

[88]:
```
array([-0.13330016, -2.26105872,  0.08938396,  0.08116071,  0.1082339 ,
        0.01132269, -0.07725742,  0.10441537,  0.16467852,  0.01697756,
        0.09944677, -0.03067457,  0.06987141,  0.03625946,  0.21411675,
       -0.18149687, -0.189805  , -0.0028016 , -2.03255259,  0.05738814,
        0.10247797,  0.00340568,  0.4728673 ,  0.04176336, -0.04020332,
        0.03971434, -1.42589823, -0.01295872, -1.08784492,  0.17235277,
        0.10783295,  0.09392084, -0.0780703 ,  0.08063115])
```

[89]:
```
# Calculate R^2 (coefficient of determination)
r2 = r2_score(y_test, y_pred)
print("R^2 score:", r2)

R_squared = r2  # Example R^2 value
n = len(y_test)  # Number of data points
TSS = np.sum((y_test - np.mean(y_test))**2)  # Total sum of squares

# Calculate MSE from R^2
MSE = (1 - R_squared) * TSS / n
print("Mean Squared Error (MSE):", MSE)

# Approximate MAE
MAE = 0.8 * np.sqrt(MSE)
print("Approximate Mean Absolute Error (MAE):", MAE)

# Mean Error
mean_error = np.mean(np.array(y_test) - np.array(y_pred))
print("Mean Error:", mean_error)
```

```
R^2 score: -228.97710703091906
Mean Squared Error (MSE): 0.3786606900254559
Approximate Mean Absolute Error (MAE): 0.49228329406581717
Mean Error: 0.1921495500706735
```

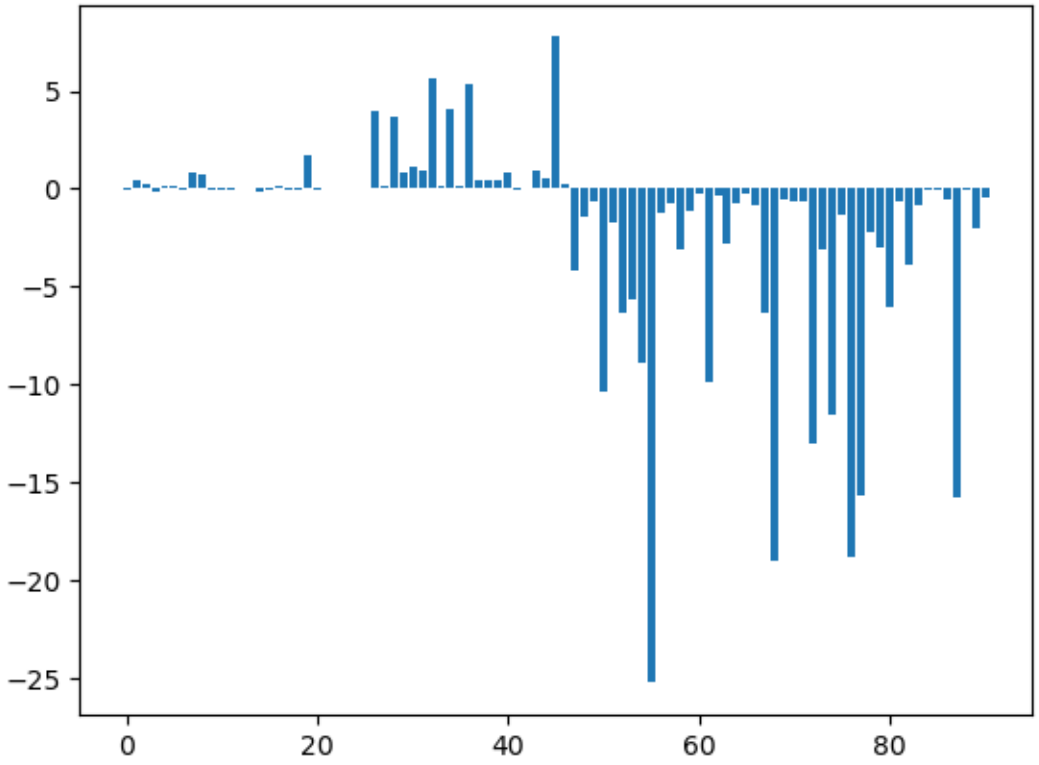
8 Multiple Linear Regression

[90]:
```
lin_model = linear_model.LinearRegression()

lin_model.fit(X_train, y_train)

importance = lin_model.coef_


for i,v in enumerate(importance):
    print(f'Feature: %0d {filled_df.columns[i]}, Score: %.5f' % (i,v))
# plot feature importance
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```

