A Predictive Analysis of Food Nutrient Density Code Document

DSC 680

Weeks 1-4

Applied Data Science Project Weeks 1-4

David Berberena

9/22/2024

Library and Dataset Importation

```
# I will import the necessary libraries needed for data mining,
exploratory data analysis, and data preparation here.

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor, StackingRegressor
from sklearn.base import BaseEstimator, RegressorMixin
import statsmodels.api as sm
import warnings

warnings.filterwarnings('ignore')

# I will read in each dataset and display them individually using the
head() function.

nutrition_1 = pd.read_csv('FOOD-DATA-GROUP1.csv')

nutrition_1.head()
```

```
   Unnamed: 0.1  Unnamed: 0                               food  Caloric
Value  \
0             0           0                       cream cheese
51
1             1           1                  neufchatel cheese
215
2             2           2  requeijao cremoso light catupiry
49
3             3           3                     ricotta cheese
30
4             4           4               cream cheese low fat
30

    Fat  Saturated Fats  Monounsaturated Fats  Polyunsaturated Fats  \
0   5.0             2.9                   1.3                 0.200
1  19.4            10.9                   4.9                 0.800
2   3.6             2.3                   0.9                 0.000
3   2.0             1.3                   0.5                 0.002
4   2.3             1.4                   0.6                 0.042

   Carbohydrates  Sugars  ...  Calcium  Copper   Iron  Magnesium
Manganese  \
0            0.8   0.500  ...    0.008  14.100  0.082      0.027
1.300
1            3.1   2.700  ...   99.500   0.034  0.100      8.500
0.088
2            0.9   3.400  ...    0.000   0.000  0.000      0.000
0.000
3            1.5   0.091  ...    0.097  41.200  0.097      0.096
4.000
4            1.2   0.900  ...   22.200   0.072  0.008      1.200
0.098

   Phosphorus  Potassium  Selenium   Zinc  Nutrition Density
0       0.091       15.5    19.100  0.039              7.070
1     117.300      129.2     0.054  0.700            130.100
2       0.000        0.0     0.000  0.000              5.400
3       0.024       30.8    43.800  0.035              5.196
4      22.800       37.1     0.034  0.053             27.007

[5 rows x 37 columns]
```

```python
# The second dataset:

nutrition_2 = pd.read_csv('FOOD-DATA-GROUP2.csv')

nutrition_2.head()
```

```
   Unnamed: 0.1  Unnamed: 0                 food  Caloric Value   Fat
\
```

```
0             0            0                  eggnog            224  10.6

1             1            1              beer light             96   0.0

2             2            2          beer budweiser             12   0.0

3             3            3   weizenbier erdinger            220  18.0

4             4            4  beer light budweiser              9   0.0


   Saturated Fats  Monounsaturated Fats  Polyunsaturated Fats
Carbohydrates  \
0             6.6                   3.3                   0.5
20.4
1             0.0                   0.0                   0.0
5.4
2             0.0                   0.0                   0.0
0.9
3            13.0                   1.0                   0.0
0.0
4             0.0                   0.0                   0.0
0.4

   Sugars  ...  Calcium  Copper   Iron  Magnesium  Manganese
Phosphorus  \
0    20.4  ...    330.2   0.051  0.500       48.3      0.024
276.9
1     0.3  ...     13.2   0.095  0.014       16.5      0.094
39.6
2     0.0  ...      1.2   0.095  0.000        2.1      0.038
3.8
3     0.0  ...      0.0   0.000  0.000        0.0      0.000
0.0
4     0.0  ...      0.9   0.088  0.000        2.1      0.007
3.2

   Potassium  Selenium   Zinc  Nutrition Density
0      419.1     0.094  1.200            377.200
1       69.3     0.077  0.044             19.456
2        9.7     0.000  0.000              2.200
3        0.0     0.000  0.000             18.000
4        7.7     0.000  0.000              1.320

[5 rows x 37 columns]

# I wish to merge the five datasets, yet I must check if each dataset
is recognized by Python as having the same columns as
# the previous dataset. I have created a simple if-else statement to
prove this by using the columns.all() function.
```

```
if nutrition_1.columns.all() == nutrition_2.columns.all():
    print("The datasets' columns are the same")
else:
    print("The datasets' columns are different")
```

```
The datasets' columns are the same
```

```
# The third dataset:

nutrition_3 = pd.read_csv('FOOD-DATA-GROUP3.csv')

nutrition_3.head()
```

```
   Unnamed: 0.1  Unnamed: 0              food  Caloric Value    Fat  \
0             0           0         nectarine             66  0.500
1             1           1    kiwifruit gold             51  0.200
2             2           2  prickly pear raw              8  0.072
3             3           3         pineapple             45  0.100
4             4           4             rowan            253  4.600

   Saturated Fats  Monounsaturated Fats  Polyunsaturated Fats
Carbohydrates  \
0           0.066                 0.100                 0.200
15.8
1           0.008                 0.099                 0.051
12.8
2           0.000                 0.000                 0.000
1.9
3           0.074                 0.001                 0.087
11.8
4           0.600                 0.000                 0.000
54.5

   Sugars  ...  Calcium  Copper   Iron  Magnesium  Manganese
Phosphorus  \
0    11.8  ...    0.081   9.000  0.100        0.4     13.500
0.002
1    10.0  ...   13.800   0.100  0.200        9.7      0.072
20.300
2     0.2  ...   34.200   0.051  0.021       13.1      0.100
2.100
3     8.9  ...    0.061  11.700  0.091        0.3     10.800
0.800
4    32.1  ...   34.200   2.400  5.000       73.0      0.400
118.600

   Potassium  Selenium   Zinc  Nutrition Density
0       39.0   301.500  0.000             20.735
1      255.2     0.003  0.077            159.686
2       24.7     0.023  0.073             39.263
```

```
3        7.2    98.100  0.061              13.970
4      298.7     0.000  1.000             176.400

[5 rows x 37 columns]
```

```python
# Statement checking the columnd in each dataset is here.

if nutrition_2.columns.all() == nutrition_3.columns.all():
    print("The datasets' columns are the same")
else:
    print("The datasets' columns are different")
```

```
The datasets' columns are the same
```

```python
# The fourth dataset:

nutrition_4 = pd.read_csv('FOOD-DATA-GROUP4.csv')

nutrition_4.head()
```

```
   Unnamed: 0.1  Unnamed: 0                        food  Caloric Value
Fat  \
0             0           0  chocolate pudding fat free            105
0.3
1             1           1             tapioca pudding            143
4.3
2             2           2    tapioca pudding fat free            105
0.4
3             3           3                rice pudding            122
2.4
4             4           4                corn pudding            328
12.6

   Saturated Fats  Monounsaturated Fats  Polyunsaturated Fats
Carbohydrates  \
0             0.0                  0.00                 0.000
23.6
1             1.1                  2.80                 0.088
23.9
2             0.1                  0.08                 0.067
23.9
3             1.4                  0.60                 0.100
20.8
4             6.3                  3.90                 1.400
42.4

   Sugars  ...  Calcium   Copper   Iron  Magnesium  Manganese
Phosphorus  \
0    17.8  ...   44.100    0.035  1.900       17.0      0.040
61.0
1    16.4  ...   78.100    0.026  0.100        6.6      0.096
```

```
66.0
2    15.9  ...   58.200    0.004  0.100        5.6      0.023
73.9
3    13.1  ...    0.063  107.400  0.014        0.1      9.000
0.1
4    16.5  ...    0.066   97.500  0.100        1.3     37.500
0.2

   Potassium  Selenium   Zinc  Nutrition Density
0      235.0     0.052  0.300             72.400
1      101.2     0.000  0.200            108.800
2       78.4     0.000  0.200             84.500
3       92.7   141.300  0.083             27.329
4      225.0   440.000  0.069             69.795

[5 rows x 37 columns]
```

```
# Statement checking the columnd in each dataset is here.

if nutrition_3.columns.all() == nutrition_4.columns.all():
    print("The datasets' columns are the same")
else:
    print("The datasets' columns are different")
```

```
The datasets' columns are the same
```

```
# The fifth dataset:

nutrition_5 = pd.read_csv('FOOD-DATA-GROUP5.csv')

nutrition_5.head()
```

```
   Unnamed: 0.1  Unnamed: 0                    food  Caloric Value
Fat  \
0             0           0  margarine with yoghurt             88
9.8
1             1           1   sunflower seed butter             99
8.8
2             2           2            hazelnut oil            120
13.6
3             3           3        menhaden fish oil           1966
218.0
4             4           4       cod liver fish oil            123
13.6

   Saturated Fats  Monounsaturated Fats  Polyunsaturated Fats
Carbohydrates  \
0             1.9                   5.6                   2.0
0.073
1             0.7                   6.2                   1.6
3.700
```

```
2             1.0                  10.6                   1.4
0.000
3            66.3                  58.2                  74.5
0.000
4             3.1                   6.4                   3.1
0.000

   Sugars  ...  Calcium  Copper   Iron  Magnesium  Manganese
Phosphorus  \
0     0.0  ...      2.8   0.001  0.027        0.3        0.0
2.2
1     1.7  ...     10.2   0.300  0.700       49.8        0.3
106.6
2     0.0  ...      0.0   0.000  0.000        0.0        0.0
0.0
3     0.0  ...      0.0   0.000  0.000        0.0        0.0
0.0
4     0.0  ...      0.0   0.000  0.000        0.0        0.0
0.0

   Potassium  Selenium   Zinc  Nutrition Density
0        3.5     0.000  0.008             12.971
1       92.2     0.075  0.800             27.500
2        0.0     0.000  0.000             13.600
3        0.0     0.000  0.000            218.000
4        0.0     0.000  0.000             17.700

[5 rows x 37 columns]
```

```
# Statement checking the columnd in each dataset is here.

if nutrition_4.columns.all() == nutrition_5.columns.all():
    print("The datasets' columns are the same")
else:
    print("The datasets' columns are different")
```

```
The datasets' columns are the same
```

```
# Now that I have verified the columns in each dataset to be the same,
I will print each dataset's number of observations to
# ensure that when the datasets are merged, the number of observations
in the final dataset will be the same as the total
# observations of all five datasets.

nutrition_total_observations = (nutrition_1.shape[0] +
nutrition_2.shape[0]
                                + nutrition_3.shape[0]) +
(nutrition_4.shape[0] + nutrition_5.shape[0])

print('The number of food ingredients in the first dataset is',
nutrition_1.shape[0])
```

```
print('The number of food ingredients in the second dataset is', nutrition_2.shape[0])
print('The number of food ingredients in the third dataset is', nutrition_3.shape[0])
print('The number of food ingredients in the fourth dataset is', nutrition_4.shape[0])
print('The number of food ingredients in the fifth dataset is', nutrition_5.shape[0])
print('The total number of observations across all five datasets is', nutrition_total_observations)
```

```
The number of food ingredients in the first dataset is 551
The number of food ingredients in the second dataset is 319
The number of food ingredients in the third dataset is 571
The number of food ingredients in the fourth dataset is 232
The number of food ingredients in the fifth dataset is 722
The total number of observations across all five datasets is 2395
```

Data Preparation

```
# I will merge the datasets by stacking them on top of one another vertically with the pd.concat() function.

food_nutrition = pd.concat([nutrition_1, nutrition_2, nutrition_3, nutrition_4, nutrition_5], ignore_index = True)

# I will confirm that the merged dataset contains the same number of observations as the five separate datasets with an
# if-else statement.

if nutrition_total_observations == food_nutrition.shape[0]:
    print('The number of observations for both the total of all five datasets and the merged dataset are the same')
else:
    print('The number of observations are not the same')
```

```
The number of observations for both the total of all five datasets and the merged dataset are the same
```

```
# Here is the first few observations of the merged dataset using head().

food_nutrition.head()
```

```
   Unnamed: 0.1  Unnamed: 0                          food  Caloric Value  \
0             0           0                  cream cheese             51
1             1           1             neufchatel cheese            215
2             2           2  requeijao cremoso light catupiry
```

```
49
3              3           3                     ricotta cheese
30
4              4           4               cream cheese low fat
30

    Fat  Saturated Fats  Monounsaturated Fats  Polyunsaturated Fats  \
0   5.0             2.9                   1.3                 0.200
1  19.4            10.9                   4.9                 0.800
2   3.6             2.3                   0.9                 0.000
3   2.0             1.3                   0.5                 0.002
4   2.3             1.4                   0.6                 0.042

   Carbohydrates  Sugars  ...  Calcium  Copper   Iron  Magnesium
Manganese  \
0            0.8   0.500  ...    0.008  14.100  0.082      0.027
1.300
1            3.1   2.700  ...   99.500   0.034  0.100      8.500
0.088
2            0.9   3.400  ...    0.000   0.000  0.000      0.000
0.000
3            1.5   0.091  ...    0.097  41.200  0.097      0.096
4.000
4            1.2   0.900  ...   22.200   0.072  0.008      1.200
0.098

   Phosphorus  Potassium  Selenium   Zinc  Nutrition Density
0       0.091       15.5    19.100  0.039              7.070
1     117.300      129.2     0.054  0.700            130.100
2       0.000        0.0     0.000  0.000              5.400
3       0.024       30.8    43.800  0.035              5.196
4      22.800       37.1     0.034  0.053             27.007

[5 rows x 37 columns]

# There are two columns that have no relevance to the dataset as a
whole, so I will drop them using the drop() function.

food_nutrition = food_nutrition.drop(['Unnamed: 0.1', 'Unnamed: 0'],
axis = 1)

food_nutrition.head()

                               food  Caloric Value   Fat  Saturated
Fats  \
0                      cream cheese             51   5.0
2.9
1                 neufchatel cheese            215  19.4
10.9
2  requeijao cremoso light catupiry             49   3.6
```

```
2.3
3                    ricotta cheese             30   2.0
1.3
4              cream cheese low fat             30   2.3
1.4

   Monounsaturated Fats  Polyunsaturated Fats  Carbohydrates  Sugars
Protein  \
0                   1.3                 0.200            0.8   0.500
0.9
1                   4.9                 0.800            3.1   2.700
7.8
2                   0.9                 0.000            0.9   3.400
0.8
3                   0.5                 0.002            1.5   0.091
1.5
4                   0.6                 0.042            1.2   0.900
1.2

   Dietary Fiber  ...  Calcium  Copper   Iron  Magnesium  Manganese  \
0            0.0  ...    0.008  14.100  0.082      0.027      1.300
1            0.0  ...   99.500   0.034  0.100      8.500      0.088
2            0.1  ...    0.000   0.000  0.000      0.000      0.000
3            0.0  ...    0.097  41.200  0.097      0.096      4.000
4            0.0  ...   22.200   0.072  0.008      1.200      0.098

   Phosphorus  Potassium  Selenium   Zinc  Nutrition Density
0       0.091       15.5    19.100  0.039              7.070
1     117.300      129.2     0.054  0.700            130.100
2       0.000        0.0     0.000  0.000              5.400
3       0.024       30.8    43.800  0.035              5.196
4      22.800       37.1     0.034  0.053             27.007

[5 rows x 35 columns]

# I will now check the data types of each column's observations within
the dataset. I can do this using .dtypes.

food_nutrition.dtypes

food                        object
Caloric Value                int64
Fat                        float64
Saturated Fats             float64
Monounsaturated Fats       float64
Polyunsaturated Fats       float64
Carbohydrates              float64
Sugars                     float64
Protein                    float64
Dietary Fiber              float64
```

```
Cholesterol             float64
Sodium                  float64
Water                   float64
Vitamin A               float64
Vitamin B1              float64
Vitamin B11             float64
Vitamin B12             float64
Vitamin B2              float64
Vitamin B3              float64
Vitamin B5              float64
Vitamin B6              float64
Vitamin C               float64
Vitamin D               float64
Vitamin E               float64
Vitamin K               float64
Calcium                 float64
Copper                  float64
Iron                    float64
Magnesium               float64
Manganese               float64
Phosphorus              float64
Potassium               float64
Selenium                float64
Zinc                    float64
Nutrition Density       float64
dtype: object
```

```python
# To keep all of the dtypes the same (save for the food variable), I will change the dtype of the Caloric Value variable
# from the int type to the float type using the astype() function.

food_nutrition['Caloric Value'] = food_nutrition['Caloric Value'].astype(float)

food_nutrition.dtypes
```

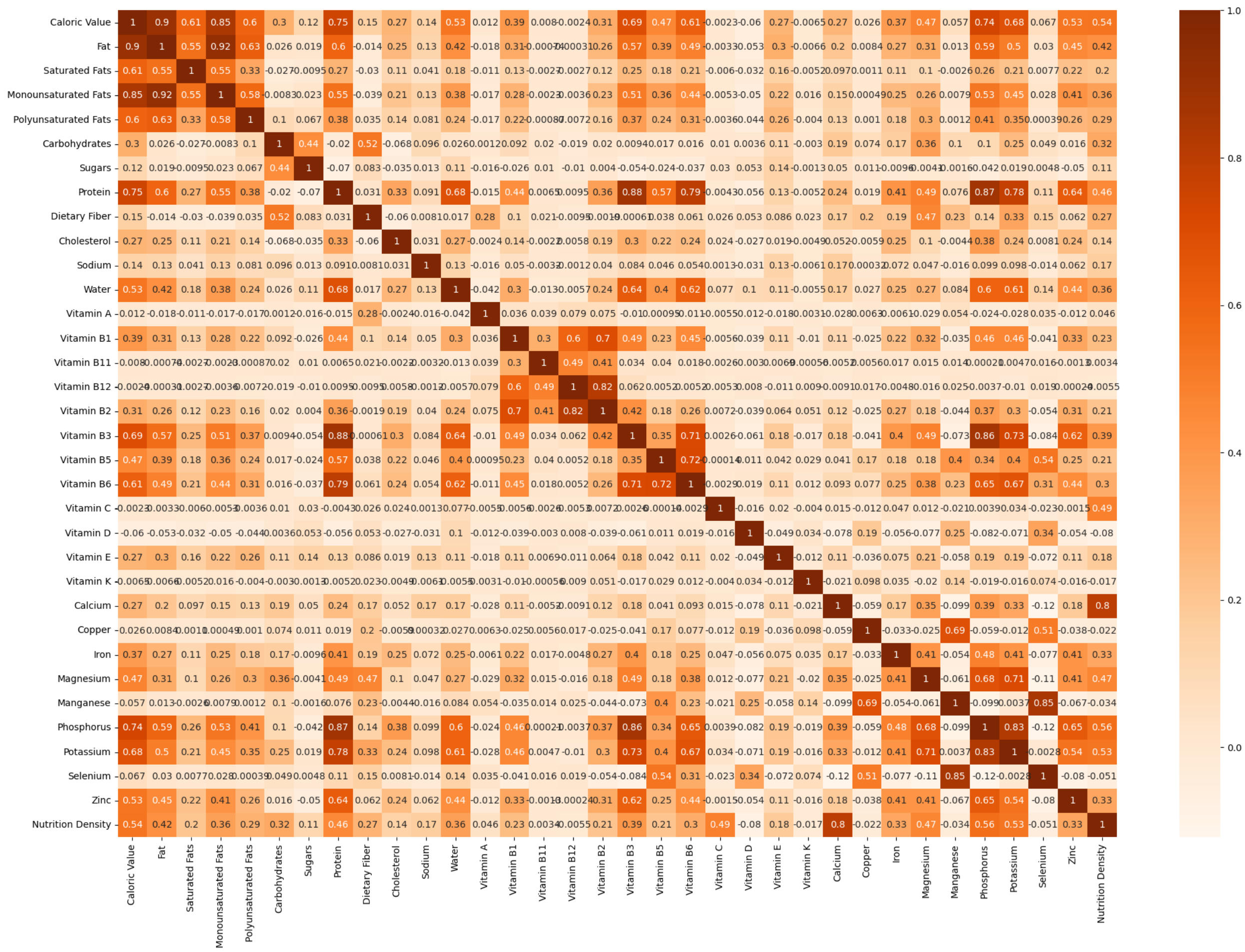
```
food                     object
Caloric Value           float64
Fat                     float64
Saturated Fats          float64
Monounsaturated Fats    float64
Polyunsaturated Fats    float64
Carbohydrates           float64
Sugars                  float64
Protein                 float64
Dietary Fiber           float64
Cholesterol             float64
Sodium                  float64
Water                   float64
Vitamin A               float64
```

```
Vitamin B1                  float64
Vitamin B11                 float64
Vitamin B12                 float64
Vitamin B2                  float64
Vitamin B3                  float64
Vitamin B5                  float64
Vitamin B6                  float64
Vitamin C                   float64
Vitamin D                   float64
Vitamin E                   float64
Vitamin K                   float64
Calcium                     float64
Copper                      float64
Iron                        float64
Magnesium                   float64
Manganese                   float64
Phosphorus                  float64
Potassium                   float64
Selenium                    float64
Zinc                        float64
Nutrition Density           float64
dtype: object

# To end the data preparation stage, I want to capitalize the food
column name and the values within that column. To do
# this, I will use both str.replace() and str.upper().

food_nutrition.columns = food_nutrition.columns.str.replace('food',
'Food')

food_nutrition['Food'] = food_nutrition['Food'].str.upper()

food_nutrition.head()

                               Food  Caloric Value   Fat  Saturated
Fats  \
0                      CREAM CHEESE           51.0   5.0
2.9
1                 NEUFCHATEL CHEESE          215.0  19.4
10.9
2  REQUEIJAO CREMOSO LIGHT CATUPIRY           49.0   3.6
2.3
3                    RICOTTA CHEESE           30.0   2.0
1.3
4              CREAM CHEESE LOW FAT           30.0   2.3
1.4

   Monounsaturated Fats  Polyunsaturated Fats  Carbohydrates  Sugars
Protein  \
0                   1.3                 0.200            0.8   0.500
```

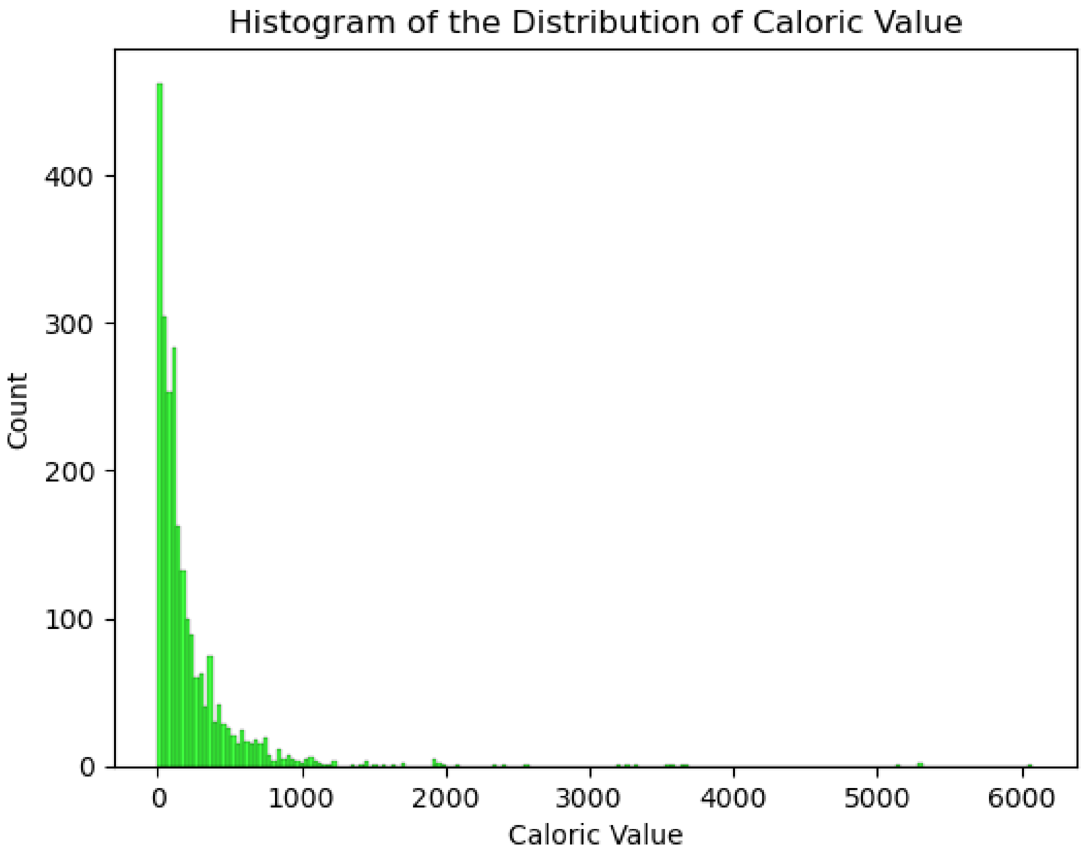
```
0.9
1                   4.9                 0.800               3.1   2.700
7.8
2                   0.9                 0.000               0.9   3.400
0.8
3                   0.5                 0.002               1.5   0.091
1.5
4                   0.6                 0.042               1.2   0.900
1.2

   Dietary Fiber  ...  Calcium  Copper   Iron  Magnesium  Manganese  \
0            0.0  ...    0.008  14.100  0.082      0.027      1.300
1            0.0  ...   99.500   0.034  0.100      8.500      0.088
2            0.1  ...    0.000   0.000  0.000      0.000      0.000
3            0.0  ...    0.097  41.200  0.097      0.096      4.000
4            0.0  ...   22.200   0.072  0.008      1.200      0.098

   Phosphorus  Potassium  Selenium   Zinc  Nutrition Density
0       0.091       15.5    19.100  0.039              7.070
1     117.300      129.2     0.054  0.700            130.100
2       0.000        0.0     0.000  0.000              5.400
3       0.024       30.8    43.800  0.035              5.196
4      22.800       37.1     0.034  0.053             27.007

[5 rows x 35 columns]
```

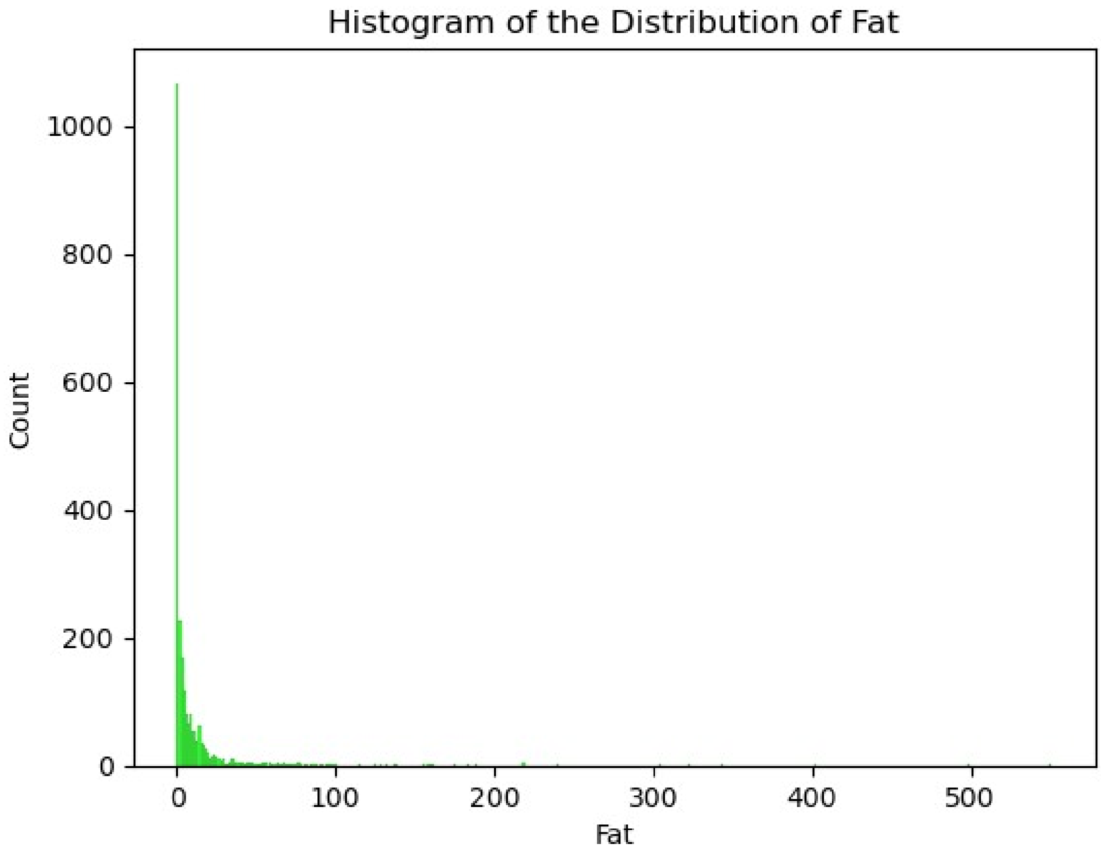
Initial Exploratory Data Analysis

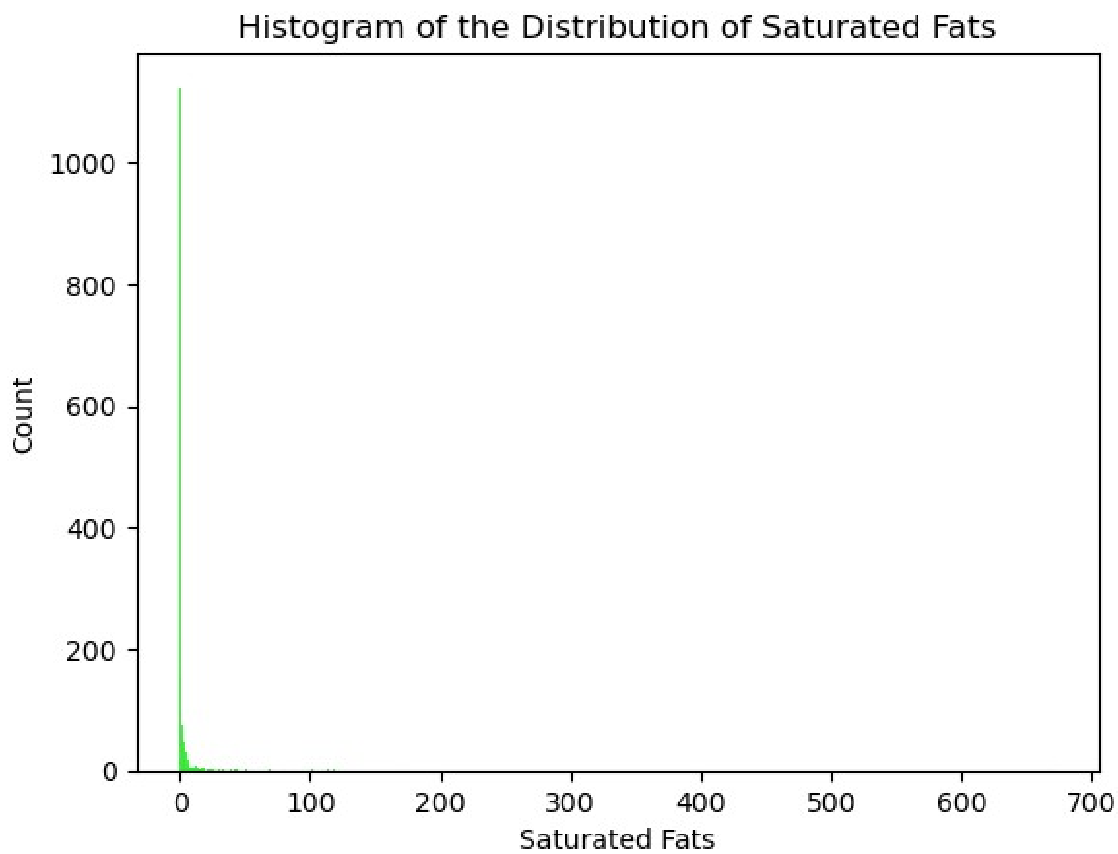
```
# To visualize the correlation between nutrient density and the other variables, I will create a heatmap using a correlation
# matrix.

food_correlation = food_nutrition.drop(['Food'], axis = 1).corr()
plt.figure(figsize = (24,16))
sns.heatmap(food_correlation, cmap = "Oranges", annot = True)
plt.show()
```

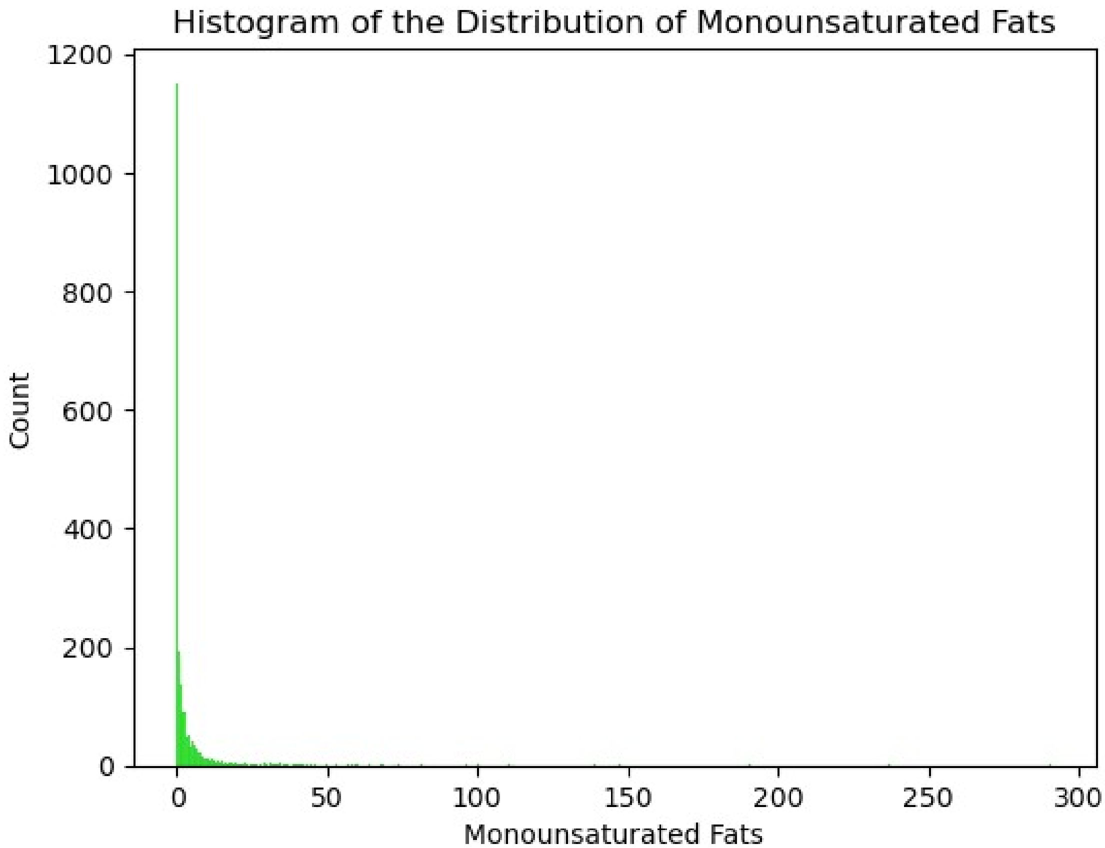



```python
# To visualize the distribution of the data, I will craft a for loop that will output a histogram for each variable except
# for the Food variable. This will help to confirm if the initial model selection of ordinary least squares is appropriate.
# Crafting the for loop involves me creating a function that outputs a histogram using Seaborn's histplot() function, which
# takes in a DataFrame and a column.

def make_nutrient_histogram(data, col):
    sns.histplot(data[col], color = 'lime')
    plt.title(f'Histogram of the Distribution of {col}')
    plt.show()

for col in food_nutrition.drop(['Food'], axis = 1).columns:
    make_nutrient_histogram(food_nutrition, col)
```

