Christian Campbell

Female Malnutrition

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import shap
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [2]:
```
malnutrition_df = pd.read_csv(r"C:\Users\chris\Documents\Bellevue University\10 - Applied Data Science\Pr
```

In [3]:
```
malnutrition_df.head(10)
```

Out[3]:

	Unnamed: 0	URBAN_RURA	alt	chrps	country	deathcount	latnum	longnum	lst	marketm0	...	si
0	0	1	5.60252	0.437479	Bangladesh	0	22.981516	90.155785	-1.065822	NaN	...	-0.74544
1	1	1	4.67774	0.447809	Bangladesh	0	22.444431	90.329185	-0.963057	NaN	...	-0.84062
2	2	1	5.14527	0.453208	Bangladesh	0	22.487263	90.206123	-0.963257	NaN	...	-0.84166
3	3	1	6.17460	0.433324	Bangladesh	0	23.016359	90.192879	-1.074995	NaN	...	-0.74453
4	4	1	5.39076	0.406196	Bangladesh	0	22.952404	90.454414	-1.065344	NaN	...	-0.74849
5	5	1	6.28571	0.445932	Bangladesh	0	22.679934	90.265038	-1.000196	NaN	...	-0.83983
6	6	1	4.33951	0.418451	Bangladesh	0	22.738653	90.504379	-0.983299	NaN	...	-0.80912
7	7	1	7.18288	0.422089	Bangladesh	0	22.962793	90.321823	-1.068017	NaN	...	-0.75738
8	8	1	7.65300	0.441328	Bangladesh	0	22.834478	90.232010	-1.043728	NaN	...	-0.80404
9	9	1	5.73404	0.397856	Bangladesh	0	22.714678	90.675880	-0.974491	NaN	...	-0.78005

10 rows × 117 columns

Removing market columns

In [4]:

```
# Drops the first column and the market columns
malnutrition_df1 = malnutrition_df.drop(
    columns=[
        malnutrition_df.columns[0],  # The first column by position
        'marketm0',
        *[f'marketm{i}' for i in range(1, 48)],  # marketm1 to marketm46
        'markets0',
        *[f'markets{i}' for i in range(1, 48)]   # markets1 to markets47
    ]
)

# Displays the modified DataFrame
print(malnutrition_df1)
```

```
       URBAN_RURA        alt     chrps     country  deathcount     latnum  \
0               1    5.60252  0.437479  Bangladesh           0  22.981516   
1               1    4.67774  0.447809  Bangladesh           0  22.444431   
2               1    5.14527  0.453208  Bangladesh           0  22.487263   
3               1    6.17460  0.433324  Bangladesh           0  23.016359   
4               1    5.39076  0.406196  Bangladesh           0  22.952404   
...           ...        ...       ...         ...         ...        ...   
14333           0  212.91400  0.622106     Nigeria         382   7.368560   
14334           1  392.34100  1.081900     Nigeria         311   8.570109   
14335           1  394.41600  1.080867     Nigeria         311   8.765458   
14336           1  389.46400  1.084190     Nigeria         311   8.660406   
14337           0  435.48100  1.094852     Nigeria         311   8.687992   

         longnum       lst  numevents   pasture       sif     slope     tree  \
0      90.155785 -1.065822         14  0.048795 -0.745444  0.021781  16.3438   
1      90.329185 -0.963057         14  0.042821 -0.840625  0.007349   0.0000   
2      90.206123 -0.963257         14  0.030075 -0.841667  0.005347   0.0000   
3      90.192879 -1.074995         14  0.052236 -0.744531  0.027679  16.0333   
4      90.454414 -1.065344         14  0.041699 -0.748497  0.025029  18.7647   
...          ...       ...        ...       ...       ...       ...      ...   
14333   3.937503 -0.786959        345  0.004750 -0.413014  0.508662   0.0000   
14334   3.547449 -0.804163        296  0.001639 -0.269682  0.396409  16.6122   
14335   3.603125 -0.780643        296  0.059900 -0.248022  0.527184  15.6757   
14336   3.522780 -0.781408        296  0.004605 -0.257015  0.390382  21.0244   
14337   3.412814 -0.899868        296  0.114607 -0.223676  0.185043  32.9063   

       tt00_500k  year   stunted    wasted   healthy   poorest  \
0      365.67800  2004  0.294118  0.058824  0.941176  0.071429   
1      397.88600  2004  0.444444  0.000000  1.000000  0.062500   
2      344.36600  2004  0.600000  0.050000  0.950000  0.000000   
3      369.38500  2004  0.500000  0.062500  0.937500  0.062500   
4      273.92400  2004  0.555556  0.000000  1.000000  0.111111   
...          ...   ...       ...       ...       ...       ...   
14333    6.60494  2013  0.250000  0.250000  0.750000  0.000000   
14334  271.40400  2013  0.431373  0.176471  0.784314  0.969697   
14335  227.03800  2013  0.217391  0.000000  0.956522  0.000000   
14336  219.00000  2013  0.195122  0.073171  0.926829  0.379310   
14337  175.32100  2013  0.433333  0.066667  0.933333  0.000000   

       underweight_bmi  
0             0.272727  
1             0.400000  
2             0.354839  
```

```
3              0.323529
4              0.277778
...                 ...
14333          0.212121
14334          0.347222
14335          0.097561
14336          0.026316
14337          0.048780

[14338 rows x 20 columns]
```

Finding and mapping unique values in country column

In [5]:

```
# Displays the unique values in the 'country' column for malnutrition_df1
unique_countries = malnutrition_df1['country'].unique()

print(unique_countries)
```

```
['Bangladesh' 'Ethiopia' 'Ghana' 'Guatemala' 'Honduras' 'Mali' 'Nepal'
 'Kenya' 'Senegal' '8' 'Uganda' 'Nigeria']
```

In [6]:
```python
# Defines the mapping from country names to numbers
country_mapping = {
    'Bangladesh': 1,
    'Ethiopia': 2,
    'Ghana': 3,
    'Guatemala': 4,
    'Honduras': 5,
    'Mali': 6,
    'Nepal': 7,
    '8' : 8,
    'Kenya': 9,
    'Senegal': 10,
    'Uganda': 11,
    'Nigeria': 12
}

# Converts the 'country' column to numeric values based on the mapping
malnutrition_df2 = malnutrition_df1.copy()
malnutrition_df2['country'] = malnutrition_df2['country'].map(country_mapping)

malnutrition_df2.head(10)
```

Out[6]:

	URBAN_RURA	alt	chrps	country	deathcount	latnum	longnum	lst	numevents	pasture	sif	sl
0	1	5.60252	0.437479	1	0	22.981516	90.155785	-1.065822	14	0.048795	-0.745444	0.021
1	1	4.67774	0.447809	1	0	22.444431	90.329185	-0.963057	14	0.042821	-0.840625	0.007
2	1	5.14527	0.453208	1	0	22.487263	90.206123	-0.963257	14	0.030075	-0.841667	0.005
3	1	6.17460	0.433324	1	0	23.016359	90.192879	-1.074995	14	0.052236	-0.744531	0.027
4	1	5.39076	0.406196	1	0	22.952404	90.454414	-1.065344	14	0.041699	-0.748497	0.025
5	1	6.28571	0.445932	1	0	22.679934	90.265038	-1.000196	14	0.064859	-0.839834	0.012
6	1	4.33951	0.418451	1	0	22.738653	90.504379	-0.983299	14	0.039334	-0.809128	0.008
7	1	7.18288	0.422089	1	0	22.962793	90.321823	-1.068017	14	0.059080	-0.757382	0.027
8	1	7.65300	0.441328	1	0	22.834478	90.232010	-1.043728	14	0.061838	-0.804048	0.014
9	1	5.73404	0.397856	1	0	22.714678	90.675880	-0.974491	14	0.031193	-0.780057	0.011

In [7]:

```
# Displays the unique values in the 'country' column for malnutrition_df2
unique_countries1 = malnutrition_df2['country'].unique()

print(unique_countries1)
```

```
[ 1  2  3  4  5  6  7  9 10  8 11 12]
```

Finding and filling empty cells

In [8]:
```
# Calculates the number of empty cells in each column
empty_cells_count = malnutrition_df2.isnull().sum()

# Calculates the percentage of empty cells in each column
empty_cells_percentage = (empty_cells_count / len(malnutrition_df2)) * 100

# Creates a DataFrame to display both the count and percentage of empty cells
empty_cells_info = pd.DataFrame({
    'Empty Cells Count': empty_cells_count,
    'Empty Cells Percentage': empty_cells_percentage.round(1)  # Rounded to the nearest tenth
})

# Display the result
print(empty_cells_info)
```

```
                 Empty Cells Count  Empty Cells Percentage
URBAN_RURA                       0                     0.0
alt                              0                     0.0
chrps                          209                     1.5
country                          0                     0.0
deathcount                       0                     0.0
latnum                           0                     0.0
longnum                          0                     0.0
lst                            579                     4.0
numevents                        0                     0.0
pasture                          0                     0.0
sif                            229                     1.6
slope                            0                     0.0
tree                             0                     0.0
tt00_500k                        0                     0.0
year                             0                     0.0
stunted                          0                     0.0
wasted                           0                     0.0
healthy                          0                     0.0
poorest                          0                     0.0
underweight_bmi                200                     1.4
```

