Data Preprocessing Pipeline for Food and Nutrition Analysis

1. Dataset Description

- **Data Overview**: The dataset contains **1698 entries and 19 columns**, focusing on food, nutrition, and health-related metrics.
- **Challenges**:
  - **Missing Values**: Some columns have empty values.
  - **Inconsistencies**: Text-based columns have irregular formatting.
  - **Outliers**: Numerical columns contain values that deviate significantly.
  - **Redundancy**: Possible duplicate rows.
  - **Non-uniform Categories**: Categorical columns have inconsistent groupings.

2. Data Preprocessing Steps

Phase 1: Load Dataset

- Load the dataset from a file and display its contents for inspection.

Phase 2: Encoding Data

- label encoding was applied to convert categorical columns into numerical values. This method assigns a unique numeric label to each category, making the data machine-readable for further analysis.

Phase 3: Handling Missing and Noisy Data

- Use **Iterative Imputer** to fill missing values by modeling relationships between columns like protein, sugar, and calories.
  - **Advantages**:
    - Captures correlations between features.
    - Provides more accurate predictions than simpler methods like mean or KNN.

Phase 4: Outliers Detection and Removal

- **Z-Score Method**: For normally distributed or mildly skewed data, values beyond 3 standard deviations are considered outliers.
- **IQR Method**: For highly skewed data, values outside the range of Q1 - 1.5×IQR to Q3 + 1.5×IQR are identified as outliers.

Phase 5: Feature Scaling

- **StandardScaler** is applied to ensure all numeric features (like calories, protein, fat) have equal weight. This is essential for balanced and unbiased analysis.

Phase 6: PCA (Principal Component Analysis)

- Reduce the dataset's dimensions while retaining **95% of the variance**.
    - **Benefits**:
        - Speeds up computation.
        - Simplifies analysis.
        - Reduces noise and redundancy in high-dimensional data.

3. Model Evaluation

- **Tree-based Models**:
    - Models like **RandomForest** and **GradientBoosting** performed the best with basic preprocessing.
    - **Best R^2 Score**: 0.7934 (RandomForest).
- **Linear Models**:
    - Models like **Linear Regression** required full preprocessing to achieve better accuracy.
    - **Best R^2 Score**: 0.7299.
- **Conclusion**:
    - Tree-based models are more effective for this dataset, requiring less preprocessing.

4. GUI Implementation

A user-friendly **Graphical User Interface (GUI)** is built using Python's **Tkinter library**.

- **Key Features**:
    - **Load Dataset**: Import and view the dataset.
    - **Handle Missing Values**: Automatically clean missing data.
    - **Remove Outliers**: Detect and remove unusual values.
    - **Normalize Data**: Scale numeric features to uniform ranges.
    - **Encode Categorical Data**: Convert text into numeric formats.
    - **Automate Preprocessing**: Perform all steps with one button click.
    - **Visualizations**: Display data distributions and box plots after processing.

5. Key Insights

- A systematic approach to cleaning and preparing data ensures better machine learning results.
- Using a GUI makes the process accessible for non-programmers and simplifies preprocessing workflows.
- The processed dataset is ready for accurate and efficient machine learning model implementation.

Food and Nutrition

Original DataSet

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
Ages	Gender	Height	Weight	Activity_Level	Dietary_Preference	Protein	Sugar	Sodium	Calories	Carbohydrates	Fiber	Fat	Breakfast_Sugge	Lunch_Sugg	Snack_Sugg	Dinner_Su	Disease	Daily_Calorie_Target
25	Male	180	80	Moderately	Omnivore	120	125	24	2020	250	30	60	Oatmeal	Grilled	Greek	Salmon	Weight	2000
32	Female	165	65	Lightly	Vegetarian	80	100	16	1480	200	24	40	Tofu	Lentil	Apple	Vegetable	Weight	1600
48	Male	175	95	Sedentary	Vegan	100	150	20	2185	300	36	65	Veggie	Black	Trail	Lentil	Weight	2200
55	Female	160	70	Intense	Omnivore	140	175	28	2680	350	42	80	Greek	Chicken	Banana	Turkey	Weight	2500
62	Male	170	85	Sedentary	Vegetarian	80	125	16	1815	250	30	55	Scrambled	Quinoa	Fruit	Vegetarian	Weight	2000
68	Female	155	60	Lightly	Vegan	70	110		1565	220	26.4	45	Tofu	Lentil	Hummus	Vegan	Weight	1800
28	Male	190	100	Moderately	Omnivore	180	175	36	2840	350	42	80	Pancakes	Chicken	Protein	Steak	Weight	3000
35	Female	170	75	Intense	Vegetarian	120	150	24	2220	300	36	60	Tofu	Black	Cottage	Quinoa	Weight	2400
42	Male	185	110	Moderately	Vegan	160	200	32	3005	400	48	85	Tofu	Lentil	Energy	Vegan	Weight	3200
58	Female	168	78	Sedentary	Vegan	75	115	15	1670	230	27.6	50	Overnight	Lentil	Fruit	Vegan	Weight	1900
25	Male	180	80	Lightly	Omnivore	120	125	24	2020	250	30	60	Oatmeal	Grilled	Greek	Salmon	Weight	2000
32	Female	165	65	Moderately	Vegan	100	150	20	2230	300	36	70	Tofu	Lentil	Trail	Chickpea	Weight	2200
48	Male	175	95	Sedentary	Vegetarian	100	175	20	2520	350	42	80	Wholegrain	Black	Fruit	Vegetable	Weight	2400
55	Female	160	70	Moderately	Omnivore	100	100	20	1650	200	24	50	Greek	Tuna	Popcorn	Chicken	Weight	1800
62	Male	170	85	Intense			200	30	3100	400	48	##	Tofu	Lentil	Smoothie	Vegan	Weight	
28	Female	172	60	Lightly	Vegan	100	125	20	1940	250	30	60	Tofu	Black	Trail	Lentil	Weight	2000
35	Male	185	90	Intense	Omnivore	150	150	30	2520	300	36	80	Oatmeal	Grilled	Fruit	Salmon	Weight	2500
40	Female	168	75	Moderately	Vegetarian	100	150	20	2230	300	36	70	Wholegrain	Lentil	Fruit	Vegetable	Weight	2200
22	Male	190	100	Lightly	Omnivore	160	175	32	2850	350	42	90	Eggs	Chicken	Protein	Steak	Weight	2800
58	Female	155	68	Sedentary	Vegan	80	90	16	1400	180	21.6	40	Tofu	Lentil	Fruit	Vegan	Weight	1600
24	Female	165	55	Lightly	Omnivore	100	125	20	1850	250	30	50	Greek	Salmon	Apple	Chicken	Weight	1800
38	Male	180	80	Moderately	Omnivore	160	175	32	2760	350	42	80	Oatmeal	Chicken	Protein	Steak	Weight	2800
55	Female	160	70	Sedentary	Vegetarian	80	100	16	1480	200	24	40	Tofu	Lentil	Trail	Vegetarian	Weight	1600

The dataset contains **1698 entries** with **19 columns**, focusing on food, nutrition, and health-related data.

Phase 1: Load Data Set

```
# Load the dataset
file_path = 'Food_and_Nutrition.csv'
data = pd.read_csv(file_path)
```

✓ 0.0s Python

```
data.head()
```

[10] ✓ 0.0s Python

	Ages	Gender	Height	Weight	Activity_Level	Dietary_Preference	Daily_Calorie_Target	Protein	Sugar	Sodium	Calories	Carbohydr
0	25	1	180	80	3	0	2000.0	120.0	125.0	24.0	2020.0	2
1	32	0	165	65	2	3	1600.0	80.0	100.0	16.0	1480.0	2
2	48	1	175	95	4	2	2200.0	100.0	150.0	20.0	2185.0	3
3	55	0	160	70	1	0	2500.0	140.0	175.0	28.0	2680.0	3
4	62	1	170	85	4	3	2000.0	80.0	125.0	16.0	1815.0	2

```
data.shape
```

[12] ✓ 0.0s Python

```
(1698, 19)
```

Phase 2: Encoding Data

```
# Identify categorical columns
categorical_cols = data.select_dtypes(include='object').columns
```

[7] ✓ 0.0s Python

```python
# Apply Label Encoding to convert categorical data into numeric form
label_encoder = LabelEncoder()
for col in categorical_cols:
    data[col] = data[col].astype(str)  # Ensure all data is string before encoding
    data[col] = label_encoder.fit_transform(data[col])
```

[8] ✓ 0.0s Python

Splitting data into Test Train

```python
# Split dataset into features and target
X = data.drop(columns='Daily_Calorie_Target')
y = data['Daily_Calorie_Target']
```

✓ 0.0s Python

```python
# Split into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

✓ 0.0s Python

```python
# Combine X_train and y_train for imputation
train_data = pd.concat([X_train, y_train], axis=1)
```

✓ 0.0s Python

Phase 3: Handling Missing and Noisy Data

```python
#Missing values
missing_data = data.isnull().sum()
missing_columns = missing_data[missing_data > 0]
print("Columns with Missing Data:")
print(missing_columns)
```

[14] ✓ 0.0s Python

```
Columns with Missing Data:
Daily_Calorie_Target    2
Protein                 1
Sugar                   4
Sodium                  3
Calories                2
Carbohydrates           1
dtype: int64
```

```python
# Percentage of missing values per column
data.isnull().mean() * 100
```

[17] ✓ 0.0s Python

```
Ages                    0.000000
Gender                  0.000000
Height                  0.000000
Weight                  0.000000
Activity_Level          0.000000
Dietary_Preference      0.000000
Daily_Calorie_Target    0.117786
Protein                 0.058893
Sugar                   0.235571
Sodium                  0.176678
Calories                0.117786
Carbohydrates           0.058893
Fiber                   0.000000
Fat                     0.000000
Breakfast_Suggestion    0.000000
Lunch_Suggestion        0.000000
Snack_Suggestion        0.000000
Dinner_Suggestion       0.000000
Disease                 0.000000
dtype: float64
```

```
# Apply Iterative Imputer
iter_imputer = IterativeImputer(random_state=42)
train_data_iter_imputed = pd.DataFrame(iter_imputer.fit_transform(train_data), columns=train_data.columns)
```

[26] ✓ 1.7s Python

```
# Check for missing values after imputation
print("Missing Values After Iterative Imputation:\n", train_data_iter_imputed.isnull().sum())
```

[25] ✓ 0.0s Python