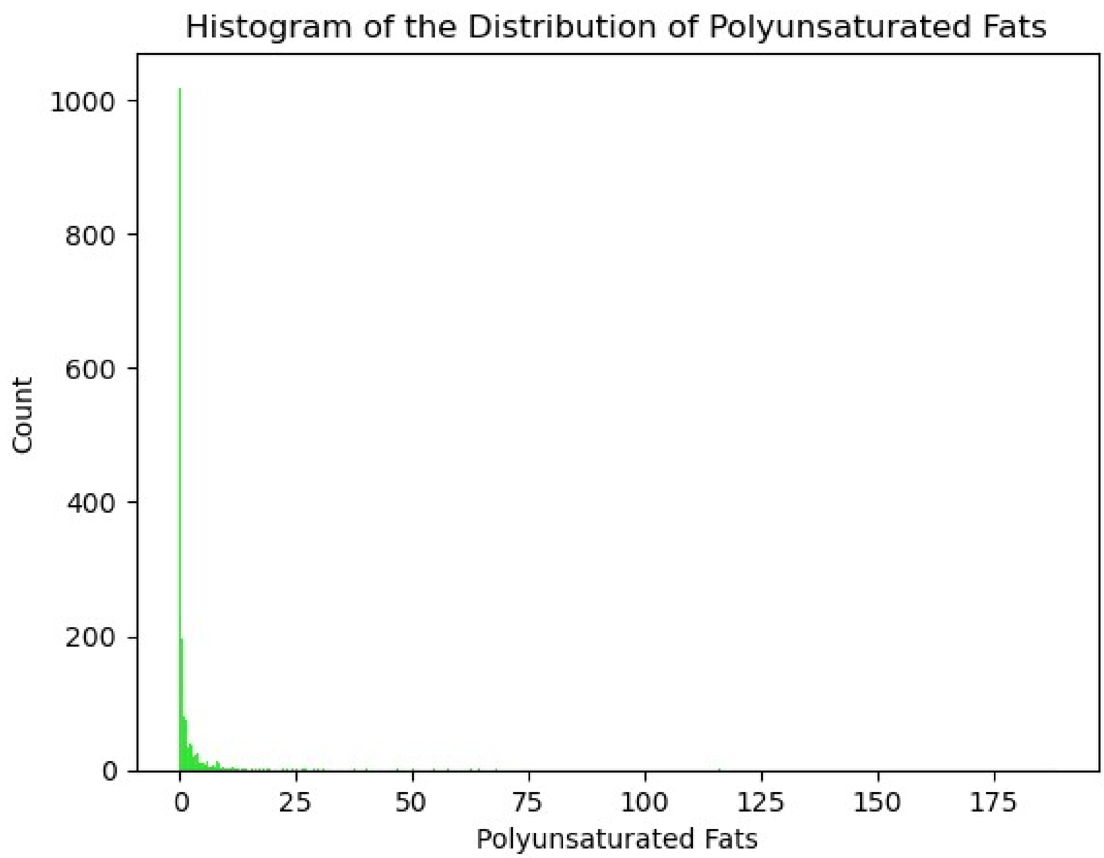

Histogram of the Distribution of Caloric Value
Count
0
100
200
300
400
0
1000
2000
3000
4000
5000
6000
Caloric Value


Histogram of the Distribution of Fat
Count
1000
800
600
400
200
0
100
200
300
400
500
Fat

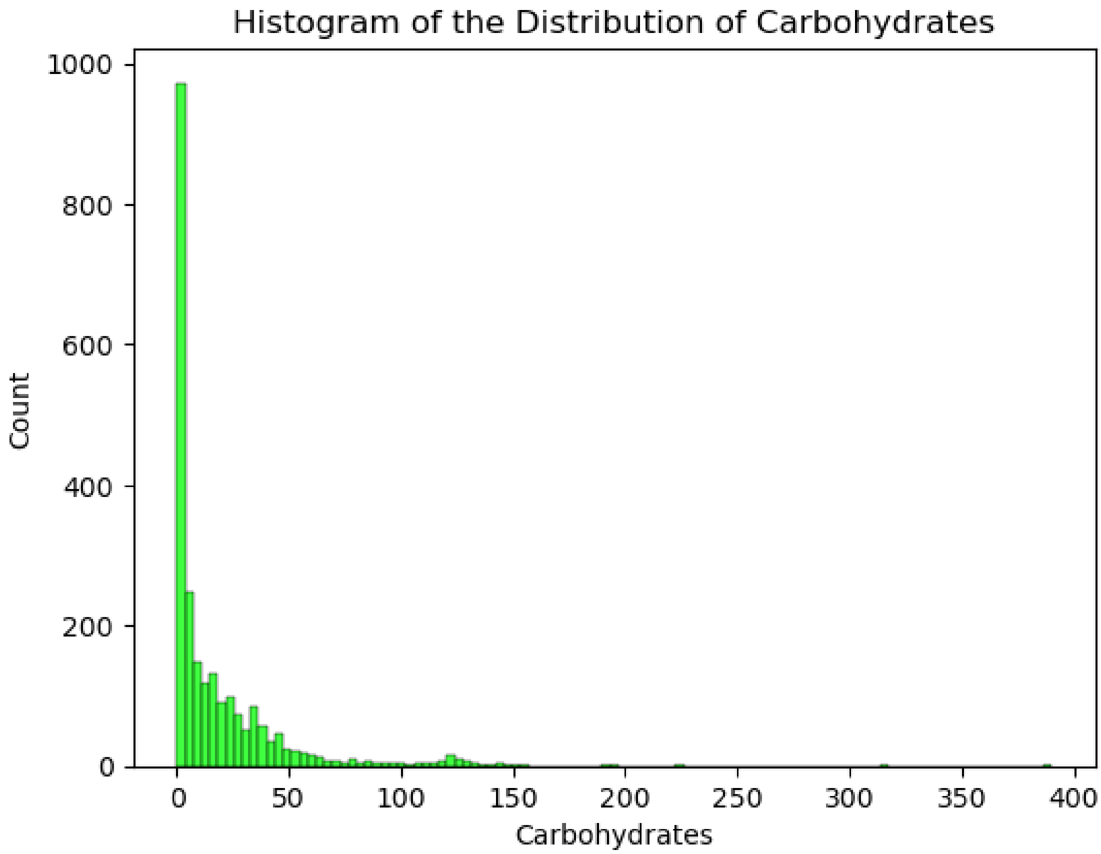

Histogram of the Distribution of Saturated Fats
Count
Saturated Fats
0
200
400
600
800
1000
0
100
200
300
400
500
600
700

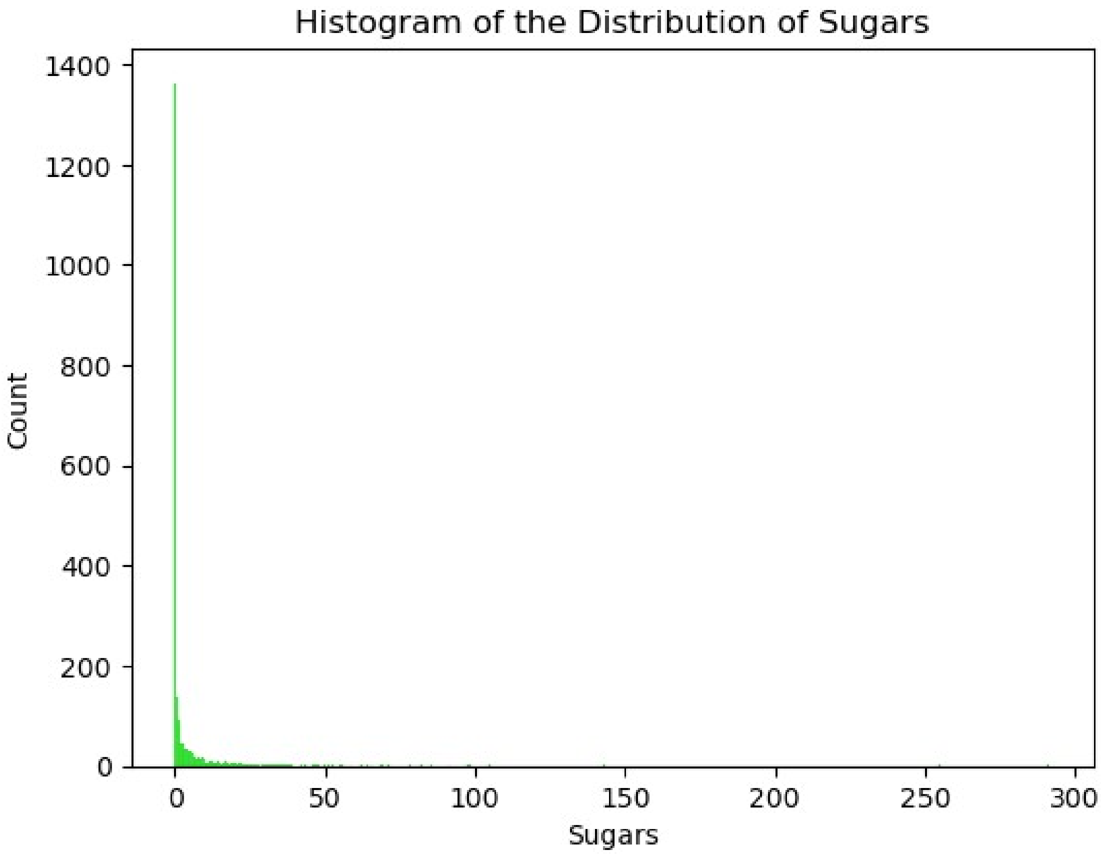


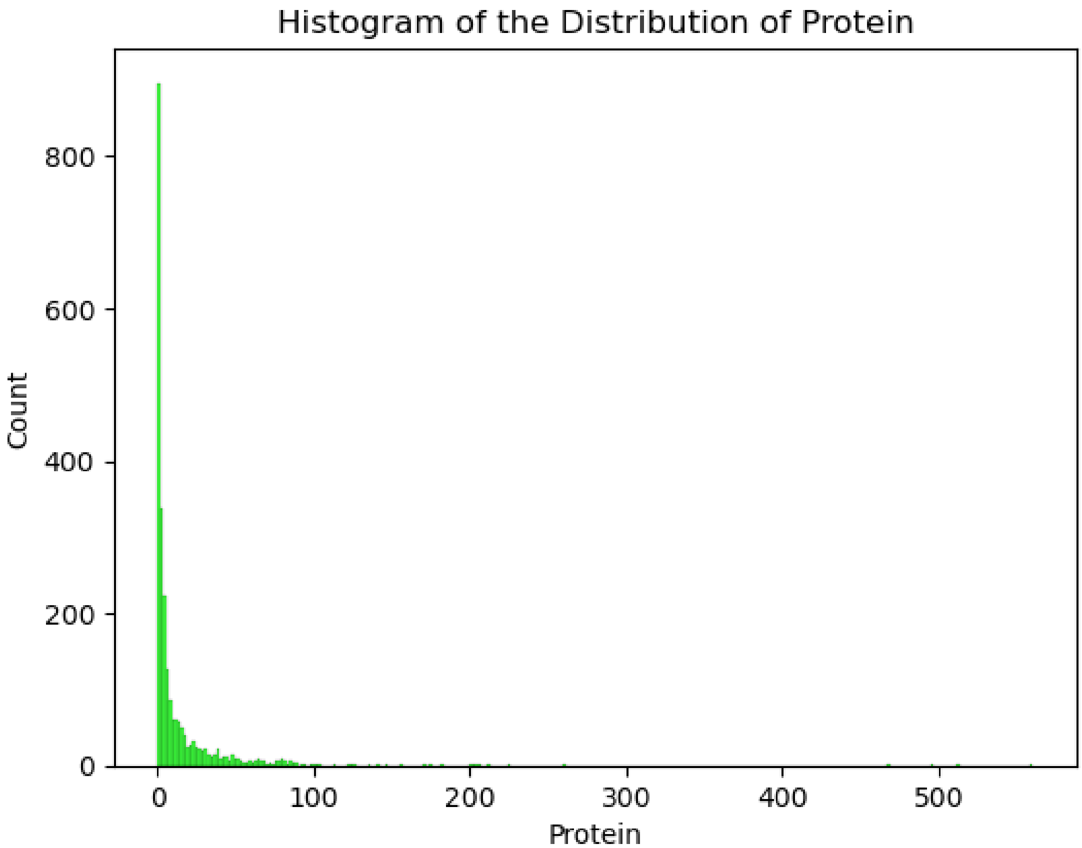
Histogram of the Distribution of Monounsaturated Fats
Count
Monounsaturated Fats
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300

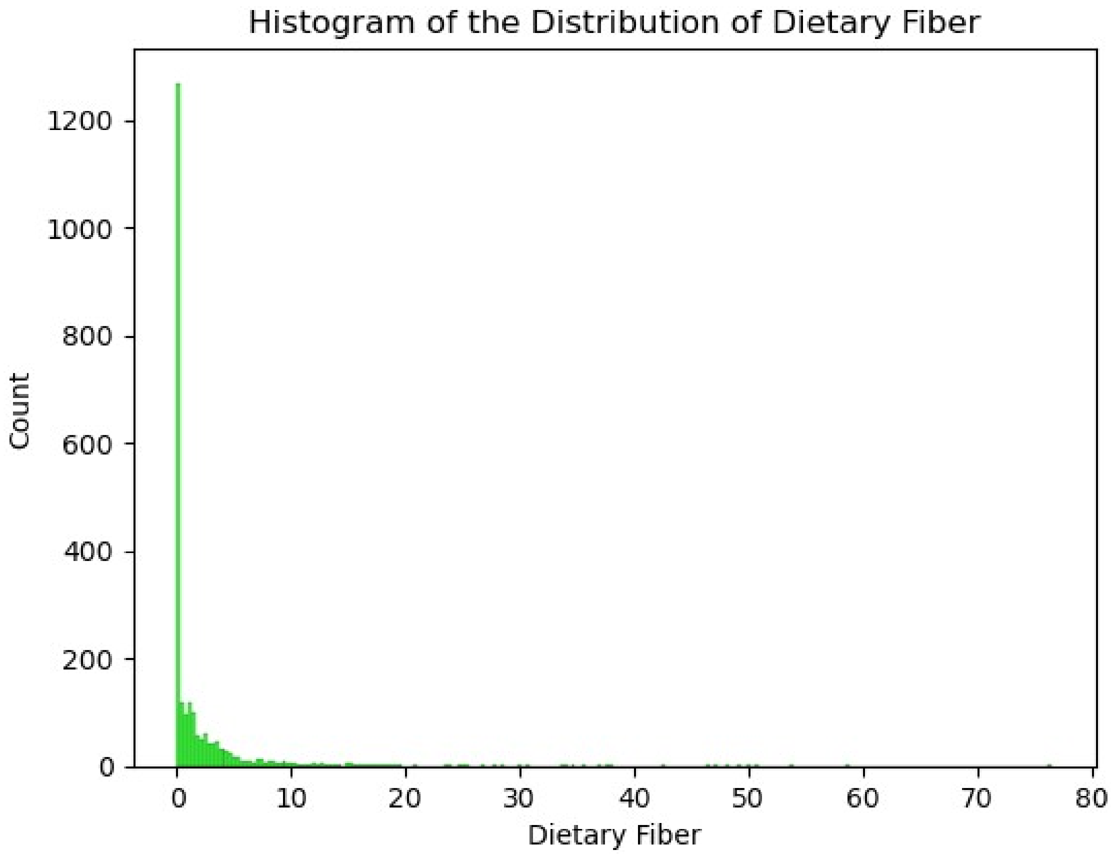


Histogram of the Distribution of Polyunsaturated Fats
1000
800
600
400
200
0
Count
0
25
50
75
100
125
150
175
Polyunsaturated Fats

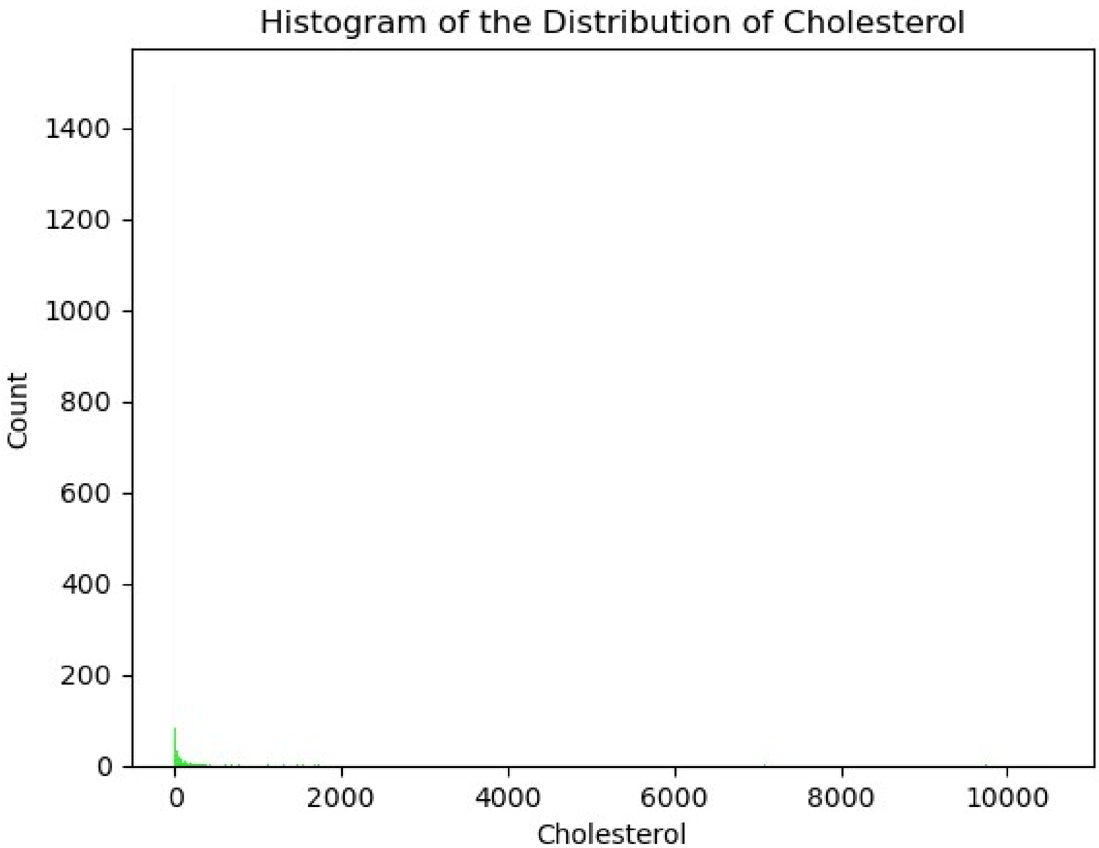

Histogram of the Distribution of Carbohydrates
Count
1000
800
600
400
200
0
0
50
100
150
200
250
300
350
400
Carbohydrates

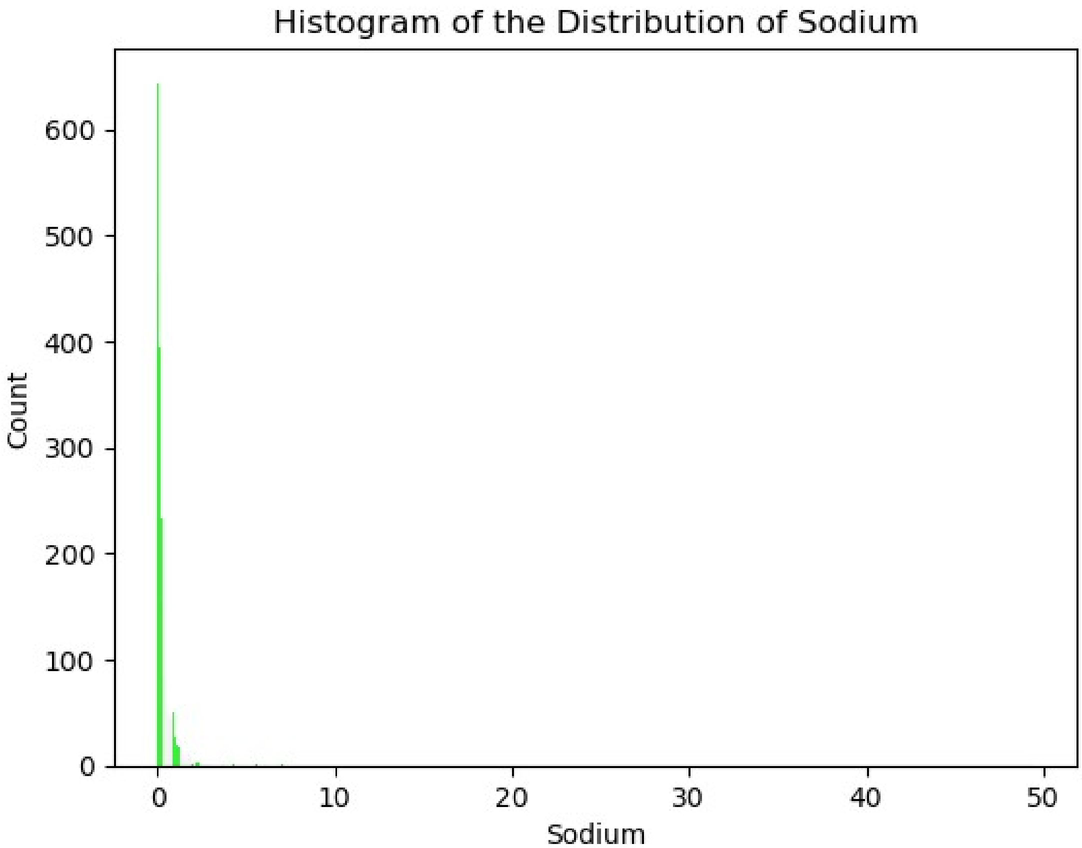

Histogram of the Distribution of Sugars
Count
0
200
400
600
800
1000
1200
1400
0
50
100
150
200
250
300
Sugars

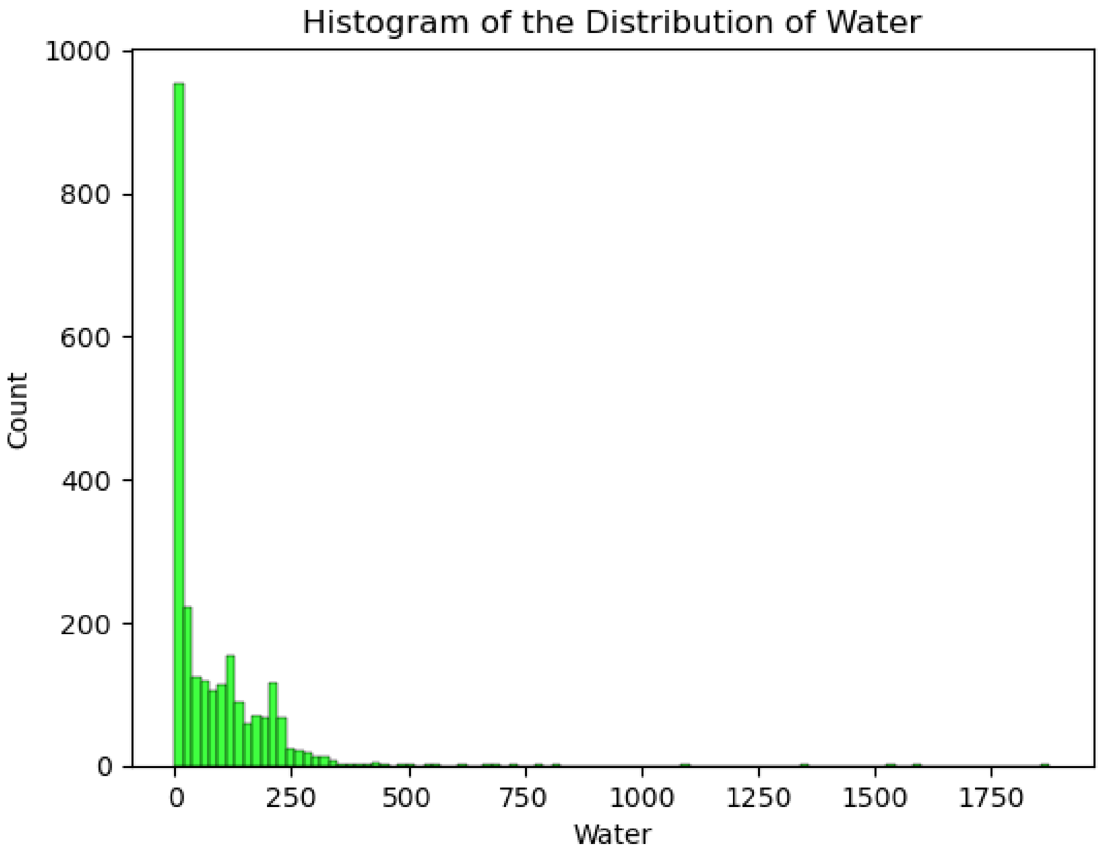

Histogram of the Distribution of Protein
Count
800
600
400
200
0
0
100
200
300
400
500
Protein



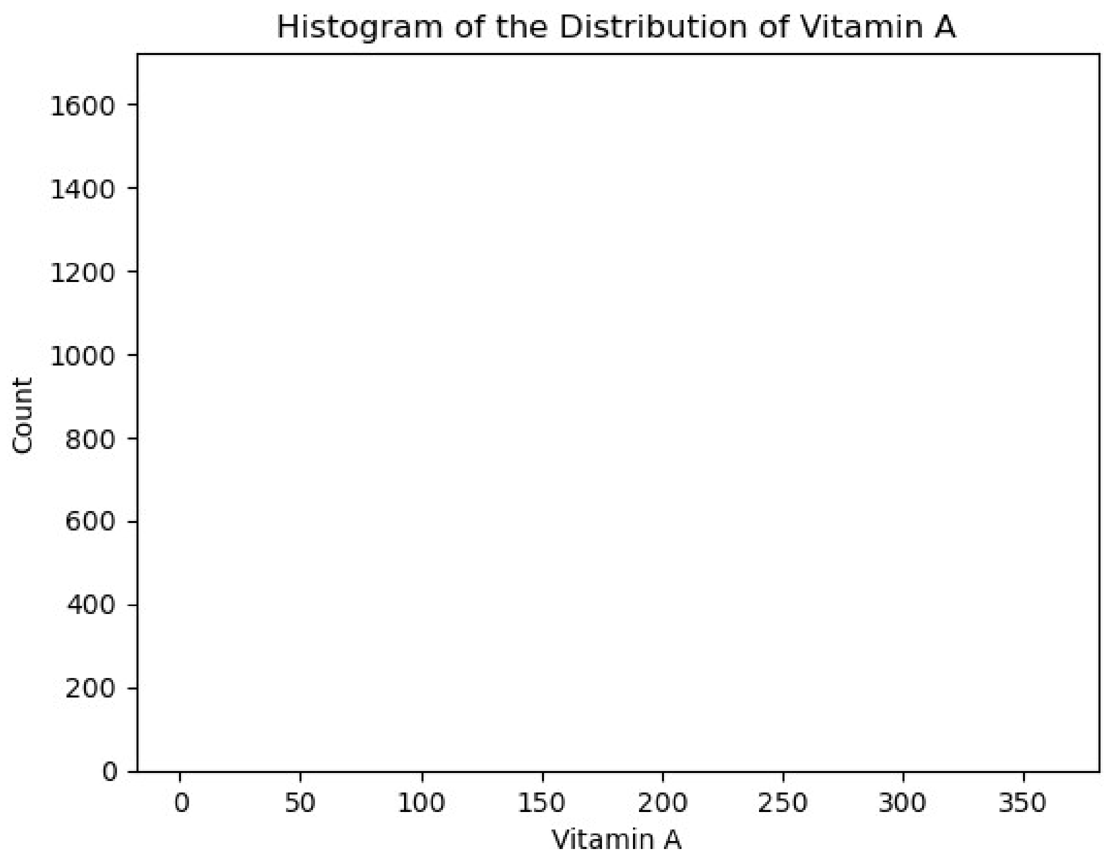
Histogram of the Distribution of Dietary Fiber
Count
0
200
400
600
800
1000
1200
0
10
20
30
40
50
60
70
80
Dietary Fiber

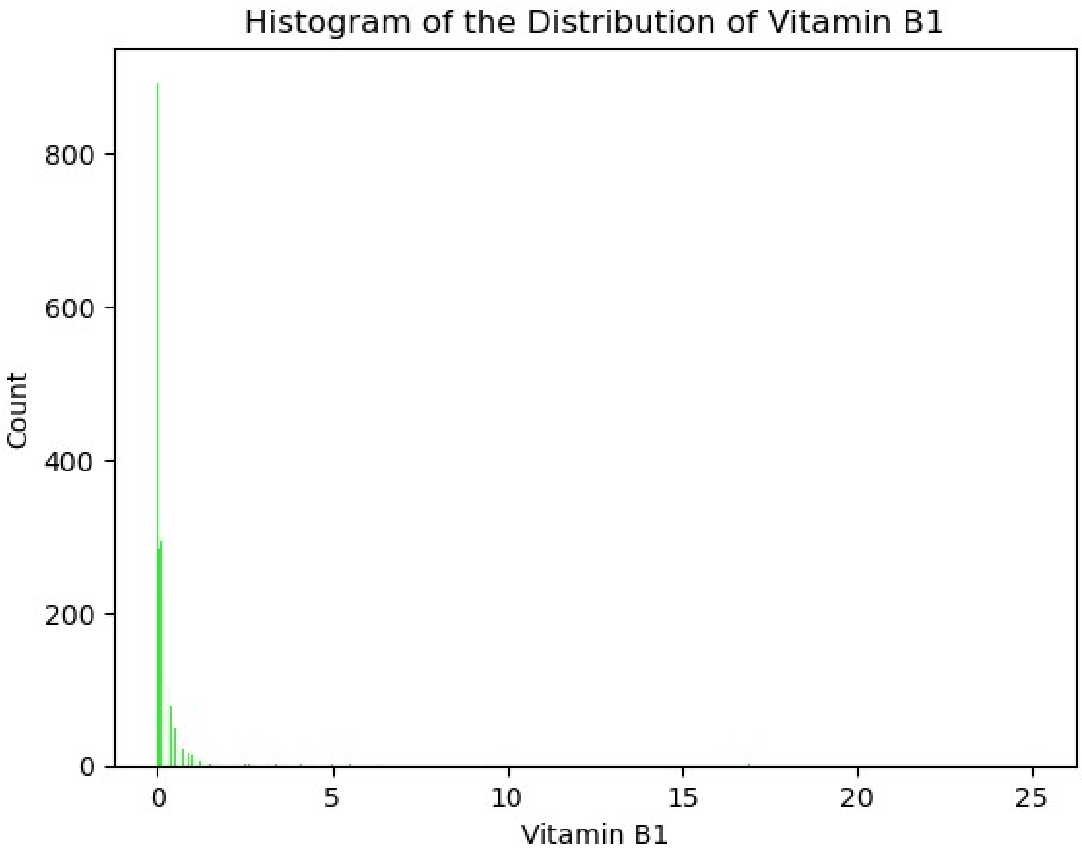

Histogram of the Distribution of Cholesterol
Count
1400
1200
1000
800
600
400
200
0
0
2000
4000
6000
8000
10000
Cholesterol

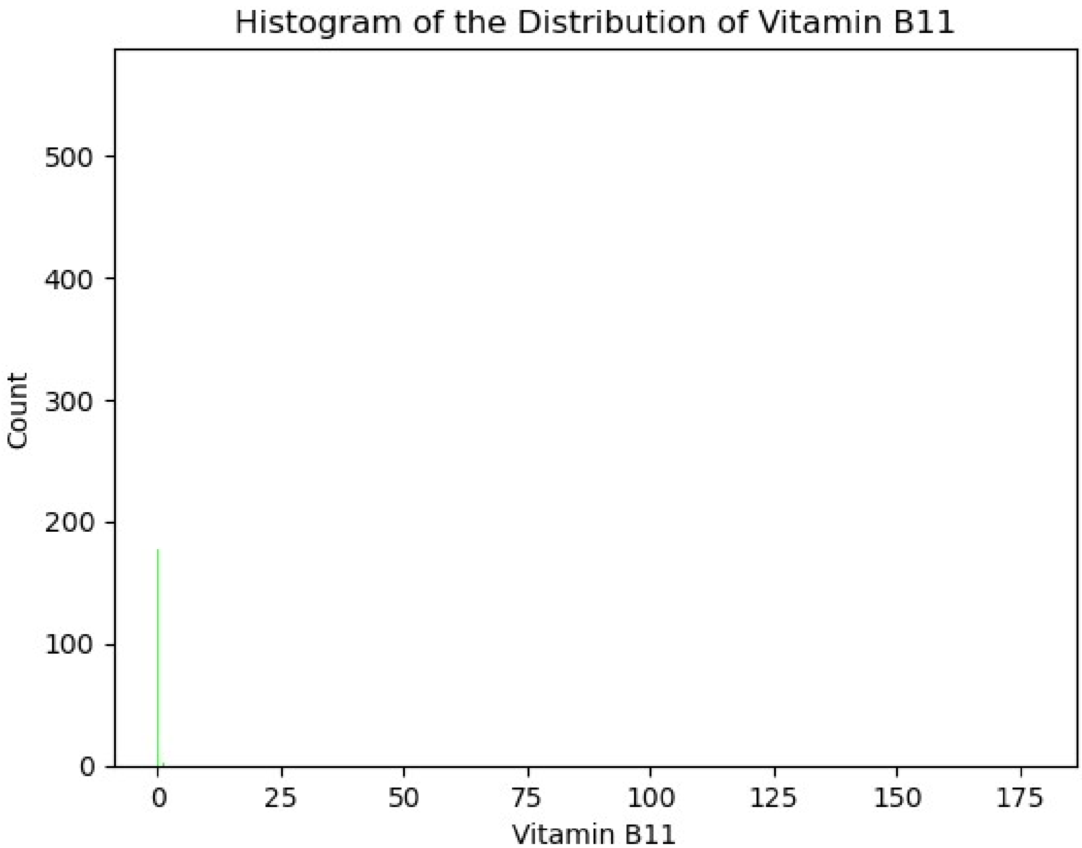

Histogram of the Distribution of Sodium
Count
600
500
400
300
200
100
0
0
10
20
30
40
50
Sodium