In [9]:

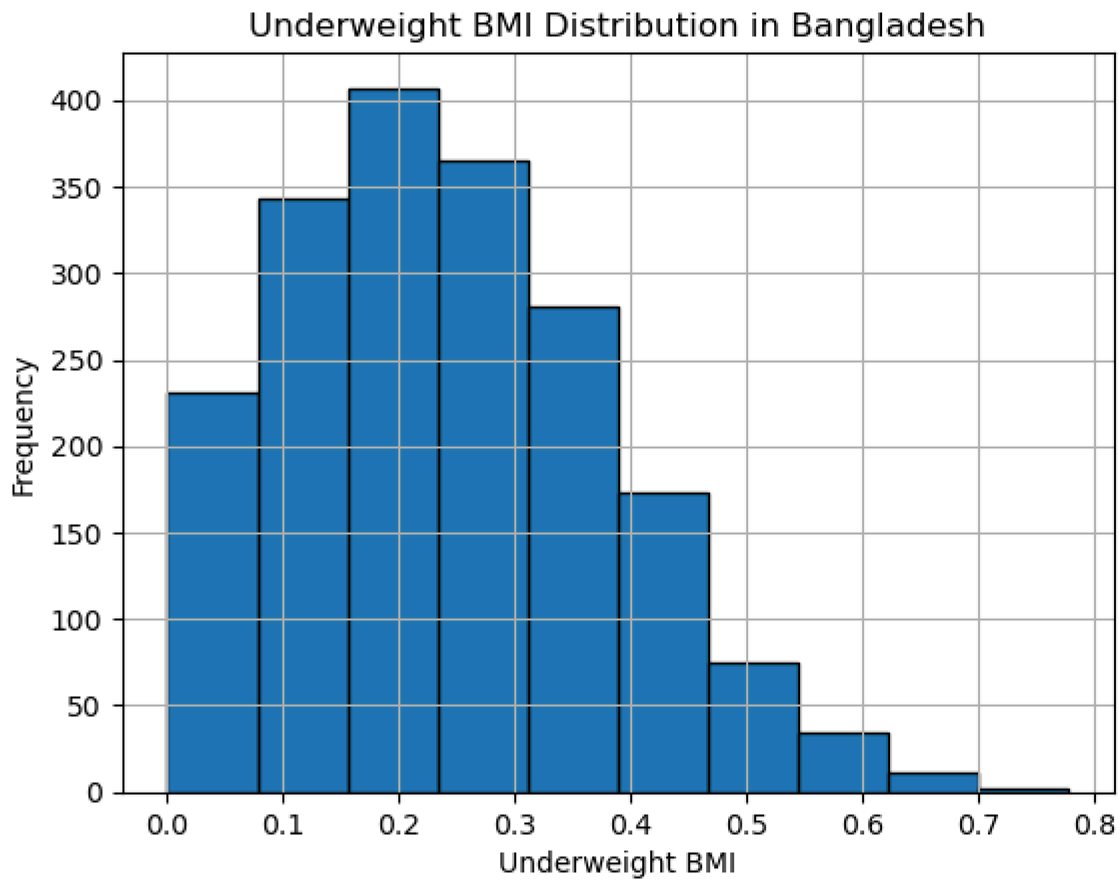
```
# List of columns to fill empty cells with their mean values
columns_to_fill = ['chrps', 'lst', 'sif', 'underweight_bmi']

# Creates a copy of malnutrition_df2 to preserve the original DataFrame
malnutrition_df3 = malnutrition_df2.copy()

# Fills empty cells in the specified columns with their respective means
for column in columns_to_fill:
    column_mean = malnutrition_df3[column].mean()  # Calculate the mean of the column
    malnutrition_df3[column].fillna(column_mean, inplace=True)  # Fill missing values with the mean
```

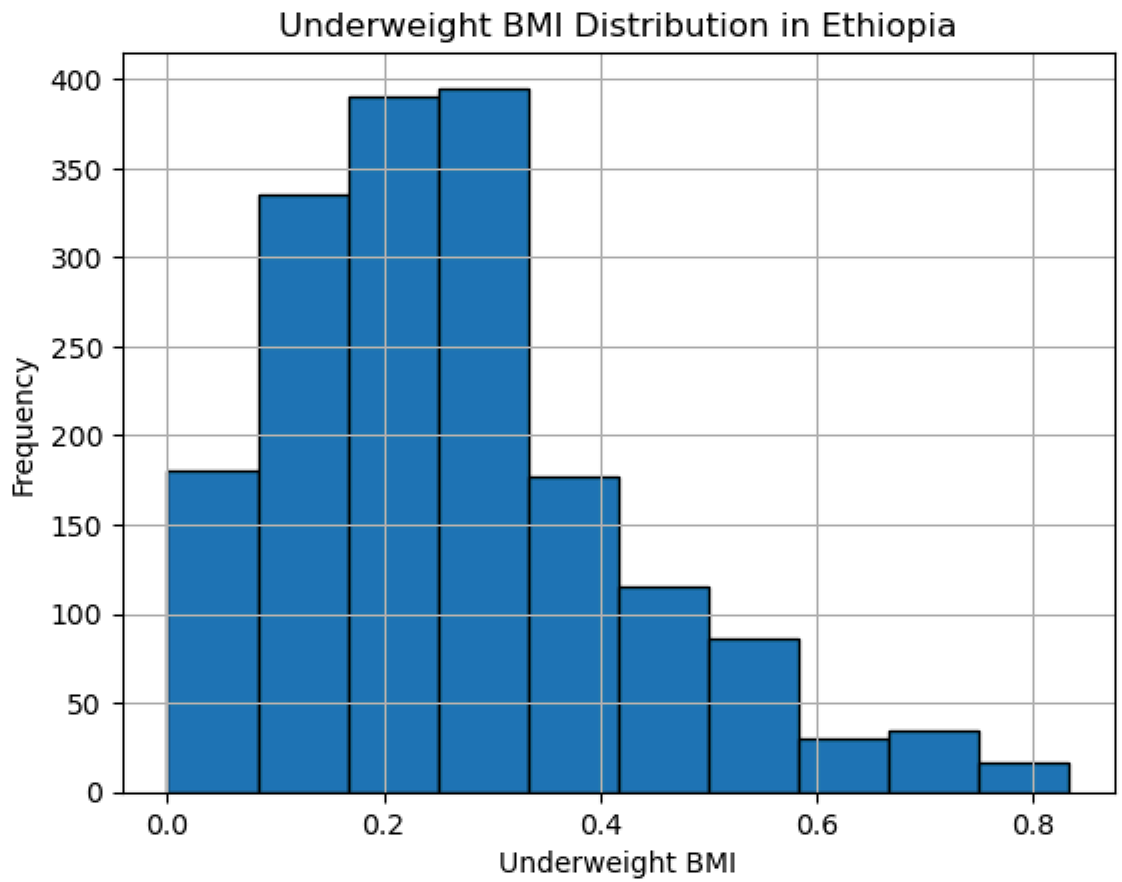
In [10]:
```python
# Calculates the number of missing values in each column
missing_values_count = malnutrition_df3.isnull().sum()

# Calculates the percentage of missing values in each column
missing_values_percentage = (missing_values_count / len(malnutrition_df3)) * 100

# Creates a DataFrame to display both the count and percentage of missing values
missing_data_summary = pd.DataFrame({
    'Missing Values Count': missing_values_count,
    'Missing Values Percentage': missing_values_percentage.round(1)  # Rounded to the nearest tenth
})

# Displays the summary
print(missing_data_summary)
```

```
                 Missing Values Count  Missing Values Percentage
URBAN_RURA                          0                        0.0
alt                                 0                        0.0
chrps                               0                        0.0
country                             0                        0.0
deathcount                          0                        0.0
latnum                              0                        0.0
longnum                             0                        0.0
lst                                 0                        0.0
numevents                           0                        0.0
pasture                             0                        0.0
sif                                 0                        0.0
slope                               0                        0.0
tree                                0                        0.0
tt00_500k                           0                        0.0
year                                0                        0.0
stunted                             0                        0.0
wasted                              0                        0.0
healthy                             0                        0.0
poorest                             0                        0.0
underweight_bmi                     0                        0.0
```