```
Feature: 0 Alcoholic Beverages_cal_int, Score: -0.03894
Feature: 1 Animal fats_cal_int, Score: 0.41804
Feature: 2 Aquatic Products, Other_cal_int, Score: 0.19711
Feature: 3 Cereals - Excluding Beer_cal_int, Score: -0.14964
Feature: 4 Eggs_cal_int, Score: 0.10450
Feature: 5 Fish, Seafood_cal_int, Score: 0.17500
Feature: 6 Fruits - Excluding Wine_cal_int, Score: -0.00279
Feature: 7 Meat_cal_int, Score: 0.84885
Feature: 8 Milk - Excluding Butter_cal_int, Score: 0.69532
Feature: 9 Miscellaneous_cal_int, Score: -0.07858
Feature: 10 Offals_cal_int, Score: -0.02336
Feature: 11 Oilcrops_cal_int, Score: -0.01845
Feature: 12 Pulses_cal_int, Score: 0.07300
Feature: 13 Spices_cal_int, Score: -0.00148
Feature: 14 Starchy Roots_cal_int, Score: -0.11039
Feature: 15 Stimulants_cal_int, Score: -0.04747
Feature: 16 Sugar Crops_cal_int, Score: 0.12209
Feature: 17 Sugar & Sweeteners_cal_int, Score: -0.03339
Feature: 18 Treenuts_cal_int, Score: -0.03473
Feature: 19 Vegetal Products_cal_int, Score: 1.75639
Feature: 20 Vegetable Oils_cal_int, Score: -0.02976
Feature: 21 Vegetables_cal_int, Score: 0.01123
Feature: 22 Obesity_cal_int, Score: 0.00292
Feature: 23 Undernourished_cal_int, Score: 0.00460
Feature: 24 Confirmed_cal_int, Score: 0.00856
Feature: 25 Deaths_cal_int, Score: 0.01161
Feature: 26 Recovered_cal_int, Score: 4.02107
Feature: 27 Active_cal_int, Score: 0.16145
Feature: 28 Population_cal_int, Score: 3.64793
Feature: 29 Alcoholic Beverages_fat_int, Score: 0.80536
Feature: 30 Animal fats_fat_int, Score: 1.11179
Feature: 31 Aquatic Products, Other_fat_int, Score: 0.95432
Feature: 32 Cereals - Excluding Beer_fat_int, Score: 5.62596
```

```
Feature: 33 Eggs_fat_int, Score: 0.10676
Feature: 34 Fish, Seafood_fat_int, Score: 4.02750
Feature: 35 Fruits - Excluding Wine_fat_int, Score: 0.13984
Feature: 36 Meat_fat_int, Score: 5.32023
Feature: 37 Miscellaneous_fat_int, Score: 0.45400
Feature: 38 Milk - Excluding Butter_fat_int, Score: 0.48757
Feature: 39 Offals_fat_int, Score: 0.42754
Feature: 40 Oilcrops_fat_int, Score: 0.85767
Feature: 41 Pulses_fat_int, Score: -0.00830
Feature: 42 Spices_fat_int, Score: 0.00397
Feature: 43 Starchy Roots_fat_int, Score: 0.92506
Feature: 44 Stimulants_fat_int, Score: 0.55678
Feature: 45 Sugar Crops_fat_int, Score: 7.75556
Feature: 46 Sugar & Sweeteners_fat_int, Score: 0.22910
Feature: 47 Treenuts_fat_int, Score: -4.21844
Feature: 48 Vegetal Products_fat_int, Score: -1.38671
Feature: 49 Vegetable Oils_fat_int, Score: -0.60746
Feature: 50 Vegetables_fat_int, Score: -10.37562
Feature: 51 Alcoholic Beverages_food_int, Score: -1.70258
Feature: 52 Animal fats_food_int, Score: -6.30815
Feature: 53 Aquatic Products, Other_food_int, Score: -5.60801
Feature: 54 Cereals - Excluding Beer_food_int, Score: -8.90798
Feature: 55 Eggs_food_int, Score: -25.23133
Feature: 56 Fish, Seafood_food_int, Score: -1.21720
Feature: 57 Fruits - Excluding Wine_food_int, Score: -0.77403
Feature: 58 Meat_food_int, Score: -3.14305
Feature: 59 Milk - Excluding Butter_food_int, Score: -1.09594
Feature: 60 Miscellaneous_food_int, Score: -0.26366
Feature: 61 Offals_food_int, Score: -9.87039
Feature: 62 Oilcrops_food_int, Score: -0.32676
Feature: 63 Pulses_food_int, Score: -2.75895
Feature: 64 Spices_food_int, Score: -0.77756
Feature: 65 Starchy Roots_food_int, Score: -0.23304
Feature: 66 Stimulants_food_int, Score: -0.81039
Feature: 67 Sugar & Sweeteners_food_int, Score: -6.35657
Feature: 68 Sugar Crops_food_int, Score: -19.00679
Feature: 69 Treenuts_food_int, Score: -0.49459
Feature: 70 Vegetable Oils_food_int, Score: -0.60000
Feature: 71 Vegetables_food_int, Score: -0.60472
Feature: 72 Vegetal Products_food_int, Score: -12.99565
Feature: 73 Alcoholic Beverages_protein_int, Score: -3.12269
Feature: 74 Animal fats_protein_int, Score: -11.50380
Feature: 75 Aquatic Products, Other_protein_int, Score: -1.32418
Feature: 76 Cereals - Excluding Beer_protein_int, Score: -18.75715
Feature: 77 Eggs_protein_int, Score: -15.70640
Feature: 78 Fish, Seafood_protein_int, Score: -2.16888
Feature: 79 Fruits - Excluding Wine_protein_int, Score: -3.01997
Feature: 80 Meat_protein_int, Score: -6.01851
```