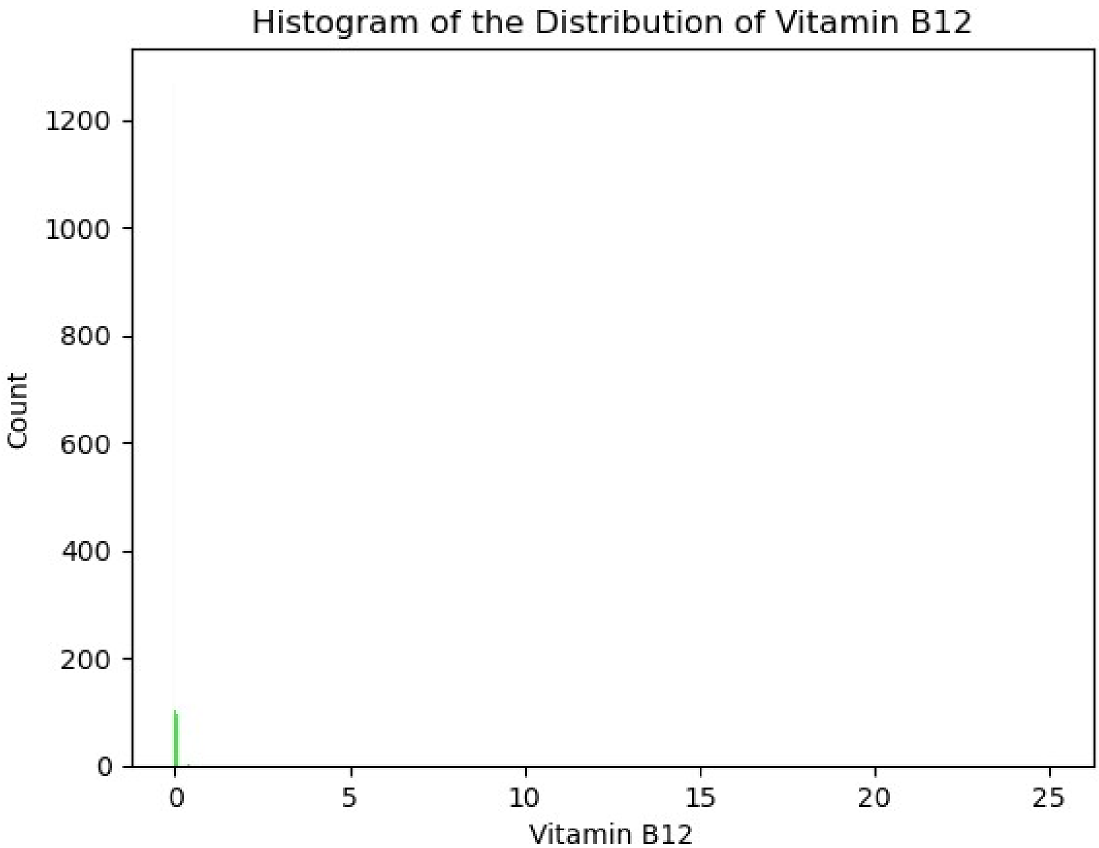


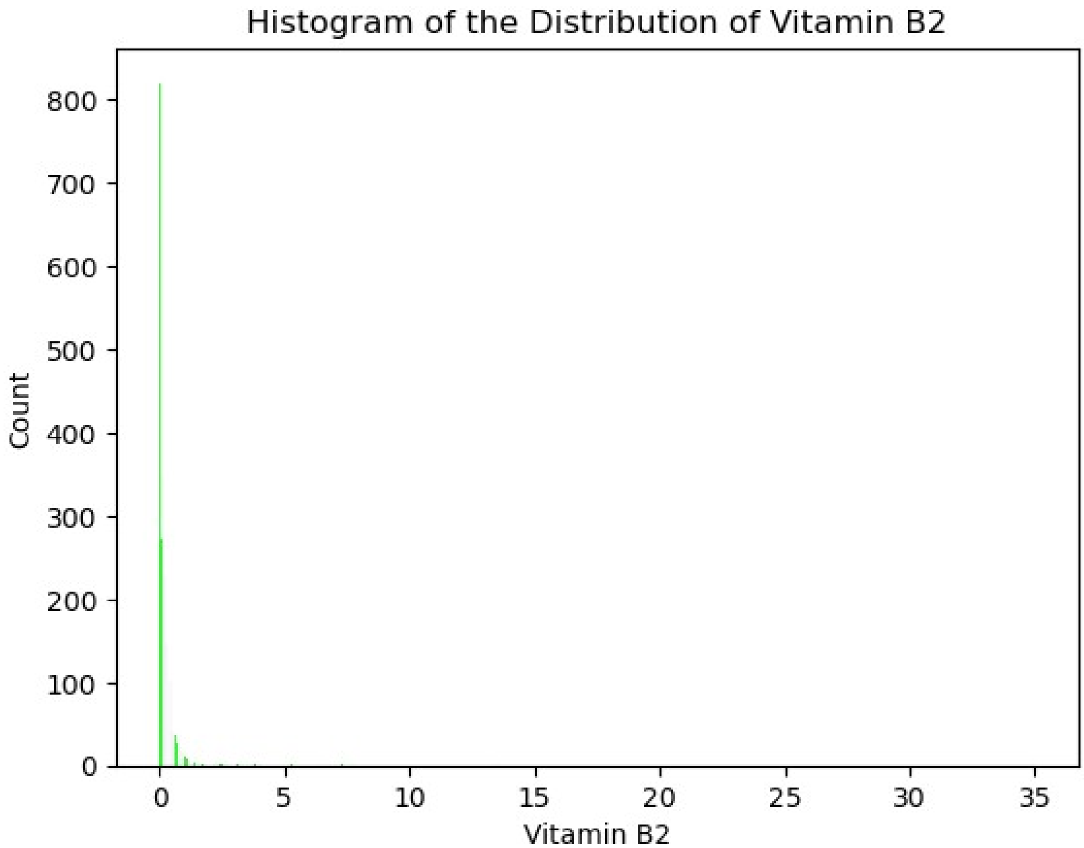
Histogram of the Distribution of Water
1000
800
600
400
200
0
Count
0
250
500
750
1000
1250
1500
1750
Water

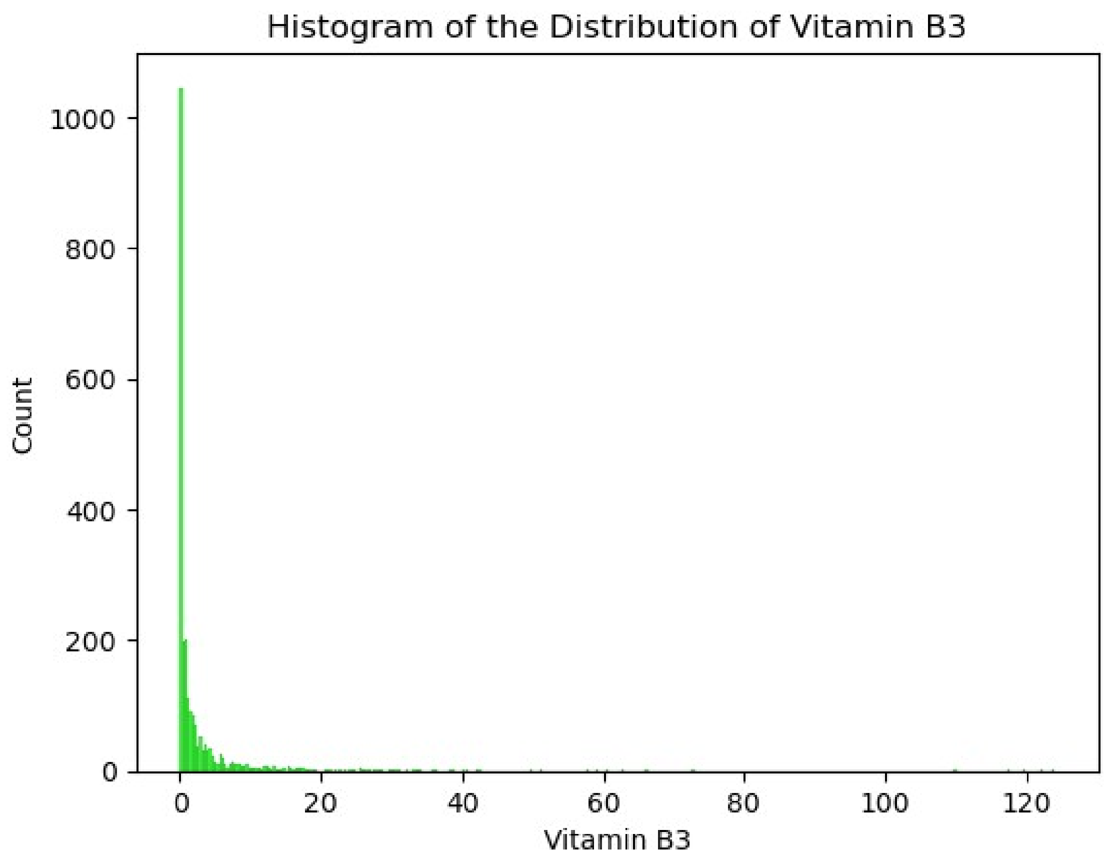

Histogram of the Distribution of Vitamin A
Count
1600
1400
1200
1000
800
600
400
200
0
0
50
100
150
200
250
300
350
Vitamin A


Histogram of the Distribution of Vitamin B1
Count
Vitamin B1
800
600
400
200
0
0
5
10
15
20
25

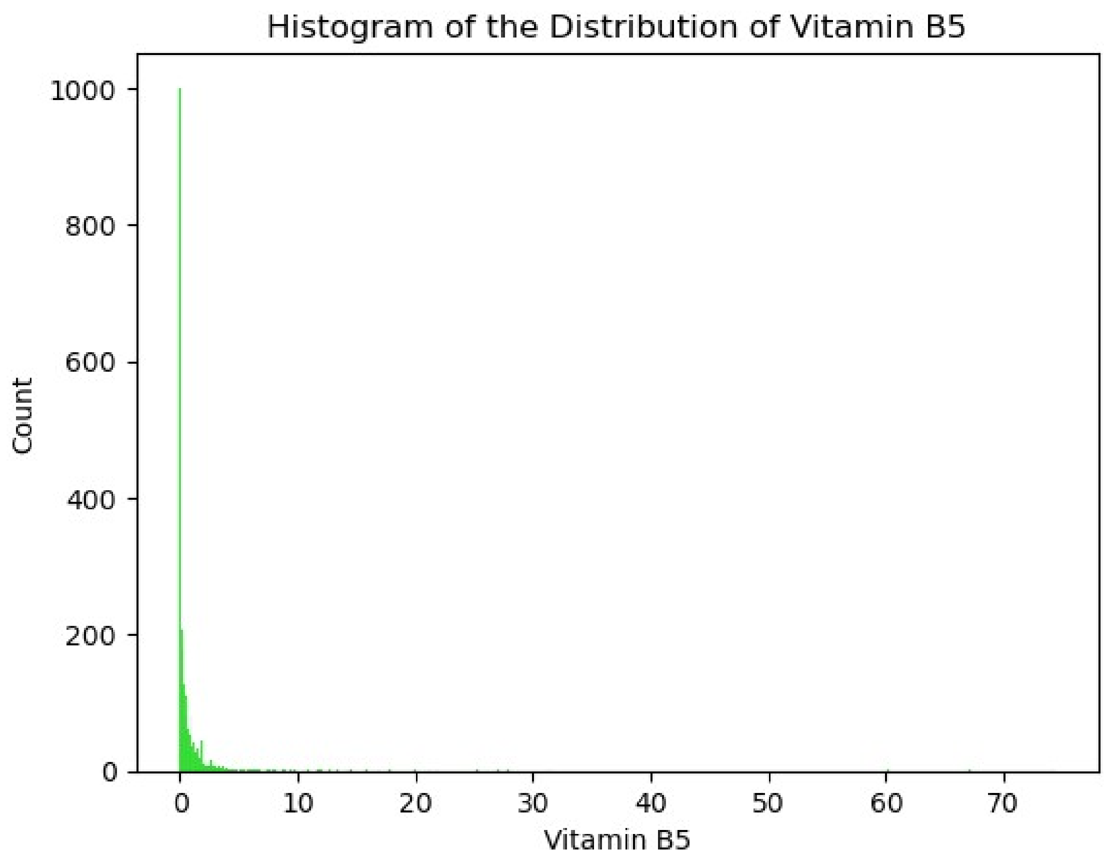

Histogram of the Distribution of Vitamin B11
Count
500
400
300
200
100
0
0
25
50
75
100
125
150
175
Vitamin B11

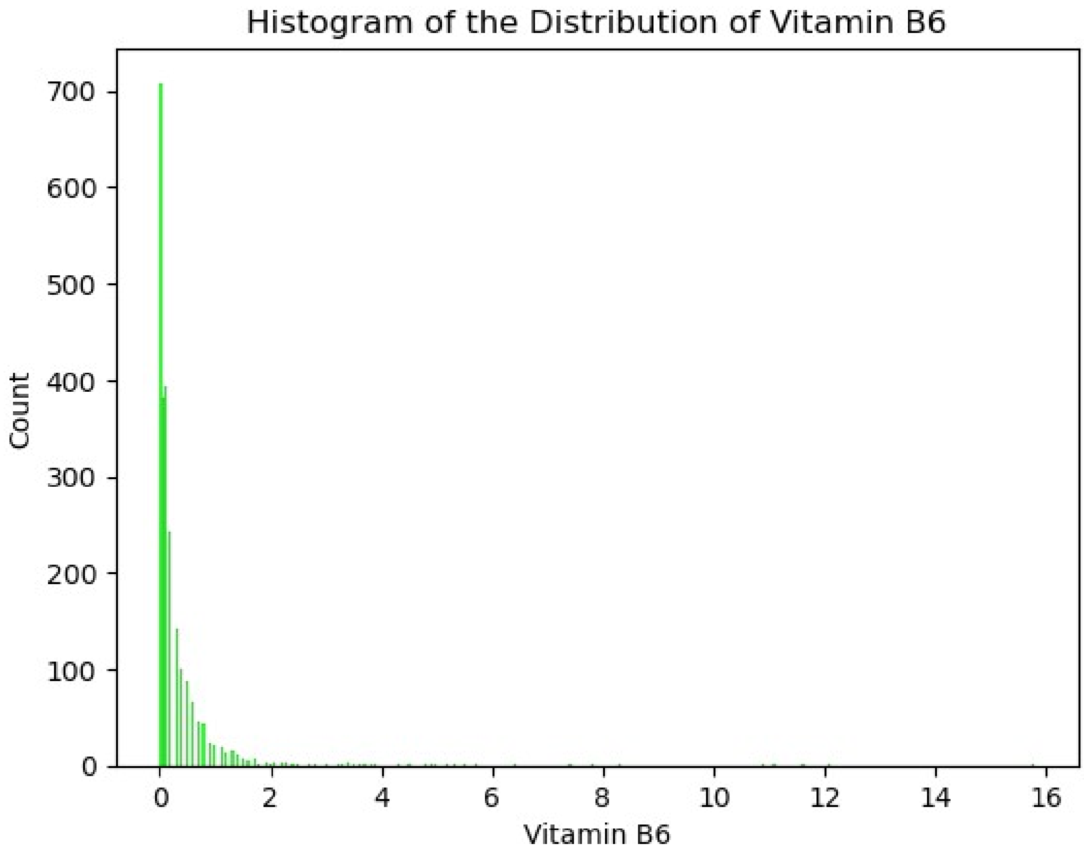

Histogram of the Distribution of Vitamin B12
Count
Vitamin B12
0
200
400
600
800
1000
1200
0
5
10
15
20
25

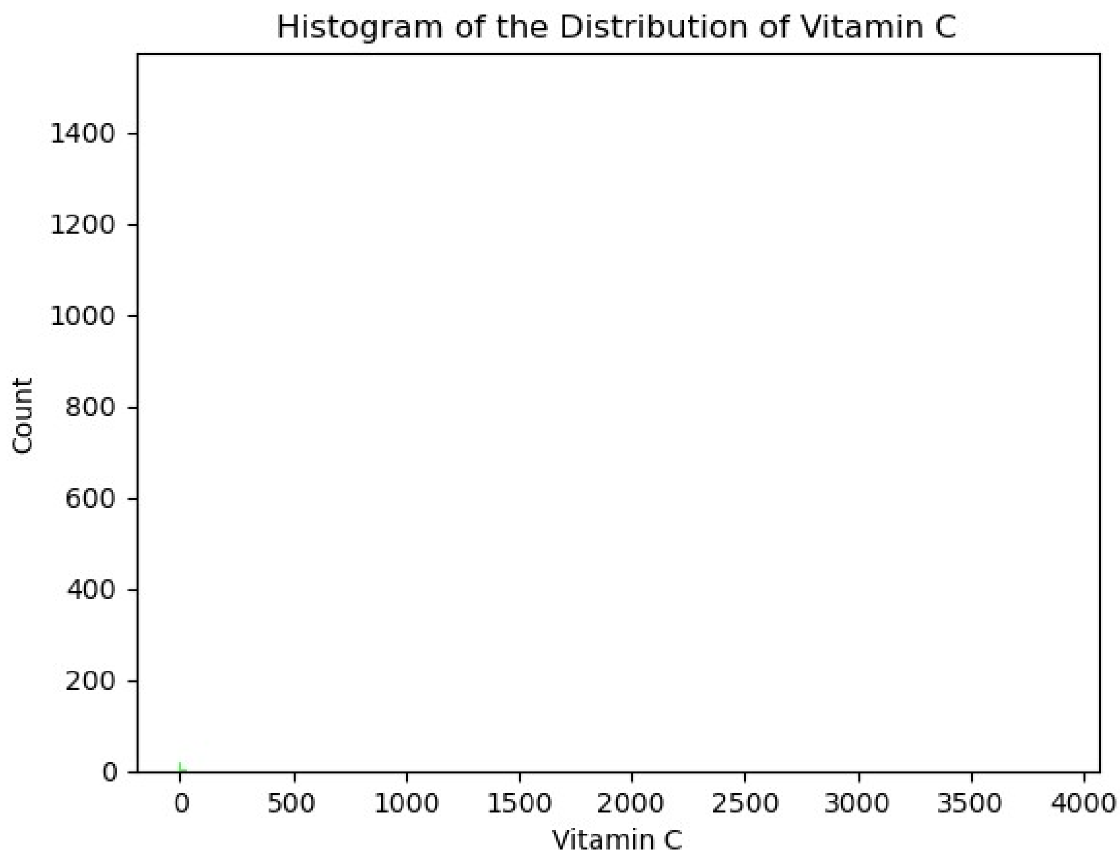

Histogram of the Distribution of Vitamin B2
Count
800
700
600
500
400
300
200
100
0
0
5
10
15
20
25
30
35
Vitamin B2

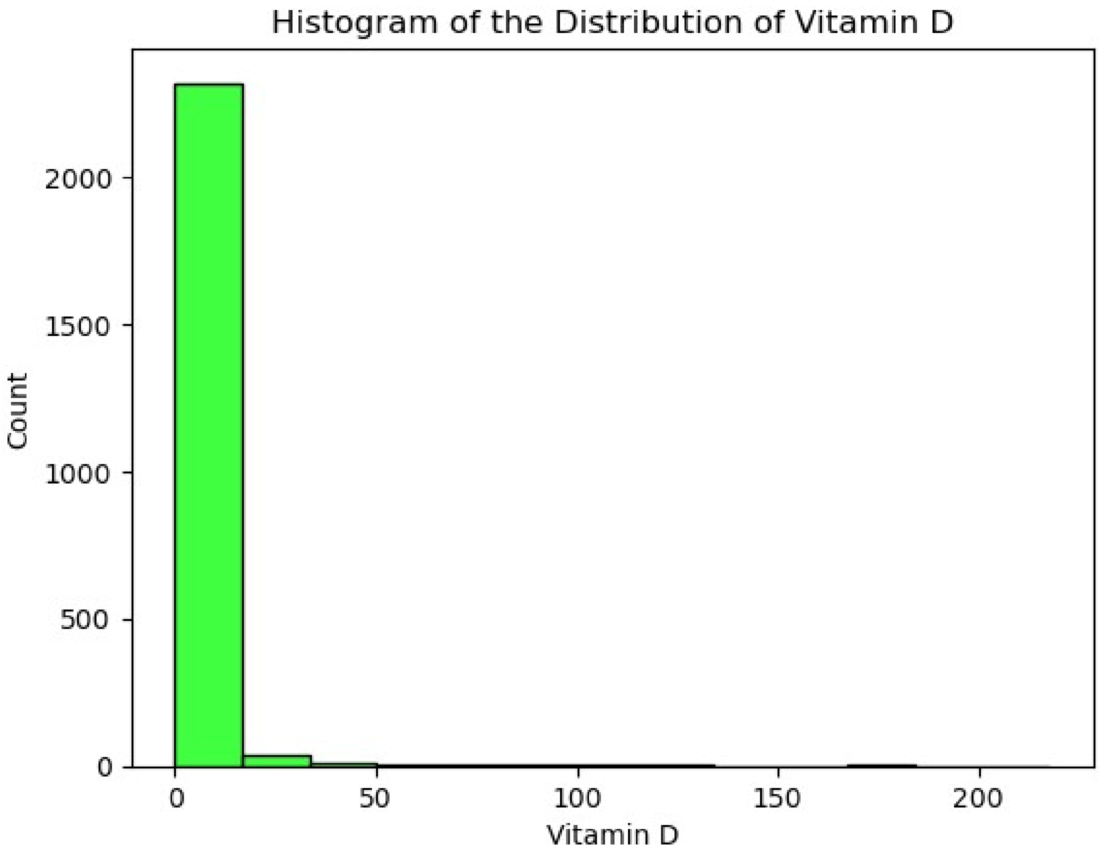


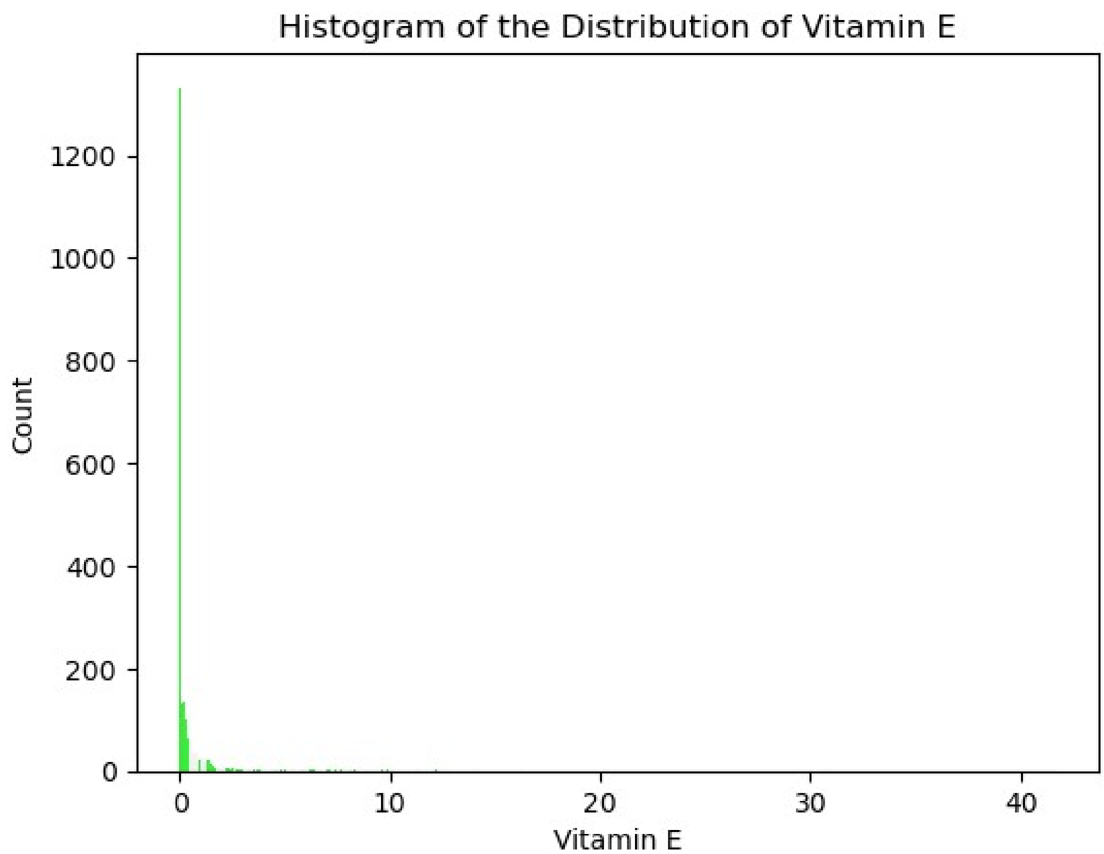
Histogram of the Distribution of Vitamin B3
Count
1000
800
600
400
200
0
0
20
40
60
80
100
120
Vitamin B3


Histogram of the Distribution of Vitamin B5
Count
1000
800
600
400
200
0
0
10
20
30
40
50
60
70
Vitamin B5

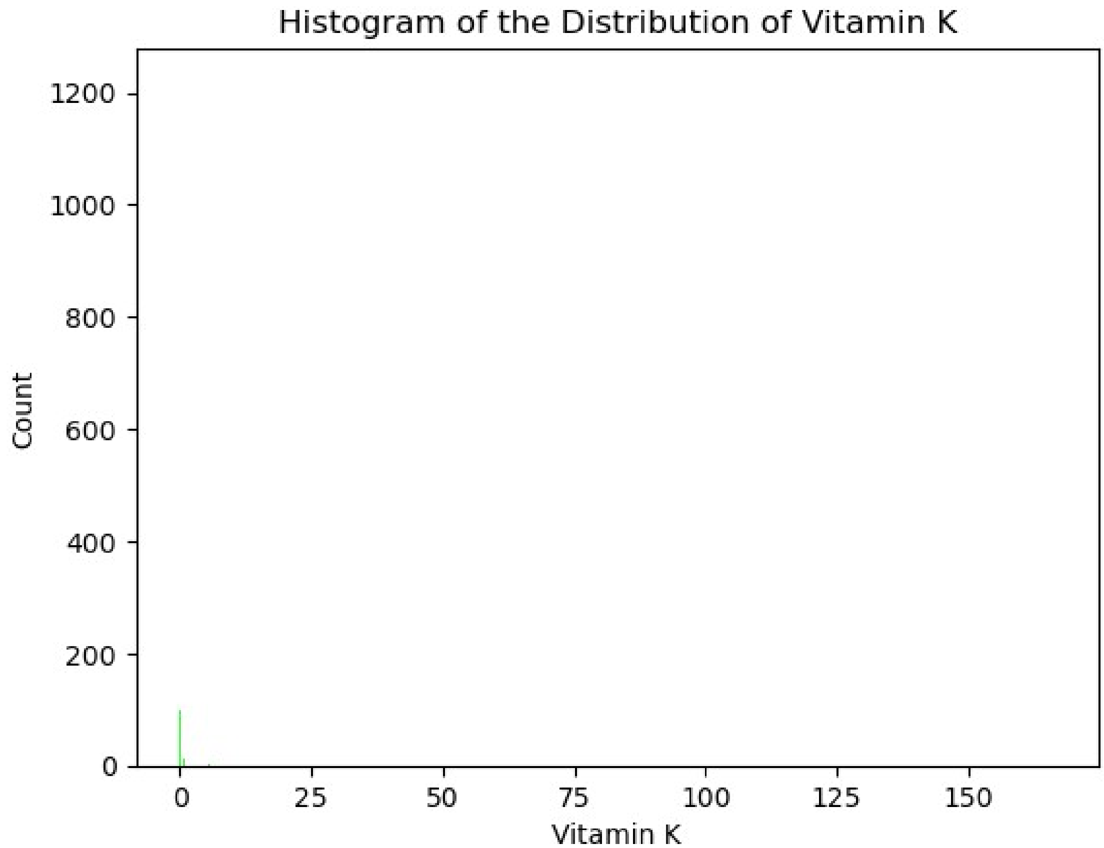

Histogram of the Distribution of Vitamin B6
Count
700
600
500
400
300
200
100
0
0
2
4
6
8
10
12
14
16
Vitamin B6

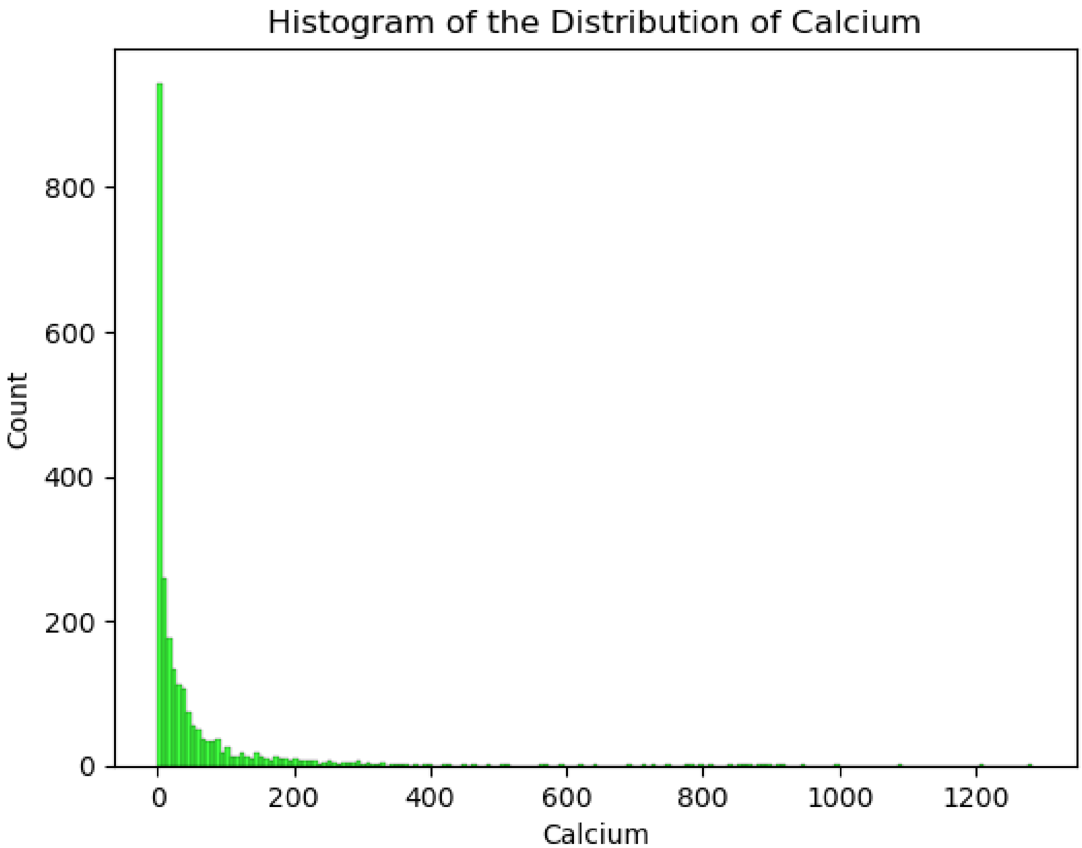

Histogram of the Distribution of Vitamin C
Count
1400
1200
1000
800
600
400
200
0
0
500
1000
1500
2000
2500
3000
3500
4000
Vitamin C

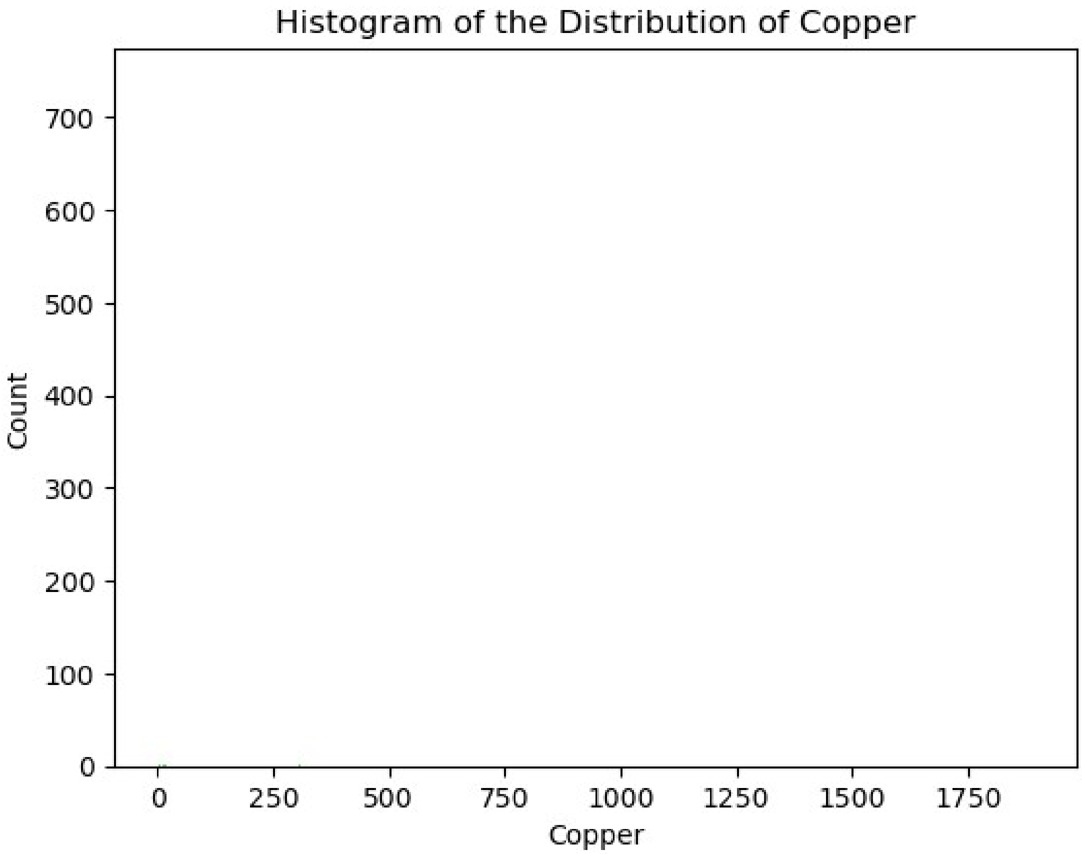

Histogram of the Distribution of Vitamin D
Count
2000
1500
1000
500
0
0
50
100
150
200
Vitamin D