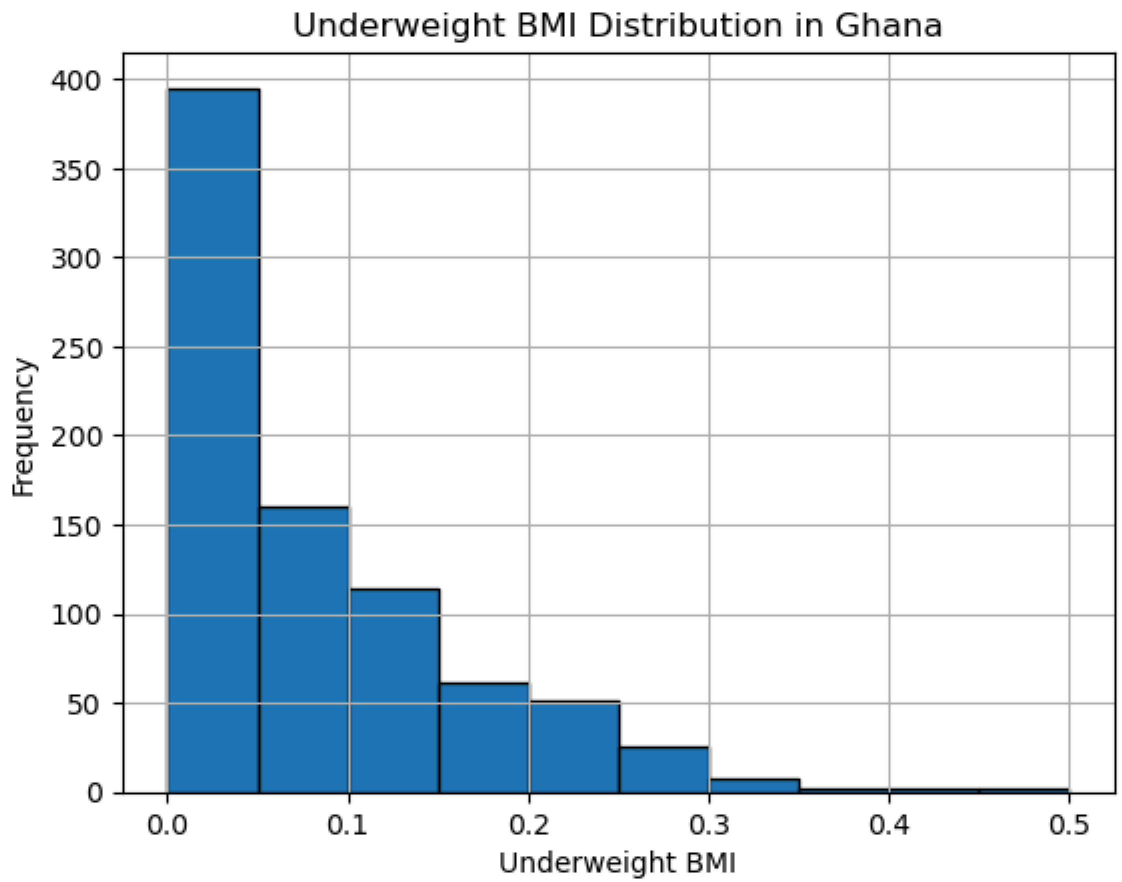
EDA

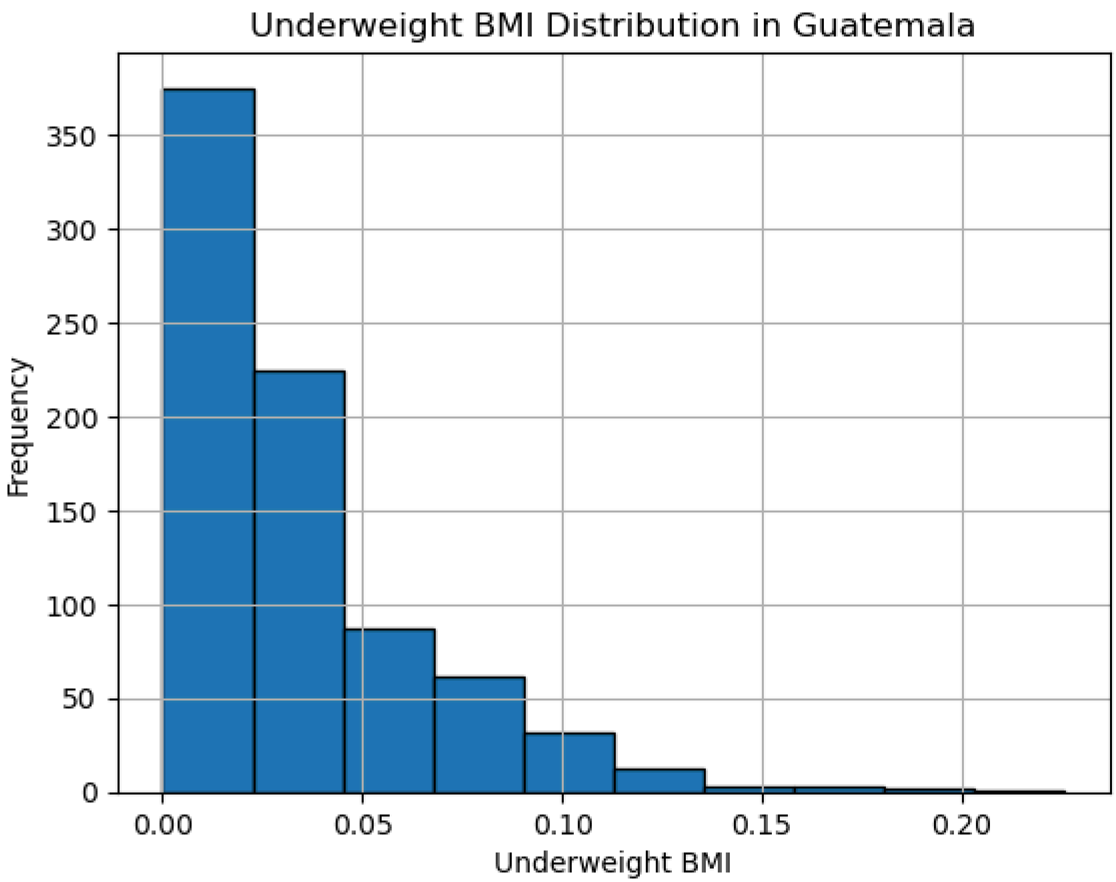
In [11]:
```
# Defines the mapping of country values to names
country_map = {
    1: 'Bangladesh',
    2: 'Ethiopia',
    3: 'Ghana',
    4: 'Guatemala',
    5: 'Honduras',
    6: 'Mali',
    7: 'Nepal',
    9: 'Kenya',
    10: 'Senegal',
    11: 'Uganda',
    12: 'Nigeria'
}

# Iterates over the country_map to create histograms
for country_code, country_name in country_map.items():
    # Filter the dataframe for the current country
    country_df = malnutrition_df3[malnutrition_df3['country'] == country_code]

    # Creates histogram for 'underweight_bmi'
    plt.figure()
    plt.hist(country_df['underweight_bmi'], bins=10, edgecolor='black')
    plt.title(f'Underweight BMI Distribution in {country_name}')
    plt.xlabel('Underweight BMI')
    plt.ylabel('Frequency')
    plt.grid(True)

    # Shows the plot
    plt.show()
```

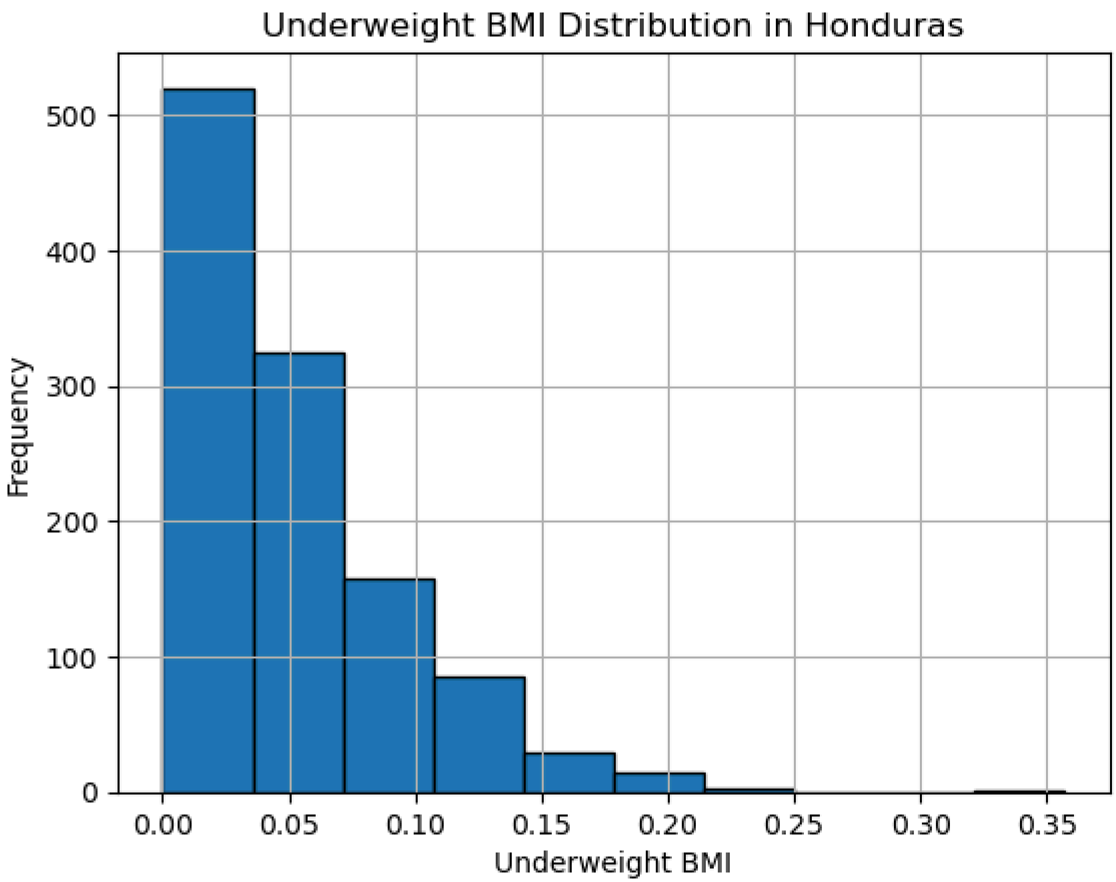

Underweight BMI Distribution in Bangladesh
400
350
300
250
200
150
100
50
0
Frequency
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Underweight BMI