```
Missing Values After Iterative Imputation:
 Ages                    0
Gender                   0
Height                   0
Weight                   0
Activity_Level           0
Dietary_Preference       0
Protein                  0
Sugar                    0
Sodium                   0
Calories                 0
Carbohydrates            0
Fiber                    0
Fat                      0
Breakfast_Suggestion     0
Lunch_Suggestion         0
Snack_Suggestion         0
Dinner_Suggestion        0
Disease                  0
Daily_Calorie_Target     0
dtype: int64
```

```
# Replace original training data with Iterative Imputer results
data.update(train_data_iter_imputed)
```

[27] ✓ 0.0s Python

```
# Save the updated DataFrame to a new CSV file
data.to_csv('Missing_data_food&_Nutrition.csv', index=False)
print("Missing data has been saved handled!")
```

[28] ✓ 0.0s Python

```
Missing data has been saved handled!
```

Post-Encoding and Missing Value Imputation

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
Ages	Gender	Height	Weight	Activity_L	Dietary_P	Daily_Calc	Protein	Sugar	Sodium	Calories	Carbohyd	Fiber	Fat	Breakfast	Lunch_Su	Snack_Su	Dinner_Su	Disease
28	0	156	101	3	3	2739	122	141	24.4	2444	282	33.84	92	13	6	22	19	3
68	0	160	70	4	3	1600	80	90	16	1400	180	21.6	40	19	15	13	8	3
35	1	185	95	3	0	2500	140	140	28	2310	280	33.6	70	7	3	10	12	3
59	1	179	59	4	1	2147	61	191.5	12.2	2325	383	45.96	61	13	16	2	9	3
65	1	192	84	3	2	1850	56	174.5	11.2	2205	349	41.88	65	5	5	17	15	3
56	1	196	61	2	2	2145	160	107	32	2135	214	25.68	71	14	6	3	17	3
29	1	180	58	2	2	2751	114	149.5	22.8	1949	299	35.88	33	5	5	22	15	3
25	1	180	85	3	0	2200	120	125	24	2020	250	30	60	6	3	2	12	3
69	1	195	84	3	3	3164	237	158	47.4	3157	316	37.92	105	12	6	9	8	3
36	0	200	100	2	0	2124	159	106	31.8	2114	212	25.44	70	12	2	22	8	3
70	0	164	61	1	1	1944	141	142.5	28.2	2352	285	34.2	72	3	12	2	15	3
25	0	165	55	2	0	1800	100	100	20	1740	200	24	60	6	3	22	12	3
38	1	197	50	4	1	2754	83	197.5	16.6	2533	395	47.4	69	3	16	17	2	3
25	1	180	85	3	0	2200	120	125	24	2020	250	30	60	6	5	2	8	3
58	1	175	80	3	0	2000	100	110	20	1820	220	26.4	60	6	12	22	12	3
52	0	152	67	1	1	1916	120	122.5	24	1982	245	29.4	58	5	12	2	15	3
29	1	164	63	2	3	2583	193	129	38.6	2578	258	30.96	86	7	6	9	12	3
23	0	161	61	0	2	3046	228	152	45.6	3037	304	36.48	101	13	2	10	9	3
58	0	160	55	4	2	1600	60	100	12	1400	200	24	40	14	6	9	18	3
29	0	175	85	1	3	2600	130	160	26	2610	320	38.4	90	6	8	13	8	3
66	0	164	90	1	0	2973	222	148.5	44.4	2967	297	35.64	99	7	8	2	12	3
32	0	165	68	2	3	2000	100	125	20	1940	250	30	60	14	6	13	5	3
28	0	165	60	2	0	1800	100	100	20	1740	200	24	60	6	6	2	7	3

Iterative Imputer is best for this dataset because:

- **Captures Correlations:** Nutritional features like **Protein**, **Sugar**, and **Calories** are related, and Iterative Imputer models these relationships.
- **More Accurate:** Uses regression for precise predictions, unlike KNN's averaging.
- **Handles Complexity:** Better for datasets with hidden patterns.
- **Reduces Bias:** Considers all features, ensuring balanced imputations.

Phase 4: **Outliers Detection and Removal**

```
numerical_cols = data.select_dtypes(include='number')
```
[29] ✓ 0.0s Python

```
#Calculate skewness for each numerical column
skewness = numerical_cols.skew()
```
[30] ✓ 0.0s Python

```
#Display the skewness for each numerical column
skewness
```
[31] ✓ 0.0s Python

```
Ages                    0.157585
Gender                  0.016505
Height                  0.084227
Weight                  0.228899
Activity_Level         -0.211837
Dietary_Preference     -0.196832
Daily_Calorie_Target    0.345640
Protein                 0.362170
Sugar                   0.289450
Sodium                -41.206796
Calories              -41.206796
Carbohydrates           0.512206
Fiber                   0.287235
Fat                     0.261982
Breakfast_Suggestion   -0.129258
Lunch_Suggestion        1.101071
Snack_Suggestion        0.396130
Dinner_Suggestion      -0.243729
Disease                -4.004116
dtype: float64
```

```
# Calculate skewness for each numeric column
skewness = data.select_dtypes(include='number').skew()
```
[32] ✓ 0.0s Python

```
# Define the threshold for mild skewness (you can adjust this value)
mild_skew_threshold = 0.5  # Anything below this value will use Z-score
```
[33] ✓ 0.0s Python

```
# Identify columns with mild skewness (Z-score method) and high skewness (IQR method)
mild_skewed_cols = skewness[abs(skewness) < mild_skew_threshold].index
highly_skewed_cols = skewness[abs(skewness) >= mild_skew_threshold].index
```
[34] ✓ 0.0s Python

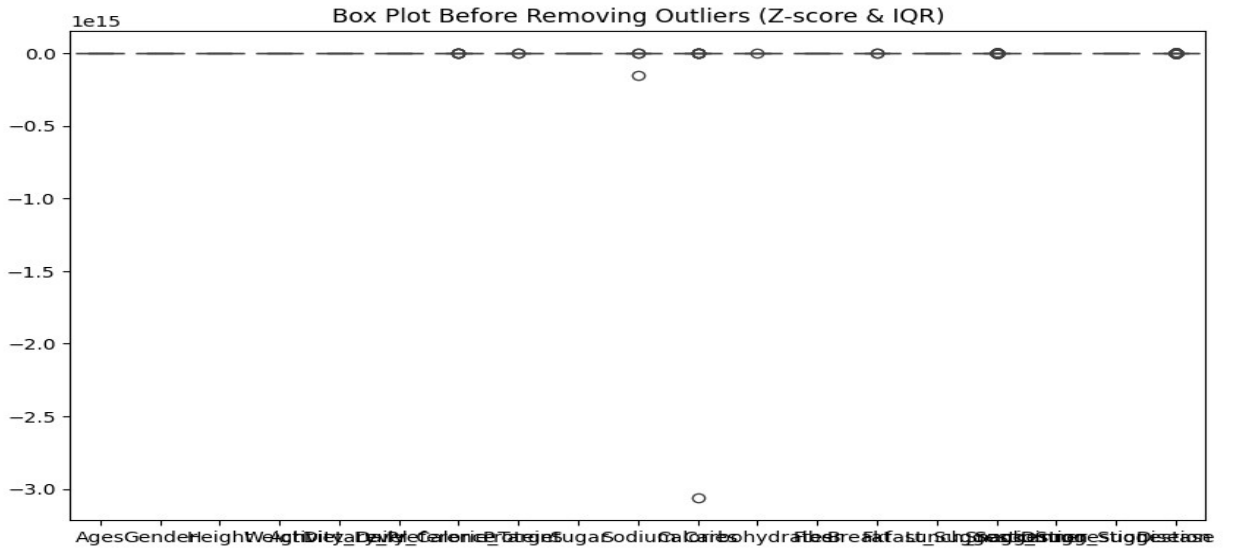
```python
# Display the results
print("Columns with Mild Skewness (Z-score Method):")
print(mild_skewed_cols)

print("\nColumns with High Skewness (IQR Method):")
print(highly_skewed_cols)
```

[36] ✓ 0.0s Python

```
Columns with Mild Skewness (Z-score Method):
Index(['Ages', 'Gender', 'Height', 'Weight', 'Activity_Level',
       'Dietary_Preference', 'Daily_Calorie_Target', 'Protein', 'Sugar',
       'Fiber', 'Fat', 'Breakfast_Suggestion', 'Snack_Suggestion',
       'Dinner_Suggestion'],
      dtype='object')

Columns with High Skewness (IQR Method):
Index(['Sodium', 'Calories', 'Carbohydrates', 'Lunch_Suggestion', 'Disease'], dtype='object')
```

```python
# Box plot before removing outliers
plt.figure(figsize=(10, 6))
sns.boxplot(data=data.select_dtypes(include=np.number))
plt.title("Box Plot Before Removing Outliers (Z-score & IQR)")
plt.show()
```

[37] ✓ 1.5s Python




```python
# Apply Z-score Method to Columns with Mild Skewness
# Calculate Z-scores for each numeric column
z_scores = np.abs(zscore(data[mild_skewed_cols]))
```

[38] ✓ 0.0s Python

```python
# Set a threshold for the Z-score (commonly 3)
threshold = 3
```

[39] ✓ 0.0s Python

```python
# Filter out outliers based on Z-score for mild skewed columns
data_zscore_cleaned = data[(z_scores < threshold).all(axis=1)]
```

[40] ✓ 0.0s Python

```python
# Apply IQR Method to Columns with High Skewness
# Calculate IQR for each numeric column with high skewness
Q1 = data[highly_skewed_cols].quantile(0.25)
Q3 = data[highly_skewed_cols].quantile(0.75)
IQR = Q3 - Q1
```

[41] ✓ 0.0s Python

```python
# Set the lower and upper bounds to filter out the outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

[42] ✓ 0.0s Python

```python
# Filter out outliers based on IQR for highly skewed columns
data_iqr_cleaned = data[~((data[highly_skewed_cols] < lower_bound) | (data[highly_skewed_cols] > upper_bound)).any(axis=1)]
```