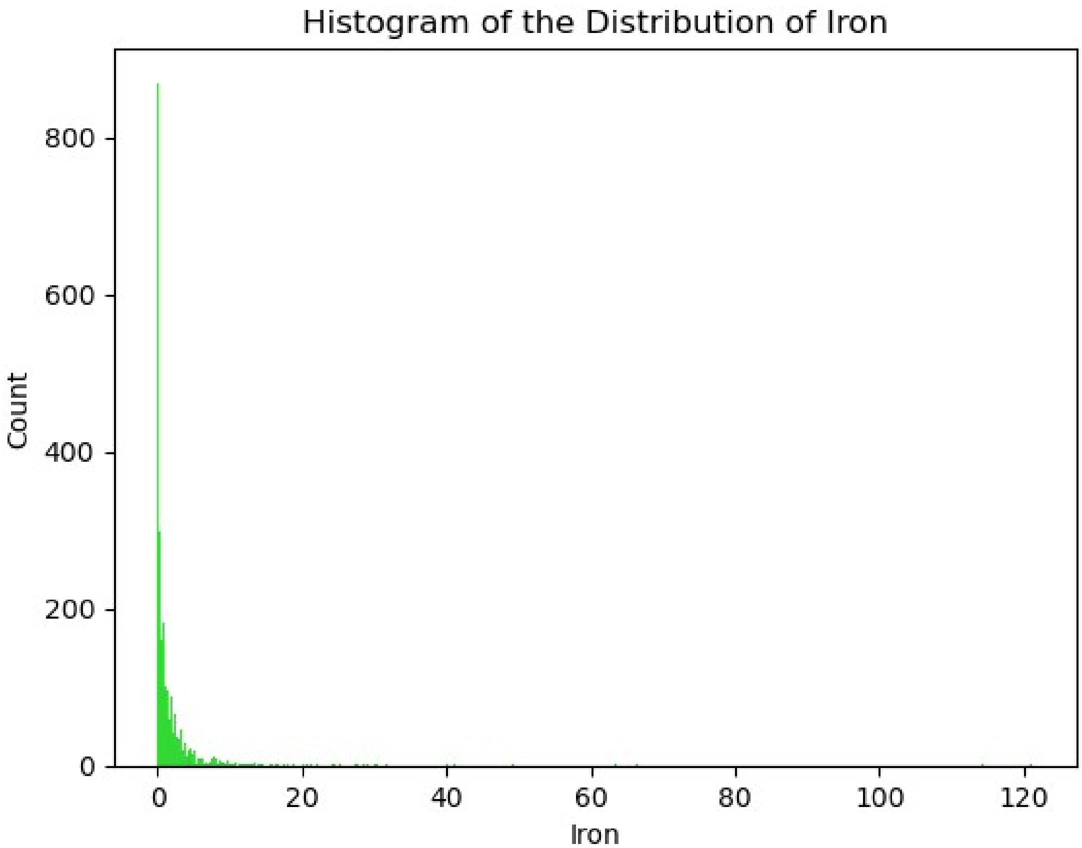


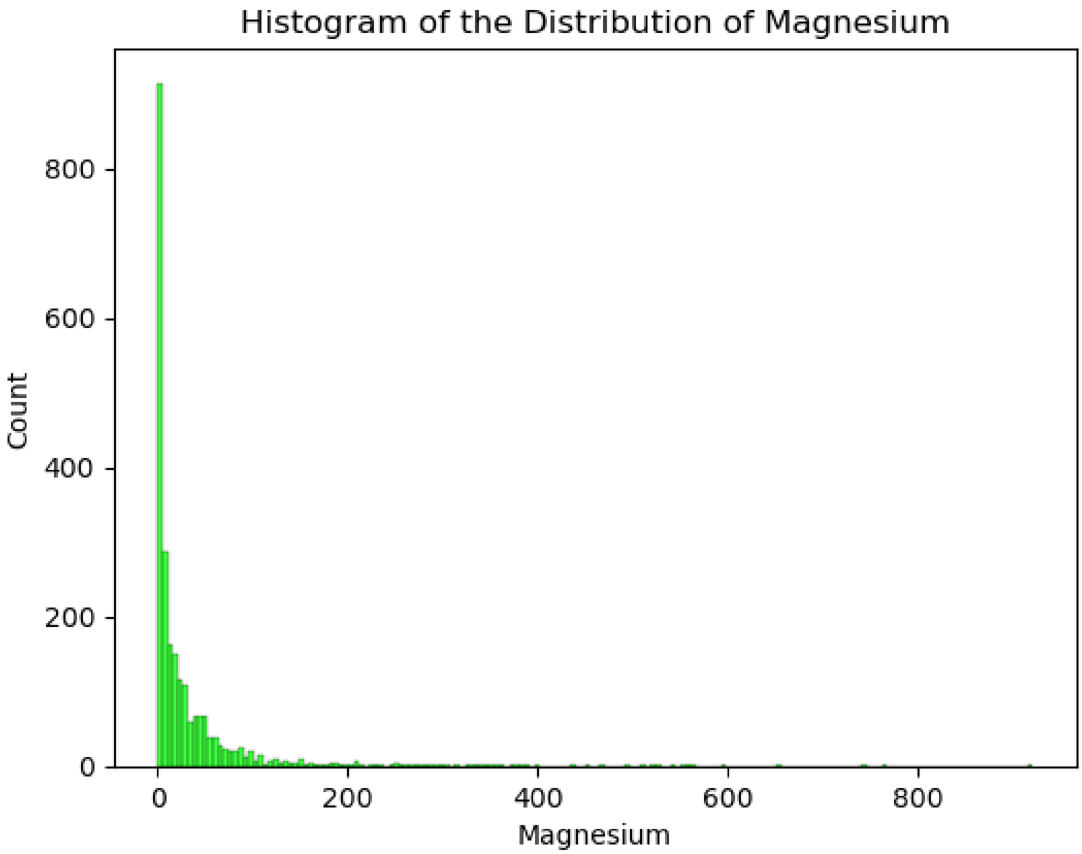
Histogram of the Distribution of Vitamin E
Count
1200
1000
800
600
400
200
0
0
10
20
30
40
Vitamin E


Histogram of the Distribution of Vitamin K
Count
Vitamin K
0
200
400
600
800
1000
1200
0
25
50
75
100
125
150

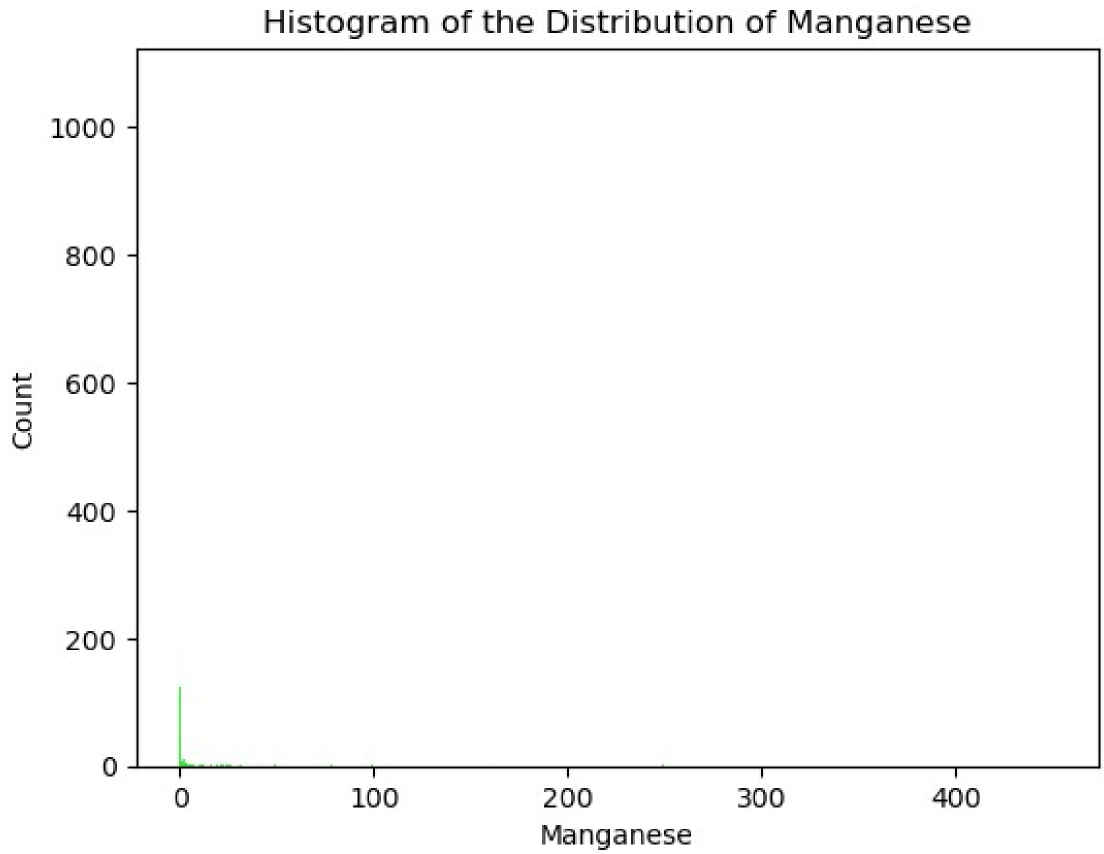

Histogram of the Distribution of Calcium
Count
800
600
400
200
0
0
200
400
600
800
1000
1200
Calcium

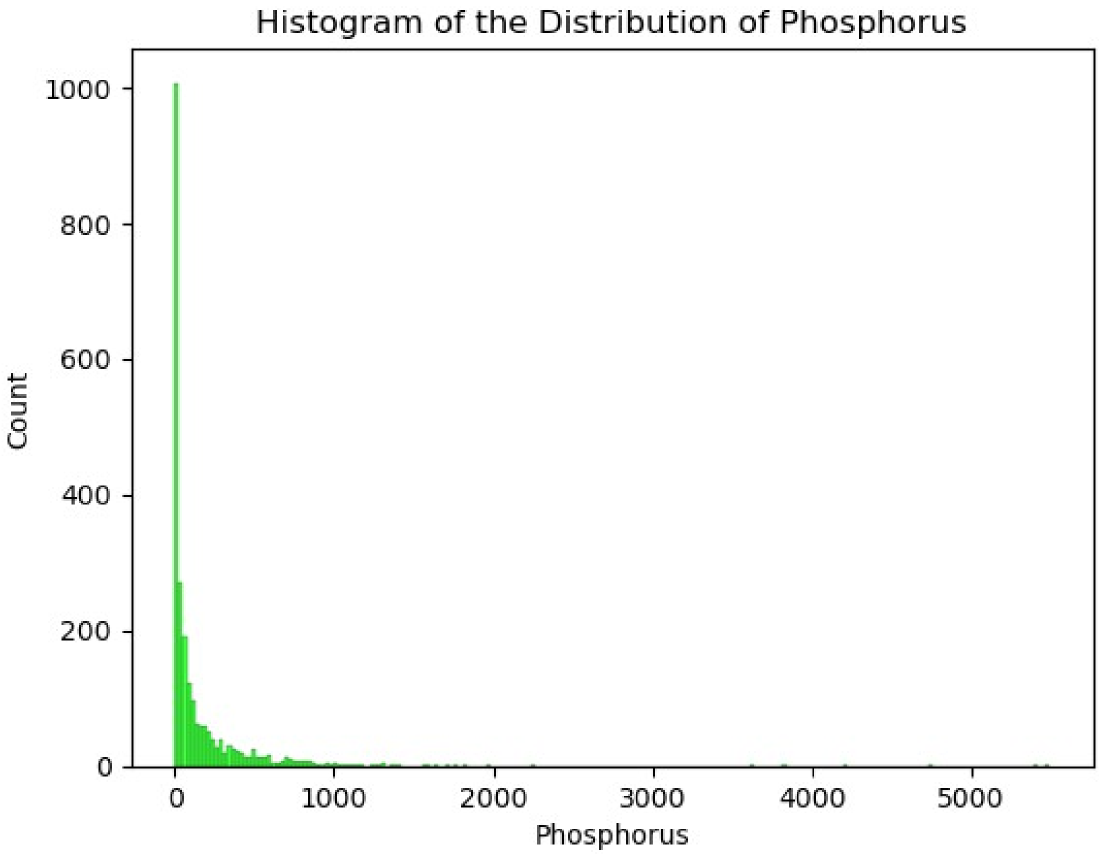

Histogram of the Distribution of Copper
Count
Copper
0
100
200
300
400
500
600
700
0
250
500
750
1000
1250
1500
1750

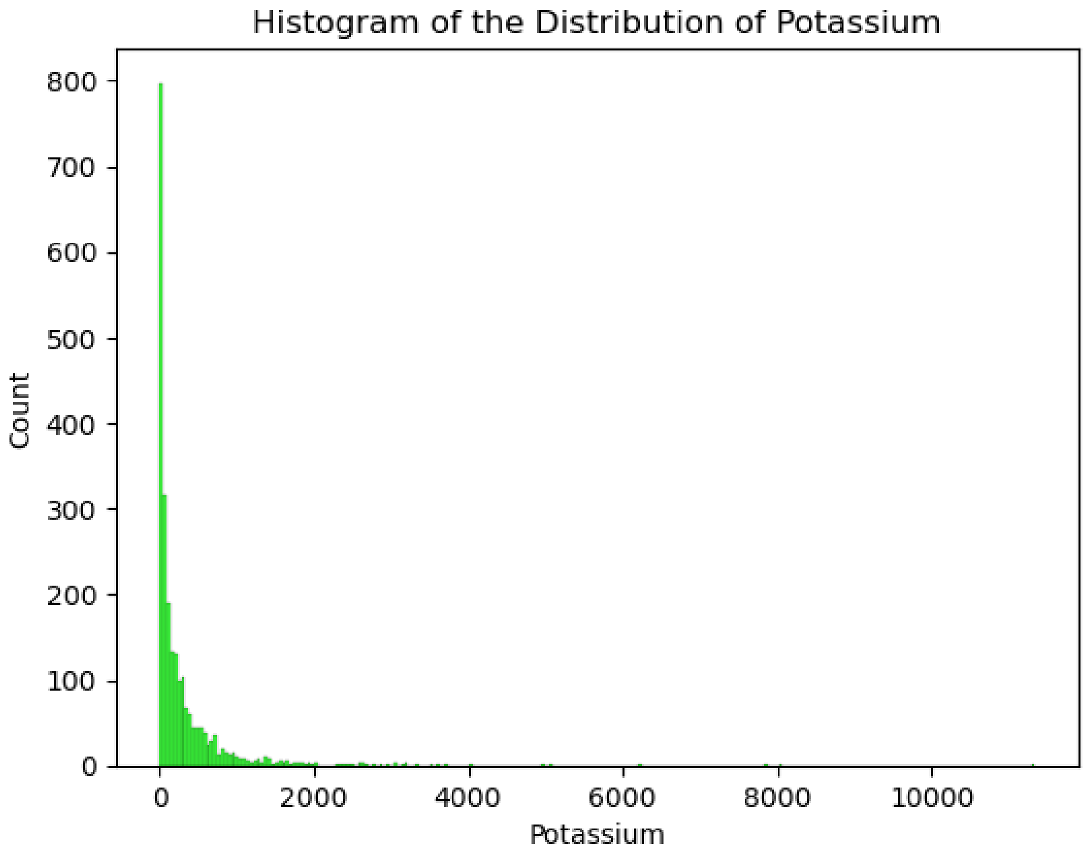

Histogram of the Distribution of Iron
Count
800
600
400
200
0
0
20
40
60
80
100
120
Iron



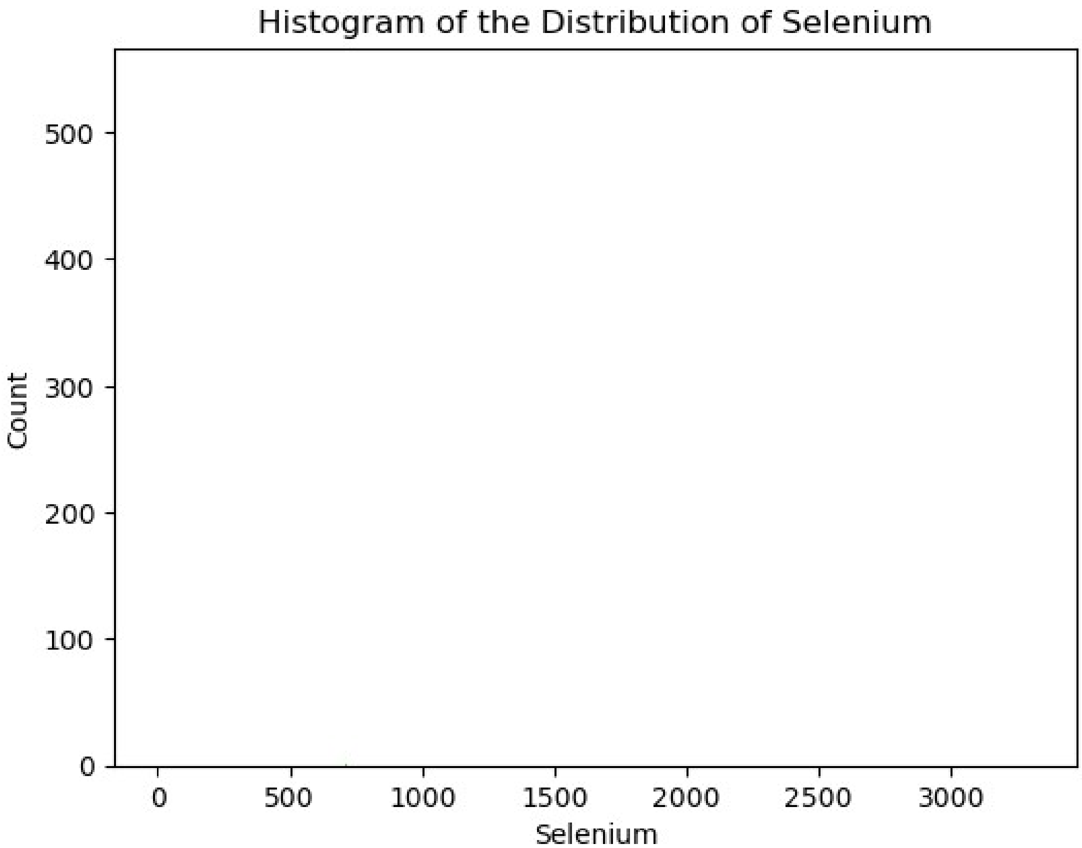
Histogram of the Distribution of Magnesium
800
600
400
200
0
Count
0
200
400
600
800
Magnesium

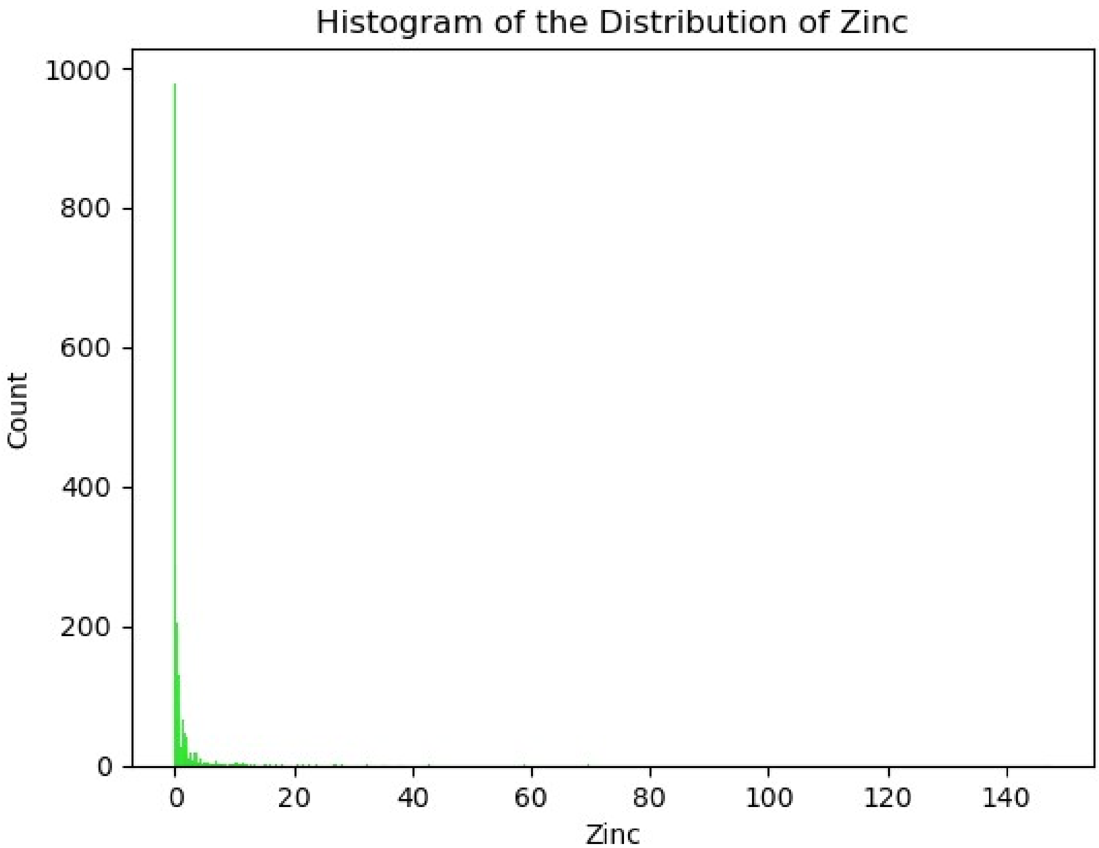

Histogram of the Distribution of Manganese
Count
1000
800
600
400
200
0
0
100
200
300
400
Manganese


Histogram of the Distribution of Phosphorus
Count
1000
800
600
400
200
0
0
1000
2000
3000
4000
5000
Phosphorus

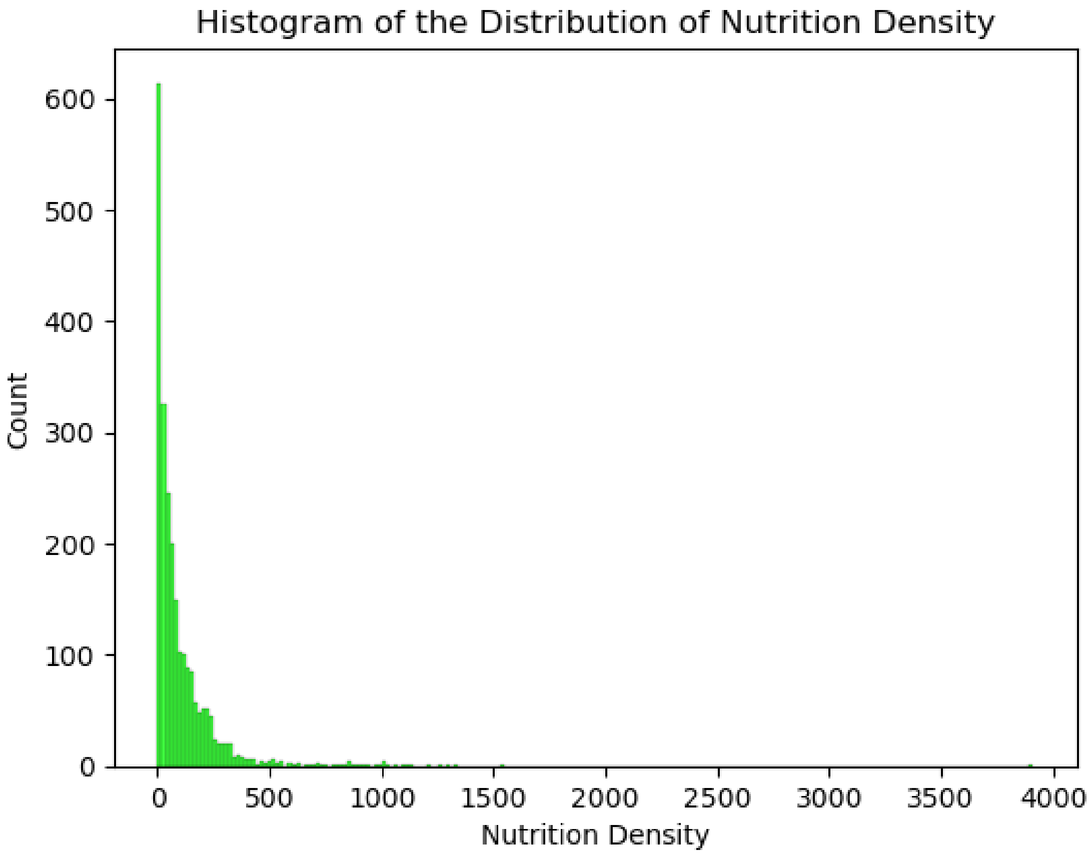


Histogram of the Distribution of Potassium
Count
800
700
600
500
400
300
200
100
0
0
2000
4000
6000
8000
10000
Potassium


Histogram of the Distribution of Selenium
Count
Selenium
0
100
200
300
400
500
0
500
1000
1500
2000
2500
3000



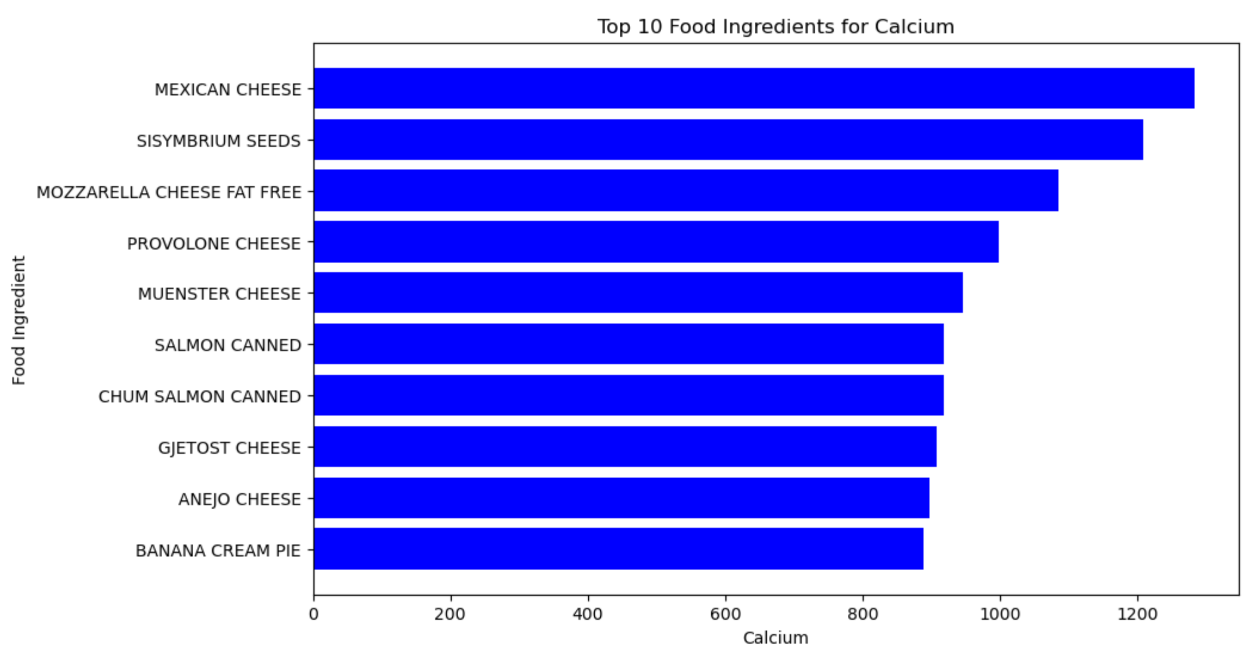
Histogram of the Distribution of Zinc
1000
800
600
400
200
0
Count
0
20
40
60
80
100
120
140
Zinc

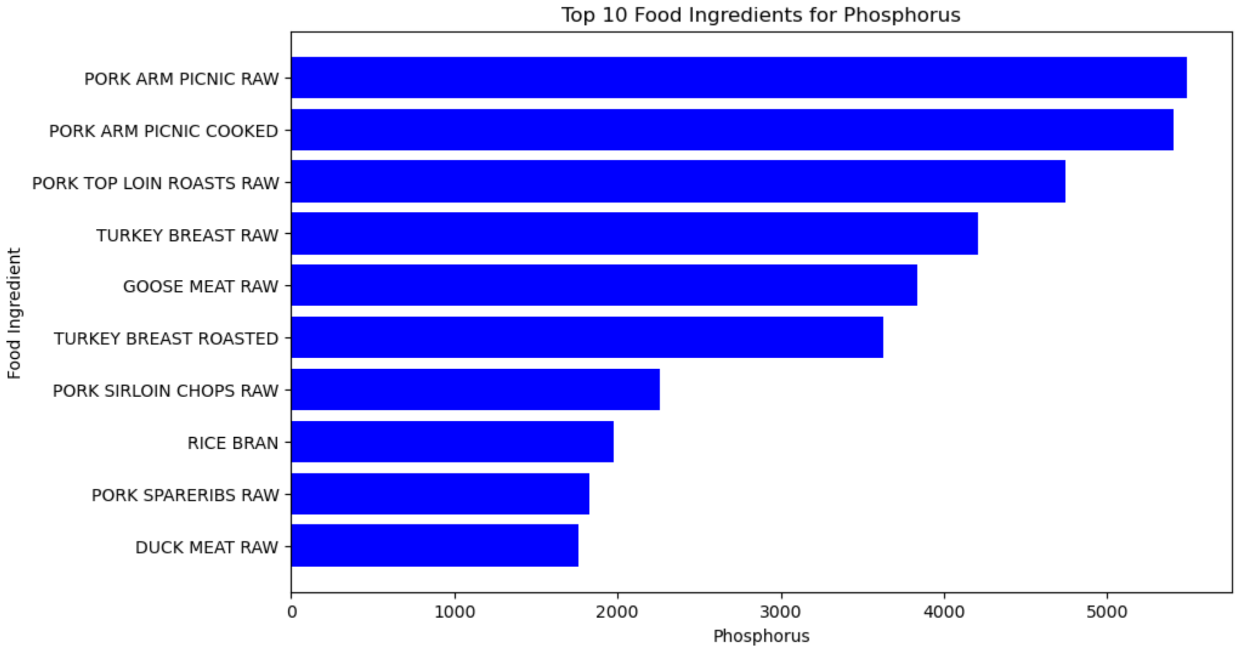



```
# To answer some potential questions that this study may yield at first glance, I'd like to see what foods are highest in
# specific variables that are highly correlated with nutrition density. Since there are many variables within the dataset, I
# will extract the top twenty most positively correlated predictor variables to nutrient density.

nutrient_density_corr = food_correlation['Nutrition Density']

top_20_nutrient_corr = nutrient_density_corr.drop('Nutrition Density').sort_values(ascending = False).head(20)

print(top_20_nutrient_corr)
```

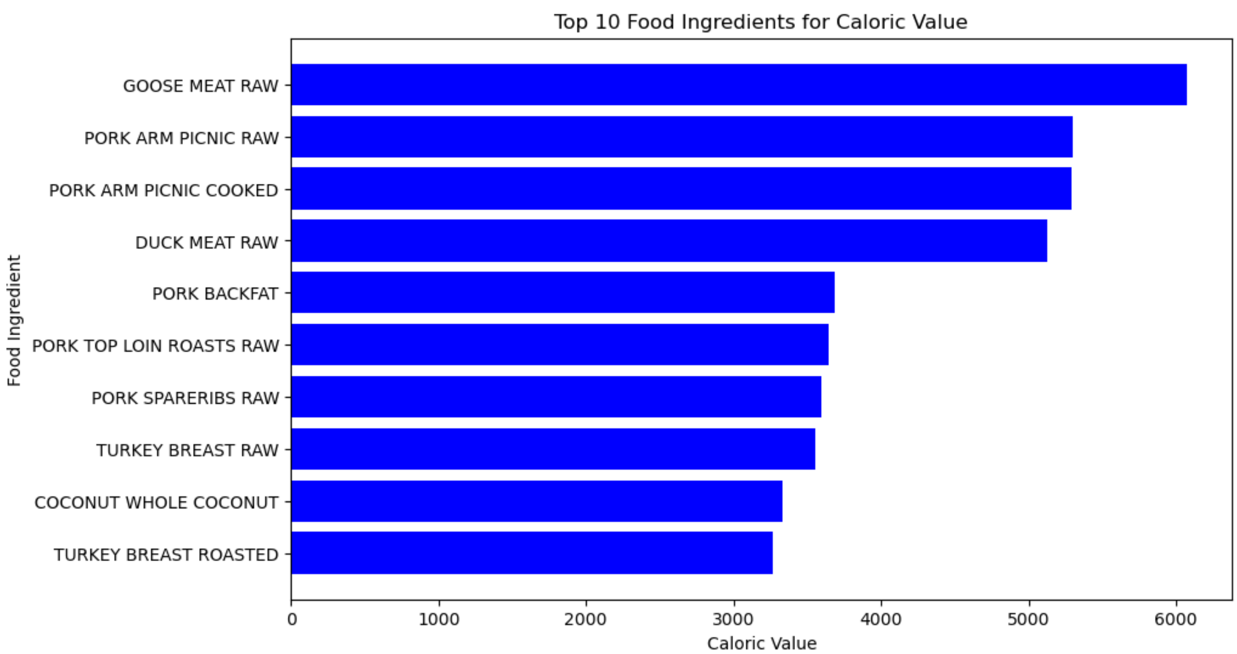
```
Calcium                  0.796068
Phosphorus               0.557906
Caloric Value            0.535323
Potassium                0.528733
Vitamin C                0.490566
Magnesium                0.471353
Protein                  0.455231
Fat                      0.422081
Vitamin B3               0.393261
```

```
Water                        0.358863
Monounsaturated Fats         0.358096
Zinc                         0.331521
Iron                         0.325229
Carbohydrates                0.323416
Vitamin B6                   0.303067
Polyunsaturated Fats         0.285149
Dietary Fiber                0.274237
Vitamin B1                   0.233346
Vitamin B2                   0.209983
Vitamin B5                   0.209027
Name: Nutrition Density, dtype: float64

# I'd like to see the top ten food ingredients with their values for
each of the highly correlated variables that have been
# identified.

for var in top_20_nutrient_corr.index:
    top_10_values = food_nutrition[[var, 'Food']].sort_values(by =
var, ascending = False).head(10)

    plt.figure(figsize = (10, 6))
    plt.barh(top_10_values['Food'], top_10_values[var], color =
'blue')
    plt.xlabel(f'{var}')
    plt.ylabel('Food Ingredient')
    plt.title(f'Top 10 Food Ingredients for {var}')
    plt.gca().invert_yaxis()
    plt.show()
```

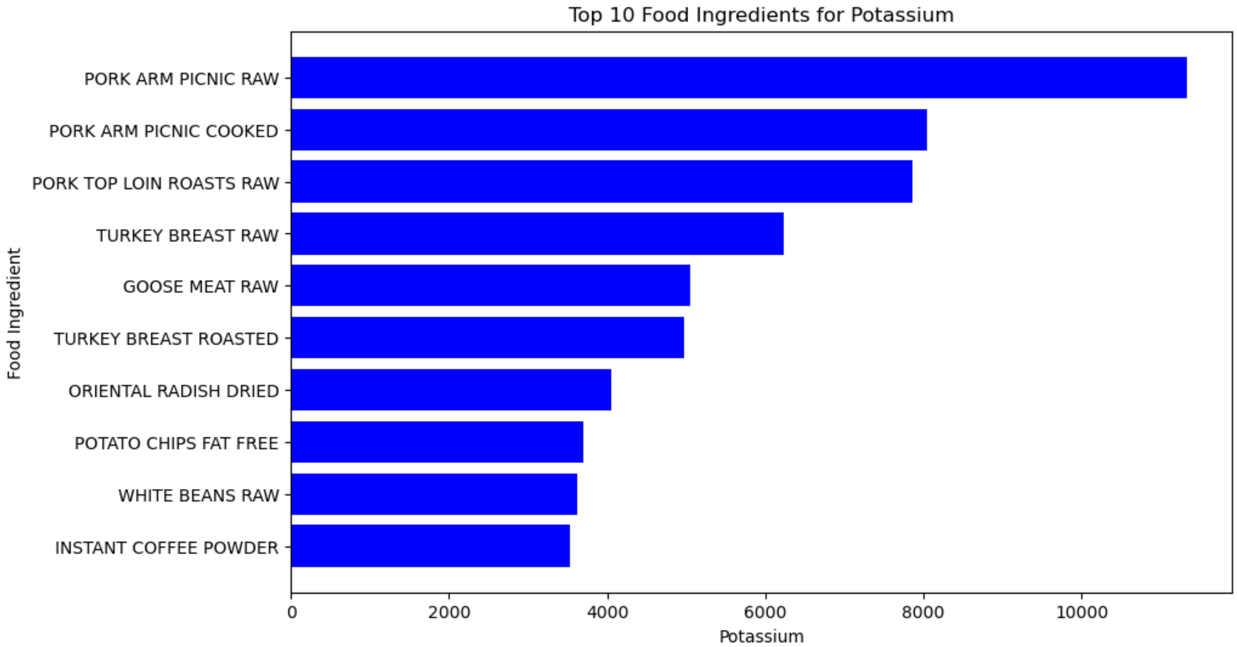




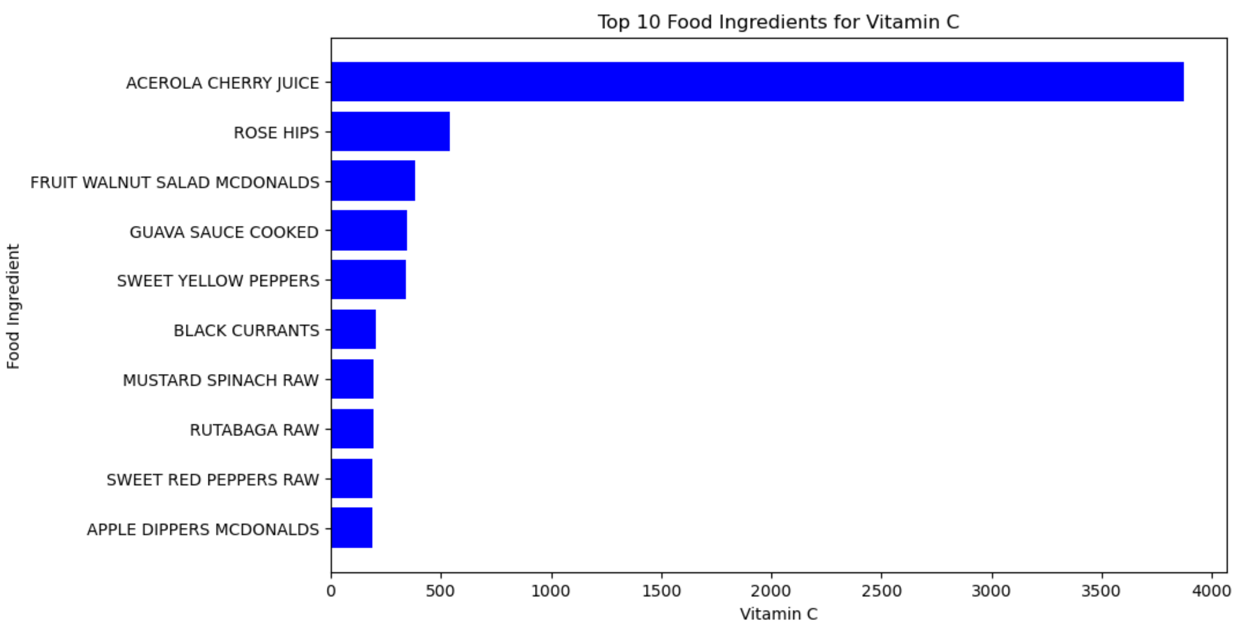
Top 10 Food Ingredients for Phosphorus
PORK ARM PICNIC RAW
PORK ARM PICNIC COOKED
PORK TOP LOIN ROASTS RAW
TURKEY BREAST RAW
GOOSE MEAT RAW
TURKEY BREAST ROASTED
PORK SIRLOIN CHOPS RAW
RICE BRAN
PORK SPARERIBS RAW
DUCK MEAT RAW
Food Ingredient
0
1000
2000
3000
4000
5000
Phosphorus