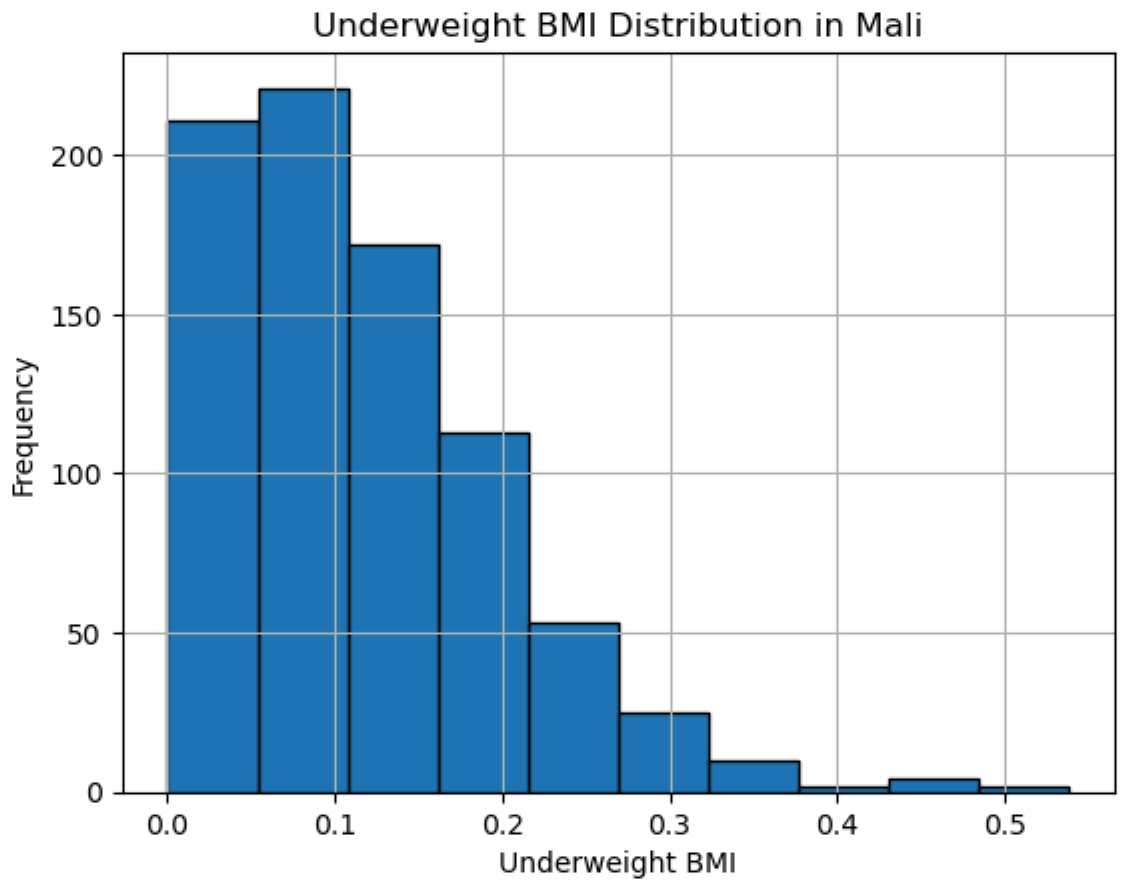


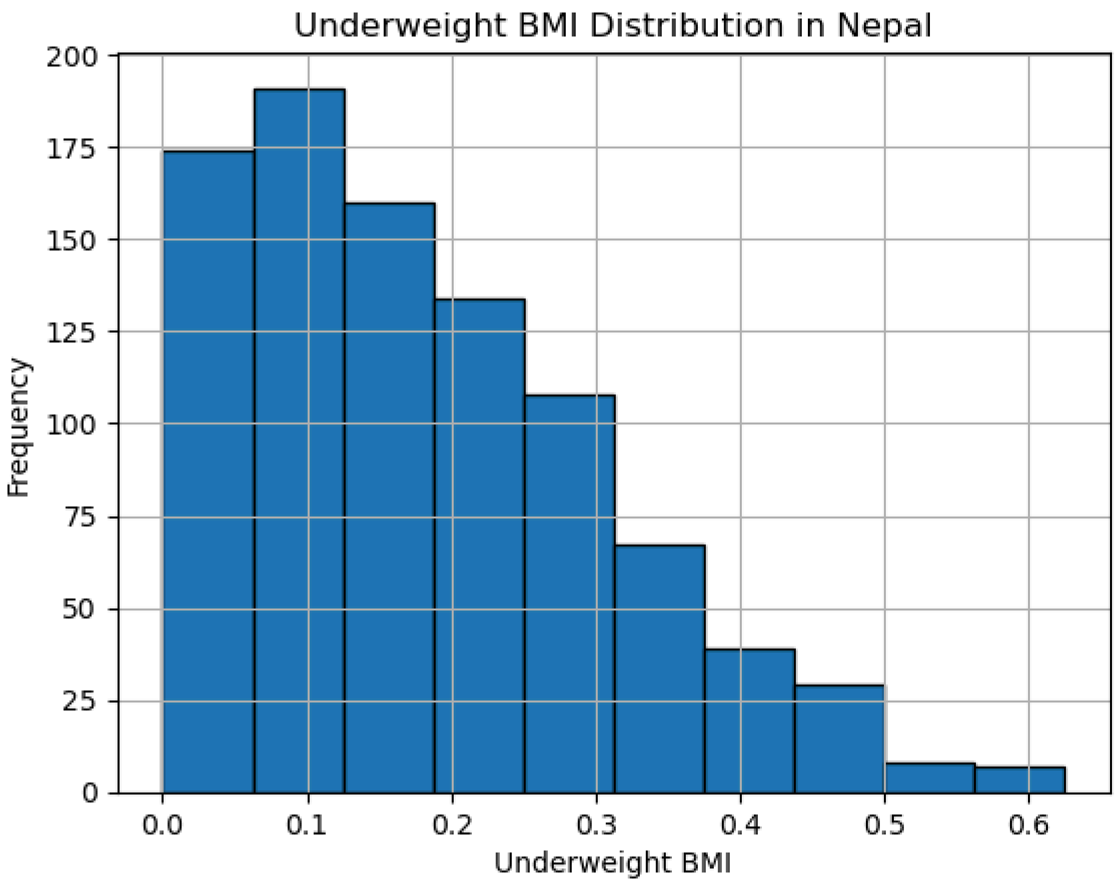
Underweight BMI Distribution in Ethiopia
Frequency
400
350
300
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
Underweight BMI