```
Feature: 81 Milk - Excluding Butter_protein_int, Score: -0.59377
Feature: 82 Offals_protein_int, Score: -3.87423
Feature: 83 Oilcrops_protein_int, Score: -0.79697
Feature: 84 Pulses_protein_int, Score: -0.02230
Feature: 85 Spices_protein_int, Score: -0.05567
Feature: 86 Starchy Roots_protein_int, Score: -0.56588
Feature: 87 Stimulants_protein_int, Score: -15.73155
Feature: 88 Sugar Crops_protein_int, Score: -0.05425
Feature: 89 Sugar & Sweeteners_protein_int, Score: -2.02269
Feature: 90 Treenuts_protein_int, Score: -0.40201
```



[91]:
```
# Evaluate the model
print("Intercept ( 0):", lin_model.intercept_)
print("Coefficients ( 1,  2, ...):", lin_model.coef_)
```

```
Intercept ( 0): 0.05404622706589582
Coefficients ( 1,  2, …): [-3.89404224e-02  4.18035810e-01  1.97111629e-01
-1.49641247e-01
  1.04495759e-01  1.75003877e-01 -2.79243628e-03  8.48851887e-01
  6.95316217e-01 -7.85764390e-02 -2.33619086e-02 -1.84497338e-02
  7.30019282e-02 -1.48159173e-03 -1.10390062e-01 -4.74694580e-02
  1.22086465e-01 -3.33904871e-02 -3.47340593e-02  1.75638855e+00
 -2.97555711e-02  1.12292610e-02  2.92432967e-03  4.59543123e-03
```