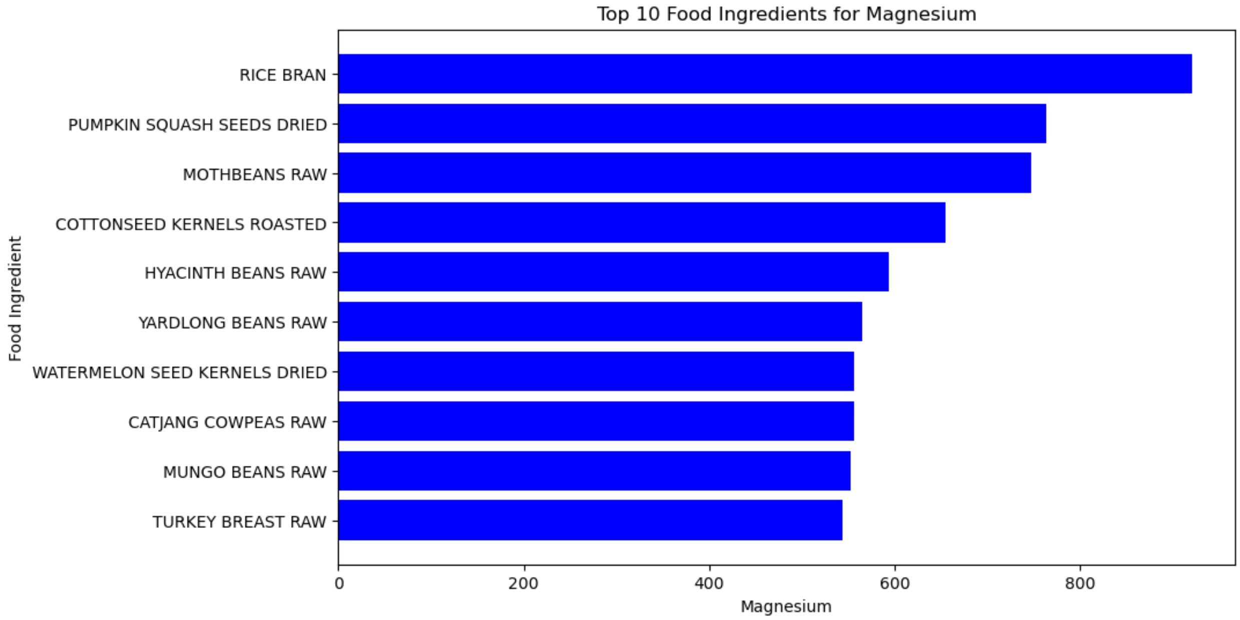
Top 10 Food Ingredients for Caloric Value
GOOSE MEAT RAW
PORK ARM PICNIC RAW
PORK ARM PICNIC COOKED
DUCK MEAT RAW
PORK BACKFAT
PORK TOP LOIN ROASTS RAW
PORK SPARERIBS RAW
TURKEY BREAST RAW
COCONUT WHOLE COCONUT
TURKEY BREAST ROASTED
Food Ingredient
0
1000
2000
3000
4000
5000
6000
Caloric Value



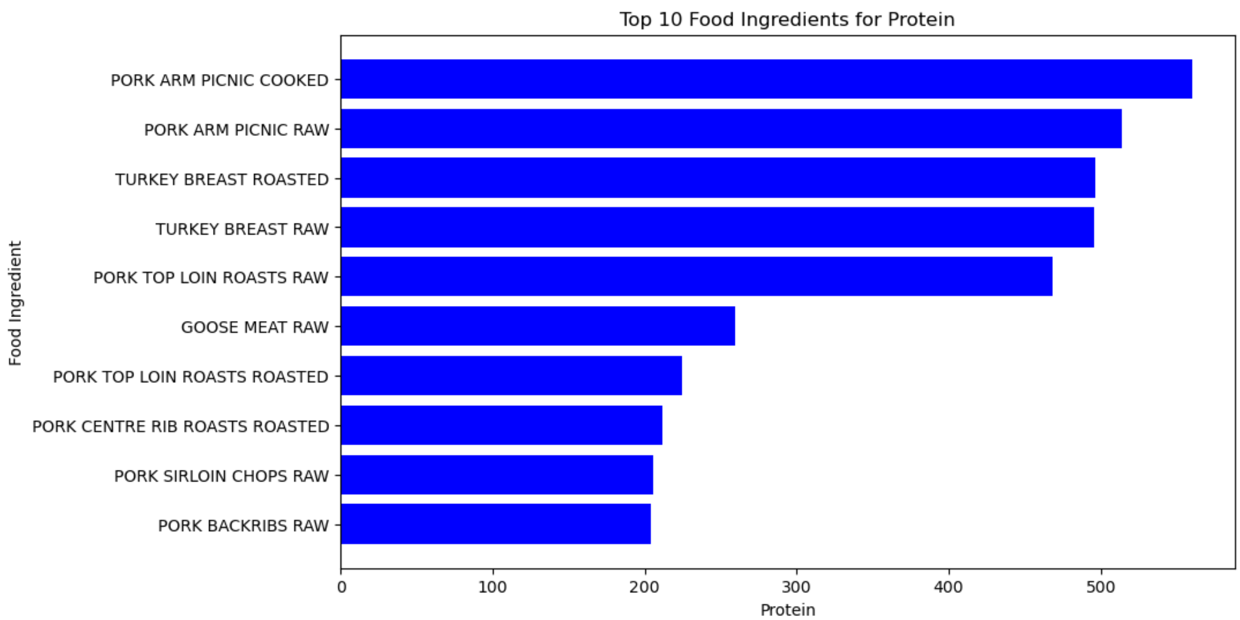
Top 10 Food Ingredients for Potassium
PORK ARM PICNIC RAW
PORK ARM PICNIC COOKED
PORK TOP LOIN ROASTS RAW
TURKEY BREAST RAW
GOOSE MEAT RAW
TURKEY BREAST ROASTED
ORIENTAL RADISH DRIED
POTATO CHIPS FAT FREE
WHITE BEANS RAW
INSTANT COFFEE POWDER
Food Ingredient
0
2000
4000
6000
8000
10000
Potassium



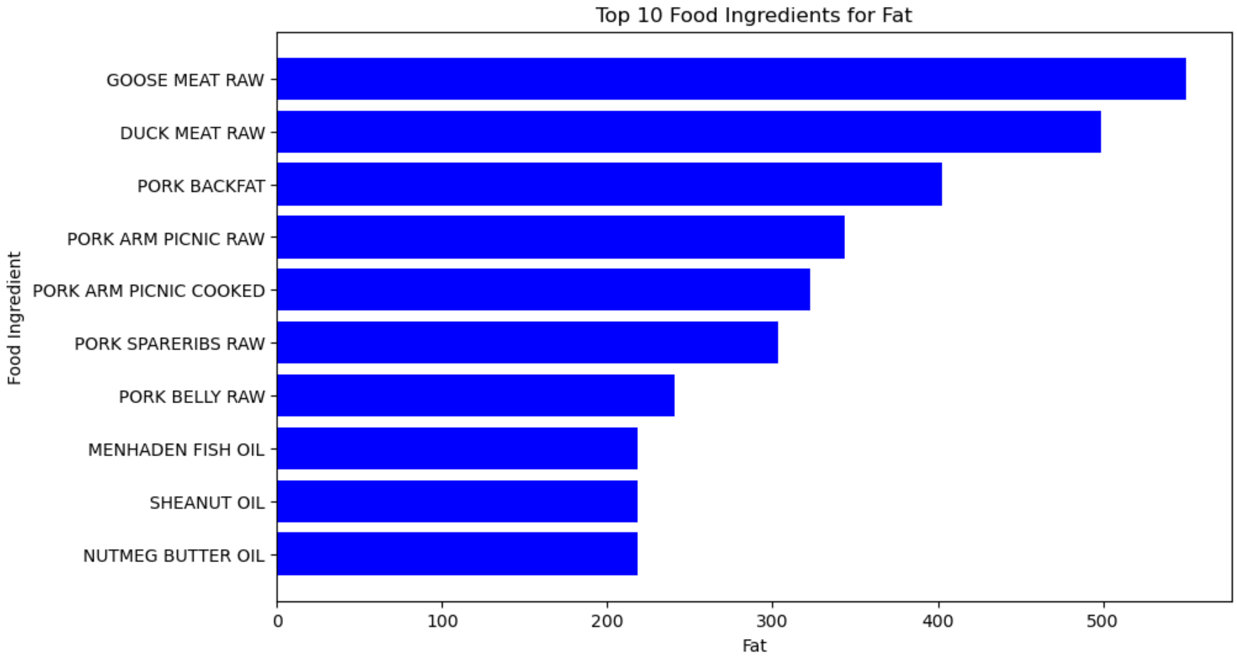
Top 10 Food Ingredients for Vitamin C
ACEROLA CHERRY JUICE
ROSE HIPS
FRUIT WALNUT SALAD MCDONALDS
GUAVA SAUCE COOKED
SWEET YELLOW PEPPERS
BLACK CURRANTS
MUSTARD SPINACH RAW
RUTABAGA RAW
SWEET RED PEPPERS RAW
APPLE DIPPERS MCDONALDS
Food Ingredient
0
500
1000
1500
2000
2500
3000
3500
4000
Vitamin C

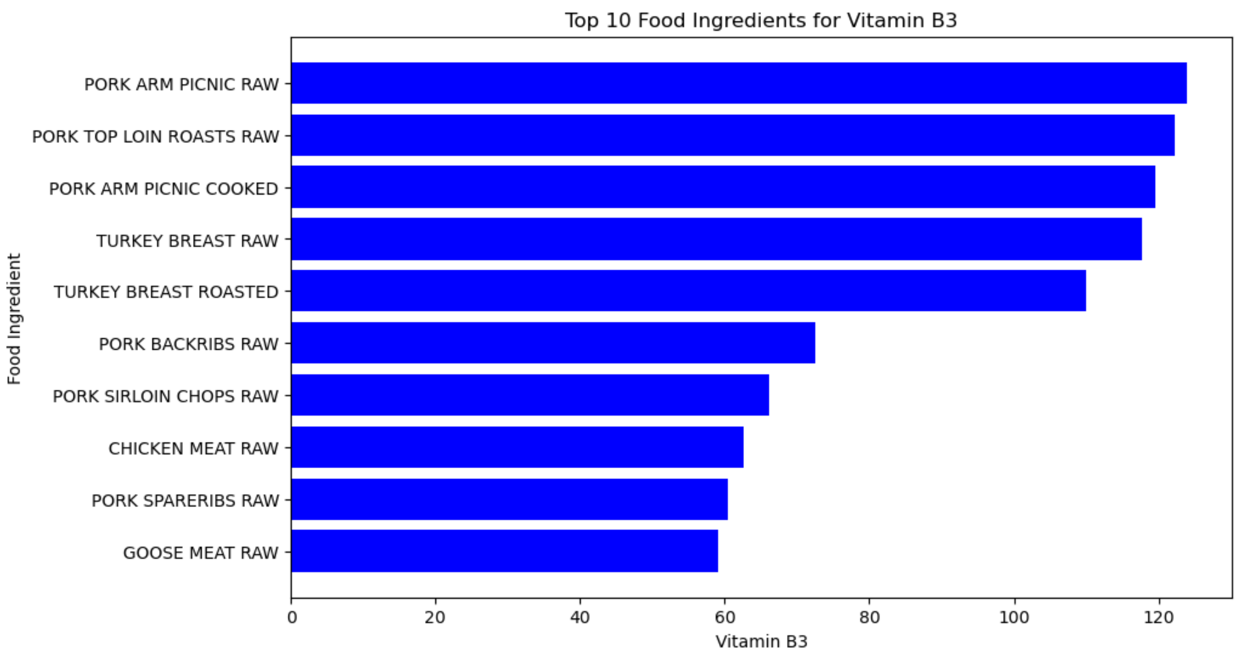


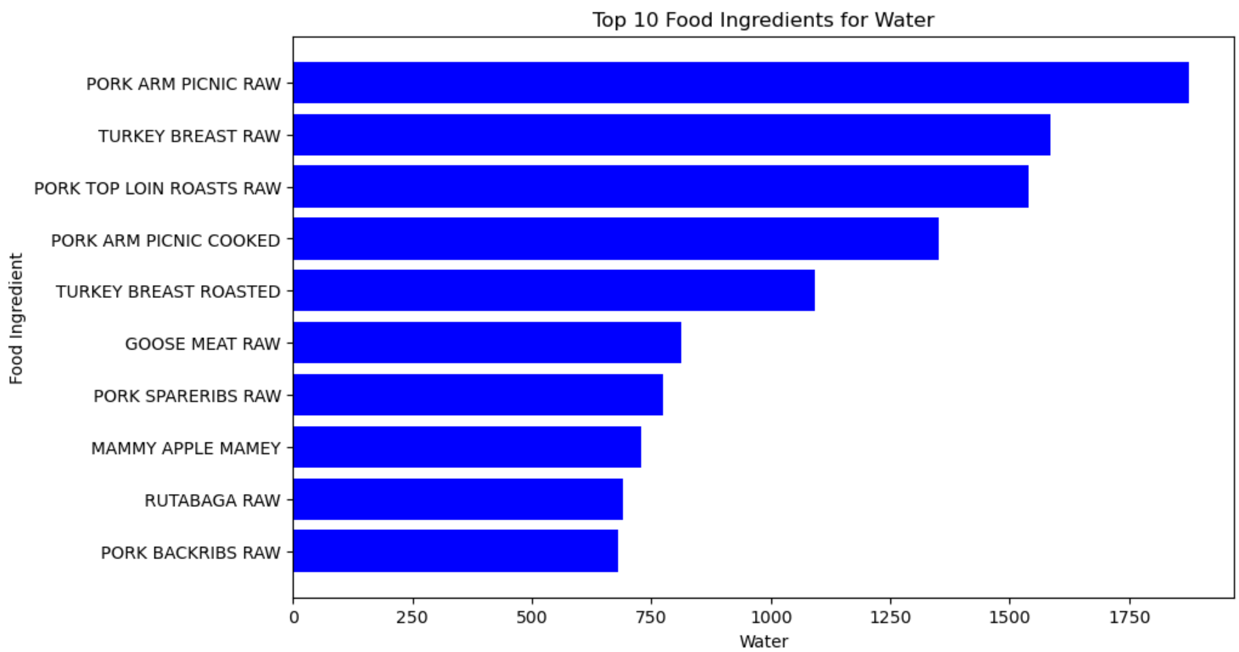
Top 10 Food Ingredients for Magnesium
RICE BRAN
PUMPKIN SQUASH SEEDS DRIED
MOTHBEANS RAW
COTTONSEED KERNELS ROASTED
HYACINTH BEANS RAW
YARDLONG BEANS RAW
WATERMELON SEED KERNELS DRIED
CATJANG COWPEAS RAW
MUNGO BEANS RAW
TURKEY BREAST RAW
Food Ingredient
0
200
400
600
800
Magnesium

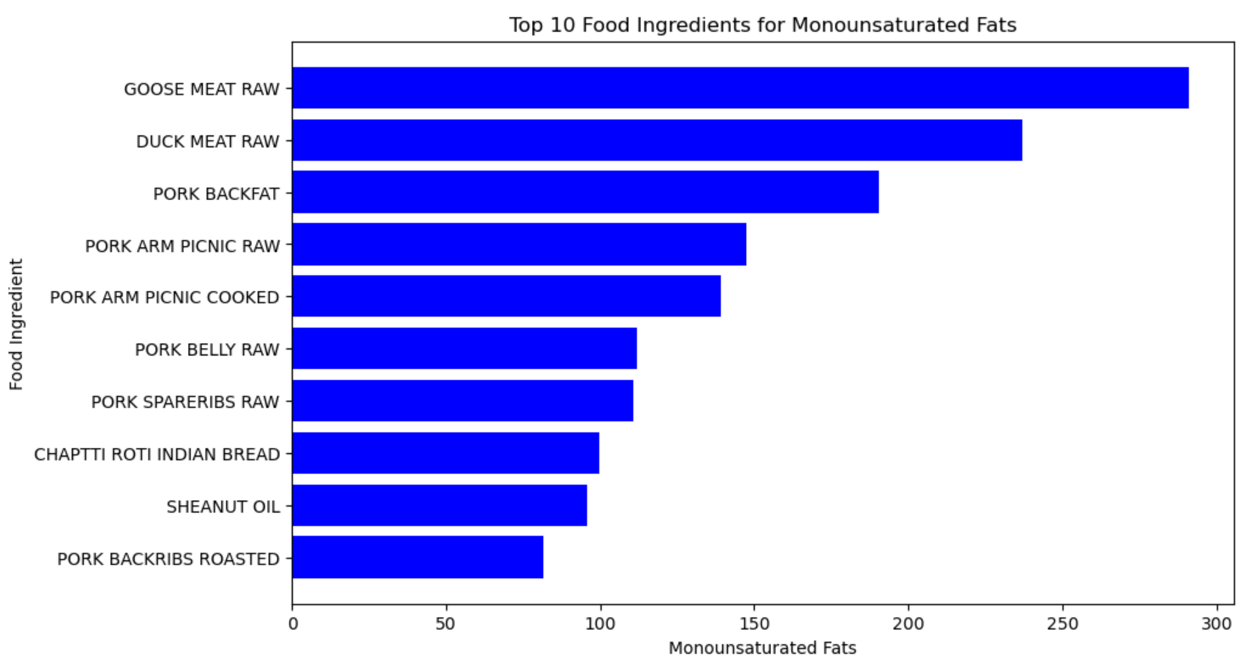


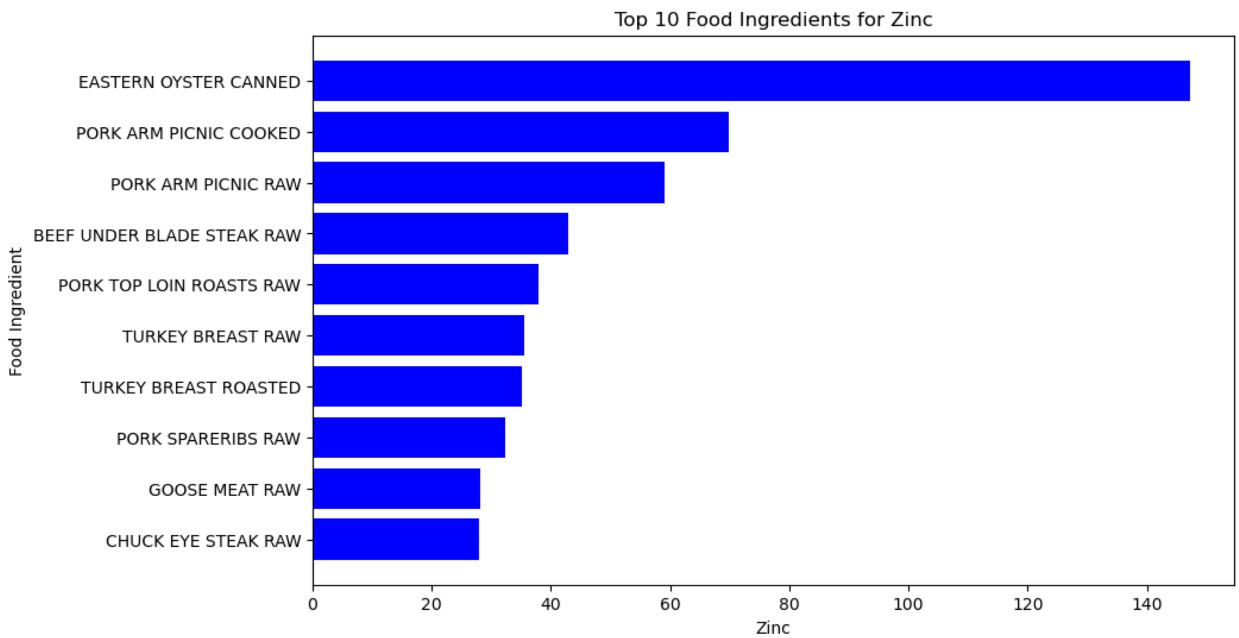
Top 10 Food Ingredients for Protein
PORK ARM PICNIC COOKED
PORK ARM PICNIC RAW
TURKEY BREAST ROASTED
TURKEY BREAST RAW
PORK TOP LOIN ROASTS RAW
GOOSE MEAT RAW
PORK TOP LOIN ROASTS ROASTED
PORK CENTRE RIB ROASTS ROASTED
PORK SIRLOIN CHOPS RAW
PORK BACKRIBS RAW
Food Ingredient
0
100
200
300
400
500
Protein

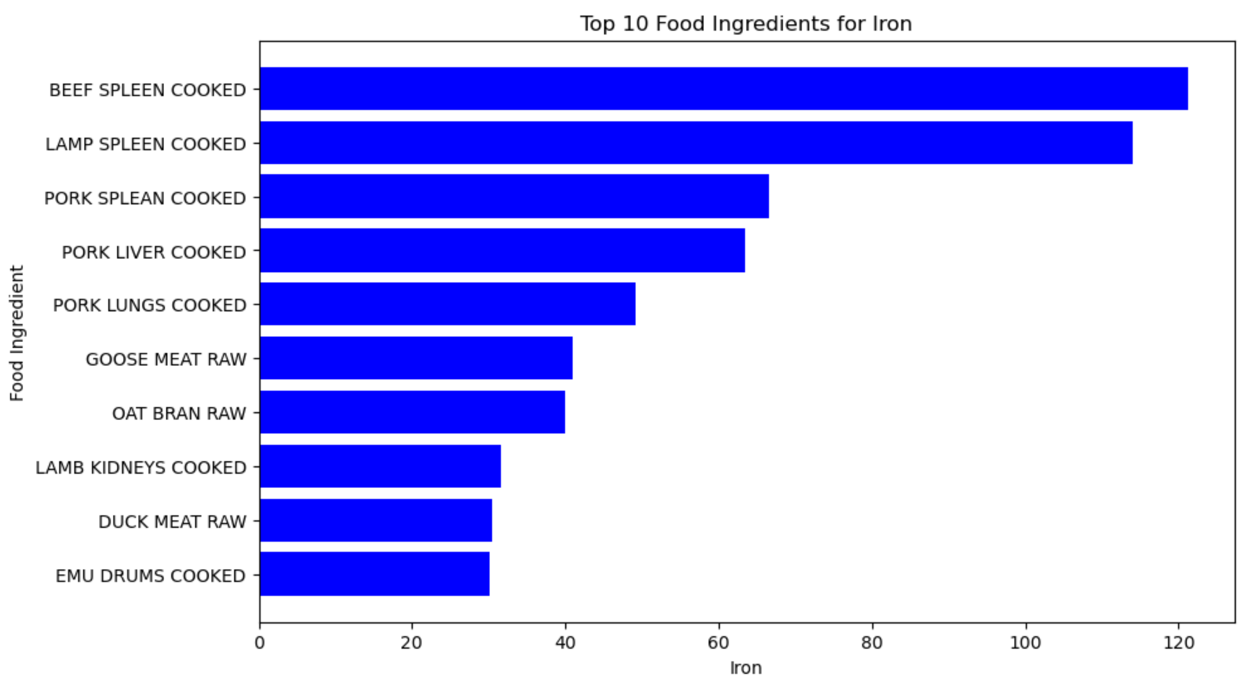

Top 10 Food Ingredients for Fat
GOOSE MEAT RAW
DUCK MEAT RAW
PORK BACKFAT
PORK ARM PICNIC RAW
PORK ARM PICNIC COOKED
PORK SPARERIBS RAW
PORK BELLY RAW
MENHADEN FISH OIL
SHEANUT OIL
NUTMEG BUTTER OIL
Food Ingredient
0
100
200
300
400
500
Fat

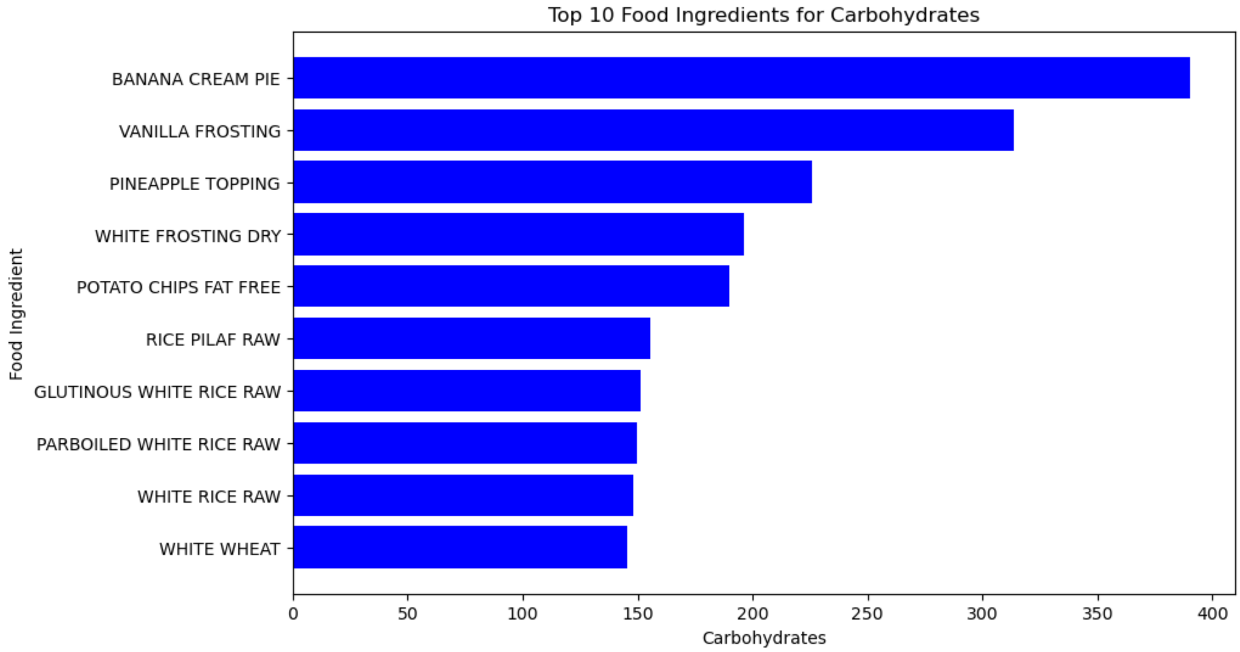

Top 10 Food Ingredients for Vitamin B3
PORK ARM PICNIC RAW
PORK TOP LOIN ROASTS RAW
PORK ARM PICNIC COOKED
TURKEY BREAST RAW
TURKEY BREAST ROASTED
PORK BACKRIBS RAW
PORK SIRLOIN CHOPS RAW
CHICKEN MEAT RAW
PORK SPARERIBS RAW
GOOSE MEAT RAW
Food Ingredient
0
20
40
60
80
100
120
Vitamin B3

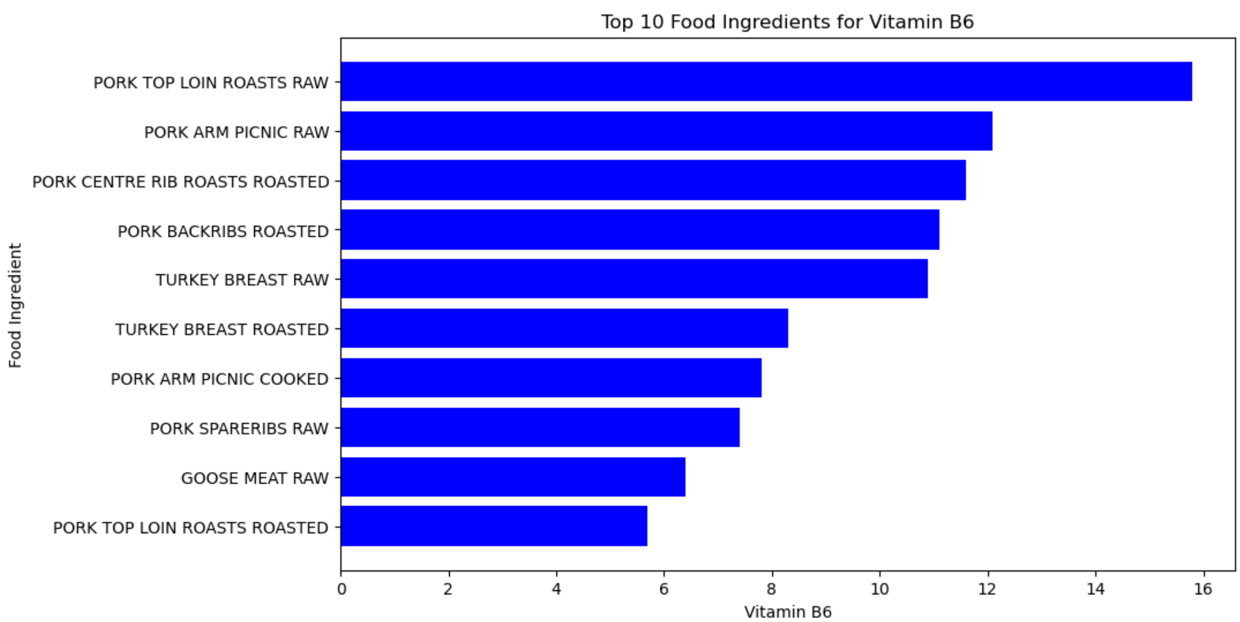

Top 10 Food Ingredients for Water
PORK ARM PICNIC RAW
TURKEY BREAST RAW
PORK TOP LOIN ROASTS RAW
PORK ARM PICNIC COOKED
TURKEY BREAST ROASTED
GOOSE MEAT RAW
PORK SPARERIBS RAW
MAMMY APPLE MAMEY
RUTABAGA RAW
PORK BACKRIBS RAW
Food Ingredient
0
250
500
750
1000
1250
1500
1750
Water

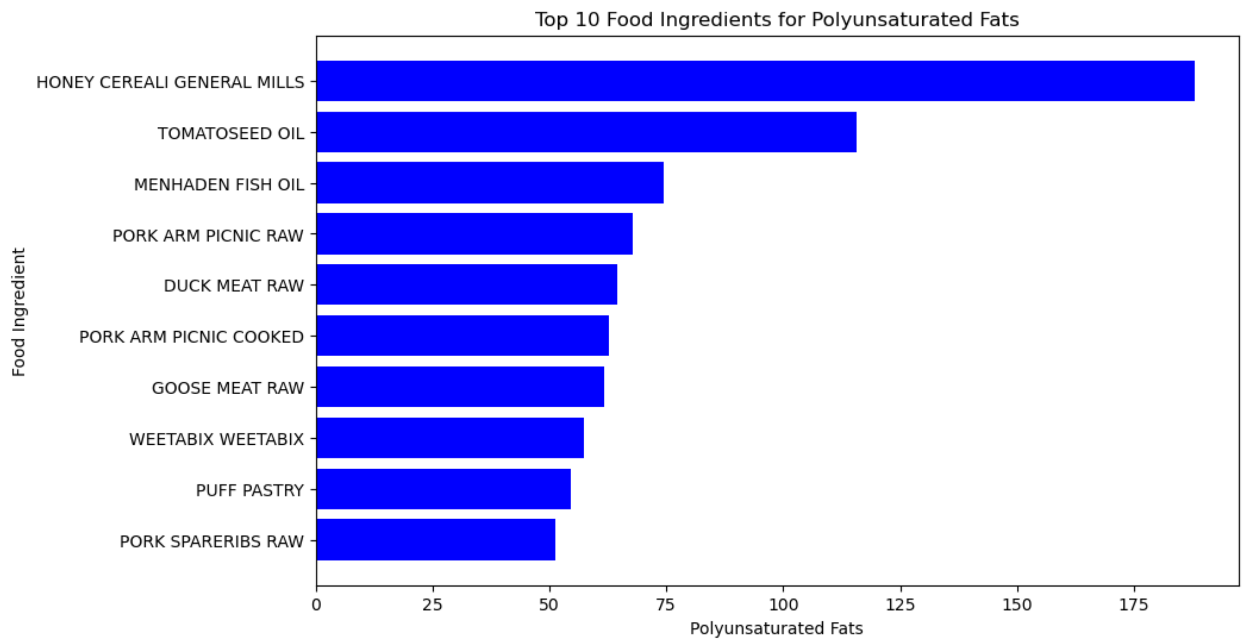

Top 10 Food Ingredients for Monounsaturated Fats
GOOSE MEAT RAW
DUCK MEAT RAW
PORK BACKFAT
PORK ARM PICNIC RAW
PORK ARM PICNIC COOKED
PORK BELLY RAW
PORK SPARERIBS RAW
CHAPTTI ROTI INDIAN BREAD
SHEANUT OIL
PORK BACKRIBS ROASTED
Food Ingredient
0
50
100
150
200
250
300
Monounsaturated Fats


Top 10 Food Ingredients for Zinc
EASTERN OYSTER CANNED
PORK ARM PICNIC COOKED
PORK ARM PICNIC RAW
BEEF UNDER BLADE STEAK RAW
PORK TOP LOIN ROASTS RAW
TURKEY BREAST RAW
TURKEY BREAST ROASTED
PORK SPARERIBS RAW
GOOSE MEAT RAW
CHUCK EYE STEAK RAW
Food Ingredient
0
20
40
60
80
100
120
140
Zinc