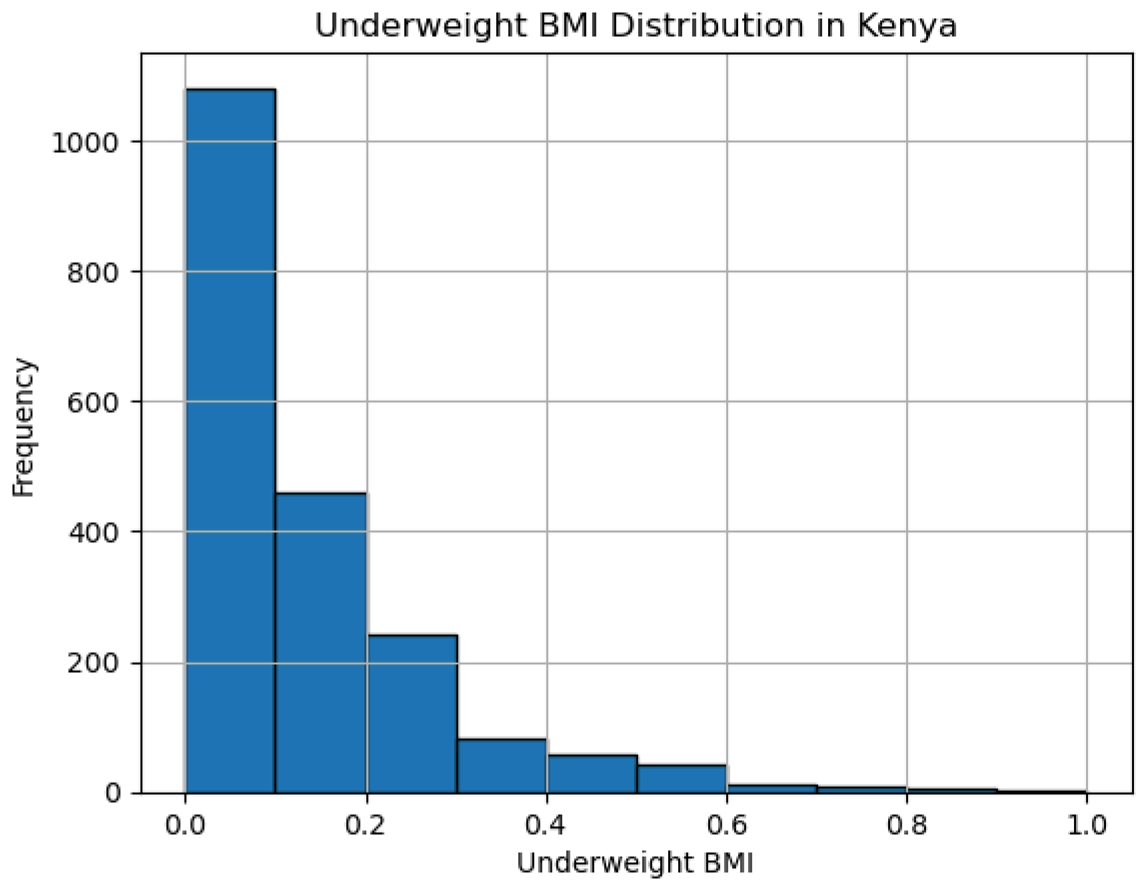

Underweight BMI Distribution in Ghana
Frequency
0
50
100
150
200
250
300
350
400
0.0
0.1
0.2
0.3
0.4
0.5
Underweight BMI

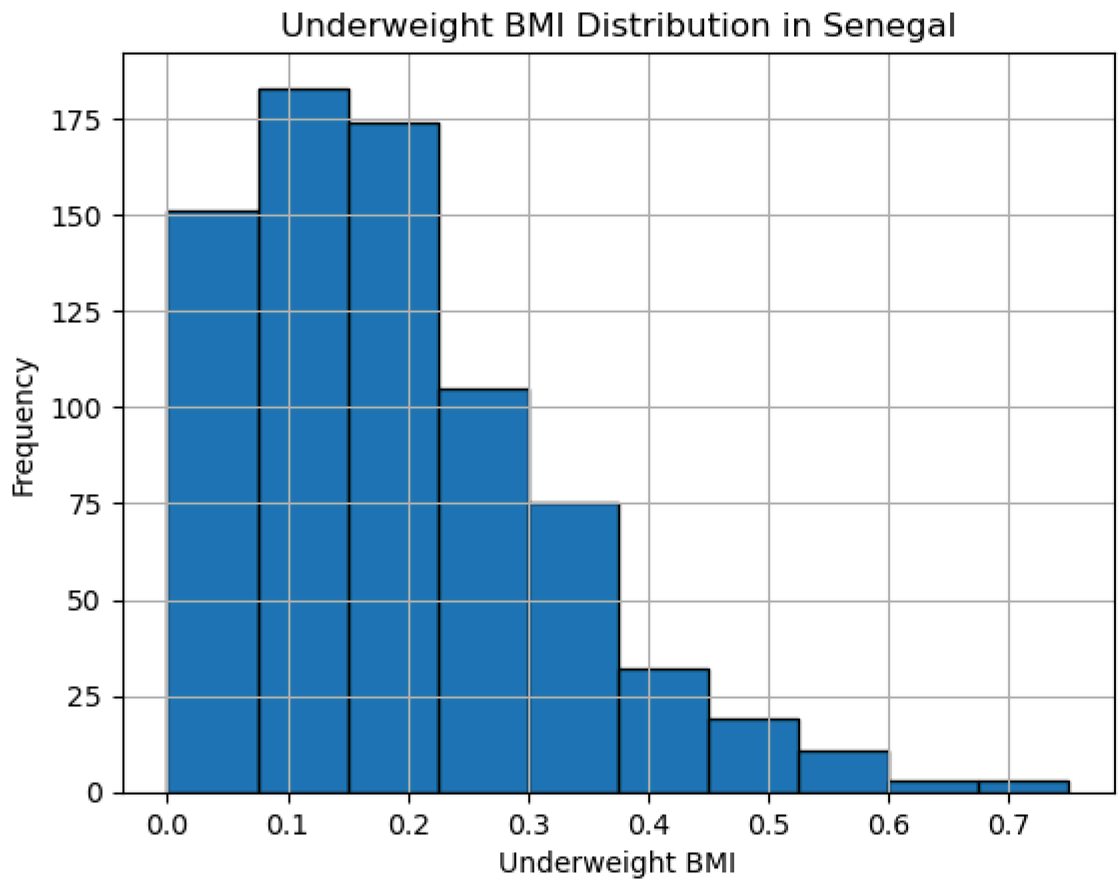

Underweight BMI Distribution in Guatemala
Frequency
350
300
250
200
150
100
50
0
0.00
0.05
0.10
0.15
0.20
Underweight BMI

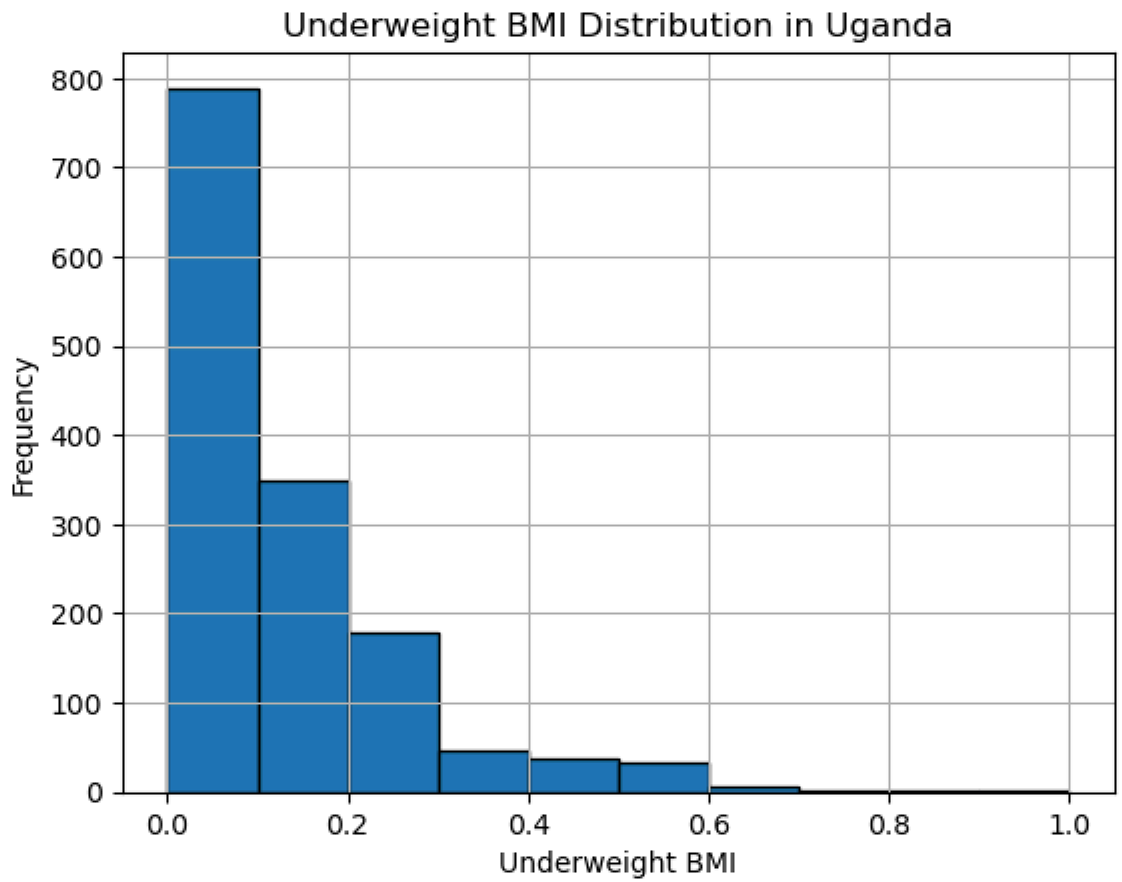

Underweight BMI Distribution in Honduras
Frequency
0
100
200
300
400
500
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Underweight BMI