```
 8.55955157e-03  1.16067193e-02  4.02106725e+00  1.61453645e-01
 3.64793131e+00  8.05363658e-01  1.11179155e+00  9.54320480e-01
 5.62595711e+00  1.06756939e-01  4.02749713e+00  1.39838707e-01
 5.32023121e+00  4.54002488e-01  4.87574420e-01  4.27535348e-01
 8.57673114e-01 -8.30336041e-03  3.96917757e-03  9.25063460e-01
 5.56783673e-01  7.75555988e+00  2.29104473e-01 -4.21843758e+00
-1.38670590e+00 -6.07459217e-01 -1.03756174e+01 -1.70257681e+00
-6.30814977e+00 -5.60800548e+00 -8.90797805e+00 -2.52313316e+01
-1.21719712e+00 -7.74032484e-01 -3.14304861e+00 -1.09594443e+00
-2.63661533e-01 -9.87039417e+00 -3.26758289e-01 -2.75895499e+00
-7.77561640e-01 -2.33043421e-01 -8.10386572e-01 -6.35657171e+00
-1.90067919e+01 -4.94586658e-01 -5.99997168e-01 -6.04724415e-01
-1.29956536e+01 -3.12269319e+00 -1.15037982e+01 -1.32417950e+00
-1.87571548e+01 -1.57064011e+01 -2.16888038e+00 -3.01996632e+00
-6.01850699e+00 -5.93773558e-01 -3.87422636e+00 -7.96972996e-01
-2.23012569e-02 -5.56691522e-02 -5.65878562e-01 -1.57315540e+01
-5.42452980e-02 -2.02268884e+00 -4.02009846e-01]
```

[92]:
```
y_pred = lin_model.predict(X_test)
y_pred
```

[92]:
```
array([-0.13330016,  0.40136833,  0.08938396,  0.08116071,  0.1082339 ,
        0.01132269, -0.07725742,  0.10441537,  0.16467852,  0.01697756,
        0.09944677, -0.03067457,  0.06987141,  0.03625946,  0.21411675,
       -0.18149687, -0.189805  , -0.0028016 ,  0.25134845,  0.05738814,
        0.10247797,  0.00340568,  0.4728673 ,  0.04176336, -0.04020332,
        0.03971434,  1.48887949, -0.01295872,  0.23890786,  0.17235277,
        0.10783295,  0.09392084, -0.0780703 ,  0.08063115])
```

[93]:
```
# Calculate R^2 (coefficient of determination)
r2 = r2_score(y_test, y_pred)
print("R^2 score:", r2)

R_squared = r2  # Example R^2 value
n = len(y_test)  # Number of data points
TSS = np.sum((y_test - np.mean(y_test))**2)  # Total sum of squares

# Calculate MSE from R^2
MSE = (1 - R_squared) * TSS / n
print("Mean Squared Error (MSE):", MSE)

# Approximate MAE
MAE = 0.8 * np.sqrt(MSE)
print("Approximate Mean Absolute Error (MAE):", MAE)

# Mean Error
mean_error = np.mean(np.array(y_test) - np.array(y_pred))
print("Mean Error:", mean_error)
```

```
R^2 score: -49.417673751494775
Mean Squared Error (MSE): 0.08301344154943488
Approximate Mean Absolute Error (MAE): 0.23049642641836843
Mean Error: -0.07808158525879574
```

9 KNN Regression

[94]:
```
# Define and fit the model to the training data
knn = KNeighborsRegressor(n_neighbors = 5)
knn.fit(X_train, y_train)
```

[94]:
```
KNeighborsRegressor()
```

[95]:
```
y_pred = knn.predict(X_test)
```

[96]:
```
r2 = r2_score(y_test, y_pred)
print("R^2 score:", r2 )

R_squared = r2  # Example R^2 value
n = len(y_test)  # Number of data points
TSS = np.sum((y_test - np.mean(y_test))**2)  # Total sum of squares

# Calculate MSE from R^2
MSE = (1 - R_squared) * TSS / n
print("Mean Squared Error (MSE):", MSE)

# Approximate MAE
MAE = 0.8 * np.sqrt(MSE)
print("Approximate Mean Absolute Error (MAE):", MAE)

# Mean Error
mean_error = np.mean(np.array(y_test) - np.array(y_pred))
print("Mean Error:", mean_error)
```

```
R^2 score: 0.4453643268994034
Mean Squared Error (MSE): 0.0009132157952607402
Approximate Mean Absolute Error (MAE): 0.02417556843110155
Mean Error: -0.011957631368430319
```

10 Random Forest with RFECV

[97]:
```
# Initialize RandomForestRegressor
model = RandomForestRegressor(random_state= 42)

# 10-fold cross validation
rfecv = RFECV(estimator=model, step=1, cv=10, scoring = 'r2')
rfecv.fit(X_train, y_train)
```

```
print(f"Optimal number of features: {rfecv.n_features_}")
print("Selected Features:", rfecv.support_)
```

```
Optimal number of features: 32
Selected Features: [ True  True False False  True False False False  True False
True  True
  True False False False False False False False  True False  True False
 False False  True False  True  True False False  True False  True False
 False False False  True False False False False  True False False  True
  True False False  True  True False False  True  True False  True False
 False False False  True False False False  True False  True False False
 False  True False False False  True  True False False False False False
 False False  True False  True False  True]
```

[98]:
```
# Apply RFE
rfe = RFE(estimator=model, n_features_to_select= 32)
X_rfe = rfe.fit(X_train, y_train)