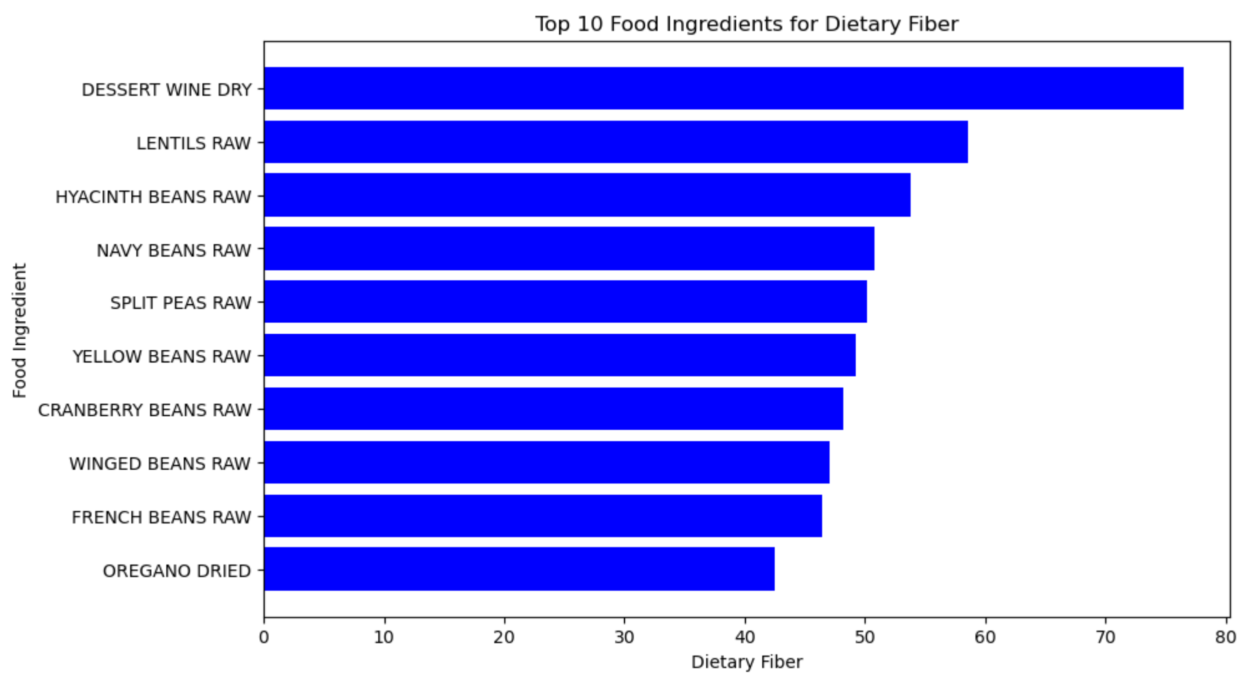

Top 10 Food Ingredients for Iron
BEEF SPLEEN COOKED
LAMP SPLEEN COOKED
PORK SPLEAN COOKED
PORK LIVER COOKED
PORK LUNGS COOKED
GOOSE MEAT RAW
OAT BRAN RAW
LAMB KIDNEYS COOKED
DUCK MEAT RAW
EMU DRUMS COOKED
Food Ingredient
0
20
40
60
80
100
120
Iron


Top 10 Food Ingredients for Carbohydrates
BANANA CREAM PIE
VANILLA FROSTING
PINEAPPLE TOPPING
WHITE FROSTING DRY
POTATO CHIPS FAT FREE
RICE PILAF RAW
GLUTINOUS WHITE RICE RAW
PARBOILED WHITE RICE RAW
WHITE RICE RAW
WHITE WHEAT
Food Ingredient
0
50
100
150
200
250
300
350
400
Carbohydrates

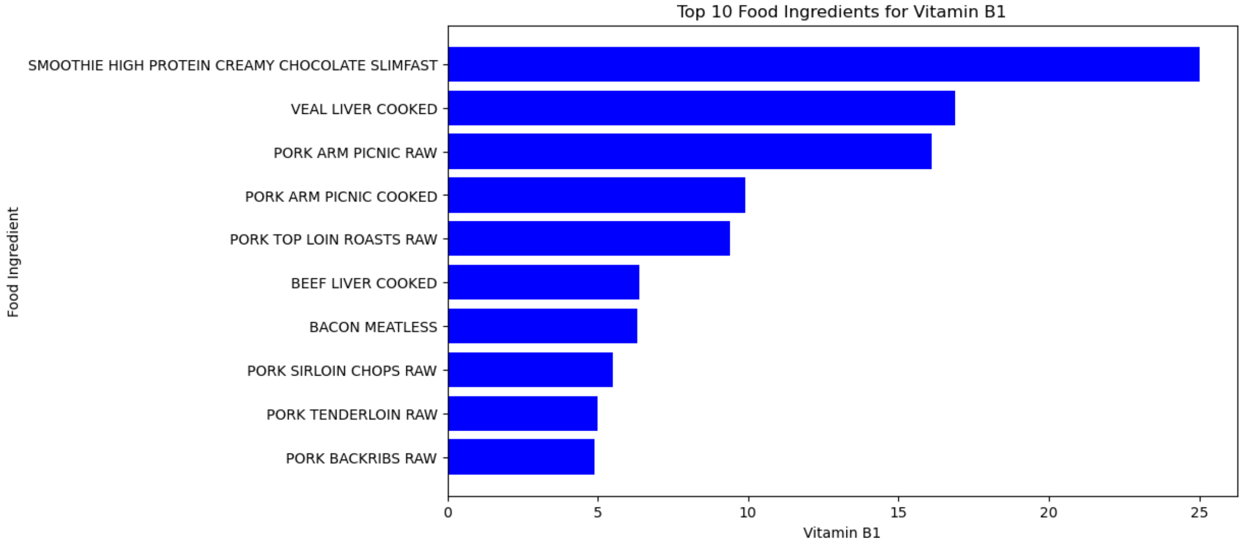

Top 10 Food Ingredients for Vitamin B6
PORK TOP LOIN ROASTS RAW
PORK ARM PICNIC RAW
PORK CENTRE RIB ROASTS ROASTED
PORK BACKRIBS ROASTED
TURKEY BREAST RAW
TURKEY BREAST ROASTED
PORK ARM PICNIC COOKED
PORK SPARERIBS RAW
GOOSE MEAT RAW
PORK TOP LOIN ROASTS ROASTED
Food Ingredient
0
2
4
6
8
10
12
14
16
Vitamin B6

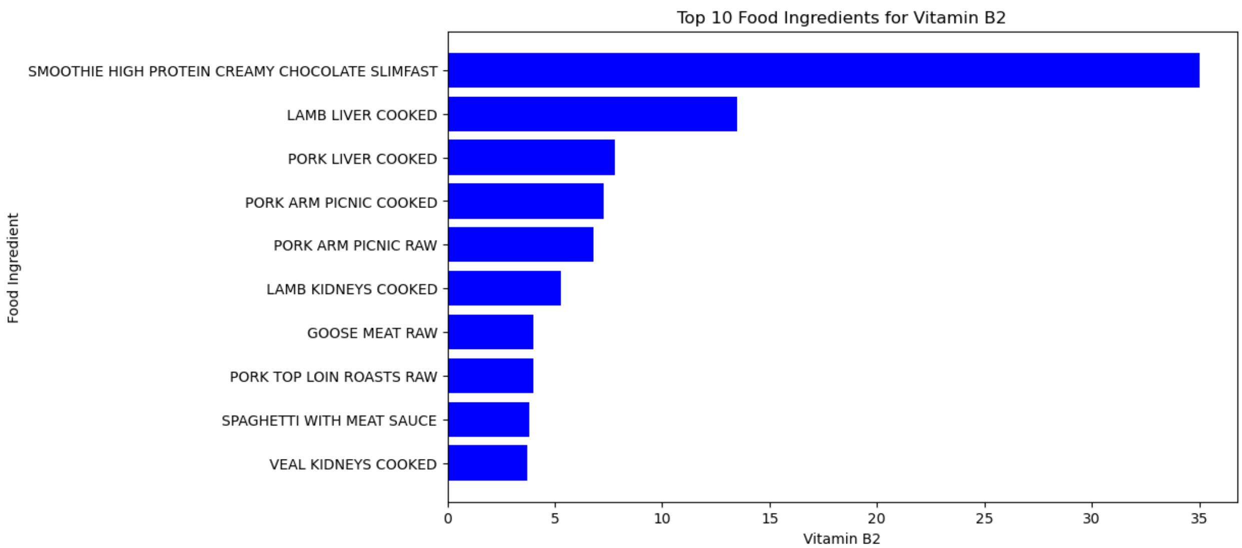


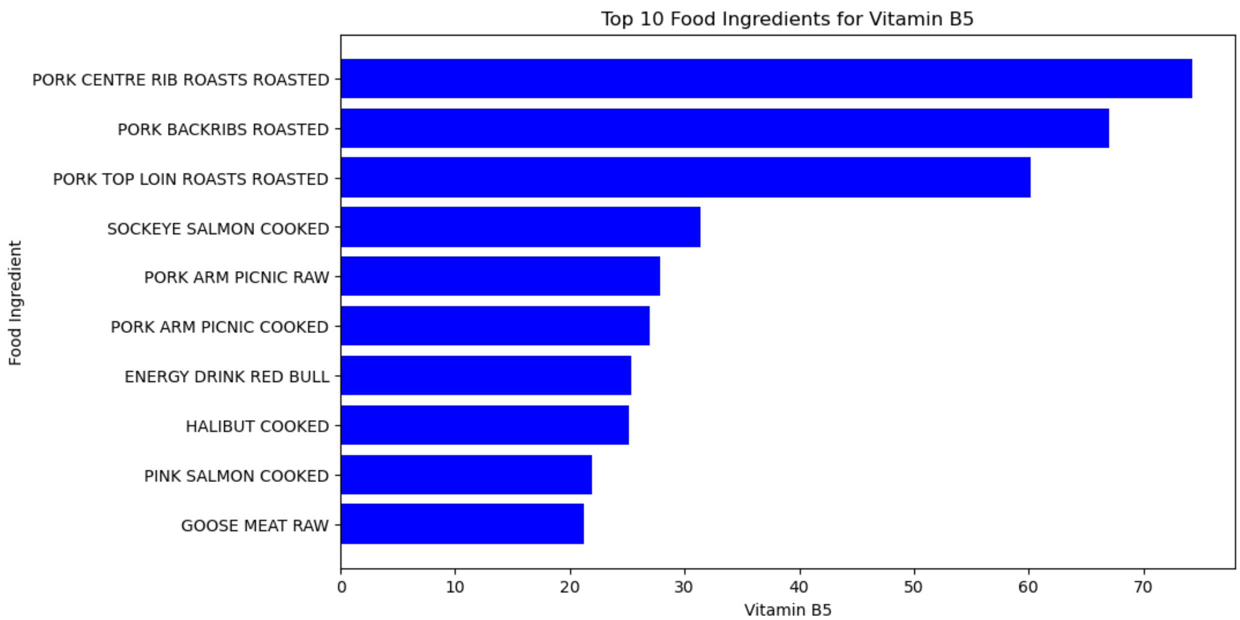
Top 10 Food Ingredients for Polyunsaturated Fats
Food Ingredient
HONEY CEREALI GENERAL MILLS
TOMATOSEED OIL
MENHADEN FISH OIL
PORK ARM PICNIC RAW
DUCK MEAT RAW
PORK ARM PICNIC COOKED
GOOSE MEAT RAW
WEETABIX WEETABIX
PUFF PASTRY
PORK SPARERIBS RAW
0
25
50
75
100
125
150
175
Polyunsaturated Fats



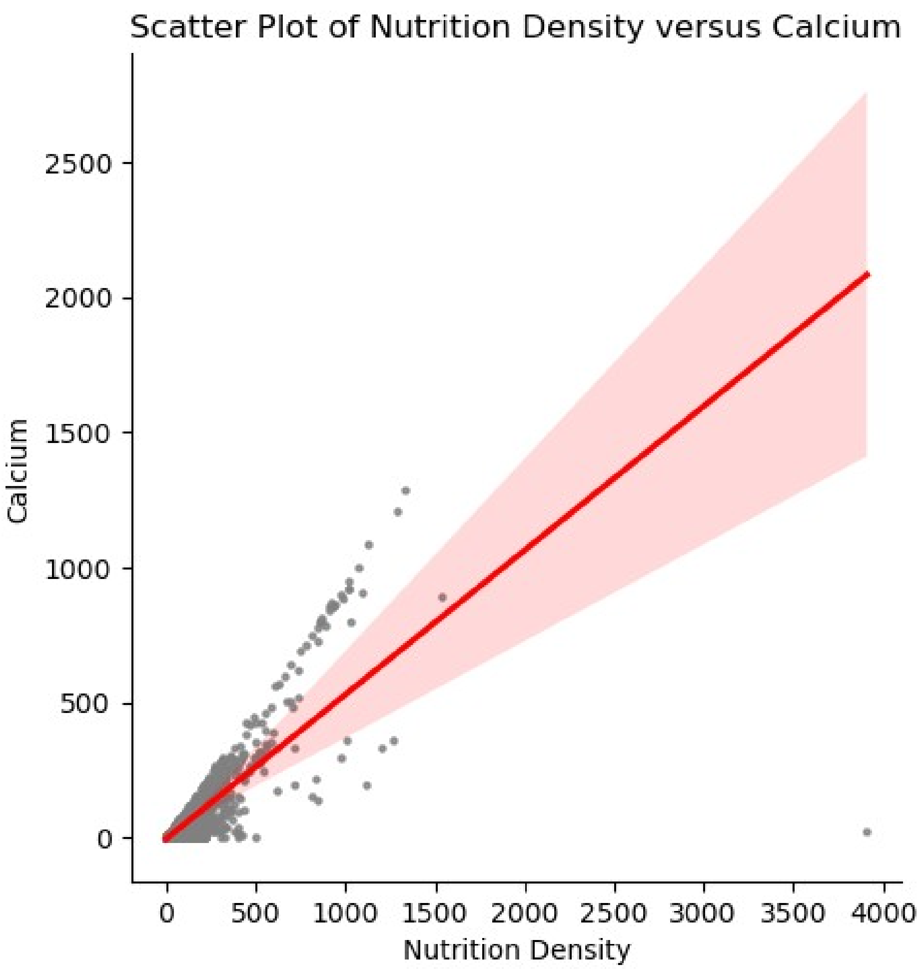
Top 10 Food Ingredients for Dietary Fiber
Food Ingredient
DESSERT WINE DRY
LENTILS RAW
HYACINTH BEANS RAW
NAVY BEANS RAW
SPLIT PEAS RAW
YELLOW BEANS RAW
CRANBERRY BEANS RAW
WINGED BEANS RAW
FRENCH BEANS RAW
OREGANO DRIED
0
10
20
30
40
50
60
70
80
Dietary Fiber


Top 10 Food Ingredients for Vitamin B1
SMOOTHIE HIGH PROTEIN CREAMY CHOCOLATE SLIMFAST
VEAL LIVER COOKED
PORK ARM PICNIC RAW
PORK ARM PICNIC COOKED
PORK TOP LOIN ROASTS RAW
BEEF LIVER COOKED
BACON MEATLESS
PORK SIRLOIN CHOPS RAW
PORK TENDERLOIN RAW
PORK BACKRIBS RAW
Food Ingredient
0
5
10
15
20
25
Vitamin B1

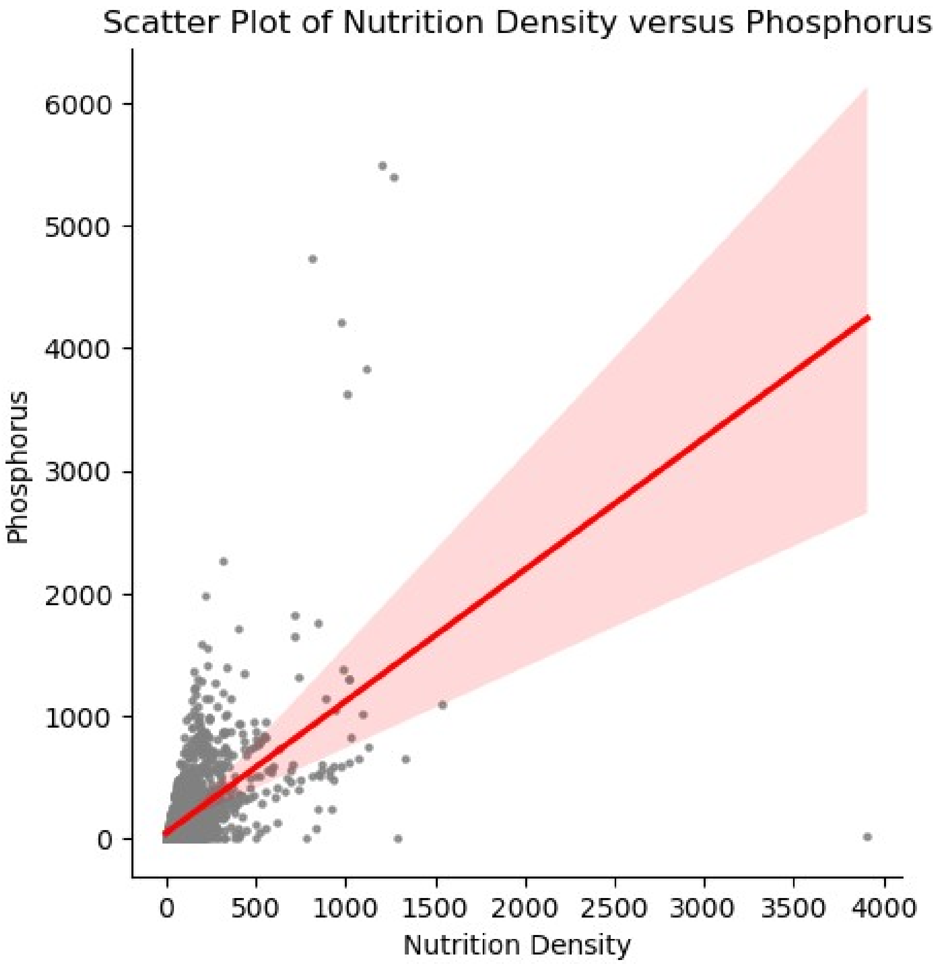

Top 10 Food Ingredients for Vitamin B2
SMOOTHIE HIGH PROTEIN CREAMY CHOCOLATE SLIMFAST
LAMB LIVER COOKED
PORK LIVER COOKED
PORK ARM PICNIC COOKED
PORK ARM PICNIC RAW
LAMB KIDNEYS COOKED
GOOSE MEAT RAW
PORK TOP LOIN ROASTS RAW
SPAGHETTI WITH MEAT SAUCE
VEAL KIDNEYS COOKED
Food Ingredient
0
5
10
15
20
25
30
35
Vitamin B2

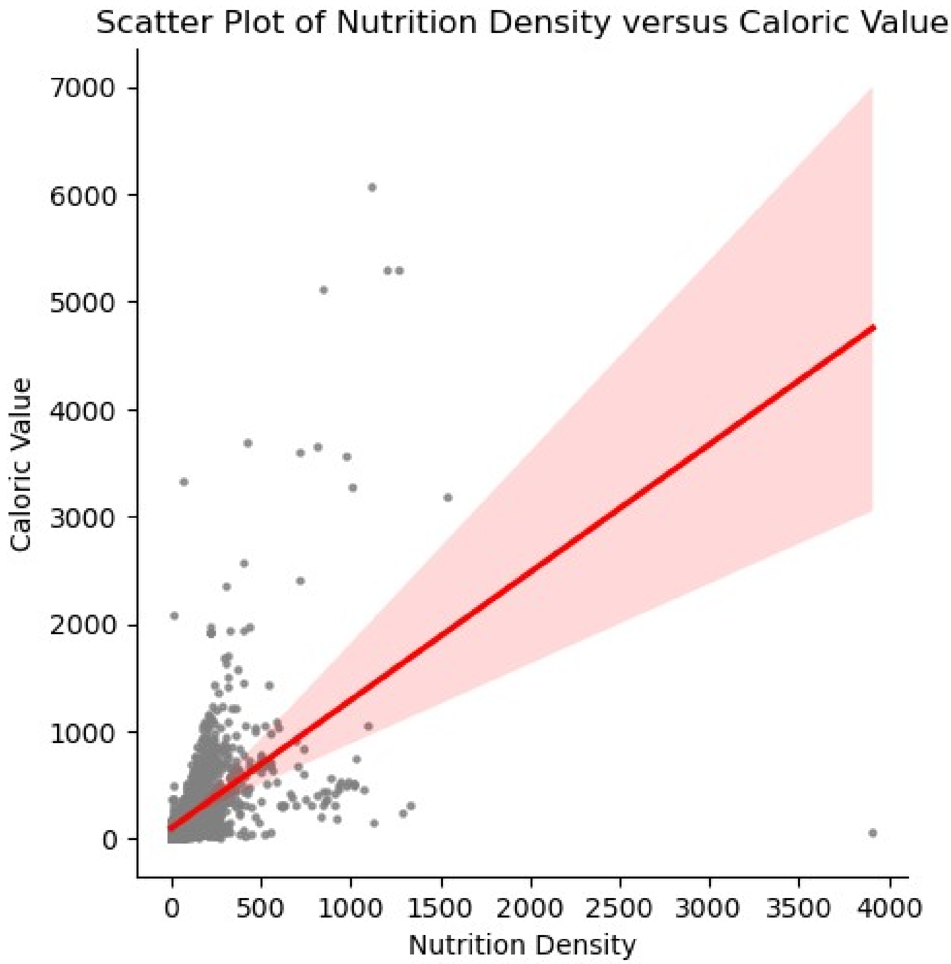


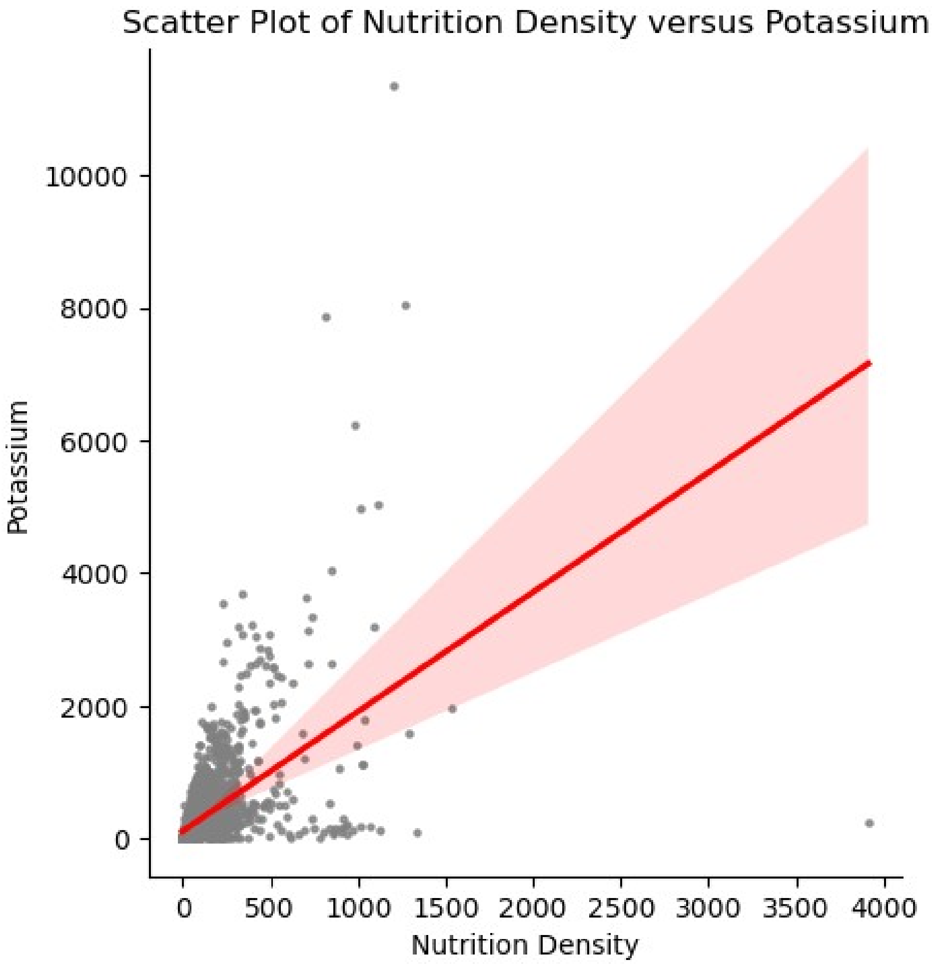

```
# The final element of my exploratory data analysis, I wish to see the
relationship between the top 20 highly correlated
# variables and the Nutrition Density variable. This can be seen best
on a scatter plot, which I will craft a function to
# generate multiple scatter plots with a trend line and a for loop
that creates the 20 scatter plots needed.

def make_nutrient_scatter_plot(data, col1, col2):
    sns.lmplot(data = data, x = col1, y = col2, line_kws = {'color':
'red'}, scatter_kws = {'color': 'gray', 's': 5})
    plt.title(f'Scatter Plot of {col1} versus {col2}')
    plt.show()

for col in top_20_nutrient_corr.index:
    make_nutrient_scatter_plot(food_nutrition, 'Nutrition Density',
col)
```

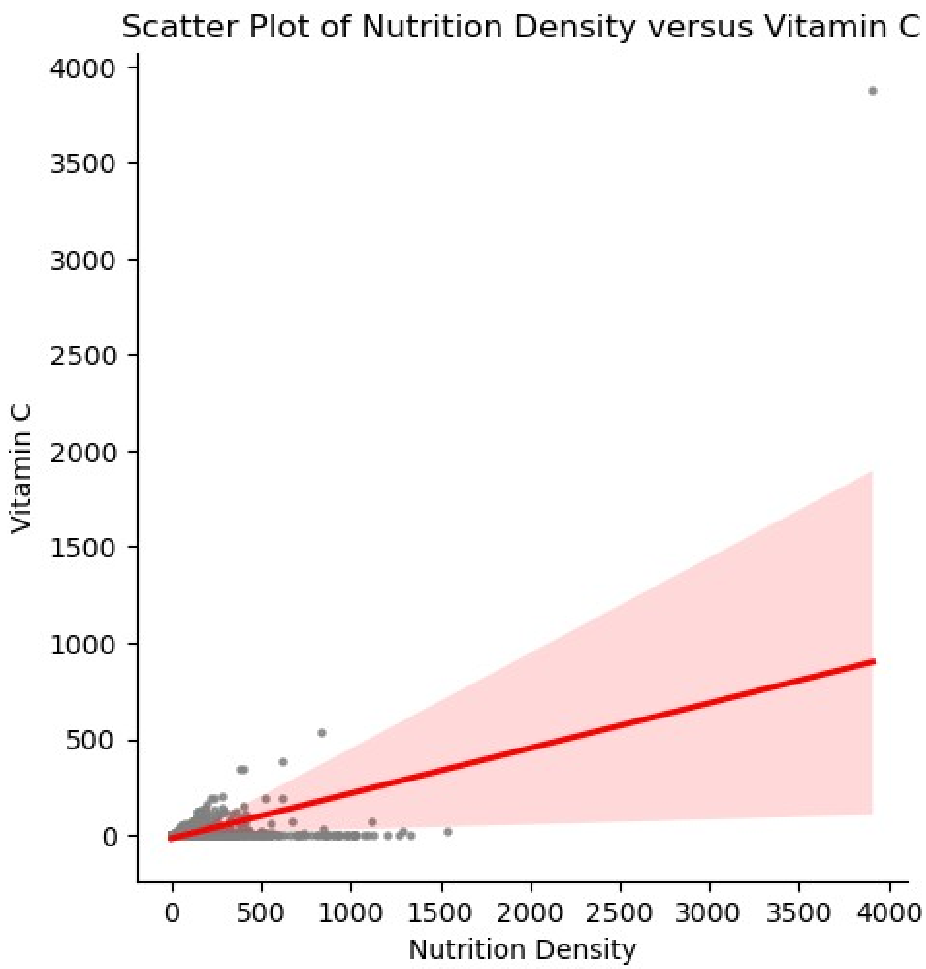


Scatter Plot of Nutrition Density versus Calcium
Calcium
Nutrition Density
0
500
1000
1500
2000
2500
3000
3500
4000

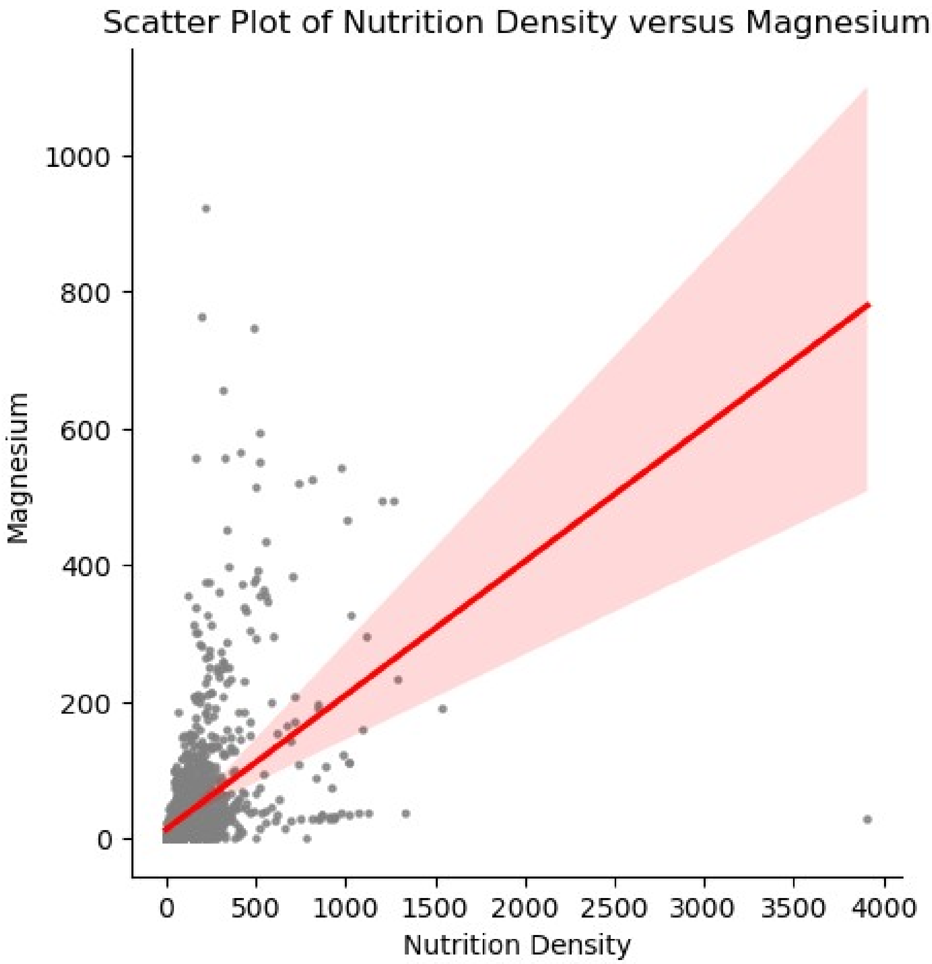

Scatter Plot of Nutrition Density versus Phosphorus
Phosphorus
6000
5000
4000
3000
2000
1000
0
0
500
1000
1500
2000
2500
3000
3500
4000
Nutrition Density

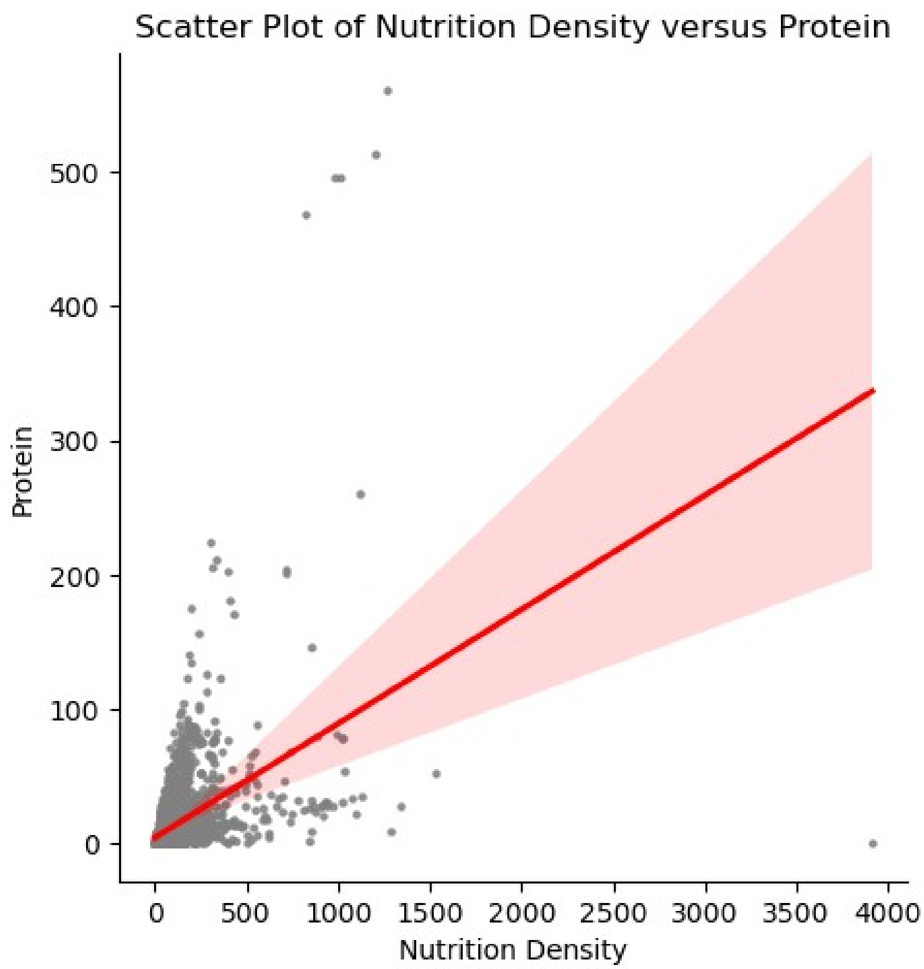

Scatter Plot of Nutrition Density versus Caloric Value
Caloric Value
Nutrition Density
0
1000
2000
3000
4000
5000
6000
7000
0
500
1000
1500
2000
2500
3000
3500
4000