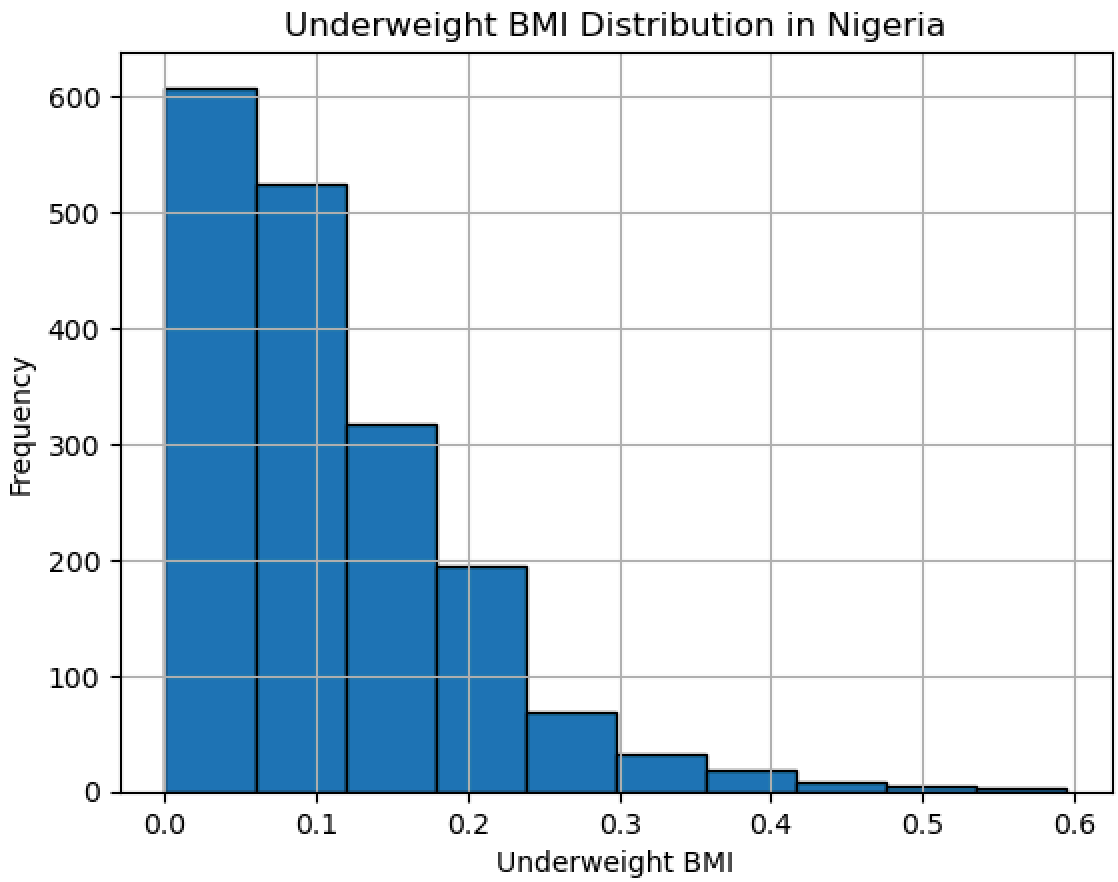

Underweight BMI Distribution in Mali
Frequency
200
150
100
50
0
0.0
0.1
0.2
0.3
0.4
0.5
Underweight BMI


Underweight BMI Distribution in Nepal
Frequency
200
175
150
125
100
75
50
25
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Underweight BMI


Underweight BMI Distribution in Kenya
Frequency
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0
Underweight BMI


Underweight BMI Distribution in Senegal
Frequency
0
25
50
75
100
125
150
175
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Underweight BMI


Underweight BMI Distribution in Uganda
Frequency
800
700
600
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0
Underweight BMI



Machine learning model

Random Forest Regressor

In [12]:

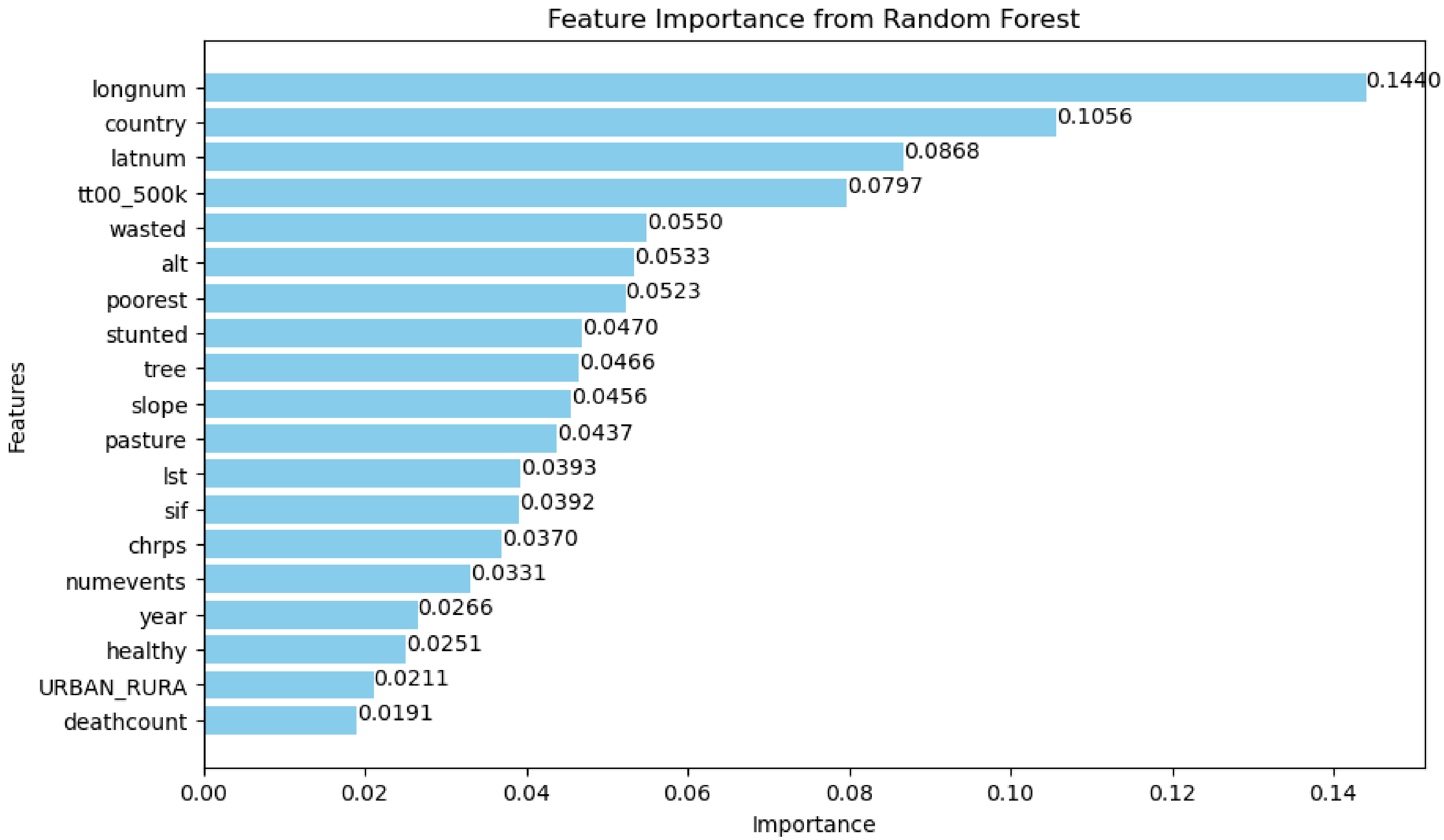
```
# Defines the target and features
X = malnutrition_df3.drop(columns=['underweight_bmi'])
y = malnutrition_df3['underweight_bmi']

# Splits the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initializes the Random Forest Regressor
rf_model = RandomForestRegressor(random_state=42)

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Performs grid search
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=3, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)

# Retrieves the best parameters
best_params = grid_search.best_params_
print("Best Parameters: ", best_params)

# Trains the model with the best parameters
best_rf_model = grid_search.best_estimator_
best_rf_model.fit(X_train, y_train)

# Makes predictions
y_pred_train = best_rf_model.predict(X_train)
y_pred_test = best_rf_model.predict(X_test)

# Evaluates the model
train_mae = mean_absolute_error(y_train, y_pred_train)
test_mae = mean_absolute_error(y_test, y_pred_test)
train_mse = mean_squared_error(y_train, y_pred_train)
test_mse = mean_squared_error(y_test, y_pred_test)
train_rmse = np.sqrt(train_mse)
test_rmse = np.sqrt(test_mse)
train_r2 = r2_score(y_train, y_pred_train)
```

```python
test_r2 = r2_score(y_test, y_pred_test)

# Prints the results
print(f'Training MAE: {train_mae:.4f}')
print(f'Testing MAE: {test_mae:.4f}')
print(f'Training MSE: {train_mse:.4f}')
print(f'Testing MSE: {test_mse:.4f}')
print(f'Training RMSE: {train_rmse:.4f}')
print(f'Testing RMSE: {test_rmse:.4f}')
print(f'Training R^2: {train_r2:.4f}')
print(f'Testing R^2: {test_r2:.4f}')
```

```
Fitting 3 folds for each of 324 candidates, totalling 972 fits
Best Parameters:  {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_spli
t': 2, 'n_estimators': 200}
Training MAE: 0.0361
Testing MAE: 0.0732
Training MSE: 0.0025
Testing MSE: 0.0099
Training RMSE: 0.0505
Testing RMSE: 0.0994
Training R^2: 0.8742
Testing R^2: 0.5295
```