# Transform both training and test sets
X_train_rfe = rfe.transform(X_train)
X_test_rfe = rfe.transform(X_test)

# Train the RandomForestRegressor on the selected features
model.fit(X_train_rfe, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test_rfe)

# Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Display feature selection results
print("Selected Features (True means selected):", rfe.support_)
print("Feature Rankings:", rfe.ranking_)
```

```
Mean Squared Error: 0.0010966107772282557
Selected Features (True means selected): [ True  True False False  True False
False False  True False  True  True
  True False False False False False False False  True False  True False
 False False  True False  True  True False False  True False  True False
 False False False  True False False False False  True False False  True
  True False False  True  True False False  True  True False  True False
 False False False  True False False False  True False  True False False
 False  True False False False  True  True False False False False False
 False False  True False  True False  True]
Feature Rankings: [ 1  1 59 43  1 49 40 50  1 21  1  1  1 36 41 18 53 10  9  3
```

```
1 25  1  2
 22 58  1 60  1  1  7 34  1 16  1 17 24 39 13  1  6 54 55 23  1 11 48  1
  1 38 31  1  1 47 44  1  1 33  1 19 28  4 12  1 56 35 30  1 14  1  5 52
 29  1 46  8 20  1  1 32 45 51 26 15 57 27  1 42  1 37  1]
```

[99]:
```
r2 = r2_score(y_test, y_pred)
print("R^2 score:", r2 )

R_squared = r2  # Example R^2 value
n = len(y_test)  # Number of data points
TSS = np.sum((y_test - np.mean(y_test))**2)  # Total sum of squares

# Calculate MSE from R^2
MSE = (1 - R_squared) * TSS / n
print("Mean Squared Error (MSE):", MSE)

# Approximate MAE
MAE = 0.8 * np.sqrt(MSE)
print("Approximate Mean Absolute Error (MAE):", MAE)

# Mean Error
mean_error = np.mean(np.array(y_test) - np.array(y_pred))
print("Mean Error:", mean_error)
```

```
R^2 score: 0.33398057752198207
Mean Squared Error (MSE): 0.0010966107772282557
Approximate Mean Absolute Error (MAE): 0.026492091224100896
Mean Error: -0.005174702250455016
```

11 Regresion without RFECV

[100]:
```
# Initialize RandomForestRegressor
model = RandomForestRegressor(random_state=42)

# Apply RFE
rfe = RFE(estimator=model, n_features_to_select= 82)
X_rfe = rfe.fit(X_train, y_train)

# Transform both training and test sets
X_train_rfe = rfe.transform(X_train)
X_test_rfe = rfe.transform(X_test)

# Train the RandomForestRegressor on the selected features
model.fit(X_train_rfe, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test_rfe)
```

```
# Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Display feature selection results
print("Selected Features (True means selected):", rfe.support_)
print("Feature Rankings:", rfe.ranking_)
```

```
Mean Squared Error: 0.0011273901058126575
Selected Features (True means selected): [ True  True False  True  True  True
True  True  True  True  True  True
  True  True  True  True False  True  True  True  True  True  True  True
  True False  True False  True  True  True  True  True  True  True  True
  True  True  True  True  True False False  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True False  True  True  True  True  True  True False
  True  True  True  True  True  True  True  True  True  True  True  True
 False  True  True  True  True  True  True]
Feature Rankings: [ 1  1  9  1  1  1  1  1  1  1  1  1  1  1  1  1  3  1  1  1
1  1  1  1
  1  8  1 10  1  1  1  1  1  1  1  1  1  1  1  1  1  4  5  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  6  1  1  1  1  1  1  2
  1  1  1  1  1  1  1  1  1  1  1  1  7  1  1  1  1  1  1]
```

[101]:
```
r2 = r2_score(y_test, y_pred)
print("R^2 score:", r2 )

R_squared = r2  # Example R^2 value
n = len(y_test)  # Number of data points
TSS = np.sum((y_test - np.mean(y_test))**2)  # Total sum of squares

# Calculate MSE from R^2
MSE = (1 - R_squared) * TSS / n
print("Mean Squared Error (MSE):", MSE)

# Approximate MAE
MAE = 0.8 * np.sqrt(MSE)
print("Approximate Mean Absolute Error (MAE):", MAE)

# Mean Error
mean_error = np.mean(np.array(y_test) - np.array(y_pred))
print("Mean Error:", mean_error)
```

```
R^2 score: 0.3152869525150691
Mean Squared Error (MSE): 0.0011273901058126575
Approximate Mean Absolute Error (MAE): 0.026861304281812173
Mean Error: -0.0071461337951813884
```

12 SVM Model for Regression

[102]:
```
# Create an SVR model
svr_model = SVR(kernel='rbf')