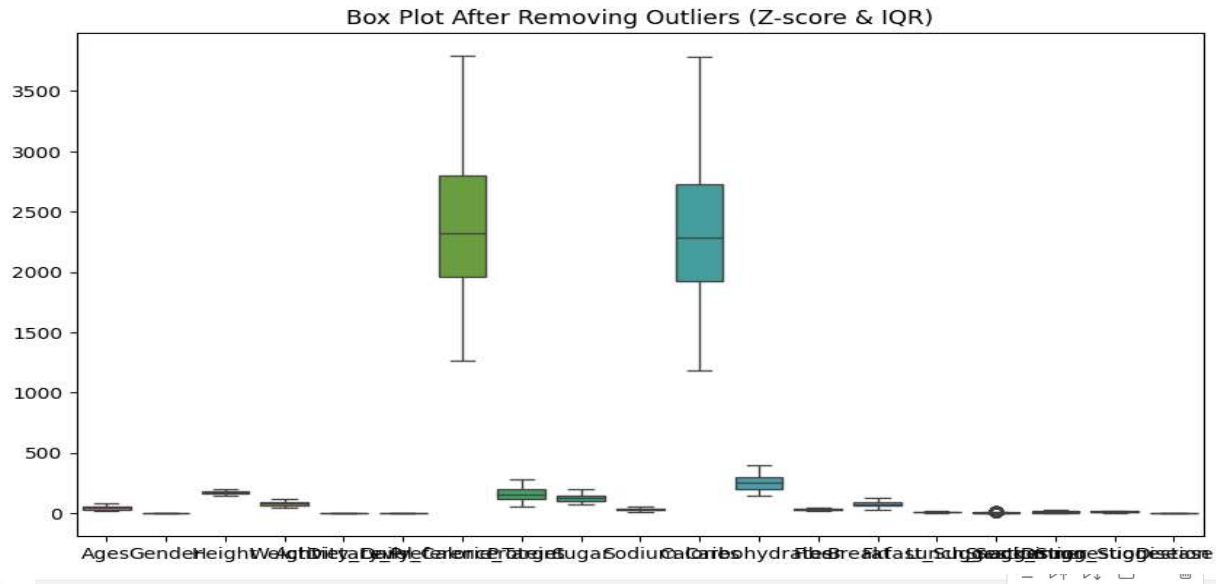
[43] ✓ 0.0s Python

```python
# Combine the cleaned data from both methods
# Since we've cleaned separately for each method, we need to combine the results
final_cleaned_data = data_zscore_cleaned.merge(data_iqr_cleaned, how='inner')
```

[44] ✓ 0.1s Python

```python
# Box plot after removing outliers (Z-score and IQR)
plt.figure(figsize=(10, 6))
sns.boxplot(data=final_cleaned_data.select_dtypes(include=np.number))
plt.title("Box Plot After Removing Outliers (Z-score & IQR)")
plt.show()
```

[45] ✓ 1.4s Python




```python
# Save the cleaned data to a CSV file after removing outliers
data_zscore_cleaned.merge(data_iqr_cleaned, how='inner').to_csv('outliers_removed.csv', index=False)
```

[46] ✓ 0.2s Python

Outliers Removal Complete

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
Ages	Gender	Height	Weight	Activity_Le	Dietary_Pr	Daily_Calc	Protein	Sugar	Sodium	Calories	Carbohydr	Fiber	Fat	Breakfast_	Lunch_Sug	Snack_Sug	Dinner_Su	Disease
28	0	156	101	3	3	2739	122	141	24.4	2444	282	33.84	92	13	6	22	19	3
68	0	160	70	4	3	1600	80	90	16	1400	180	21.6	40	19	15	13	8	3
35	1	185	95	3	0	2500	140	140	28	2310	280	33.6	70	7	3	10	12	3
65	1	192	84	3	2	1850	56	174.5	11.2	2205	349	41.88	65	5	5	17	15	3
56	1	196	61	2	2	2145	160	107	32	2135	214	25.68	71	14	6	3	17	3
29	1	180	58	2	2	2751	114	149.5	22.8	1949	299	35.88	33	5	5	22	15	3
25	1	180	85	3	0	2200	120	125	24	2020	250	30	60	6	3	2	12	3
69	1	195	84	3	3	3164	237	158	47.4	3157	316	37.92	105	12	6	9	8	3
69	1	195	84	3	3	3164	237	158	47.4	3157	316	37.92	105	12	6	9	8	3
36	0	200	100	2	0	2124	159	106	31.8	2114	212	25.44	70	12	2	22	8	3
70	0	164	61	1	1	1944	141	142.5	28.2	2352	285	34.2	72	3	12	2	15	3
25	0	165	55	2	0	1800	100	100	20	1740	200	24	60	6	3	22	12	3
25	1	180	85	3	0	2200	120	125	24	2020	250	30	60	6	5	2	8	3
58	1	175	80	3	0	2000	100	110	20	1820	220	26.4	60	6	12	22	12	3
52	0	152	67	1	1	1916	120	122.5	24	1982	245	29.4	58	5	12	2	15	3
29	1	164	63	2	3	2583	193	129	38.6	2578	258	30.96	86	7	6	9	12	3
29	1	164	63	2	3	2583	193	129	38.6	2578	258	30.96	86	7	6	9	12	3
23	0	161	61	0	2	3046	228	152	45.6	3037	304	36.48	101	13	2	10	9	3
58	0	160	55	4	2	1600	60	100	12	1400	200	24	40	14	6	9	18	3
29	0	175	85	1	3	2600	130	160	26	2610	320	38.4	90	6	8	13	8	3
66	0	164	90	1	0	2973	222	148.5	44.4	2967	297	35.64	99	7	8	2	12	3
66	0	164	90	1	0	2973	222	148.5	44.4	2967	297	35.64	99	7	8	2	12	3
32	0	165	68	2	3	2000	100	125	20	1940	250	30	60	14	6	13	5	3
28	0	165	60	2	0	1800	100	100	20	1740	200	24	60	6	6	2	7	3
28	1	191	80	2	1	2868	128	127	25.6	2203	254	30.48	75	5	5	17	2	3

Z-Score Method (for Mildly Skewed or Normal Data)

- Applied to: Columns with mild skewness (e.g., Ages, Height, Weight, Protein, Sugar, Fiber).
- Threshold: Z-score > 3 or < -3.
- Data points far from the mean (more than 3 standard deviations) were marked as outliers.

IQR Method (for Highly Skewed Data)

- Applied to: Highly skewed columns (e.g., Sodium, Calories, Disease).
    - Q1 (25th percentile) and Q3 (75th percentile) were calculated.
    - Outliers were values below Q1 - 1.5×IQR or above Q3 + 1.5×IQR.

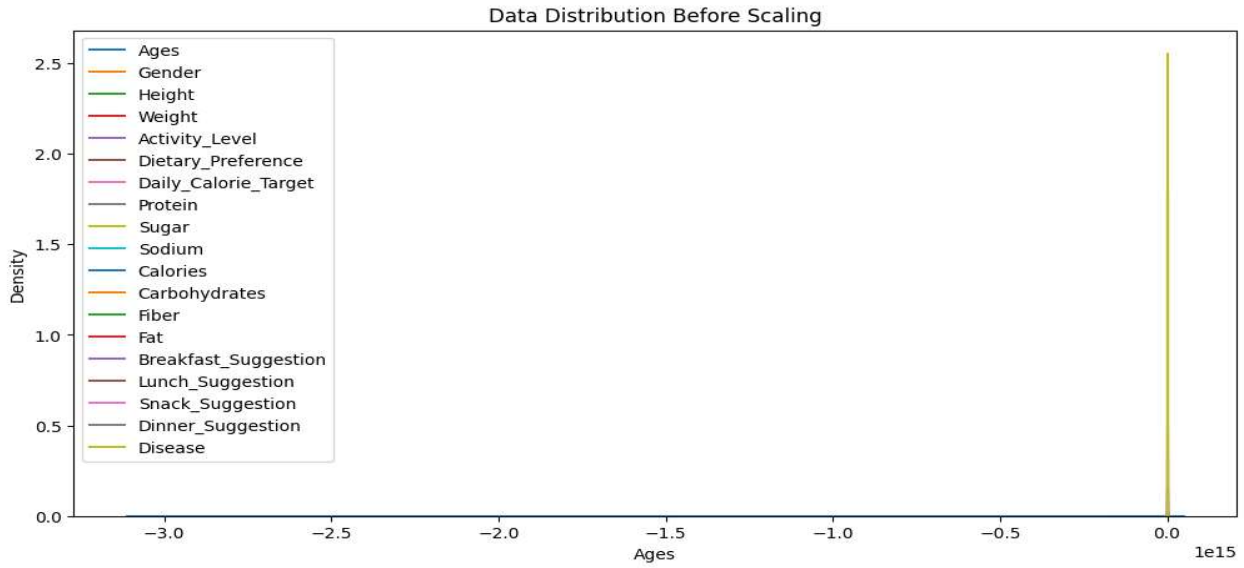
Phase 5: feature scaling

```
# Select numeric columns for scaling
numeric_coloumns= data.select_dtypes(include=np.number).columns
```

[47] ✓ 0.0s Python

```
# Visualize the data before scaling
plt.figure(figsize=(12, 6))
plt.title("Data Distribution Before Scaling")
for col in numeric_coloumns:
    sns.kdeplot(data[col], label=col)
plt.legend()
plt.show()
```

[48] ✓ 2.1s Python




```
# Apply StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data[numeric_coloumns])
```

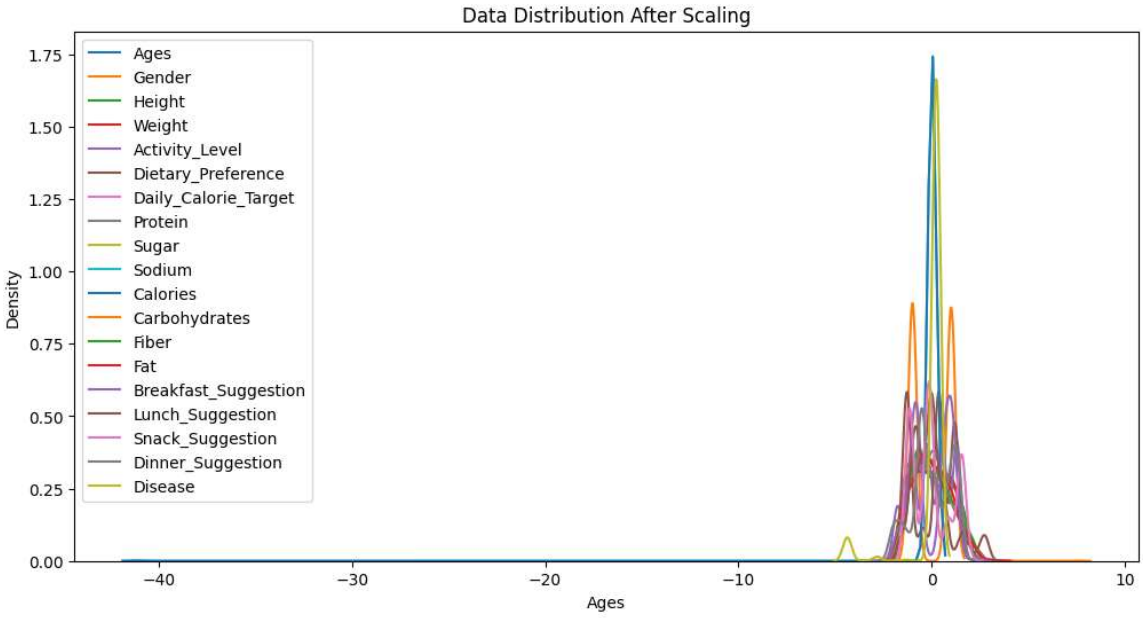
[49] ✓ 0.0s Python

```
# Convert scaled data back to a DataFrame
scaled_df = pd.DataFrame(scaled_data, columns=numeric_coloumns)
```

[50] ✓ 0.0s Python

```
# Visualize the data after scaling
plt.figure(figsize=(12, 6))
plt.title("Data Distribution After Scaling")
for col in numeric_coloumns:
    sns.kdeplot(scaled_df[col], label=col)
plt.legend()
plt.show()
```

[51] 2.7s Python



```
# Save the scaled data to a new CSV file
data_scaled = data.copy()
data_scaled[numeric_coloumns] = scaled_df
data_scaled.to_csv('Food_and_Nutrition_Scaled.csv', index=False)
```
[52] ✓ 0.2s Python

```
print("Feature scaling applied and results saved to 'Food_and_Nutrition_Scaled.csv'.")
```
[37] Python

```
Feature scaling applied and results saved to 'Food_and_Nutrition_Scaled.csv'.
```

Scaled Result

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Ages	Gender	Height	Weight	Activity_Le	Dietary_Pr	Daily_Calc	Protein	Sugar	Sodium	Calories	Carbohydr	Fiber	Fat	Breakfast_	Lunch_Sug	Snack_Sug	Dinner_Su	Disease
2	-0.99429	-0.99179	-1.31686	1.432032	0.537777	1.195421	0.762941	-0.483	0.468217	0.024275	0.024275	0.45495	0.466967	0.92918	0.8149	0.074045	1.551645	1.34214	0.240518
3	1.520967	-0.99179	-1.02588	-0.45109	1.301915	1.195421	-1.22504	-1.25455	-1.02681	0.024275	0.024275	-1.01513	-1.02821	-1.4736	2.362664	2.486742	0.320032	-0.60296	0.240518
4	-0.55412	1.008279	0.79274	1.067558	0.537777	-1.28098	0.345796	-0.15233	0.438903	0.024275	0.024275	0.426125	0.43765	-0.08738	-0.73286	-0.73019	-0.09051	0.104347	0.240518
5	0.955034	1.008279	0.35627	-1.11929	1.301915	-0.45551	-0.27032	-1.60359	1.948588	0.024275	0.024275	1.910622	1.947481	-0.50325	0.8149	2.75482	-1.18527	-0.42614	0.240518
6	1.332323	1.008279	1.301955	0.399355	0.537777	0.369953	-0.7887	-1.69544	1.450245	0.024275	0.024275	1.420594	1.44909	-0.31842	-1.24878	-0.19403	0.867416	0.63483	0.240518
7	0.76639	1.008279	1.592934	-0.9978	-0.22636	0.369953	-0.27381	0.21507	-0.52847	0.024275	0.024275	-0.52511	-0.52981	-0.04117	1.072861	0.074045	-1.04843	0.988485	0.240518
8	-0.93141	1.008279	0.429015	-1.18003	-0.22636	0.369953	0.783886	-0.62996	0.717388	0.024275	0.024275	0.699964	0.716163	-1.79705	-1.24878	-0.19403	1.551645	0.63483	0.240518
9	-1.18293	1.008279	0.429015	0.460101	0.537777	-1.28098	-0.17782	-0.51974	-0.00081	0.024275	0.024275	-0.00625	-0.00211	-0.54945	-0.99082	-0.73019	-1.18527	0.104347	0.240518
10	1.583848	1.008279	1.520189	0.399355	0.537777	1.195421	1.504725	1.629581	0.96656	0.024275	0.024275	0.944978	0.965358	1.529874	0.55694	0.074045	-0.22735	-0.60296	0.240518
11	-0.49124	-0.99179	1.883914	1.371287	-0.22636	-1.28098	-0.31046	0.1967	-0.55778	0.024275	0.024275	-0.55393	-0.55913	-0.08738	0.55694	-0.99826	1.551645	-0.60296	0.240518
12	1.646729	-0.99179	-0.7349	-0.9978	-0.9905	-0.45551	-0.62463	-0.13396	0.512188	0.024275	0.024275	0.498188	0.510943	0.005034	-1.76471	1.68251	-1.18527	0.63483	0.240518
13	-1.18293	-0.99179	-0.66216	-1.36227	-0.22636	-1.28098	-0.87597	-0.88715	-0.73367	0.024275	0.024275	-0.72688	-0.73503	-0.54945	-0.99082	-0.73019	1.551645	0.104347	0.240518
14	-0.36547	1.008279	1.665679	-1.666	1.301915	-0.45551	0.789122	-1.19944	2.124473	0.024275	0.024275	2.083573	2.123384	-0.13359	-1.76471	2.75482	0.867416	-1.66393	0.240518
15	-1.18293	1.008279	0.429015	0.460101	0.537777	-1.28098	-0.17782	-0.51974	-0.00081	0.024275	0.024275	-0.00625	-0.00211	-0.54945	-0.99082	-0.19403	-1.18527	-0.60296	0.240518
16	0.892153	1.008279	0.065291	0.156372	0.537777	-1.28098	-0.52689	-0.88715	-0.44053	0.024275	0.024275	-0.43863	-0.44186	-0.54945	-0.99082	1.68251	1.551645	0.104347	0.240518
17	0.514865	-0.99179	-1.60784	-0.63332	-0.9905	-0.45551	-0.6735	-0.51974	-0.0741	0.024275	0.024275	-0.07832	-0.0754	-0.64187	-1.24878	1.68251	-1.18527	0.63483	0.240518
18	-0.93141	1.008279	-0.7349	-0.87631	-0.22636	1.195421	0.490663	0.821289	0.116446	0.024275	0.024275	0.109048	0.115162	0.651936	-0.73286	0.074045	-0.22735	0.104347	0.240518
19	-1.3087	-0.99179	-0.95314	-0.9978	-1.75464	0.369953	1.298771	1.464248	0.790674	0.024275	0.024275	0.772027	0.789455	1.345045	0.8149	-0.99826	-0.09051	-0.42614	0.240518
20	0.892153	-0.99179	-1.02588	-1.36227	1.301915	0.369953	-1.22504	-1.62196	-0.73367	0.024275	0.024275	-0.72688	-0.73503	-1.4736	1.072861	0.074045	-0.22735	1.165313	0.240518
21	-0.93141	-0.99179	0.065291	0.460101	-0.9905	1.195421	0.520334	-0.33604	1.025188	0.024275	0.024275	1.002628	1.023992	0.836765	-0.99082	0.6102	0.320032	-0.60296	0.240518
22	1.395204	-0.99179	-0.7349	0.763829	-0.9905	-1.28098	1.171359	1.354027	0.688074	0.024275	0.024275	0.671139	0.686846	1.25263	-0.73286	0.6102	-1.18527	0.104347	0.240518
23	-0.74276	-0.99179	-0.66216	-0.57258	-0.22636	1.195421	-0.52689	-0.88715	-0.00081	0.024275	0.024275	-0.00625	-0.00211	-0.54945	1.072861	0.074045	0.320032	-1.13345	0.240518
24	-0.99429	-0.99179	-0.66216	-1.05854	-0.22636	-1.28098	-0.87597	-0.88715	-0.73367	0.024275	0.024275	-0.72688	-0.73503	-0.54945	-0.99082	0.074045	-1.18527	-0.77979	0.240518
25	-0.99429	1.008279	1.22921	0.156372	-0.22636	-0.45551	0.988094	-0.37278	0.057817	0.024275	0.024275	0.051397	0.056528	0.143656	-1.24878	-0.19403	0.867416	-1.66393	0.240518
26	1.080797	-0.99179	-0.37118	-0.57258	-1.75464	-1.28098	1.066636	1.280546	0.600131	0.024275	0.024275	0.584663	0.598894	1.160216	-0.99082	-0.73019	1.551645	-0.42614	0.240518
27	1.332323	1.008279	-0.80765	-0.26885	0.537777	0.369953	-1.02607	-0.37278	-1.15873	0.024275	0.024275	-1.14485	-1.16013	-0.68807	0.55694	-0.19403	-1.18527	-0.42614	0.240518

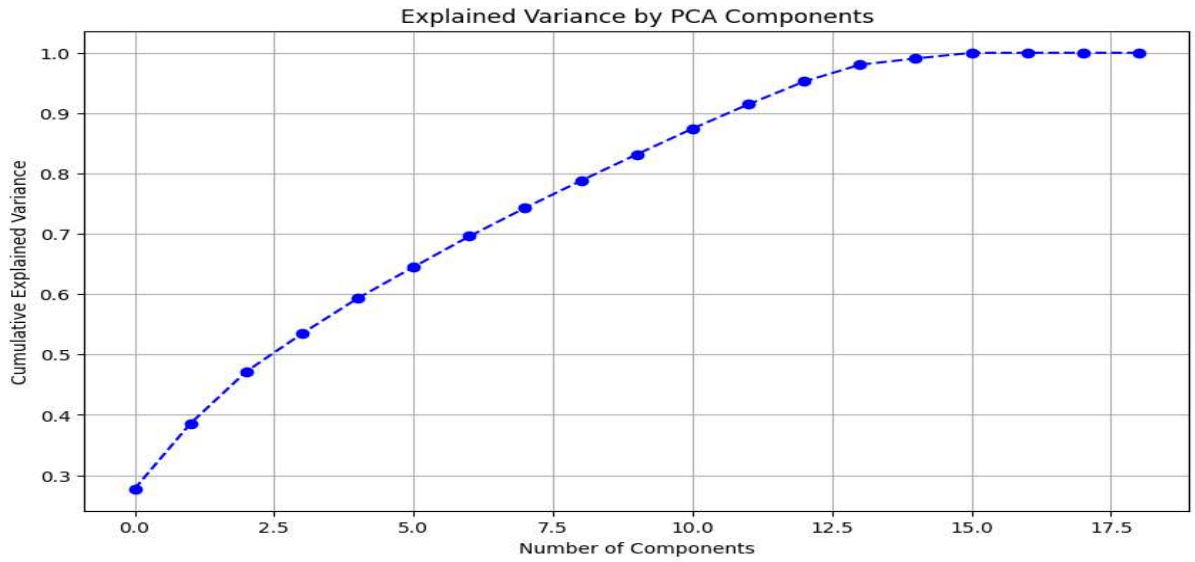
StandardScaler is applied because the numeric features in food and nutrition data (e.g., calories, protein, fat, etc.) likely have different scales and units. Standardizing ensures that larger-scale features (e.g., calorie counts) don't dominate smaller-scale ones (e.g., vitamin percentages), making the dataset balanced and suitable for dimensionality reduction techniques like PCA and distance-based models.