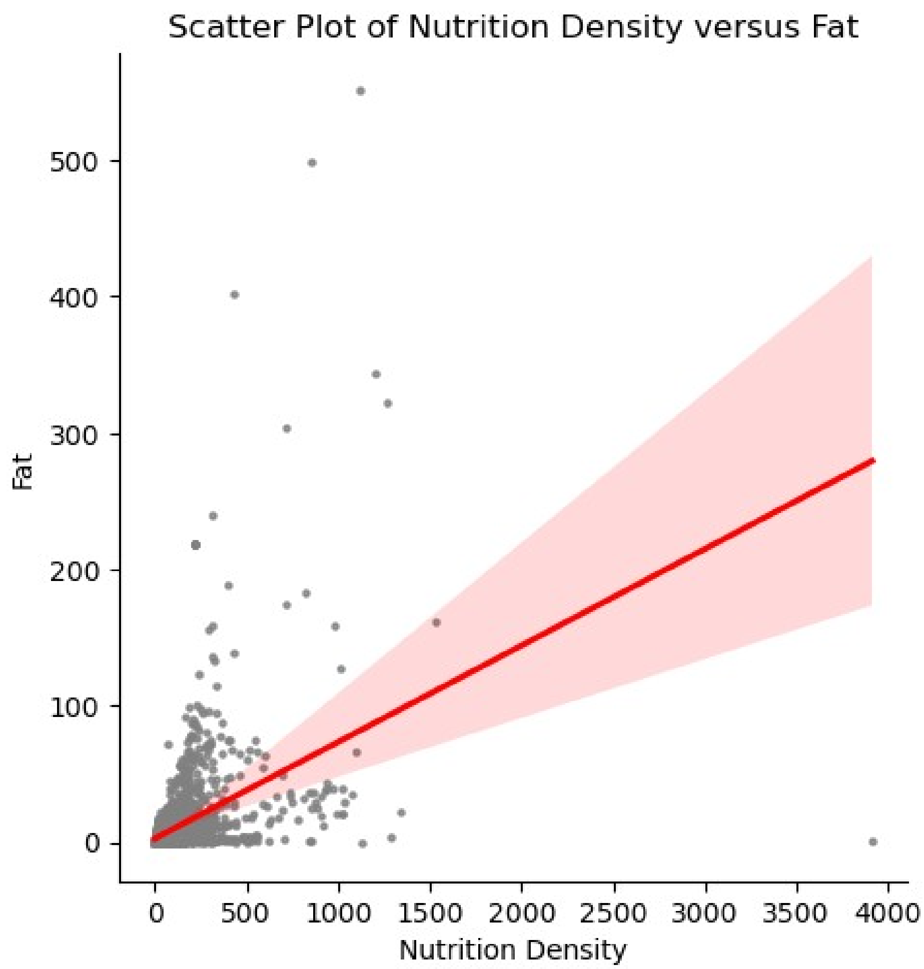

Scatter Plot of Nutrition Density versus Potassium
Potassium
0
2000
4000
6000
8000
10000
0
500
1000
1500
2000
2500
3000
3500
4000
Nutrition Density

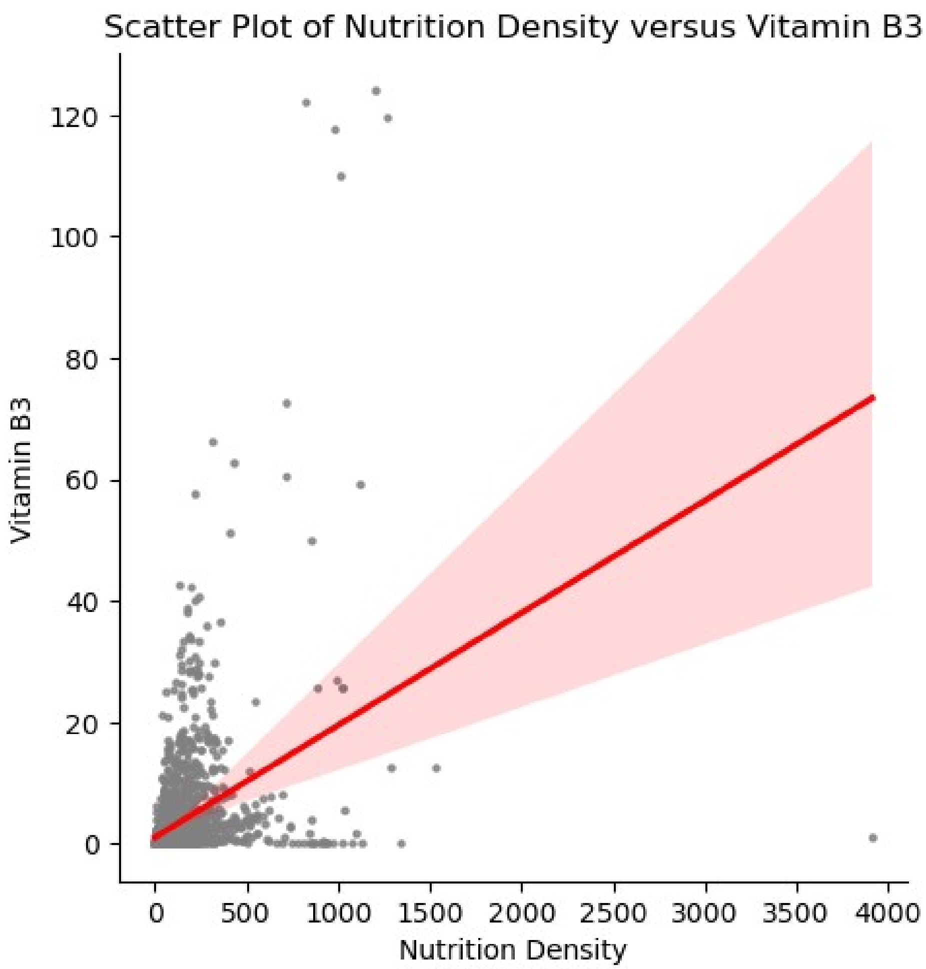


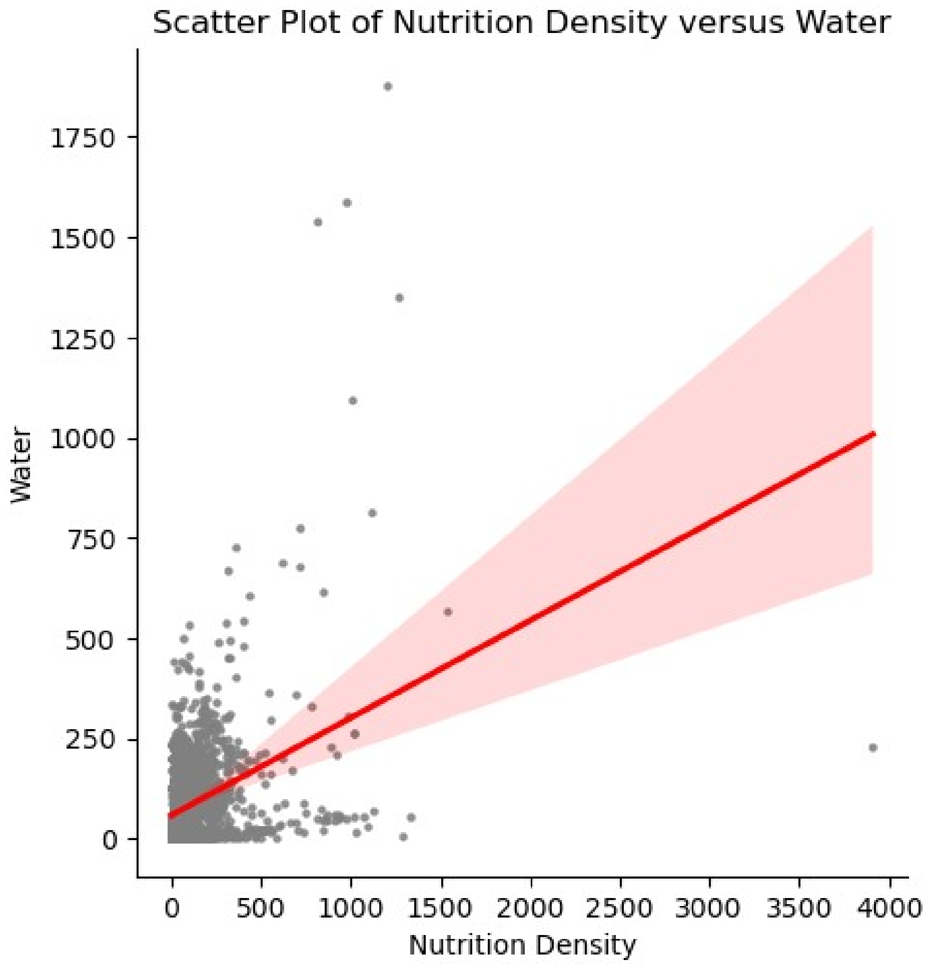
Scatter Plot of Nutrition Density versus Vitamin C
Vitamin C
Nutrition Density
0
500
1000
1500
2000
2500
3000
3500
4000


Scatter Plot of Nutrition Density versus Magnesium
Magnesium
Nutrition Density
0
200
400
600
800
1000
0
500
1000
1500
2000
2500
3000
3500
4000


Scatter Plot of Nutrition Density versus Protein
Protein
Nutrition Density
0
100
200
300
400
500
0
500
1000
1500
2000
2500
3000
3500
4000

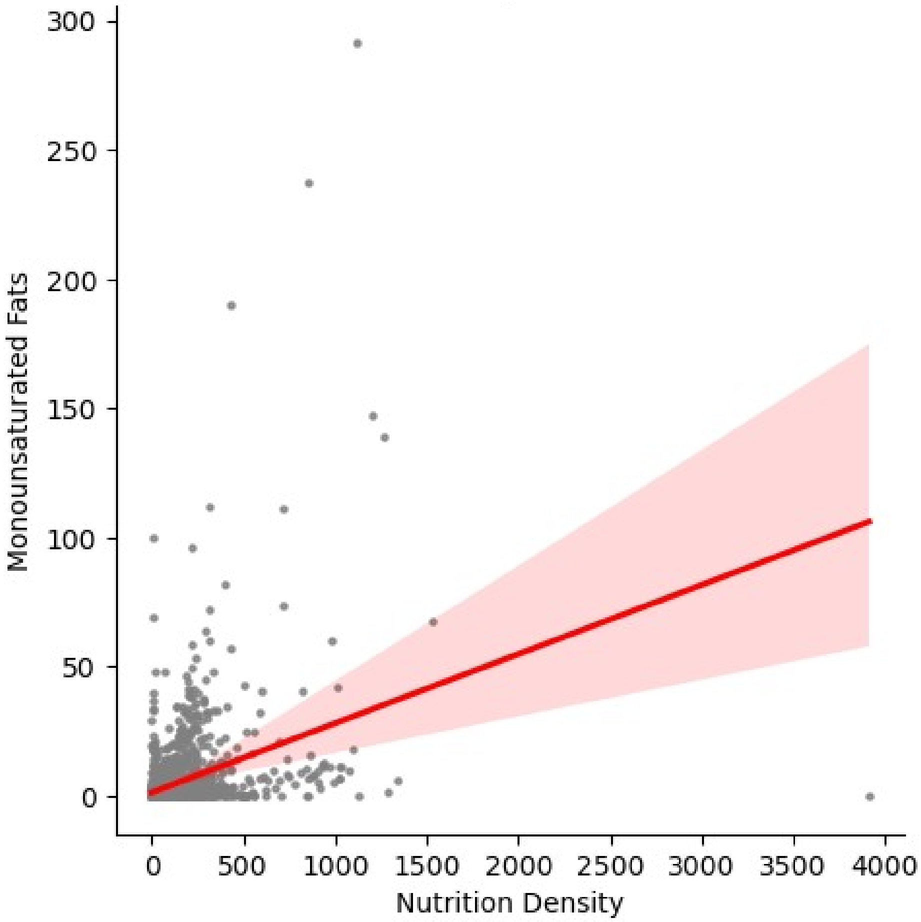

Scatter Plot of Nutrition Density versus Fat
Fat
0
100
200
300
400
500
Nutrition Density
0
500
1000
1500
2000
2500
3000
3500
4000

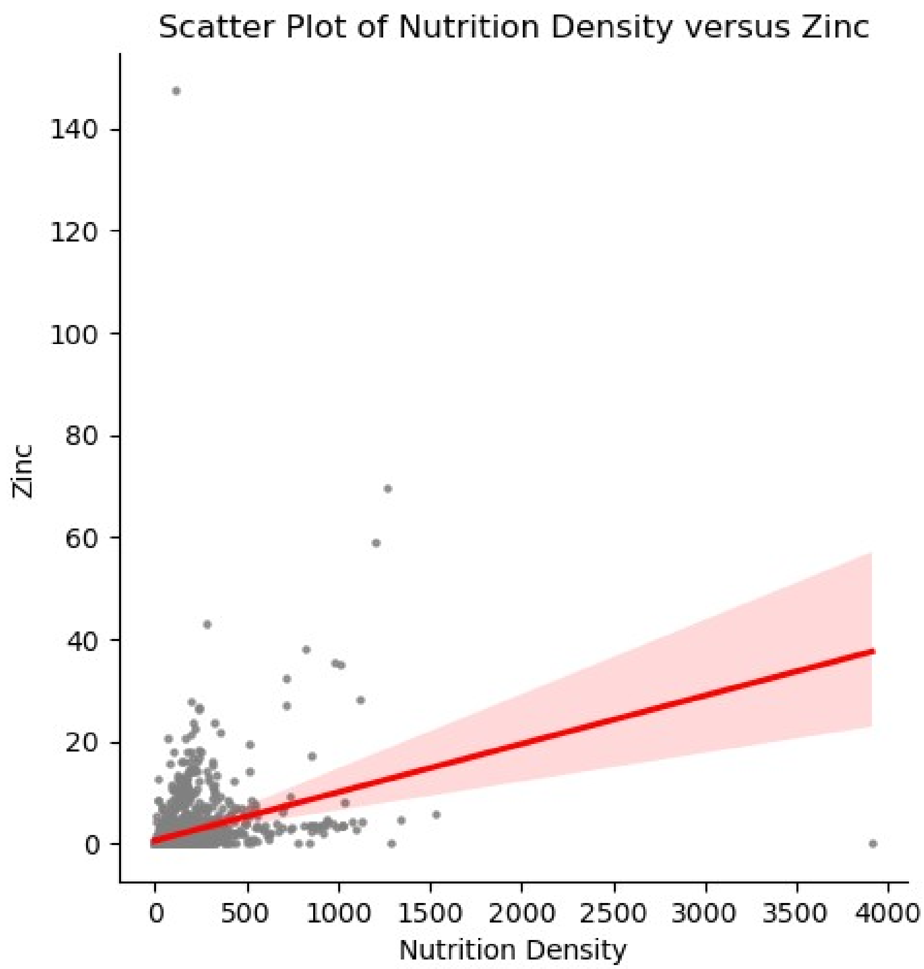

Scatter Plot of Nutrition Density versus Vitamin B3
Vitamin B3
Nutrition Density
0
20
40
60
80
100
120
0
500
1000
1500
2000
2500
3000
3500
4000

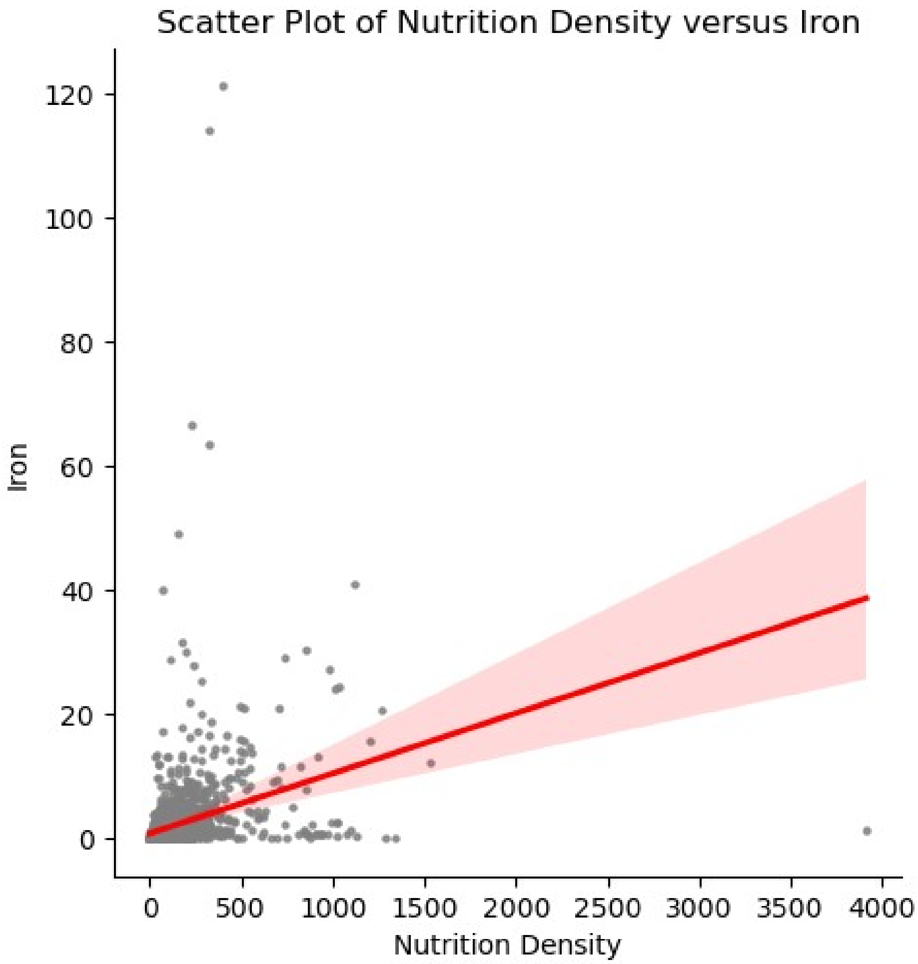


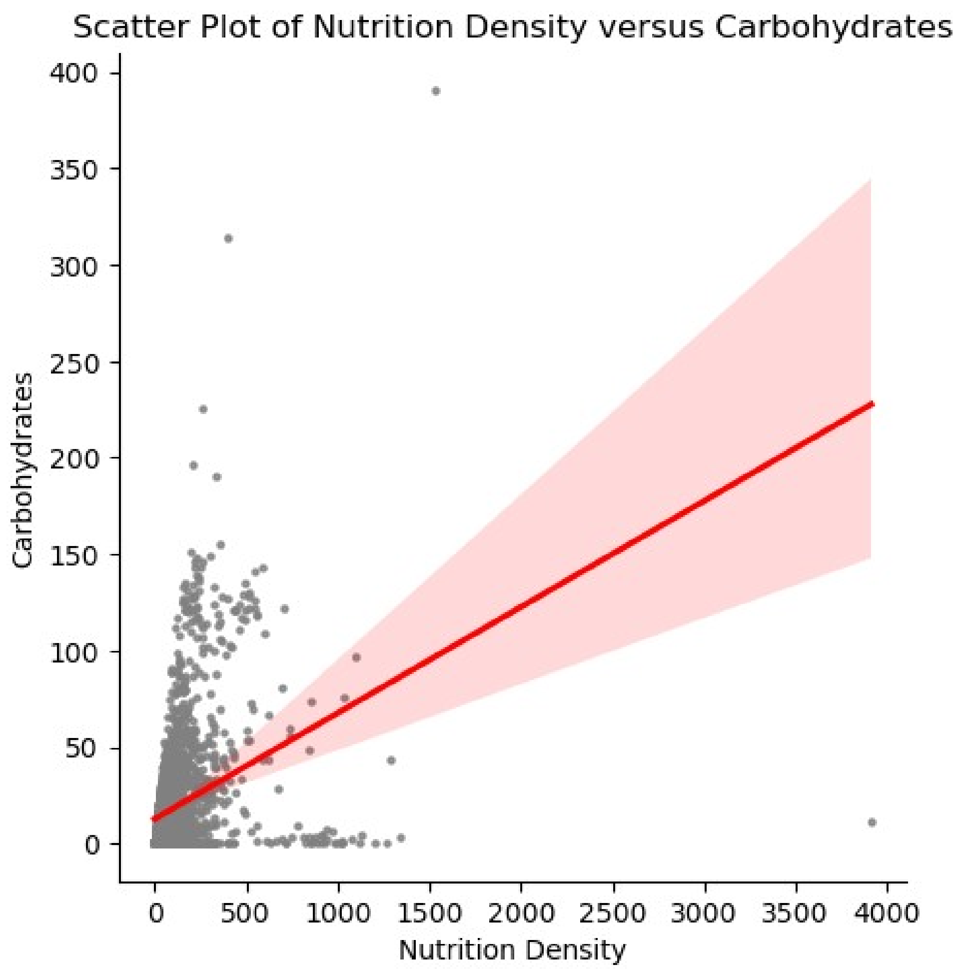
Scatter Plot of Nutrition Density versus Water
Water
1750
1500
1250
1000
750
500
250
0
0
500
1000
1500
2000
2500
3000
3500
4000
Nutrition Density

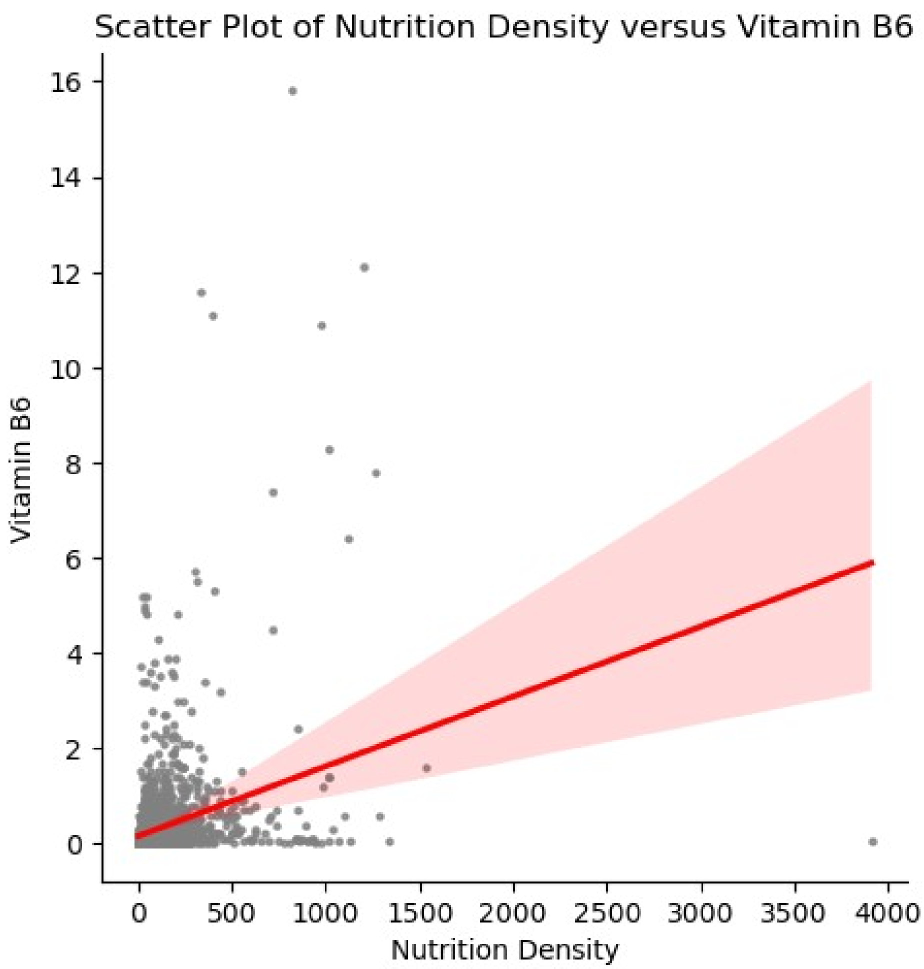
Scatter Plot of Nutrition Density versus Monounsaturated Fats



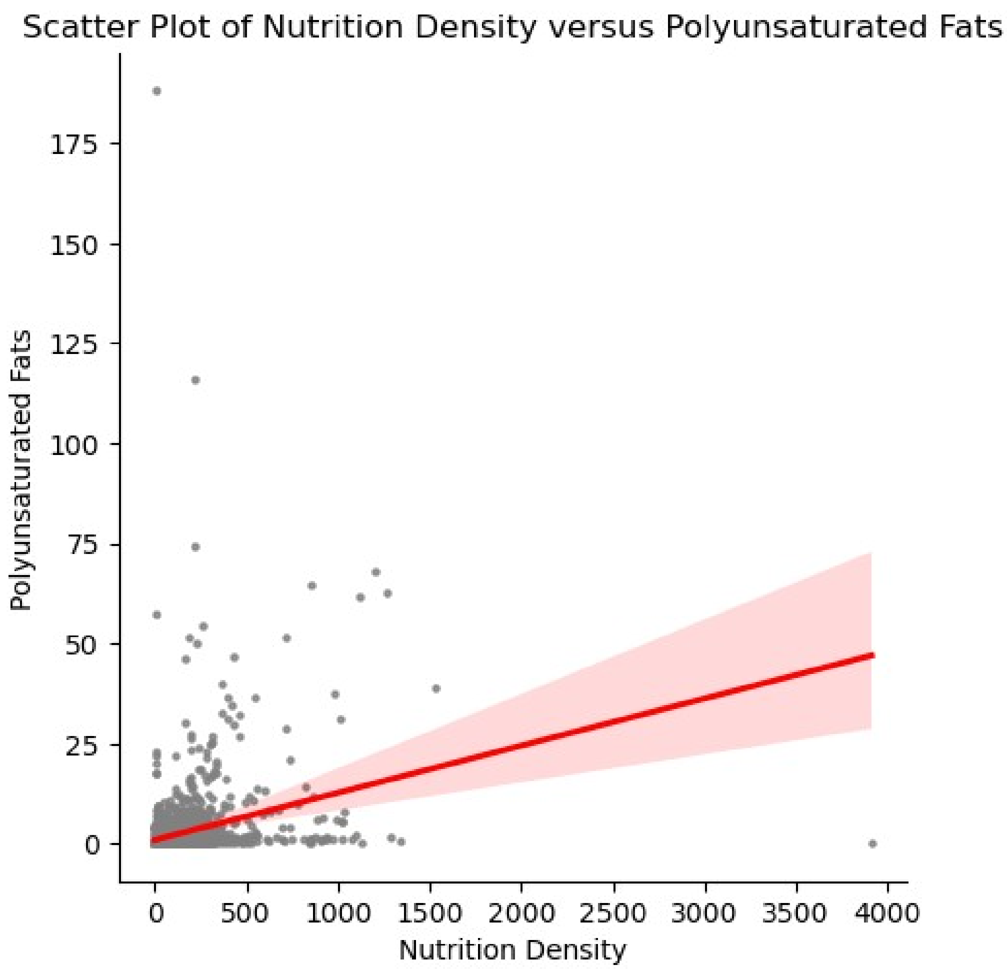

Scatter Plot of Nutrition Density versus Zinc
Zinc
0
20
40
60
80
100
120
140
0
500
1000
1500
2000
2500
3000
3500
4000
Nutrition Density

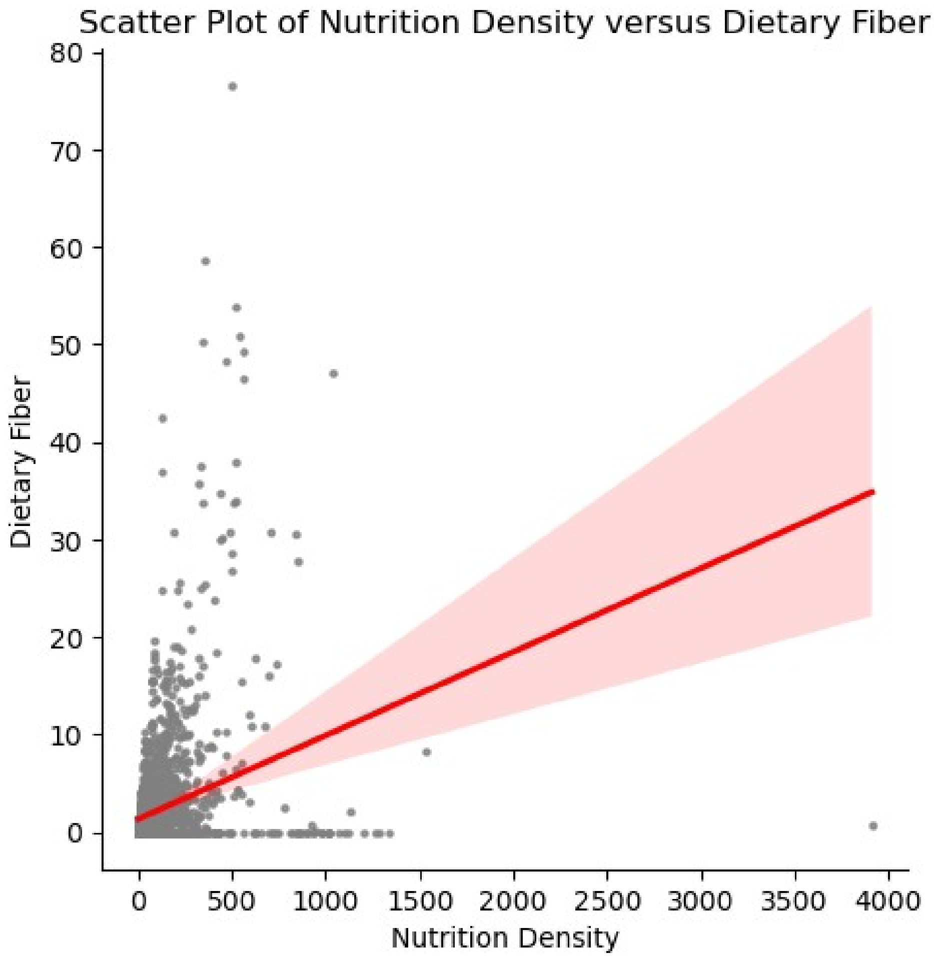

Scatter Plot of Nutrition Density versus Iron
Iron
Nutrition Density
0
20
40
60
80
100
120
0
500
1000
1500
2000
2500
3000
3500
4000

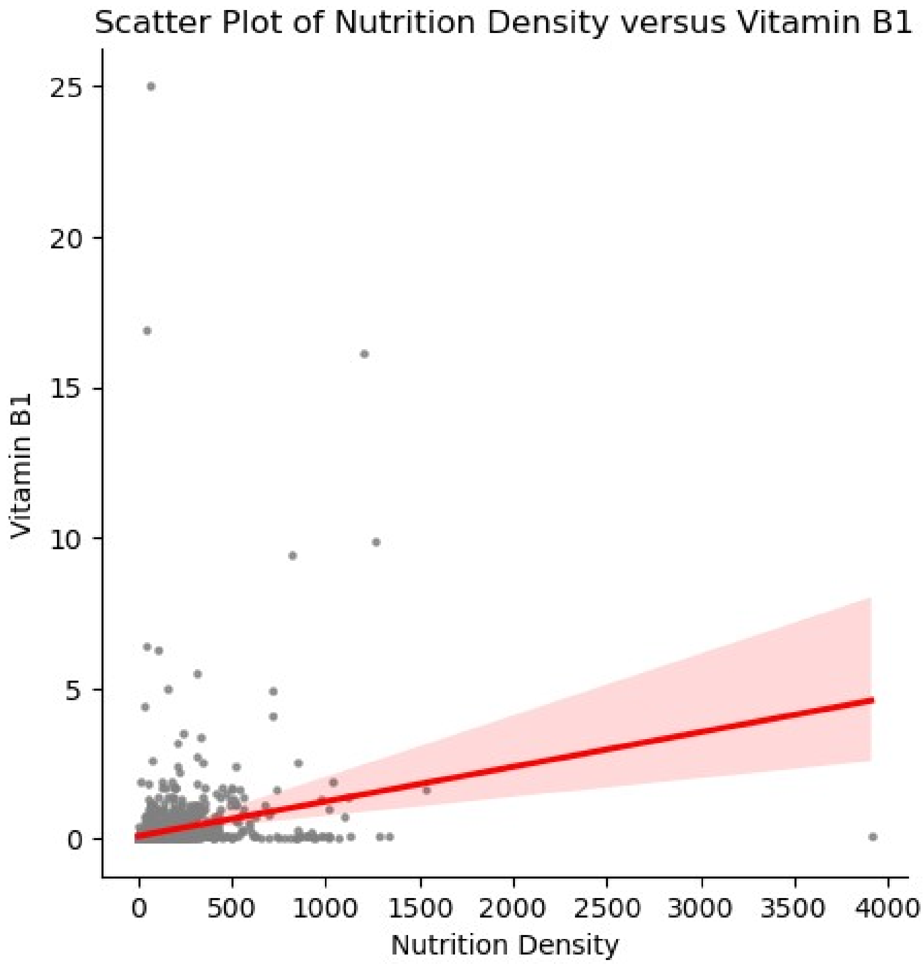

Scatter Plot of Nutrition Density versus Carbohydrates
Carbohydrates
Nutrition Density
0
50
100
150
200
250
300
350
400
0
500
1000
1500
2000
2500
3000
3500
4000

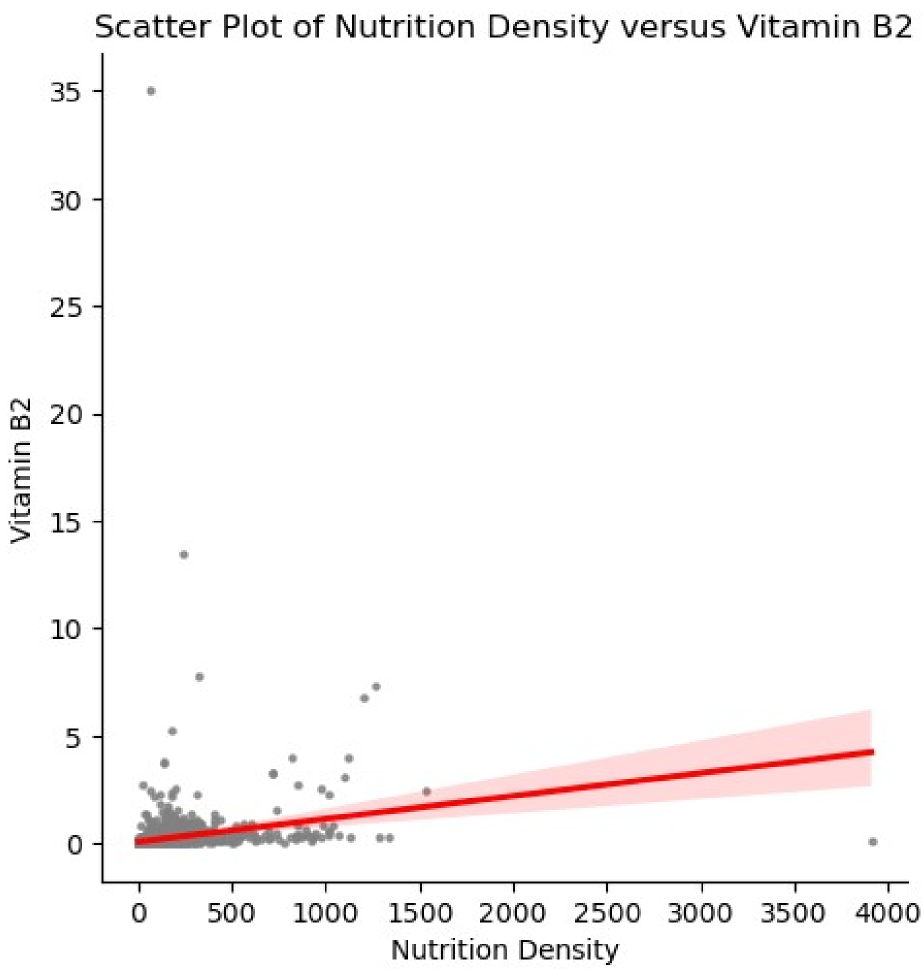


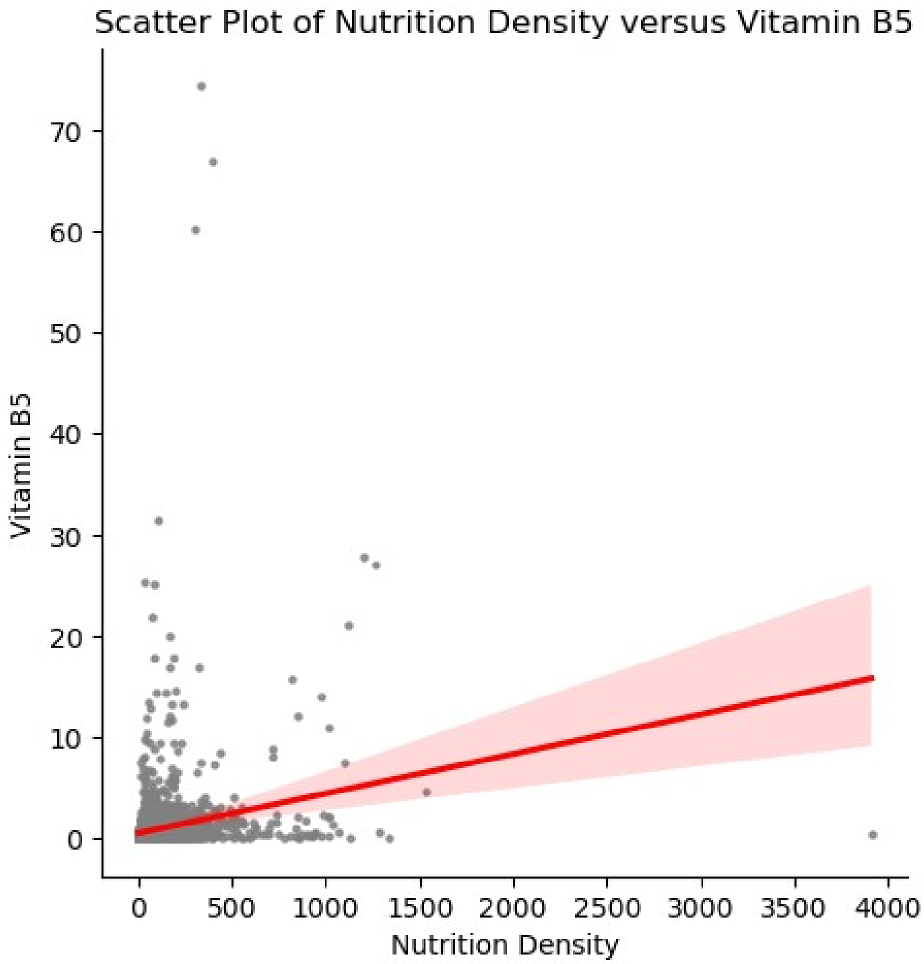
Scatter Plot of Nutrition Density versus Vitamin B6
Vitamin B6
Nutrition Density
0
2
4
6
8
10
12
14
16
0
500
1000
1500
2000
2500
3000
3500
4000