In [13]:

```
# Calculates feature importances from the best Random Forest model
feature_importances = best_rf_model.feature_importances_

# Creates a DataFrame for feature importances
importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Bar chart for feature importance
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
for index, value in enumerate(importance_df['Importance']):
    plt.text(value, index, f'{value:.4f}')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Feature Importance from Random Forest')
plt.gca().invert_yaxis()
plt.show()
```

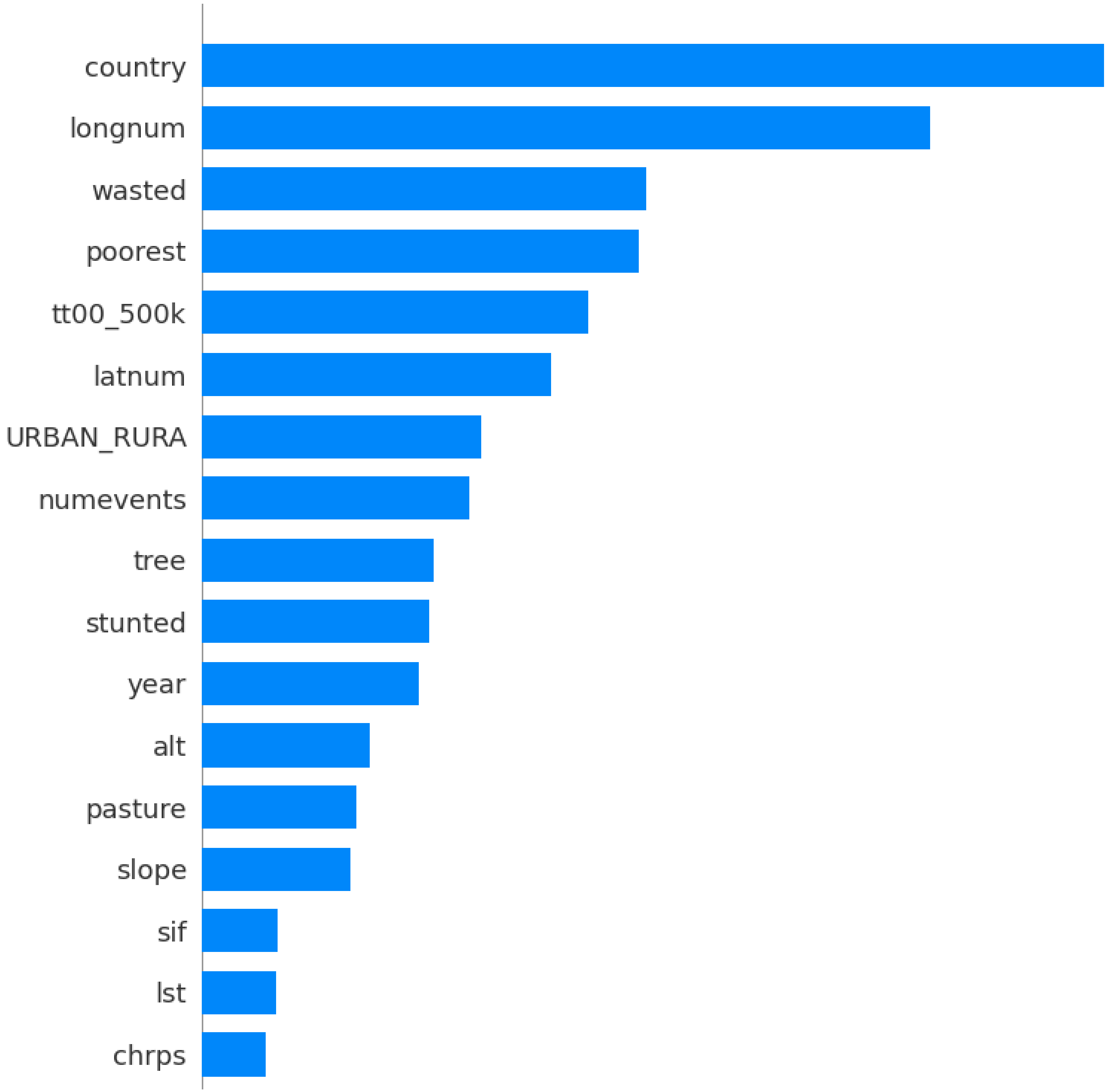

Feature Importance from Random Forest
longnum
country
latnum
tt00_500k
wasted
alt
poorest
stunted
tree
slope
pasture
lst
sif
chrps
numevents
year
healthy
URBAN_RURA
deathcount
0.1440
0.1056
0.0868
0.0797
0.0550
0.0533
0.0523
0.0470
0.0466
0.0456
0.0437
0.0393
0.0392
0.0370
0.0331
0.0266
0.0251
0.0211
0.0191
Features
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
Importance

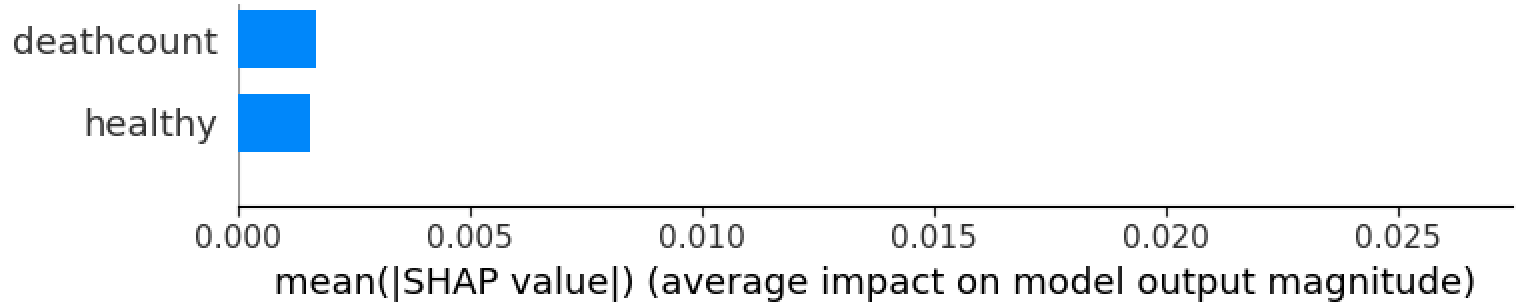
In [14]:
```
# Display the ranked table of feature importances
print(importance_df)
```