Phase 6: **PCA**

```
# Determine the optimal number of components
pca = PCA()
pca.fit(scaled_data)
```
[53] ✓ 0.0s Python

```
PCA()
```

```
# Plot the cumulative explained variance
plt.figure(figsize=(10, 6))
plt.plot(np.cumsum(pca.explained_variance_ratio_), marker='o', linestyle='--', color='b')
plt.title('Explained Variance by PCA Components')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.grid()
plt.show()
```
[54] ✓ 0.5s Python




```
# Choose the number of components that explain at least 95% of the variance
explained_variance_threshold = 0.95
n_components = np.argmax(np.cumsum(pca.explained_variance_ratio_) >= explained_variance_threshold) + 1
print(f"Number of components explaining at least {explained_variance_threshold*100}% variance: {n_components}")
```

[55] ✓ 0.0s Python

```
Number of components explaining at least 95.0% variance: 13
```

```
# Apply PCA with the chosen number of components
pca = PCA(n_components=n_components)
pca_data = pca.fit_transform(scaled_data)
```

[56] ✓ 0.0s Python

```
# Convert PCA-transformed data back to a DataFrame
pca_columns = [f'PC{i+1}' for i in range(n_components)]
pca_df = pd.DataFrame(pca_data, columns=pca_columns)
```

[57] ✓ 0.0s Python

```
# Save the PCA-transformed data to a new CSV file
pca_df.to_csv('Food_and_Nutrition_PCA.csv', index=False)
print(f"PCA applied and results saved to 'Food_and_Nutrition_PCA.csv'.")
```

[58] ✓ 0.2s Python

PCA Result

A	B	C	D	E	F	G	H	I	J	K	L	M
PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13
0.95928	0.2281	0.32112	-1.5546	0.63209	1.54551	0.13786	1.15295	0.52738	0.62007	1.99203	0.46841	0.9544
-3.441	-0.1909	-0.8705	-2.3478	2.20747	-1.1597	-0.4435	-0.444	0.10569	1.69319	0.41651	0.17197	-0.3156
1.00611	-0.3001	-0.9609	1.85864	-0.4243	0.56	-0.1013	0.02889	0.06468	-0.4232	0.56146	0.48637	-0.2026
1.01085	-0.8785	-2.9454	-0.9561	0.8318	-1.7684	-1.5134	-0.7877	0.03952	0.35992	1.05137	-2.0091	-2.3411
0.96399	-0.6399	-2.3918	0.29702	1.1729	0.85848	0.54119	1.2822	0.27253	-1.2298	0.03526	-1.0692	-1.3478
-0.5243	0.37268	1.25053	0.9119	1.49785	-0.4567	-0.9429	-0.4696	0.17515	0.15541	-0.8206	-0.4629	-1.2422
0.51753	-0.3249	-1.031	-0.4608	-0.6713	2.17421	-0.0559	-0.0423	0.88804	-0.738	-0.3612	-1.2173	-0.8977
-0.1166	-0.314	-0.6011	1.89754	-1.2589	0.03629	-0.8522	-0.125	0.44871	-0.8091	0.59698	0.46511	-0.2776
2.91322	0.25504	-0.05	0.57435	2.46585	-0.8449	-0.0517	-0.2894	-0.0802	-0.48	-0.9412	-0.5865	0.05349
-0.1249	0.02293	-0.079	0.86976	0.38941	1.65333	0.66573	-0.5158	-1.7819	0.19182	-0.2556	2.3726	0.1116
0.01827	-0.1131	-0.5943	-0.9006	-1.3066	-1.8491	-0.0362	1.99738	-0.3396	0.63922	-1.0656	-1.2695	-1.2163
-1.7968	-0.0576	0.08539	-0.4932	-2.2457	1.80593	0.86541	-0.2258	0.06458	0.08056	-0.3517	0.67605	-0.7589
2.17287	-1.1706	-4.1219	-0.8797	-0.5049	0.62415	-1.6622	-0.9843	-0.4543	-0.5596	-1.0926	-2.0127	-1.8035
-0.1372	-0.4331	-1.0754	1.64903	-1.2873	-0.2473	-1.0396	-0.483	0.28105	-0.6437	0.29866	0.50582	-0.017
-1.1765	-0.5385	-2.0889	0.75469	-0.0384	0.84735	0.27558	0.40793	0.23839	1.79989	-0.8335	-0.4036	-0.4944
-1.0168	-0.1014	-0.3148	-1.1171	-1.8026	-1.5402	-0.3937	1.59048	0.11903	1.03046	-0.2709	-0.7031	-0.4523
0.82029	0.26245	0.6832	-0.3208	-0.604	0.18787	-0.7187	-0.2013	1.51088	-0.365	-0.6247	-0.9786	0.70493
2.51914	0.7179	2.03159	-1.654	-1.2999	-0.2003	0.50362	-1.0644	-0.0535	-0.3038	0.11933	0.14748	0.42406
-3.1942	-0.0048	0.30583	-1.1569	0.61481	-0.3256	0.18584	0.44413	0.70058	0.06622	1.06847	-0.1036	-1.7349
1.87898	-0.0666	-0.6563	-1.7734	-0.5992	0.35141	-0.5923	0.71584	-0.4502	-0.7016	-0.3569	0.01437	0.88258
2.19063	0.25286	0.21496	0.27437	-0.959	-2.1228	0.7024	1.38184	-0.8905	1.06132	-0.2511	0.06384	-0.0548
-0.8579	-0.005	-0.0898	-2.2508	0.11747	0.00521	-0.1042	-0.9791	0.11072	-0.5464	0.35301	0.54475	0.36475

PCA reduces my dataset's dimensions while retaining 95% variance, improving model efficiency, reducing noise, and addressing feature redundancy. It simplifies analysis and speeds up computations, making it highly beneficial for high-dimensional data like yours.

Phase 7: **Pipelining**

```
# Separate features and target
X = data.drop(columns='Daily_Calorie_Target')
y = data['Daily_Calorie_Target']
```
[59] ✓ 0.0s Python

```
numerical_cols = X.select_dtypes(include='number').columns
```
[62] ✓ 0.0s Python

```
# Separate features and target
X = data.drop(columns='Daily_Calorie_Target')
y = data['Daily_Calorie_Target']
```
[61] ✓ 0.0s Python

```
# Identify numerical columns
numerical_cols = X.select_dtypes(include='number').columns
```
[47] Python

```
# Define preprocessing steps for numerical data
numerical_preprocessor = Pipeline(steps=[
    ('imputer', IterativeImputer(random_state=42)),  # Impute missing values
    ('scaler', StandardScaler())  # Standardize the data
])
```
[63] ✓ 0.0s Python

```
# Full Preprocessing Pipeline (with PCA)
full_pipeline = Pipeline(steps=[
    ('preprocessor', numerical_preprocessor),
    ('pca', PCA(n_components=0.95))  # Retain 95% variance
])
```
[64] ✓ 0.0s Python

```
# Basic Preprocessing Pipeline (without PCA)
basic_pipeline = Pipeline(steps=[
    ('preprocessor', numerical_preprocessor)
])
```
[65] ✓ 0.0s Python

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```
[66] ✓ 0.0s Python

```
# Define regression models to evaluate
models = {
    "RandomForest": RandomForestRegressor(random_state=42),
    "LinearRegression": LinearRegression(),
    "GradientBoosting": GradientBoostingRegressor(random_state=42)
}
```
[67] ✓ 0.0s Python

```python
# Function to evaluate regression models with a given pipeline
def evaluate_models(pipeline, X_train, X_test, y_train, y_test, models):
    results = {}
    for name, model in models.items():
        model_pipeline = Pipeline(steps=[
            ('pipeline', pipeline),
            ('model', model)
        ])
        model_pipeline.fit(X_train, y_train)
        y_pred = model_pipeline.predict(X_test)
        mse = mean_squared_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)
        results[name] = {"MSE": mse, "R²": r2}
    return results
```

[70] ✓ 0.0s Python

```python
# Evaluate with both preprocessing strategies
print("Evaluating models with full preprocessing (including PCA)...")
full_results = evaluate_models(full_pipeline, X_train, X_test, y_train, y_test, models)
```

[71] 3.7s Python

```
Evaluating models with full preprocessing (including PCA)...
```

```python
print("Evaluating models with basic preprocessing (without PCA)...")
basic_results = evaluate_models(basic_pipeline, X_train, X_test, y_train, y_test, models)
```

[72] ✓ 6.5s Python

```
Evaluating models with basic preprocessing (without PCA)...
```

```python
# Compare results
print("\nModel Performance Comparison:")
print("\nFull Preprocessing:")
for model, metrics in full_results.items():
    print(f"  {model}:")
    print(f"    - MSE: {metrics['MSE']:.2f}")
    print(f"    - R²: {metrics['R²']:.4f}")

print("\nBasic Preprocessing:")
for model, metrics in basic_results.items():
    print(f"  {model}:")
    print(f"    - MSE: {metrics['MSE']:.2f}")
    print(f"    - R²: {metrics['R²']:.4f}")
```

[73] ✓ 0.0s Python

```
Model Performance Comparison:

Full Preprocessing:
  RandomForest:
    - MSE: 73502.01
    - R²: 0.7873
  LinearRegression:
    - MSE: 93332.75
    - R²: 0.7299
  GradientBoosting:
    - MSE: 76720.64
    - R²: 0.7780

Basic Preprocessing:
  RandomForest:
    - MSE: 70701.82
    - R²: 0.7954
  LinearRegression:
    - MSE: 55579340.25
    - R²: -159.8385
  GradientBoosting:
    - MSE: 74996.86
    - R²: 0.7830
```

```
# Create the final cleaned dataset by combining scaled features
final_cleaned_data = final_cleaned_data.copy()  # Replace with your cleaned dataset
numeric_columns = final_cleaned_data.select_dtypes(include='number').columns  # Numeric columns for scaling
final_cleaned_data[numeric_columns] = scaled_df  # Apply the scaled numeric features
```

[74] ✓ 0.1s Python

```
# Save the final cleaned dataset to a CSV file
final_cleaned_filename = 'Cleaned_Food_and_Nutrition.csv'
final_cleaned_data.to_csv(final_cleaned_filename, index=False)
```

[75] ✓ 0.4s Python

```
print(f"Cleaned dataset saved as: {final_cleaned_filename}")
```

[76] ✓ 0.0s Python

```
Cleaned dataset saved as: Cleaned_Food_and_Nutrition.csv
```

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
Ages	Gender	Height	Weight	Activity_Le	Dietary_Pr	Daily_Calc	Protein	Sugar	Sodium	Calories	Carbohydr	Fiber	Fat	Breakfast_	Lunch_Sug	Snack_Sug	Dinner_Su	Disease
-0.99429	-0.99179	-1.31686	1.432032	0.537777	1.195421	0.762941	-0.483	0.468217	0.024275	0.024275	0.45495	0.466967	0.92918	0.8149	0.074045	1.551645	1.34214	0.240518
1.520967	-0.99179	-1.02588	-0.45109	1.301915	1.195421	-1.22504	-1.25455	-1.02681	0.024275	0.024275	-1.01513	-1.02821	-1.4736	2.362664	2.486742	0.320032	-0.60296	0.240518
-0.55412	1.008279	0.79274	1.067558	0.537777	-1.28098	0.345796	-0.15233	0.438903	0.024275	0.024275	0.426125	0.43765	-0.08738	-0.73286	-0.73019	-0.09051	0.104347	0.240518
0.955034	1.008279	0.35627	-1.11929	1.301915	-0.45551	-0.27032	-1.60359	1.948588	0.024275	0.024275	1.910622	1.947481	-0.50325	0.8149	2.75482	-1.18527	-0.42614	0.240518
1.332323	1.008279	1.301955	0.399355	0.537777	0.369953	-0.7887	-1.69544	1.450245	0.024275	0.024275	1.420594	1.44909	-0.31842	-1.24878	-0.19403	0.867416	0.63483	0.240518
0.76639	1.008279	1.592934	-0.9978	-0.22636	0.369953	-0.27381	0.21507	-0.52847	0.024275	0.024275	-0.52511	-0.52981	-0.04117	1.072861	0.074045	-1.04843	0.988485	0.240518
-0.93141	1.008279	0.429015	-1.18003	-0.22636	0.369953	0.783886	-0.62996	0.717388	0.024275	0.024275	0.699964	0.716163	-1.79705	-1.24878	-0.19403	1.551645	0.63483	0.240518
-1.18293	1.008279	0.429015	0.460101	0.537777	-1.28098	-0.17782	-0.51974	-0.00081	0.024275	0.024275	-0.00625	-0.00211	-0.54945	-0.99082	-0.73019	-1.18527	0.104347	0.240518
1.583848	1.008279	1.520189	0.399355	0.537777	1.195421	1.504725	1.629581	0.96656	0.024275	0.024275	0.944978	0.965358	1.529874	0.55694	0.074045	-0.22735	-0.60296	0.240518
-0.49124	-0.99179	1.883914	1.371287	-0.22636	-1.28098	-0.31046	0.1967	-0.55778	0.024275	0.024275	-0.55393	-0.55913	-0.08738	0.55694	-0.99826	1.551645	-0.60296	0.240518
1.646729	-0.99179	-0.7349	-0.9978	-0.9905	-0.45551	-0.62463	-0.13396	0.512188	0.024275	0.024275	0.498188	0.510943	0.005034	-1.76471	1.68251	-1.18527	0.63483	0.240518
-1.18293	-0.99179	-0.66216	-1.36227	-0.22636	-1.28098	-0.87597	-0.88715	-0.73367	0.024275	0.024275	-0.72688	-0.73503	-0.54945	-0.99082	-0.73019	1.551645	0.104347	0.240518
-0.36547	1.008279	1.665679	-1.666	1.301915	-0.45551	0.789122	-1.19944	2.124473	0.024275	0.024275	2.083573	2.123384	-0.13359	-1.76471	2.75482	0.867416	-1.66393	0.240518
-1.18293	1.008279	0.429015	0.460101	0.537777	-1.28098	-0.17782	-0.51974	-0.00081	0.024275	0.024275	-0.00625	-0.00211	-0.54945	-0.99082	-0.19403	-1.18527	-0.60296	0.240518
0.892153	1.008279	0.065291	0.156372	0.537777	-1.28098	-0.52689	-0.88715	-0.44053	0.024275	0.024275	-0.43863	-0.44186	-0.54945	-0.99082	1.68251	1.551645	0.104347	0.240518
0.514865	-0.99179	-1.60784	-0.63332	-0.9905	-0.45551	-0.6735	-0.51974	-0.0741	0.024275	0.024275	-0.07832	-0.0754	-0.64187	-1.24878	1.68251	-1.18527	0.63483	0.240518
-0.93141	1.008279	-0.7349	-0.87631	-0.22636	1.195421	0.490663	0.821289	0.116446	0.024275	0.024275	0.109048	0.115162	0.651936	-0.73286	0.074045	-0.22735	0.104347	0.240518

Preprocessing Pipeline:

- **Missing Value Imputation**: Handled with **Iterative Imputer**.
- **Outlier Removal**: Applied **Z-score** and **IQR** methods.
- **Scaling**: Used **StandardScaler** for numeric features.
- **Dimensionality Reduction**: Applied **PCA** for feature reduction.

Model Evaluation:

- **Tree-Based Models** (RandomForest, GradientBoosting):
- Performed best with **Basic Preprocessing**.
  Best R2R^2R2: **0.7934** (RandomForest).
- **Linear Model** (Linear Regression):
- Needed **Full Preprocessing** for better performance.
  Best R2R^2R2: **0.7299**.
  Tree-based models like RandomForest are more suitable for the dataset with simpler preprocessing.

Phase 8: Interface

```
import tkinter as tk
```

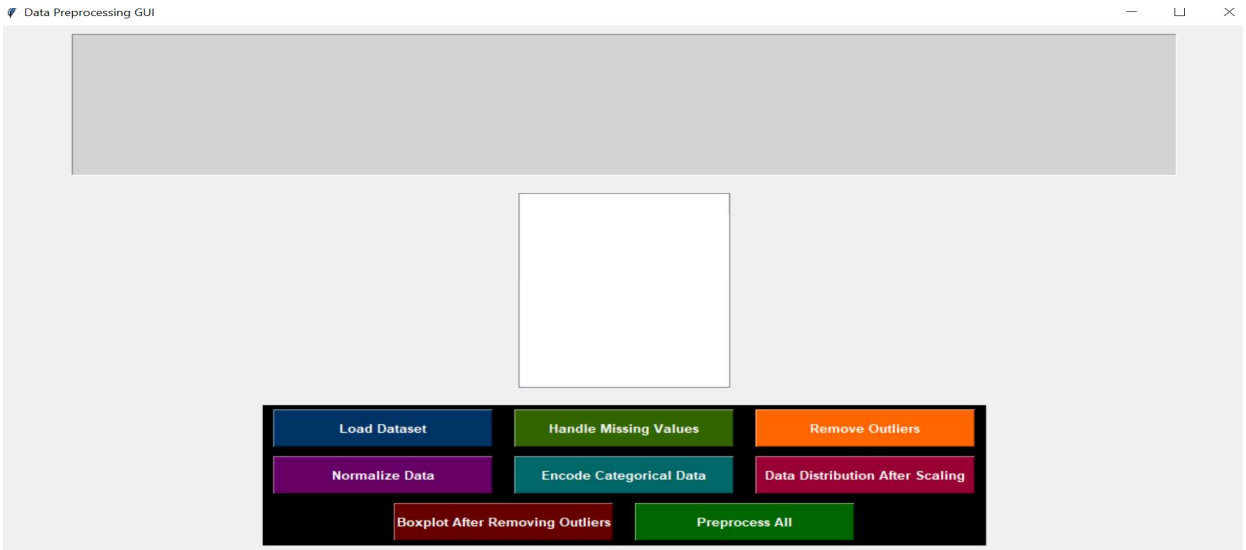
```
from tkinter import filedialog, ttk
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler

# Initialize a global dataframe
df = pd.DataFrame()

# Function definitions for data preprocessing steps
def load_dataset():
    global df
    file_path = filedialog.askopenfilename()
    if file_path:
        df = pd.read_csv(file_path)
        clear_output()
        display_table(df)
        output_text.insert(tk.END, "Dataset loaded successfully!\n\n")


def display_table(data):
    # Clear previous table
    for row in tree.get_children():
        tree.delete(row)
    # Insert new table data
    tree["columns"] = list(data.columns)
    tree["show"] = "headings"
    for col in data.columns:
        tree.heading(col, text=col)
        tree.column(col, width=100)
    for index, row in data.iterrows():
        tree.insert("", "end", values=list(row))


def clear_output():
    output_text.delete(1.0, tk.END)


def handle_missing_values():
    global df
    clear_output()
    df = df.dropna()
    display_table(df)
    output_text.insert(tk.END, "Missing values handled.\n\n")
```

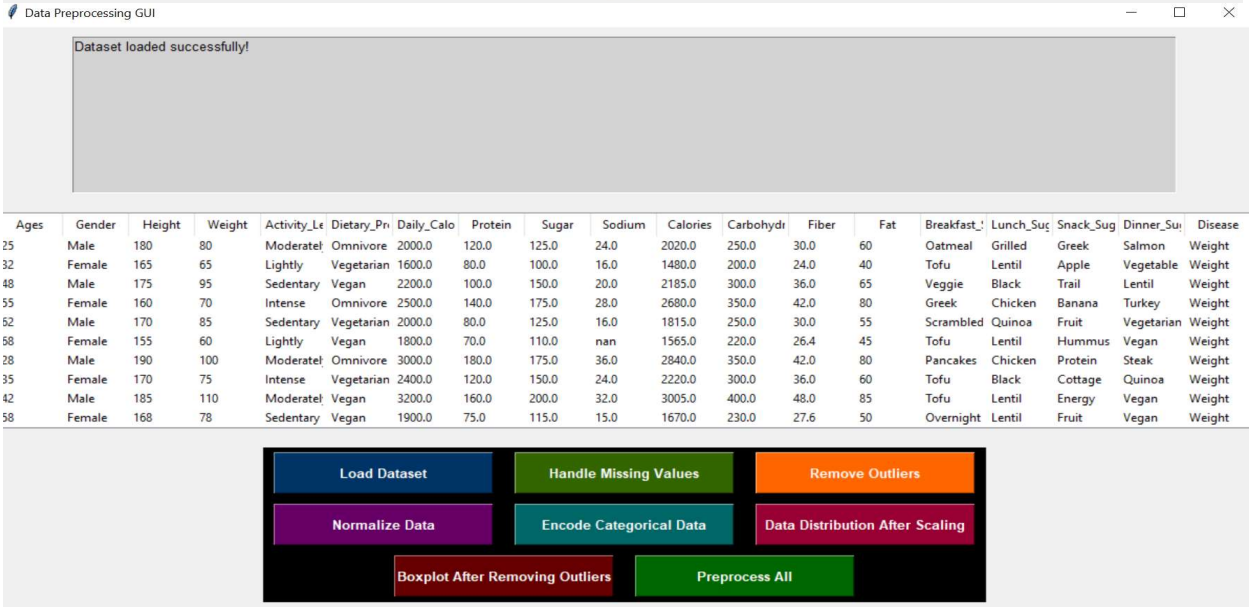
```
def remove_outliers():
    global df
    clear_output()
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
    display_table(df)
    output_text.insert(tk.END, "Outliers removed.\n\n")


def normalize_data():
    global df
    clear_output()
    numeric_columns = df.select_dtypes(include=['number']).columns
    if not numeric_columns.empty:
        scaler = RobustScaler()
        df[numeric_columns] = scaler.fit_transform(df[numeric_columns])
        display_table(df)
        output_text.insert(tk.END, "Data normalized.\n\n")
    else:
        output_text.insert(tk.END, "No numeric columns to normalize.\n\n")


def encode_categorical_data():
    global df
    clear_output()
    categorical_columns = df.select_dtypes(include=['object',
'category']).columns
    if not categorical_columns.empty:
        df = pd.get_dummies(df, columns=categorical_columns, drop_first=True)
        display_table(df)
        output_text.insert(tk.END, "Categorical data encoded.\n\n")
    else:
        output_text.insert(tk.END, "No categorical columns to encode.\n\n")


def plot_data_distribution():
    global df
    clear_output()
    output_text.insert(tk.END, "Displaying data distribution after scaling.\n\n")
    df.plot(kind='kde', subplots=True, layout=(3, 3), figsize=(12, 8),
sharex=False)
    plt.tight_layout()
    plt.show()
```

```
def plot_boxplot_after_outliers():
    global df
    clear_output()
    output_text.insert(tk.END, "Displaying boxplot after removing outliers.\n\n")
    df.boxplot(figsize=(10, 6))
    plt.show()


def preprocess_all():
    global df
    clear_output()
    file_path = filedialog.askopenfilename(title="Select Cleaned Dataset for
Preprocessing")
    if file_path:
        df = pd.read_csv(file_path)
        handle_missing_values()
        remove_outliers()
        normalize_data()
        encode_categorical_data()
        display_table(df)
        output_text.insert(tk.END, "All preprocessing steps completed and cleaned
data displayed.\n\n")

# GUI Implementation
root = tk.Tk()
root.title("Data Preprocessing GUI")
root.geometry("1200x800")  # Increased window size for better layout

# Output Text Area
output_text = tk.Text(root, wrap=tk.WORD, height=10, width=150, font=("Arial",
10), bg="lightgray")
output_text.pack(pady=10)

# Table for Displaying Data
tree_frame = tk.Frame(root)
tree_frame.pack(pady=10)
tree = ttk.Treeview(tree_frame)
tree.pack()

# Button Frame
button_frame = tk.Frame(root, bg="black")
button_frame.pack(pady=10)
```

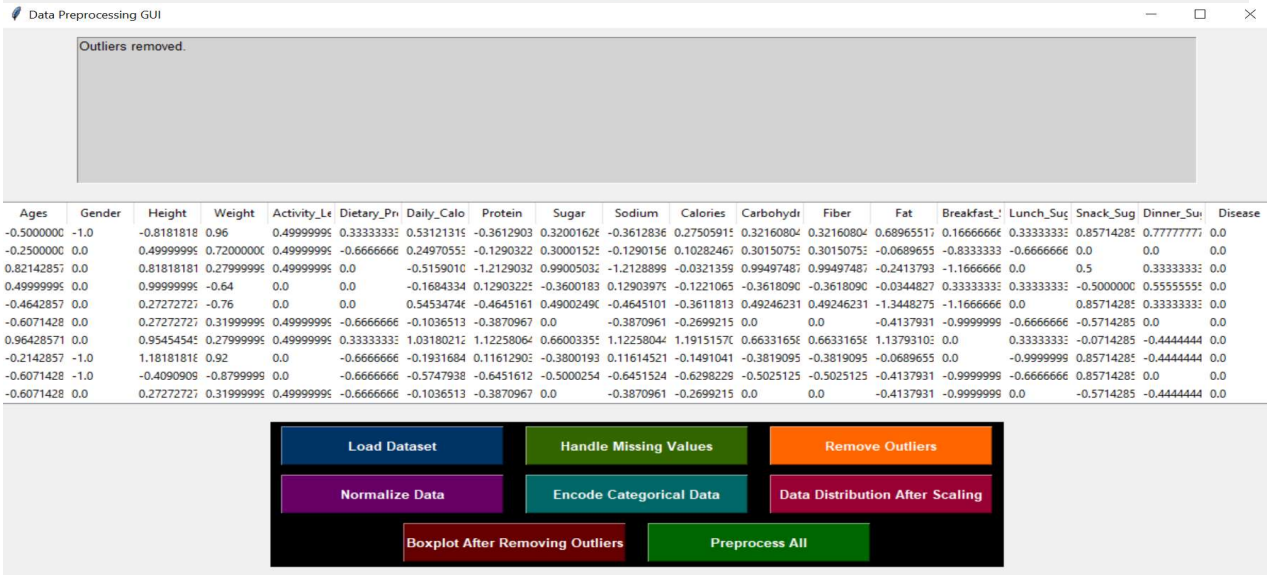
```
# Arrange buttons in rows with three per row
buttons = [
    {"text": "Load Dataset", "command": load_dataset, "bg": "#003366"},
    {"text": "Handle Missing Values", "command": handle_missing_values, "bg":
"#336600"},
    {"text": "Remove Outliers", "command": remove_outliers, "bg": "#FF6600"},
    {"text": "Normalize Data", "command": normalize_data, "bg": "#660066"},
    {"text": "Encode Categorical Data", "command": encode_categorical_data, "bg":
"#006666"},
    {"text": "Data Distribution After Scaling", "command":
plot_data_distribution, "bg": "#990033"},
    {"text": "Boxplot After Removing Outliers", "command":
plot_boxplot_after_outliers, "bg": "#660000"},
    {"text": "Preprocess All", "command": preprocess_all, "bg": "#006600"},
]

for i in range(0, len(buttons), 3):
    row_frame = tk.Frame(button_frame, bg="black")
    row_frame.pack(pady=5)
    for button in buttons[i:i+3]:
        tk.Button(
            row_frame,
            text=button["text"],
            command=button["command"],
            bg=button["bg"],
            fg="white",
            font=("Arial", 10, "bold"),
            width=25,
            height=2
        ).pack(side=tk.LEFT, padx=10)

root.mainloop()
```

Loading Dataset



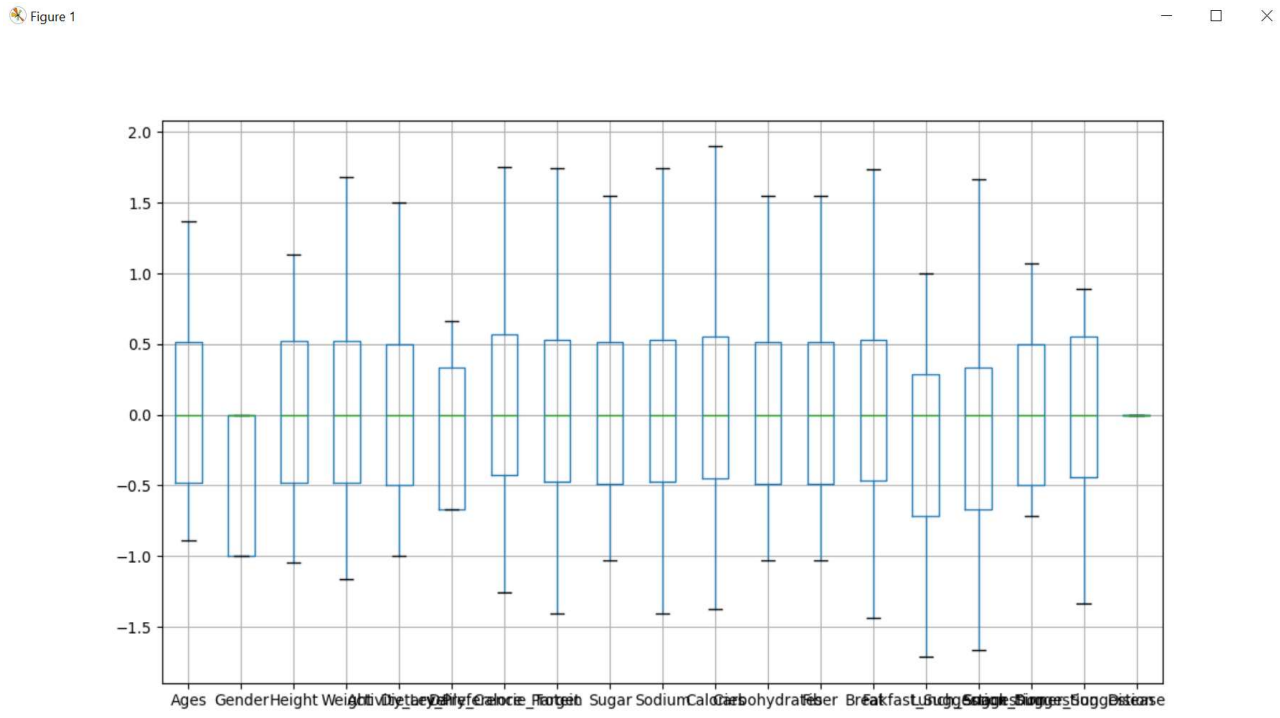


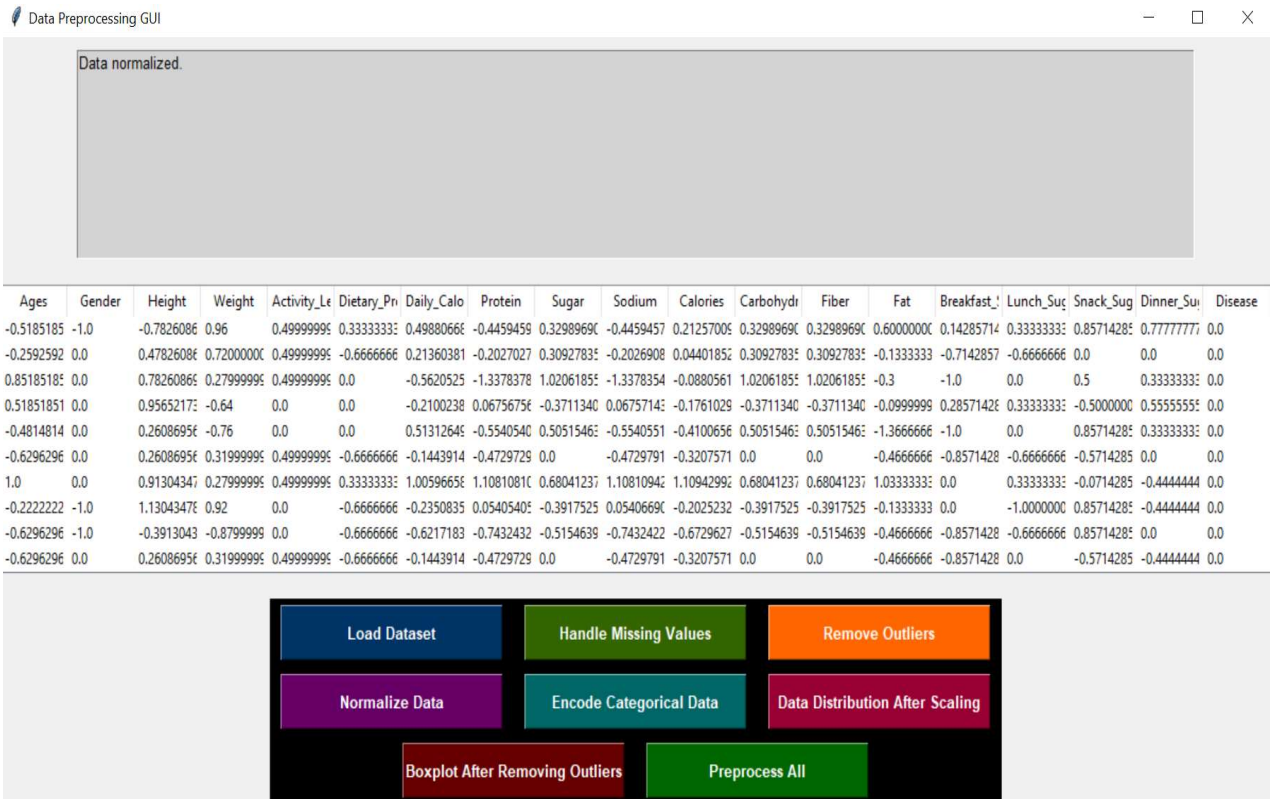
Ages	Gender	Height	Weight	Activity_Le	Dietary_Pr	Daily_Calo	Protein	Sugar	Sodium	Calories	Carbohydr	Fiber	Fat	Breakfast_	Lunch_Sug	Snack_Sug	Dinner_Su	Disease
25	Male	180	80	Moderatel	Omnivore	2000.0	120.0	125.0	24.0	2020.0	250.0	30.0	60	Oatmeal	Grilled	Greek	Salmon	Weight
32	Female	165	65	Lightly	Vegetarian	1600.0	80.0	100.0	16.0	1480.0	200.0	24.0	40	Tofu	Lentil	Apple	Vegetable	Weight
48	Male	175	95	Sedentary	Vegan	2200.0	100.0	150.0	20.0	2185.0	300.0	36.0	65	Veggie	Black	Trail	Lentil	Weight
55	Female	160	70	Intense	Omnivore	2500.0	140.0	175.0	28.0	2680.0	350.0	42.0	80	Greek	Chicken	Banana	Turkey	Weight
62	Male	170	85	Sedentary	Vegetarian	2000.0	80.0	125.0	16.0	1815.0	250.0	30.0	55	Scrambled	Quinoa	Fruit	Vegetarian	Weight
58	Female	155	60	Lightly	Vegan	1800.0	70.0	110.0	nan	1565.0	220.0	26.4	45	Tofu	Lentil	Hummus	Vegan	Weight
28	Male	190	100	Moderatel	Omnivore	3000.0	180.0	175.0	36.0	2840.0	350.0	42.0	80	Pancakes	Chicken	Protein	Steak	Weight
35	Female	170	75	Intense	Vegetarian	2400.0	120.0	150.0	24.0	2220.0	300.0	36.0	60	Tofu	Black	Cottage	Quinoa	Weight
42	Male	185	110	Moderatel	Vegan	3200.0	160.0	200.0	32.0	3005.0	400.0	48.0	85	Tofu	Lentil	Energy	Vegan	Weight
58	Female	168	78	Sedentary	Vegan	1900.0	75.0	115.0	15.0	1670.0	230.0	27.6	50	Overnight	Lentil	Fruit	Vegan	Weight



Ages	Gender	Height	Weight	Activity_Le	Dietary_Pr	Daily_Calo	Protein	Sugar	Sodium	Calories	Carbohydr	Fiber	Fat	Breakfast_	Lunch_Sug	Snack_Sug	Dinner_Su	Disease
-0.5000000	-1.0	-0.8181818	0.96	0.4999999	0.3333333	0.5312131	-0.3612903	0.3200162	-0.3612836	0.2750591	0.3216080	0.3216080	0.6896551	0.1666666	0.3333333	0.8571428	0.7777777	0.0
-0.2500000	0.0	0.4999999	0.7200000	0.4999999	-0.6666666	0.2497055	-0.1290322	0.3000152	-0.1290156	0.1028246	0.3015075	0.3015075	-0.0689655	-0.8333333	-0.6666666	0.0	0.0	0.0
0.8214285	0.0	0.8181818	0.2799999	0.4999999	0.0	-0.5159010	-1.2129032	0.9900503	-1.2128899	-0.0321359	0.9949748	0.9949748	-0.2413793	-1.1666666	0.0	0.5	0.3333333	0.0
0.4999999	0.0	0.9999999	-0.64	0.0	0.0	-0.1684334	0.1290322	-0.3600183	0.1290397	-0.1221065	-0.3618090	-0.3618090	-0.0344827	0.3333333	0.3333333	-0.5000000	0.5555555	0.0
-0.4642857	0.0	0.2727272	-0.76	0.0	0.0	0.5453474	-0.4645161	0.4900249	-0.4645101	-0.3611813	0.4924623	0.4924623	-1.3448275	-1.1666666	0.0	0.8571428	0.3333333	0.0
-0.6071428	0.0	0.2727272	0.3199999	0.4999999	-0.6666666	-0.1036513	-0.3870967	0.0	-0.3870961	-0.2699215	0.0	0.0	-0.4137931	-0.9999999	-0.6666666	-0.5714285	0.0	0.0
0.9642857	0.0	0.9545454	0.2799999	0.4999999	0.3333333	1.0318021	1.1225806	0.6600335	1.1225804	1.1915157	0.6633165	0.6633165	1.1379310	0.0	0.3333333	-0.0714285	-0.4444444	0.0
-0.2142857	-1.0	1.1818181	0.92	0.0	-0.6666666	-0.1931684	0.1161290	-0.3800193	0.1161452	-0.1491041	-0.3819095	-0.3819095	-0.0689655	0.0	-0.9999999	0.8571428	-0.4444444	0.0
-0.6071428	-1.0	-0.4090909	-0.8799999	0.0	-0.6666666	-0.5747938	-0.6451612	-0.5000254	-0.6451524	-0.6298229	-0.5025125	-0.5025125	-0.4137931	-0.9999999	-0.6666666	0.8571428	0.0	0.0
-0.6071428	0.0	0.2727272	0.3199999	0.4999999	-0.6666666	-0.1036513	-0.3870967	0.0	-0.3870961	-0.2699215	0.0	0.0	-0.4137931	-0.9999999	0.0	-0.5714285	-0.4444444	0.0

BOX PLOT AFTER REMOVING OUTLIERS





Ages	Gender	Height	Weight	Activity_Le	Dietary_Pr	Daily_Calo	Protein	Sugar	Sodium	Calories	Carbohydr	Fiber	Fat	Breakfast_	Lunch_Suc	Snack_Sug	Dinner_Su	Disease
-0.5185185	-1.0	-0.7826086	0.96	0.4999999	0.3333333	0.4988066	-0.4459459	0.3298969	-0.4459457	0.2125700	0.3298969	0.3298969	0.6000000	0.1428571	0.3333333	0.8571428	0.7777777	0.0
-0.2592592	0.0	0.4782608	0.7200000	0.4999999	-0.6666666	0.2136038	-0.2027027	0.3092783	-0.2026908	0.0440185	0.3092783	0.3092783	-0.1333333	-0.7142857	-0.6666666	0.0	0.0	0.0
0.8518518	0.0	0.7826086	0.2799999	0.4999999	0.0	-0.5620525	-1.3378378	1.0206185	-1.3378354	-0.0880561	1.0206185	1.0206185	-0.3	-1.0	0.0	0.5	0.3333333	0.0
0.5185185	0.0	0.9565217	-0.64	0.0	0.0	-0.2100238	0.0675675	-0.3711340	0.0675714	-0.1761029	-0.3711340	-0.3711340	-0.0999999	0.2857142	0.3333333	-0.5000000	0.5555555	0.0
-0.4814814	0.0	0.2608695	-0.76	0.0	0.0	0.5131264	-0.5540540	0.5051546	-0.5540551	-0.4100656	0.5051546	0.5051546	-1.3666666	-1.0	0.0	0.8571428	0.3333333	0.0
-0.6296296	0.0	0.2608695	0.3199999	0.4999999	-0.6666666	-0.1443914	-0.4729729	0.0	-0.4729791	-0.3207571	0.0	0.0	-0.4666666	-0.8571428	-0.6666666	-0.5714285	0.0	0.0
1.0	0.0	0.9130434	0.2799999	0.4999999	0.3333333	1.0059665	1.1081081	0.6804123	1.1081094	1.1094299	0.6804123	0.6804123	1.0333333	0.0	0.3333333	-0.0714285	-0.4444444	0.0
-0.2222222	-1.0	1.1304347	0.92	0.0	-0.6666666	-0.2350835	0.0540540	-0.3917525	0.0540669	-0.2025232	-0.3917525	-0.3917525	-0.1333333	0.0	-1.0000000	0.8571428	-0.4444444	0.0
-0.6296296	-1.0	-0.3913043	-0.8799999	0.0	-0.6666666	-0.6217183	-0.7432432	-0.5154639	-0.7432422	-0.6729627	-0.5154639	-0.5154639	-0.4666666	-0.8571428	-0.6666666	0.8571428	0.0	0.0
-0.6296296	0.0	0.2608695	0.3199999	0.4999999	-0.6666666	-0.1443914	-0.4729729	0.0	-0.4729791	-0.3207571	0.0	0.0	-0.4666666	-0.8571428	0.0	-0.5714285	-0.4444444	0.0