Scatter Plot of Nutrition Density versus Polyunsaturated Fats
Polyunsaturated Fats
0
25
50
75
100
125
150
175
Nutrition Density
0
500
1000
1500
2000
2500
3000
3500
4000


Scatter Plot of Nutrition Density versus Dietary Fiber
Dietary Fiber
Nutrition Density
0
10
20
30
40
50
60
70
80
0
500
1000
1500
2000
2500
3000
3500
4000


Scatter Plot of Nutrition Density versus Vitamin B1
Vitamin B1
Nutrition Density
0
5
10
15
20
25
0
500
1000
1500
2000
2500
3000
3500
4000


Scatter Plot of Nutrition Density versus Vitamin B2
Vitamin B2
Nutrition Density
0
5
10
15
20
25
30
35
0
500
1000
1500
2000
2500
3000
3500
4000



Training and Test Set Split and Model Creation

```
# The data is now ready to be split into training and test sets via
train_test_split(). I have chosen an 80/20 ratio as this
# seems to be the standard across many data science projects and the
constant ratio throughout the curriculum.

nutrition_stats = food_nutrition.drop(columns = ['Food'], axis = 1)

nutrition_features = nutrition_stats.drop(columns = ['Nutrition
Density'], axis = 1)
nutrition_density = nutrition_stats['Nutrition Density']

nutrition_xtrain, nutrition_xtest, nutrition_ytrain, nutrition_ytest =
train_test_split(nutrition_features,

nutrition_density, test_size = 0.2,

random_state = 123)

# To ensure that the split has been made successfully, I will print
```

```
the shapes of the training and test sets along with the
# nutrition features and the nutrition density variable.

print(f'The shape of the nutrition features data is
{nutrition_features.shape}.')
print(f'The shape of the nutrition density variable is
{nutrition_density.shape}.')
print(f'The shape of the nutrition features training set is
{nutrition_xtrain.shape}.')
print(f'The shape of the nutrition features test set is
{nutrition_xtest.shape}.')
print(f'The shape of the nutrition density training set is
{nutrition_ytrain.shape}.')
print(f'The shape of the nutrition density test set is
{nutrition_ytest.shape}.')
```

```
The shape of the nutrition features data is (2395, 33).
The shape of the nutrition density variable is (2395,).
The shape of the nutrition features training set is (1916, 33).
The shape of the nutrition features test set is (479, 33).
The shape of the nutrition density training set is (1916,).
The shape of the nutrition density test set is (479,).
```

```
# As stated earlier, the first model will be an ordinary least squares
model. Since many things need to happen within the
# process of crafting the OLS model, I will encase everything needed
into a class object and a function. The class object
# will define the creation of the OLS model, the predictions that will
come from the model, and the R-squared value, a
# metric that can be used to see the percentage of variance within the
data explained by the model.

class OLSWrapper(BaseEstimator, RegressorMixin):
    def __init__(self):
        self.model = None

    def fit(self, X, y):
        X = sm.add_constant(X)
        self.model = sm.OLS(y, X).fit()
        print(self.model.summary())
        return self

    def predict(self, X):
        X = sm.add_constant(X)
        return self.model.predict(X)

    def score(self, X, y):
        predictions = self.predict(X)
        return 1 - np.sum((y - predictions) ** 2) / np.sum((y -
np.mean(y)) ** 2)
```

```
# Now that the OLS Wrapper class object has been defined, I can craft
my pipeline using the StandardScaler() and
# OLSWrapper() functions.

pipe = Pipeline([('scale', StandardScaler()), ('model',
OLSWrapper())])

# I need to set up a search space so that Ridge regression can be
added to the model creation.

search_space = [{'model': [OLSWrapper()]},
               {'model': [Ridge(max_iter=10000)],
                'model__alpha': np.logspace(-4, 4, 50)}]

# Using GridSearchCV(), I can create the grid search using the
pipeline and search space, cross-validate the model created
# five times (I am choosing 5 for the cross-validation 'cv' argument),
and have the models all scored with the R-squared
# value using the 'scoring' argument.

grid_search = GridSearchCV(pipe, search_space, cv = 5, scoring = 'r2')
gs = grid_search.fit(nutrition_xtrain, nutrition_ytrain)

# With the grid search fitting the data and cross-validating the OLS
model normalized by Ridge regression five times, I will
# print the best estimator model by accessing the grid search
attrributes.

best_model = gs.best_estimator_
print(f'Best estimator: {best_model}')

# Now that the best OLS model can be shown, I can also display the
model performance and significance statistics attached to
# it, those values being both R-squared and root mean squared error
(RMSE).

pred = best_model.predict(nutrition_xtest)
r2 = r2_score(nutrition_ytest, pred)
rmse = np.sqrt(mean_squared_error(nutrition_ytest, pred))
print(f"R-squared: {r2}")
print(f"RMSE: {rmse}")

                            OLS Regression Results

======================================================================
========
Dep. Variable:      Nutrition Density   R-squared:
1.000
Model:                            OLS   Adj. R-squared:
1.000
```

```
Method:                 Least Squares   F-statistic:
1.111e+09
Date:                Sat, 14 Sep 2024   Prob (F-statistic):
0.00
Time:                        02:14:26   Log-Likelihood:
2895.9
No. Observations:                1532   AIC:
-5724.
Df Residuals:                    1498   BIC:
-5542.
Df Model:                          33

Covariance Type:            nonrobust

======================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
--------
const        107.7279      0.001   1.14e+05      0.000     107.726
107.730
x1             0.0072      0.019      0.373      0.709      -0.031
0.045
x2            30.5761      0.009   3271.091      0.000      30.558
30.594
x3            -0.0013      0.004     -0.319      0.750      -0.009
0.007
x4         -2.758e-05      0.004     -0.007      0.994      -0.007
0.007
x5            -0.0006      0.001     -0.459      0.646      -0.003
0.002
x6            29.4222      0.006   5044.854      0.000      29.411
29.434
x7             0.0004      0.001      0.313      0.754      -0.002
0.003
x8            34.5923      0.009   4007.671      0.000      34.575
34.609
x9             4.9943      0.002   3187.280      0.000       4.991
4.997
x10           -0.0002      0.001     -0.178      0.859      -0.002
0.002
x11           -0.0004      0.001     -0.375      0.707      -0.003
0.002
x12            0.0014      0.002      0.890      0.373      -0.002
0.005
x13            8.1047      0.001   7947.082      0.000       8.103
8.107
x14         5.474e-05      0.002      0.032      0.975      -0.003
```

```
0.003
x15           -0.0088      0.013     -0.679      0.497      -0.034
0.017
x16            0.0087      0.013      0.647      0.518      -0.018
0.035
x17           -0.0001      0.003     -0.036      0.971      -0.006
0.005
x18            0.0011      0.003      0.392      0.695      -0.005
0.007
x19            0.0013      0.002      0.552      0.581      -0.003
0.006
x20           -0.0020      0.003     -0.760      0.448      -0.007
0.003
x21          101.7161      0.001   1.07e+05      0.000     101.714
101.718
x22           -0.0014      0.001     -1.270      0.204      -0.003
0.001
x23           -0.0002      0.001     -0.161      0.872      -0.002
0.002
x24           -0.0011      0.001     -1.069      0.285      -0.003
0.001
x25          113.4886      0.001   9.52e+04      0.000     113.486
113.491
x26            0.0014      0.001      1.070      0.285      -0.001
0.004
x27            4.7880      0.001   3829.486      0.000       4.786
4.790
x28            0.0017      0.002      0.908      0.364      -0.002
0.005
x29        -7.425e-05      0.002     -0.035      0.972      -0.004
0.004
x30           -0.0008      0.004     -0.213      0.831      -0.008
0.007
x31           -0.0001      0.002     -0.053      0.958      -0.005
0.004
x32            0.0001      0.002      0.057      0.954      -0.004
0.005
x33           -0.0006      0.002     -0.330      0.741      -0.004
0.003
======================================================================
========
Omnibus:                      157.790   Durbin-Watson:
1.951
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
1214.245
Skew:                          -0.030   Prob(JB):
2.14e-264
Kurtosis:                       7.361   Cond. No.
76.3
```

```
==============================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
                            OLS Regression Results

==============================================================================
========
Dep. Variable:      Nutrition Density   R-squared:
1.000
Model:                            OLS   Adj. R-squared:
1.000
Method:                 Least Squares   F-statistic:
1.126e+09
Date:                Sat, 14 Sep 2024   Prob (F-statistic):
0.00
Time:                        02:14:27   Log-Likelihood:
2944.1
No. Observations:                1533   AIC:
-5820.
Df Residuals:                    1499   BIC:
-5639.
Df Model:                          33

Covariance Type:            nonrobust

==============================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
--------
const        106.5260      0.001   1.16e+05      0.000     106.524
106.528
x1            -0.0067      0.013     -0.521      0.602      -0.032
0.018
x2            31.8629      0.007   4475.730      0.000      31.849
31.877
x3             0.0011      0.002      0.445      0.657      -0.004
0.006
x4             0.0022      0.003      0.685      0.494      -0.004
0.009
x5            -0.0003      0.001     -0.251      0.802      -0.003
0.002
x6            28.7821      0.004   7446.717      0.000      28.775
28.790
x7             0.0005      0.001      0.392      0.695      -0.002
0.003
```

x8	34.3930	0.005	6805.555	0.000	34.383	34.403
x9	5.5549	0.001	3747.080	0.000	5.552	5.558
x10	-7.688e-05	0.001	-0.071	0.943	-0.002	0.002
x11	-0.0001	0.001	-0.136	0.892	-0.002	0.002
x12	0.0012	0.001	0.798	0.425	-0.002	0.004
x13	12.6584	0.001	1.11e+04	0.000	12.656	12.661
x14	-0.0002	0.001	-0.118	0.906	-0.003	0.003
x15	-2.288e-05	0.001	-0.025	0.980	-0.002	0.002
x16	0.0003	0.001	0.273	0.785	-0.002	0.002
x17	5.347e-05	0.001	0.037	0.971	-0.003	0.003
x18	0.0019	0.003	0.686	0.493	-0.003	0.007
x19	0.0002	0.002	0.087	0.931	-0.003	0.004
x20	-0.0005	0.002	-0.201	0.840	-0.005	0.004
x21	101.7471	0.001	1.1e+05	0.000	101.745	101.749
x22	-0.0010	0.001	-0.950	0.342	-0.003	0.001
x23	-0.0001	0.001	-0.096	0.924	-0.002	0.002
x24	-0.0010	0.001	-1.042	0.298	-0.003	0.001
x25	105.5202	0.001	9.39e+04	0.000	105.518	105.522
x26	0.0002	0.001	0.174	0.862	-0.002	0.003
x27	4.6796	0.001	3754.790	0.000	4.677	4.682
x28	0.0010	0.002	0.531	0.596	-0.003	0.004
x29	-0.0005	0.002	-0.223	0.824	-0.005	0.004
x30	-0.0006	0.003	-0.193	0.847	-0.007	0.006
x31	-0.0009	0.002	-0.404	0.686	-0.005	0.003
x32	0.0006	0.002	0.268	0.789	-0.004	

```
0.005
x33           -0.0002      0.002     -0.100      0.920      -0.004
0.003
======================================================================
========
Omnibus:                      154.734   Durbin-Watson:
1.934
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
1133.010
Skew:                          -0.080   Prob(JB):
9.33e-247
Kurtosis:                       7.209   Cond. No.
52.9
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.

                            OLS Regression Results

======================================================================
========
Dep. Variable:      Nutrition Density   R-squared:
1.000
Model:                            OLS   Adj. R-squared:
1.000
Method:                 Least Squares   F-statistic:
1.172e+09
Date:                Sat, 14 Sep 2024   Prob (F-statistic):
0.00
Time:                        02:14:27   Log-Likelihood:
2937.4
No. Observations:                1533   AIC:
-5807.
Df Residuals:                    1499   BIC:
-5625.
Df Model:                          33

Covariance Type:            nonrobust

======================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
--------
const        109.3162      0.001   1.19e+05      0.000     109.314
109.318
```

x1	0.0063	0.013	0.503	0.615	-0.018	0.031
x2	27.6459	0.006	4524.813	0.000	27.634	27.658
x3	-0.0015	0.003	-0.435	0.664	-0.008	0.005
x4	-0.0010	0.003	-0.352	0.725	-0.007	0.005
x5	-0.0003	0.001	-0.226	0.821	-0.003	0.003
x6	29.0358	0.004	6868.334	0.000	29.028	29.044
x7	0.0010	0.001	0.841	0.401	-0.001	0.003
x8	30.2969	0.005	6475.517	0.000	30.288	30.306
x9	5.7938	0.002	3791.603	0.000	5.791	5.797
x10	-0.0003	0.001	-0.281	0.778	-0.002	0.002
x11	-0.0002	0.001	-0.175	0.861	-0.002	0.002
x12	0.0011	0.001	0.806	0.421	-0.002	0.004
x13	11.8080	0.001	1e+04	0.000	11.806	11.810
x14	0.0003	0.002	0.162	0.872	-0.003	0.003
x15	-3.301e-05	0.001	-0.031	0.975	-0.002	0.002
x16	-0.0009	0.003	-0.302	0.763	-0.006	0.005
x17	0.0008	0.003	0.269	0.788	-0.005	0.006
x18	0.0007	0.002	0.287	0.774	-0.004	0.005
x19	1.908e-06	0.002	0.001	0.999	-0.004	0.004
x20	-0.0005	0.002	-0.215	0.830	-0.005	0.004
x21	102.6561	0.001	1.1e+05	0.000	102.654	102.658
x22	-0.0002	0.001	-0.204	0.838	-0.002	0.002
x23	3.203e-05	0.001	0.031	0.975	-0.002	0.002
x24	-0.0009	0.001	-0.908	0.364	-0.003	0.001
x25	116.6813	0.001	1.01e+05	0.000	116.679	

```
116.684
x26             0.0017      0.001      1.155      0.248      -0.001
0.004
x27             3.7882      0.002   2466.833      0.000       3.785
3.791
x28             0.0004      0.002      0.200      0.842      -0.003
0.004
x29            -0.0013      0.002     -0.536      0.592      -0.006
0.003
x30             0.0006      0.003      0.188      0.851      -0.005
0.006
x31            -0.0006      0.002     -0.342      0.732      -0.004
0.003
x32             0.0003      0.002      0.139      0.890      -0.004
0.005
x33            -0.0014      0.002     -0.798      0.425      -0.005
0.002
======================================================================
========
Omnibus:                      158.885   Durbin-Watson:
1.953
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
1226.540
Skew:                          -0.050   Prob(JB):
4.57e-267
Kurtosis:                       7.381   Cond. No.
47.9
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
                            OLS Regression Results

======================================================================
========
Dep. Variable:      Nutrition Density   R-squared:
1.000
Model:                            OLS   Adj. R-squared:
1.000
Method:                 Least Squares   F-statistic:
1.134e+09
Date:                Sat, 14 Sep 2024   Prob (F-statistic):
0.00
Time:                        02:14:27   Log-Likelihood:
2910.7
No. Observations:                1533   AIC:
-5753.
```

```
Df Residuals:                    1499   BIC:
-5572.
Df Model:                          33

Covariance Type:            nonrobust

======================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
--------
const        107.4200      0.001   1.15e+05      0.000     107.418
107.422
x1            -0.0025      0.013     -0.196      0.844      -0.027
0.022
x2            30.7091      0.007   4683.143      0.000      30.696
30.722
x3             0.0009      0.003      0.277      0.782      -0.005
0.007
x4             0.0005      0.003      0.179      0.858      -0.005
0.006
x5            -0.0004      0.001     -0.293      0.769      -0.003
0.002
x6            29.6182      0.004   7362.120      0.000      29.610
29.626
x7             0.0006      0.001      0.479      0.632      -0.002
0.003
x8            28.7150      0.004   6528.724      0.000      28.706
28.724
x9             5.3267      0.001   3644.679      0.000       5.324
5.330
x10           -0.0002      0.001     -0.231      0.817      -0.002
0.002
x11         1.621e-05      0.001      0.017      0.987      -0.002
0.002
x12            0.0018      0.001      1.279      0.201      -0.001
0.005
x13           12.7591      0.001   1.08e+04      0.000      12.757
12.761
x14            0.0002      0.002      0.120      0.904      -0.003
0.004
x15         2.356e-05      0.001      0.022      0.983      -0.002
0.002
x16            0.0006      0.003      0.204      0.838      -0.005
0.006
x17           -0.0009      0.003     -0.298      0.766      -0.006
0.005
x18            0.0003      0.002      0.114      0.910      -0.004
```

```
0.005
x19              0.0012      0.002      0.590      0.555      -0.003
0.005
x20             -0.0019      0.002     -0.811      0.418      -0.006
0.003
x21            102.4566      0.001   1.08e+05      0.000     102.455
102.458
x22             -0.0011      0.001     -1.050      0.294      -0.003
0.001
x23          -4.586e-05      0.001     -0.043      0.965      -0.002
0.002
x24             -0.0010      0.001     -1.022      0.307      -0.003
0.001
x25            116.1777      0.001   1.03e+05      0.000     116.175
116.180
x26              0.0006      0.001      0.423      0.673      -0.002
0.003
x27              4.7450      0.001   3879.037      0.000       4.743
4.747
x28              0.0006      0.002      0.358      0.720      -0.003
0.004
x29              0.0013      0.002      0.636      0.525      -0.003
0.005
x30              0.0001      0.003      0.037      0.970      -0.006
0.006
x31             -0.0006      0.002     -0.319      0.749      -0.005
0.003
x32          -7.478e-05      0.002     -0.035      0.972      -0.004
0.004
x33              0.0015      0.002      0.779      0.436      -0.002
0.005
======================================================================
========
Omnibus:                      152.053   Durbin-Watson:
1.990
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
1037.576
Skew:                          -0.136   Prob(JB):
4.93e-226
Kurtosis:                       7.021   Cond. No.
47.2
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.

                            OLS Regression Results
```

```
==============================================================================
========
Dep. Variable:      Nutrition Density   R-squared:
1.000
Model:                            OLS   Adj. R-squared:
1.000
Method:                 Least Squares   F-statistic:
7.959e+08
Date:                Sat, 14 Sep 2024   Prob (F-statistic):
0.00
Time:                        02:14:27   Log-Likelihood:
2930.5
No. Observations:                1533   AIC:
-5793.
Df Residuals:                    1499   BIC:
-5612.
Df Model:                          33

Covariance Type:            nonrobust

==============================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
--------
const        105.1559      0.001   1.14e+05      0.000     105.154
105.158
x1            -0.0004      0.012     -0.033      0.973      -0.024
0.023
x2            29.1762      0.006   4726.062      0.000      29.164
29.188
x3             0.0002      0.003      0.079      0.937      -0.006
0.006
x4             0.0008      0.003      0.285      0.776      -0.005
0.006
x5            -0.0006      0.001     -0.483      0.629      -0.003
0.002
x6            27.5388      0.004   7454.634      0.000      27.532
27.546
x7             0.0008      0.001      0.710      0.478      -0.001
0.003
x8            34.0200      0.005   6772.400      0.000      34.010
34.030
x9             5.4590      0.002   3478.309      0.000       5.456
5.462
x10            0.0002      0.001      0.208      0.835      -0.002
0.002
x11           -0.0002      0.001     -0.217      0.828      -0.002
```

```
0.002
x12            0.0011      0.002      0.701      0.483      -0.002
0.004
x13           12.6743      0.001   1.08e+04      0.000      12.672
12.677
x14           -0.0008      0.002     -0.480      0.631      -0.004
0.003
x15        -9.269e-05      0.001     -0.087      0.931      -0.002
0.002
x16           -0.0007      0.004     -0.158      0.875      -0.009
0.008
x17            0.0012      0.004      0.274      0.784      -0.008
0.010
x18            0.0021      0.003      0.716      0.474      -0.004
0.008
x19            0.0009      0.002      0.535      0.593      -0.003
0.004
x20          8.83e-05      0.002      0.038      0.969      -0.004
0.005
x21           26.4332      0.001   2.67e+04      0.000      26.431
26.435
x22            0.0004      0.001      0.347      0.728      -0.002
0.002
x23           -0.0002      0.001     -0.178      0.859      -0.002
0.002
x24           -0.0022      0.001     -1.846      0.065      -0.005
0.000
x25          111.6601      0.001   9.88e+04      0.000     111.658
111.662
x26            0.0016      0.002      0.968      0.333      -0.002
0.005
x27            4.3115      0.001   3680.297      0.000       4.309
4.314
x28            0.0014      0.002      0.780      0.435      -0.002
0.005
x29            0.0007      0.003      0.293      0.770      -0.004
0.006
x30           -0.0032      0.004     -0.917      0.359      -0.010
0.004
x31           -0.0012      0.002     -0.544      0.586      -0.006
0.003
x32           -0.0013      0.002     -0.542      0.588      -0.006
0.003
x33           -0.0004      0.002     -0.194      0.846      -0.004
0.003
======================================================================
========
Omnibus:                      142.247   Durbin-Watson:
1.954
```

```
Prob(Omnibus):                    0.000   Jarque-Bera (JB):
874.648
Skew:                             0.147   Prob(JB):
1.18e-190
Kurtosis:                         6.689   Cond. No.
47.4
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
Best estimator: Pipeline(steps=[('scale', StandardScaler()),
                ('model', Ridge(alpha=0.0029470517025518097,
max_iter=10000))])
R-squared: 0.999999961641675
RMSE: 0.032870538744169224

# Based on each of the five OLS model performances, we can see that
they all are overfit by the perfect R-squared value of
# 1.000 and the equally perfect Prob (F-Statistic) metric of 0.000.
The best OLS estimator's performance metrics are also
# indicative of overfitting as R-squared is almost 1.000 and the RMSE
is almost zero, meaning that the model's predictions
# are almost exactly the same as the actual observations. This fact
rules out OLS as a viable model choice, which takes
# me to the stacking regressor method I have chosen as a backup
consisting of a decision tree regressor, a random
# forest regressor, and a gradient boosting regressor. I have encased
the necessary model choices in a function that will
# output the best-performing base model metrics and the features that
are impactful to their results as well as the
# statistics for the stacking regressor as a whole.

def run_ensemble_model():

# Here I have defined the regressors and attached the proper names to
their functions via the estimators variable I created.

    tree = DecisionTreeRegressor(random_state = 123)
    forest = RandomForestRegressor(random_state = 123)
    booster = GradientBoostingRegressor(random_state = 123)

    param_grid_tree = {'max_depth': [None, 10, 20, 30],
'min_samples_split': [2, 5, 10]}
    param_grid_forest = {'n_estimators': [50, 100, 200], 'max_depth':
[None, 10, 20]}
    param_grid_booster = {'n_estimators': [100, 200], 'learning_rate':
[0.01, 0.1], 'max_depth': [3, 5]}
```

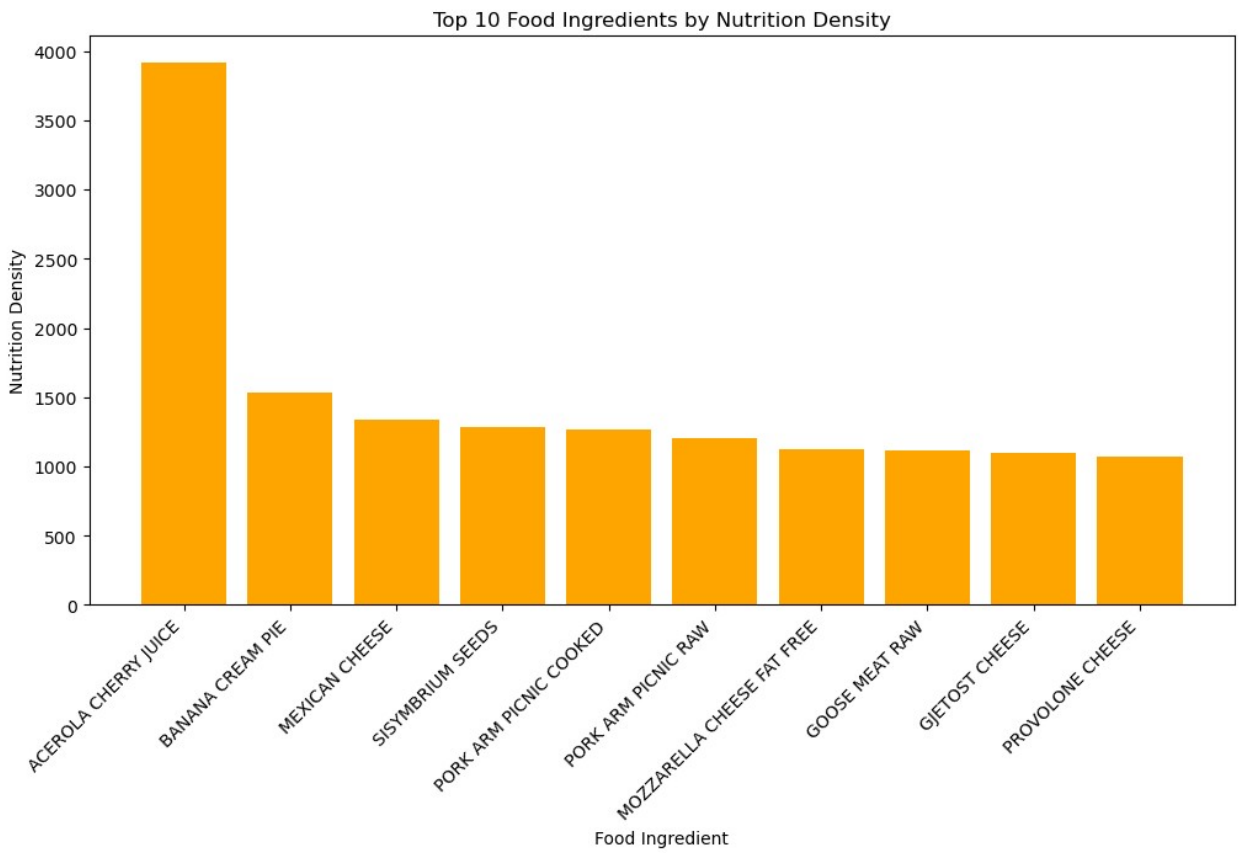
```
    grid_search_tree = GridSearchCV(tree, param_grid_tree, cv = 5)
    grid_search_forest = GridSearchCV(forest, param_grid_forest, cv =
5)
    grid_search_booster = GridSearchCV(booster, param_grid_booster, cv
= 5)

    grid_search_tree.fit(nutrition_xtrain, nutrition_ytrain)
    grid_search_forest.fit(nutrition_xtrain, nutrition_ytrain)
    grid_search_booster.fit(nutrition_xtrain, nutrition_ytrain)

    best_tree = grid_search_tree.best_estimator_
    best_forest = grid_search_forest.best_estimator_
    best_booster = grid_search_booster.best_estimator_

    estimators = [('Decision Tree', best_tree), ('Random Forest',
best_forest), ('Gradient Boosting', best_booster)]

# The stacking regressor function is then filed with my chosen models
and set to perform a five-fold cross-validation.
# Predictions are made using both the training and test sets.

    stacking_reg = StackingRegressor(estimators = estimators, cv = 5)
    stacking_reg.fit(nutrition_xtrain, nutrition_ytrain)
    ypred_train = stacking_reg.predict(nutrition_xtrain)
    ypred_test = stacking_reg.predict(nutrition_xtest)

# I have crafted two for loops: one that allows the printing of the
best estimator for each model choice along with the
# RMSE and R-squared values, and one that outputs the model's top five
most important features along with their
# corresponding coefficients.

    print("Best performing base estimators and their performance
metrics:")
    for name, model in stacking_reg.named_estimators_.items():
        print(f"\n{name} Best Estimator:\n")
        print(model.get_params())

        ypred = model.predict(nutrition_xtest)
        rmse = np.sqrt(mean_squared_error(nutrition_ytest, ypred))
        r2 = r2_score(nutrition_ytest, ypred)
        print(f"\nPerformance Metrics for {name}:\n")
        print(f"     RMSE: {rmse}")
        print(f"     R-squared: {r2}")

        importances = model.feature_importances_
        feature_names = nutrition_xtrain.columns
        sorted_indices = np.argsort(importances)[::-1]
        print("\nFeature importances:\n")
        for idx in sorted_indices[:5]:
```

```
            print(f"     {feature_names[idx]}:
{importances[idx]:.4f}")

# The final block of code in the stacking regressor function is meant
to display the performance metrics of the stacking
# regressor, which starts with the creation of the decision tree
model, allows the random forest model to be made using the
# information learned from the decision tree, and ends with the
gradient boosting regressor turning several weak decision
# trees and random forests into a strong stacked ensemble model.

    rmse = np.sqrt(mean_squared_error(nutrition_ytest, ypred_test))
    r2 = r2_score(nutrition_ytest, ypred_test)
    print('\nStacking regressor performance metrics:\n')
    print(f'     RMSE: {rmse}')
    print(f'     R-squared: {r2}')

run_ensemble_model()
```

```
Best performing base estimators and their performance metrics:

Decision Tree Best Estimator:

{'ccp_alpha': 0.0, 'criterion': 'squared_error', 'max_depth': None,
'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease':
0.0, 'min_samples_leaf': 1, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'random_state': 123, 'splitter':
'best'}

Performance Metrics for Decision Tree:

     RMSE: 41.49383966411136
     R-squared: 0.9388757928138893

Feature importances:

     Calcium: 0.5451
     Vitamin C: 0.2703
     Caloric Value: 0.1070
     Phosphorus: 0.0321
     Dietary Fiber: 0.0120

Random Forest Best Estimator:

{'bootstrap': True, 'ccp_alpha': 0.0, 'criterion': 'squared_error',
'max_depth': None, 'max_features': 1.0, 'max_leaf_nodes': None,
'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf':
1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 200, 'n_jobs': None, 'oob_score': False,
'random_state': 123, 'verbose': 0, 'warm_start': False}
```

```
Performance Metrics for Random Forest:

     RMSE: 34.89358787335249
     R-squared: 0.9567747767624285

Feature importances:

     Calcium: 0.5793
     Vitamin C: 0.2414
     Caloric Value: 0.0850
     Saturated Fats: 0.0178
     Carbohydrates: 0.0106

Gradient Boosting Best Estimator:

{'alpha': 0.9, 'ccp_alpha': 0.0, 'criterion': 'friedman_mse', 'init':
None, 'learning_rate': 0.1, 'loss': 'squared_error', 'max_depth': 3,
'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease':
0.0, 'min_samples_leaf': 1, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'n_estimators': 200,
'n_iter_no_change': None, 'random_state': 123, 'subsample': 1.0,
'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': 0, 'warm_start':
False}

Performance Metrics for Gradient Boosting:

     RMSE: 27.04256135158072
     R-squared: 0.9740377912504109

Feature importances:

     Calcium: 0.5559
     Vitamin C: 0.2746
     Caloric Value: 0.0980
     Phosphorus: 0.0160
     Potassium: 0.0068

Stacking regressor performance metrics:

     RMSE: 24.650970805242217
     R-squared: 0.978426826070024

# The stacking regressor proved much more viable in modeling the food
nutrition data, with each model within the stacking
# regressor performing better than the previous model choice. With the
RSME values rather low in regards to the nutrition
# density variable's observations and the R-squared values explaining
a large percentage of variance within the data as
# shown by their close proximity to 1, the stacking regressor is the
best model choice out of the five models crafted.
```

```
# This can be confirmed with the very high R-squared value of 0.978
and the low RMSE value of 24.65 as it relates to the
# nutrition density variable. Looking at the factors that were
considered important features to the creation of these models
# allows the definition of three consistent variables across the
decision tree, random forest, and gradient boosting base
# models: Calcium, Vitamin C, and Caloric Value. As they relate to
nutrition density, these variables have a verified
# impact. Checking back with the heatmap created in the EDA stage
confirmed that some of these important features also had
# the highest positively correlated coefficients, allowing dieticians
and nutritionists to begin recommending foods that
# are high in those vitamins, minerals, and nutrients.

# To answer some of the questions put forth in the draft paper, I will
output the answers here to showcase them in the final
# presentation of the project.

# This bar chart will answer the question of the food ingredient with
the highest nutrition density.

top_10_foods = food_nutrition.sort_values(by = 'Nutrition Density',
ascending = False).head(10)

food_names = top_10_foods['Food']
nutrition_values = top_10_foods['Nutrition Density']

plt.figure(figsize=(12, 6))
plt.bar(food_names, nutrition_values, color = 'orange')
plt.title('Top 10 Food Ingredients by Nutrition Density')
plt.xlabel('Food Ingredient')
plt.ylabel('Nutrition Density')
plt.xticks(rotation=45, ha='right')
plt.show()
```


Top 10 Food Ingredients by Nutrition Density
Nutrition Density
0
500
1000
1500
2000
2500
3000
3500
4000
ACEROLA CHERRY JUICE
BANANA CREAM PIE
MEXICAN CHEESE
SISYMBRIUM SEEDS
PORK ARM PICNIC COOKED
PORK ARM PICNIC RAW
MOZZARELLA CHEESE FAT FREE
GOOSE MEAT RAW
GJETOST CHEESE
PROVOLONE CHEESE
Food Ingredient