# Train the model
svr_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svr_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.005123653233923776
```

[103]:
```
r2 = r2_score(y_test, y_pred)
print("R^2 score:", r2 )

R_squared = r2  # Example R^2 value
n = len(y_test)  # Number of data points
TSS = np.sum((y_test - np.mean(y_test))**2)  # Total sum of squares

# Calculate MSE from R^2
MSE = (1 - R_squared) * TSS / n
print("Mean Squared Error (MSE):", MSE)

# Approximate MAE
MAE = 0.8 * np.sqrt(MSE)
print("Approximate Mean Absolute Error (MAE):", MAE)

# Mean Error
mean_error = np.mean(np.array(y_test) - np.array(y_pred))
print("Mean Error:", mean_error)
```

```
R^2 score: -2.111817464041986
Mean Squared Error (MSE): 0.005123653233923776
Approximate Mean Absolute Error (MAE): 0.05726375878084862
Mean Error: -0.058967266533761054
```

13 XGBoost

[104]:
```
# Create XGBoost model
model = xgb.XGBRegressor(
    max_depth=3,  # Adjust maximum depth to control model complexity
    learning_rate=0.1,  # Learning rate
    n_estimators=100,  # Number of trees
```

```
    objective='reg:squarederror'  # For regression problems
)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.0009378739155637516
```

[105]:
```
r2 = r2_score(y_test, y_pred)
print("R^2 score:", r2 )

R_squared = r2  # Example R^2 value
n = len(y_test)  # Number of data points
TSS = np.sum((y_test - np.mean(y_test))**2)  # Total sum of squares

# Calculate MSE from R^2
MSE = (1 - R_squared) * TSS / n
print("Mean Squared Error (MSE):", MSE)

# Approximate MAE
MAE = 0.8 * np.sqrt(MSE)
print("Approximate Mean Absolute Error (MAE):", MAE)

# Mean Error
mean_error = np.mean(np.array(y_test) - np.array(y_pred))
print("Mean Error:", mean_error)
```

```
R^2 score: 0.4303883779258626
Mean Squared Error (MSE): 0.0009378739155637516
Approximate Mean Absolute Error (MAE): 0.02449978175332999
Mean Error: -0.009874569941778901
```

[106]:
```
dtrain = xgb.DMatrix(X_train, label=y_train, missing=np.nan,␣
 ↪enable_categorical=True)
dtest = xgb.DMatrix(X_test, label=y_test, missing=np.nan,␣
 ↪enable_categorical=True)

# Set up XGBoost parameters (you can tune these parameters)
params = {
    'objective': 'reg:squarederror',  # For regression tasks
    'max_depth': 5,                   # Depth of trees
```

```
    'eta': 0.1,                          # Learning rate
    'subsample': 0.8,                    # Subsample ratio of the training␣
 ↪instances
    'colsample_bytree': 0.8,             # Subsample ratio of columns when␣
 ↪constructing each tree
}

# Train the XGBoost model
xgb_model = xgb.train(params, dtrain, num_boost_round = 100)

# Make predictions
y_pred = xgb_model.predict(dtest)

# Evaluate model performance
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 0.0012642982500524844
```

[107]:
```
r2 = r2_score(y_test, y_pred)
print("R^2 score:", r2 )

R_squared = r2  # Example R^2 value
n = len(y_test)  # Number of data points
TSS = np.sum((y_test - np.mean(y_test))**2)  # Total sum of squares

# Calculate MSE from R^2
MSE = (1 - R_squared) * TSS / n
print("Mean Squared Error (MSE):", MSE)

# Approximate MAE
MAE = 0.8 * np.sqrt(MSE)
print("Approximate Mean Absolute Error (MAE):", MAE)

# Mean Error
mean_error = np.mean(np.array(y_test) - np.array(y_pred))
print("Mean Error:", mean_error)
```

```
R^2 score: 0.2321366816508541
Mean Squared Error (MSE): 0.0012642982500524844
Approximate Mean Absolute Error (MAE): 0.02844557751274511
Mean Error: -0.00627241240313174
```