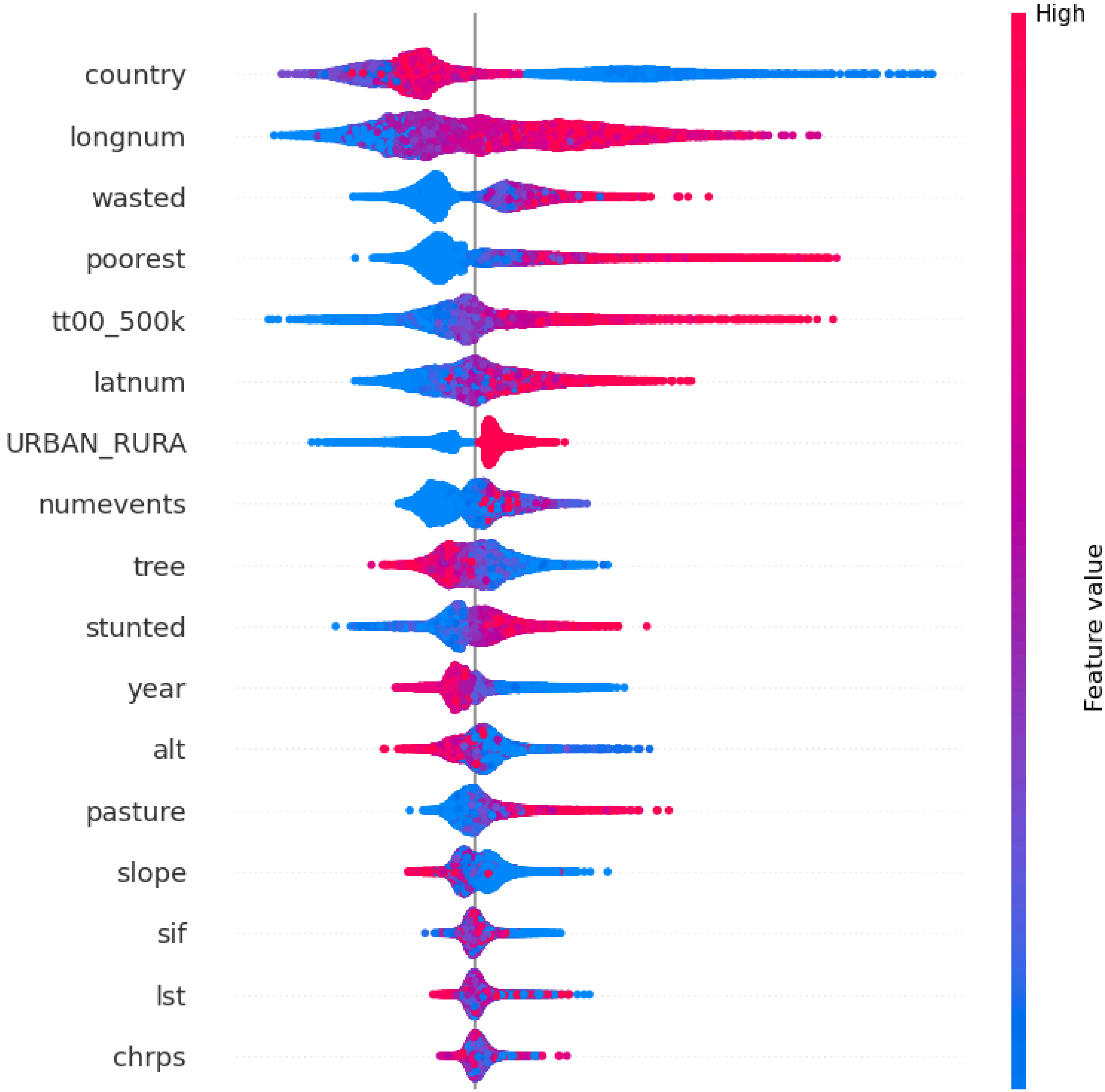
```
       Feature  Importance
6      longnum    0.144015
3      country    0.105647
5       latnum    0.086774
13   tt00_500k    0.079693
16      wasted    0.054978
1          alt    0.053303
18     poorest    0.052293
15     stunted    0.047015
12        tree    0.046564
11       slope    0.045644
9      pasture    0.043744
7          lst    0.039281
10         sif    0.039152
2        chrps    0.037005
8    numevents    0.033080
14        year    0.026553
17     healthy    0.025103
0   URBAN_RURA    0.021082
4   deathcount    0.019076
```

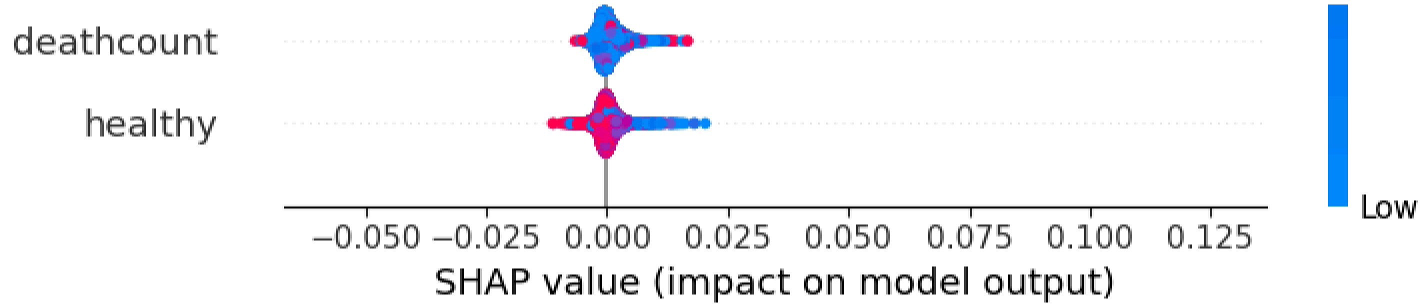
In [15]:

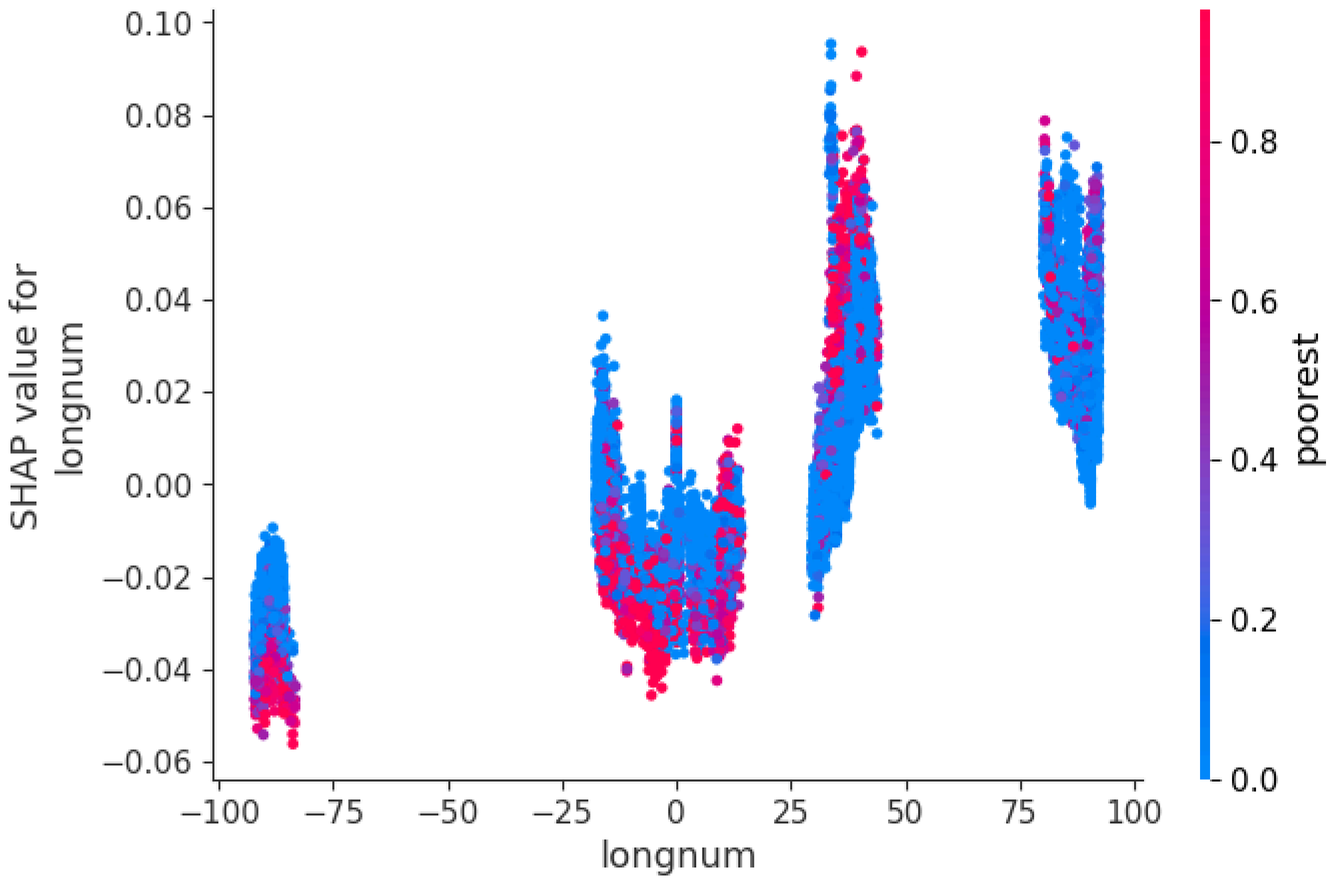
```
# Explains model predictions using SHAP values
explainer = shap.TreeExplainer(best_rf_model)
shap_values = explainer.shap_values(X)

# SHAP summary plot
shap.summary_plot(shap_values, X, plot_type="bar")

# SHAP summary plot with detailed feature impacts
shap.summary_plot(shap_values, X)

# SHAP dependence plot for a specific feature
shap.dependence_plot("longnum", shap_values, X)
```


country
longnum
wasted
poorest
tt00_500k
latnum
URBAN_RURA
numevents
tree
stunted
year
alt
pasture
slope
sif
lst
chrps


deathcount
healthy
0.000
0.005
0.010
0.015
0.020
0.025
mean(|SHAP value|) (average impact on model output magnitude)


country
longnum
wasted
poorest
tt00_500k
latnum
URBAN_RURA
numevents
tree
stunted
year
alt
pasture
slope
sif
lst
chrps
High
Feature value


deathcount
healthy
−0.050 −0.025 0.000 0.025 0.050 0.075 0.100 0.125
SHAP value (impact on model output)
Low


0.10
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
SHAP value for longnum
−100 −75 −50 −25 0 25 50 75 100
longnum
0.8
0.6
0.4
0.2
0.0
poorest

In [ ]: