THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY

PATIALA

PROJECT REPORT

DATA SCIENCE FOUNDATION

(PCS110)

TOPIC – Food Nutrition

SUBMITTED BY -

Parul Chambiyal	**8024320070**
Om Tiwari	**8024320069**
Mitul Sharma	**8024320064**
Md Sudaise Shah	**8024320065**

ACKNOWLEDGMENT

We are particularly indebted to the creators of the **Food Fact Finder**, from Kaggle whose diligent efforts in curating this rich dataset made this research possible. Their work has provided an invaluable resource for the study of mental health and its complex determinants.

We wish to acknowledge my colleagues and peers for their valuable input and collaboration during this project. Their critical discussions and suggestions have greatly enhanced the scope and depth of this research.

Lastly, I would like to express my gratitude to my family and friends for their continuous support and encouragement, which have strengthened me throughout this endeavor.

.

DATA PREPROCESSING

1. IMPORT NECESSARY LIBRARIES

```
[ ] import matplotlib.pyplot as plt
    import pandas as pd
    import seaborn as sns
```

```
[ ] from sklearn.preprocessing import StandardScaler
```

```
    from sklearn.preprocessing import MinMaxScaler
```

We have imported all the required Libraries which *includes pandas, matplotlib.pyplot, seaborn, standard scalar, min max scaler, Google Colab files*.This just sets up the environment for further code.

2. UPLOAD DATASET

We uploaded our Food Nutrition Dataset, which was downloaded from Kaggle, to the google colab.The dataset is stored in df, ready for analysis and manipulation.

```
[3] # Load dataset
    df = pd.read_csv('/content/FOOD-DATA-GROUP1.csv')
```

This will load the datatset in the environement.

3. LOAD DATASET

DataFrame *df* will contain the data from the CSV file in a structured tabular format with rows and columns.Displays the first few rows (head()) and the column names (columns) of the dataset.Prints the first 5 rows and column names.

```python
print(df.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]
```

4. FILTERING NUMERIC COLUMN

Selects columns that contain numeric data (integers and floats).A DataFrame numeric_df with only numeric columns. No print output here.

```python
[5] print(df.columns)
```

```
Index(['Unnamed: 0.1', 'Unnamed: 0', 'food', 'Caloric Value', 'Fat',
       'Saturated Fats', 'Monounsaturated Fats', 'Polyunsaturated Fats',
       'Carbohydrates', 'Sugars', 'Protein', 'Dietary Fiber', 'Cholesterol',
       'Sodium', 'Water', 'Vitamin A', 'Vitamin B1', 'Vitamin B11',
       'Vitamin B12', 'Vitamin B2', 'Vitamin B3', 'Vitamin B5', 'Vitamin B6',
       'Vitamin C', 'Vitamin D', 'Vitamin E', 'Vitamin K', 'Calcium', 'Copper',
       'Iron', 'Magnesium', 'Manganese', 'Phosphorus', 'Potassium', 'Selenium',
       'Zinc', 'Nutrition Density'],
      dtype='object')
```

```python
# We filter the numeric columns only
# Select only the numeric columns
numeric_df = df.select_dtypes(include=[int, float])
# Check the number of columns
print(f"Number of numeric columns: {numeric_df.shape[1]}")
# Now selecting the first row of numeric data (height for the bar chart)
height = numeric_df.iloc[0].values  # Get the values for the bar heights
# Get the corresponding tick labels for the selected numeric columns
tick_label = numeric_df.columns
left = list(range(len(tick_label)))
```

```
Number of numeric columns: 36
```

5. IDENTIFYING NULL VALUES

It is essential for data cleaning, as missing values can affect data analysis and machine learning models, and often need to be handled appropriately (e.g., by filling or dropping them).

```
print(df.loc[1:].isnull().sum())
```

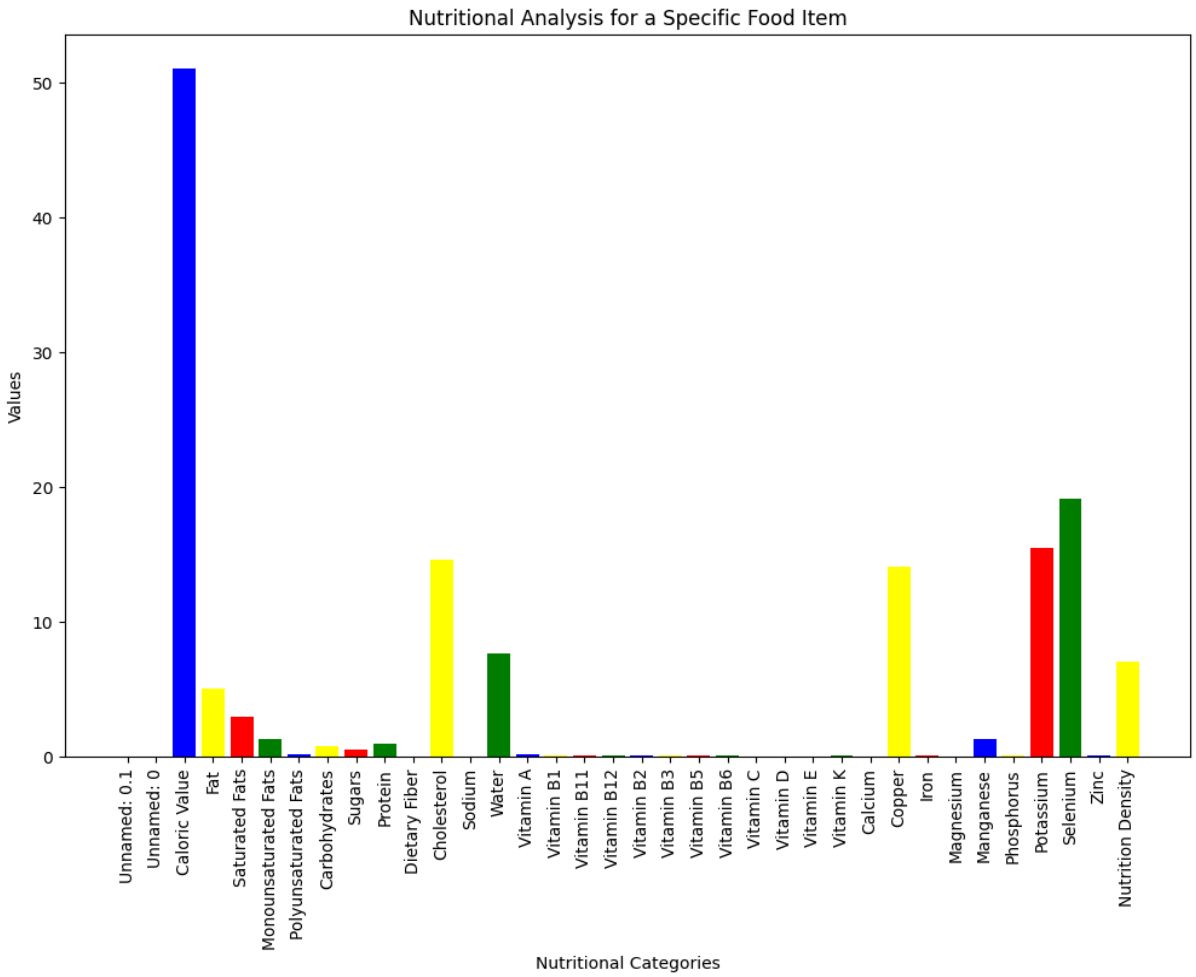
```
Unnamed: 0.1            0
Unnamed: 0              0
food                    0
Caloric Value           0
Fat                     0
Saturated Fats          0
Monounsaturated Fats    0
Polyunsaturated Fats    0
Carbohydrates           0
Sugars                  0
Protein                 0
Dietary Fiber           0
Cholesterol             0
Sodium                  0
Water                   0
Vitamin A               0
Vitamin B1              0
Vitamin B11             0
Vitamin B12             0
Vitamin B2              0
Vitamin B3              0
Vitamin B5              0
Vitamin B6              0
Vitamin C               0
Vitamin D               0
Vitamin E               0
Vitamin K               0
Calcium                 0
Copper                  0
Iron                    0
Magnesium               0
Manganese               0
Phosphorus              0
Potassium               0
Selenium                0
Zinc                    0
Nutrition Density       0
dtype: int64
```

checks for any missing (null) values in the selected rows. It returns True for each null value and False otherwise.

DATA VISUALISATION

1.Bar Chart represents categorical data where each bar's height corresponds to a value (a nutrient level like 'Protein'). The x-axis shows categories (food items), and the y-axis shows values (protein content). Higher bars indicate larger values for the category. It's useful for comparing quantities across categories.Matplotlib gives basic control for creating bar charts with full customization options (colors, labels).

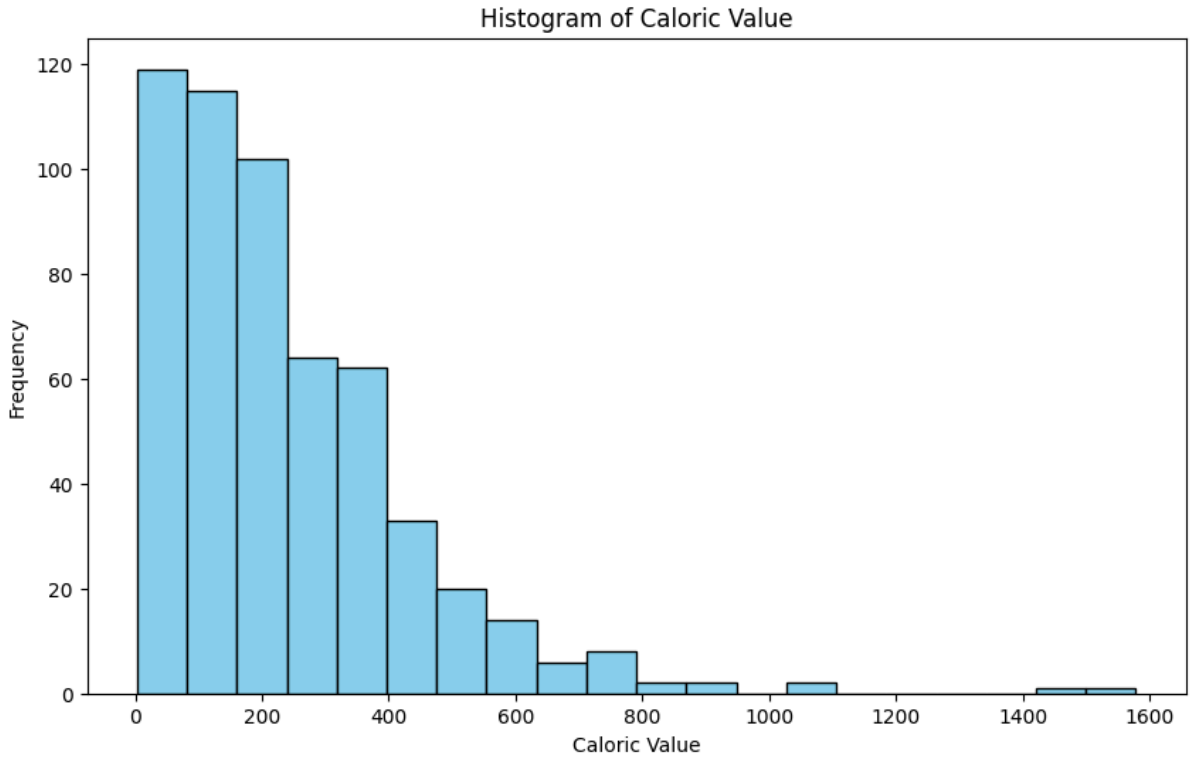
```
[8] #plotting a bar chart
    plt.figure(figsize=(12, 8))
    plt.bar(left, height, tick_label=tick_label, color=['red', 'green', 'blue', 'yellow'] * (len(left) // 4 + 1))
    # Add labels and title
    plt.xlabel('Nutritional Categories')
    plt.ylabel('Values')
    plt.title('Nutritional Analysis for a Specific Food Item')
    # Rotate the x-axis labels for better readability
    plt.xticks(rotation=90)
    # Display the chart
    plt.show()
```



2.Histogram shows the distribution of a single variable ('Caloric Value') by grouping the data into bins and displaying how many data points fall into each bin.How to Check the shape of the distribution—whether it is normal (bell-shaped), skewed, or has multiple peaks. The

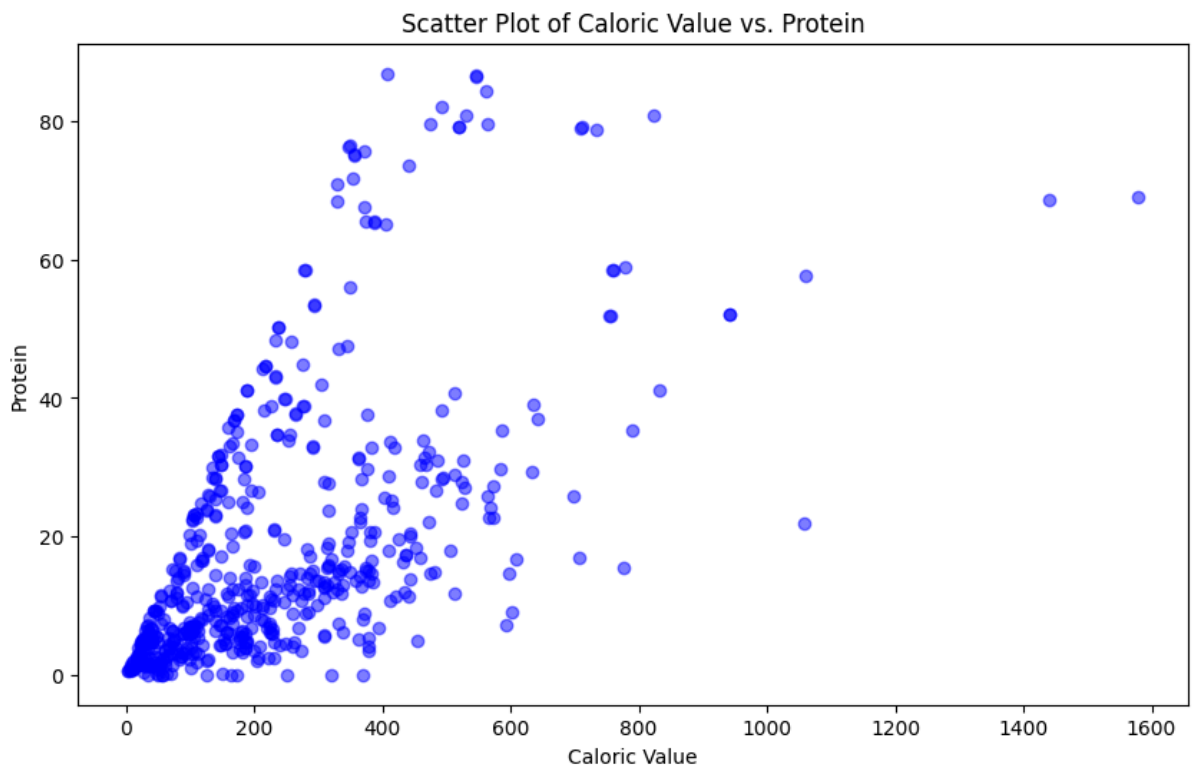
height of each bar represents the frequency of data points in that bin. Matplotlib allows you to manually define the number of bins and provides more granular control over the plot's style.

```
column_name = 'Caloric Value'
data = df[column_name]
# Plot the histogram
plt.figure(figsize=(10, 6))
plt.hist(data, bins=20, color='skyblue', edgecolor='black')
# Add labels and title
plt.xlabel(column_name)
plt.ylabel('Frequency')
plt.title(f'Histogram of {column_name}')
# Display the histogram
plt.show()
```



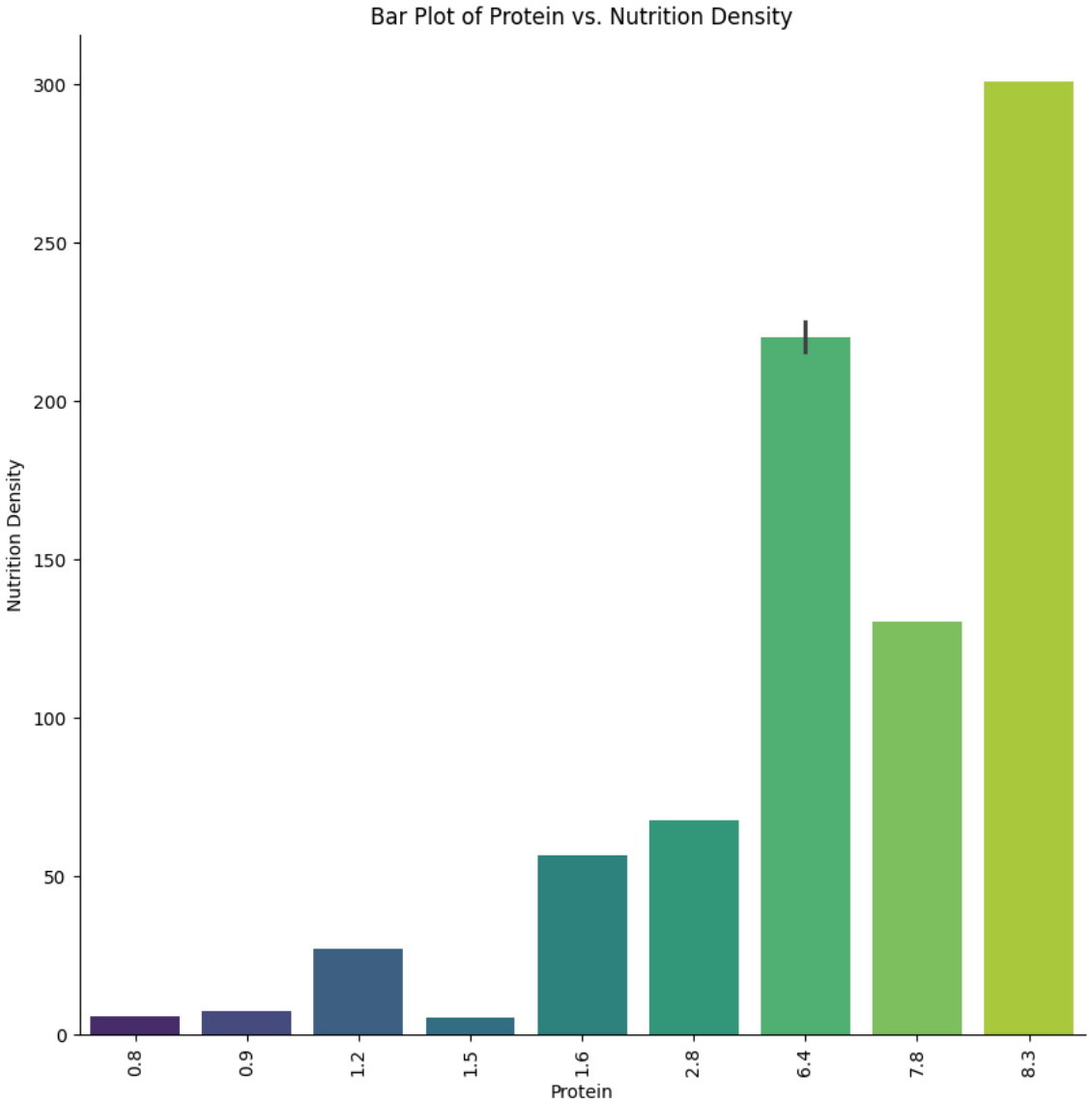
3.Scatter plot shows the relationship between two continuous variables ('Caloric Value' vs. 'Protein'). Each point represents an observation. Check for patterns (positive correlation, negative correlation, or no relationship). A cluster of points along an upward or downward trend suggests correlation.Matplotlib allows full customization of markers, colors, and axes, making it suitable for customized plots.

```
[10] x_column = 'Caloric Value'  # X-axis data
     y_column = 'Protein'        # Y-axis data
     # Extract the data
     x_data = df[x_column]
     y_data = df[y_column]
     # Plot the scatter plot
     plt.figure(figsize=(10, 6))
     plt.scatter(x_data, y_data, color='blue', alpha=0.5)
     # Add labels and title
     plt.xlabel(x_column)
     plt.ylabel(y_column)
     plt.title(f'Scatter Plot of {x_column} vs. {y_column}')
     # Display the scatter plot
     plt.show()
```



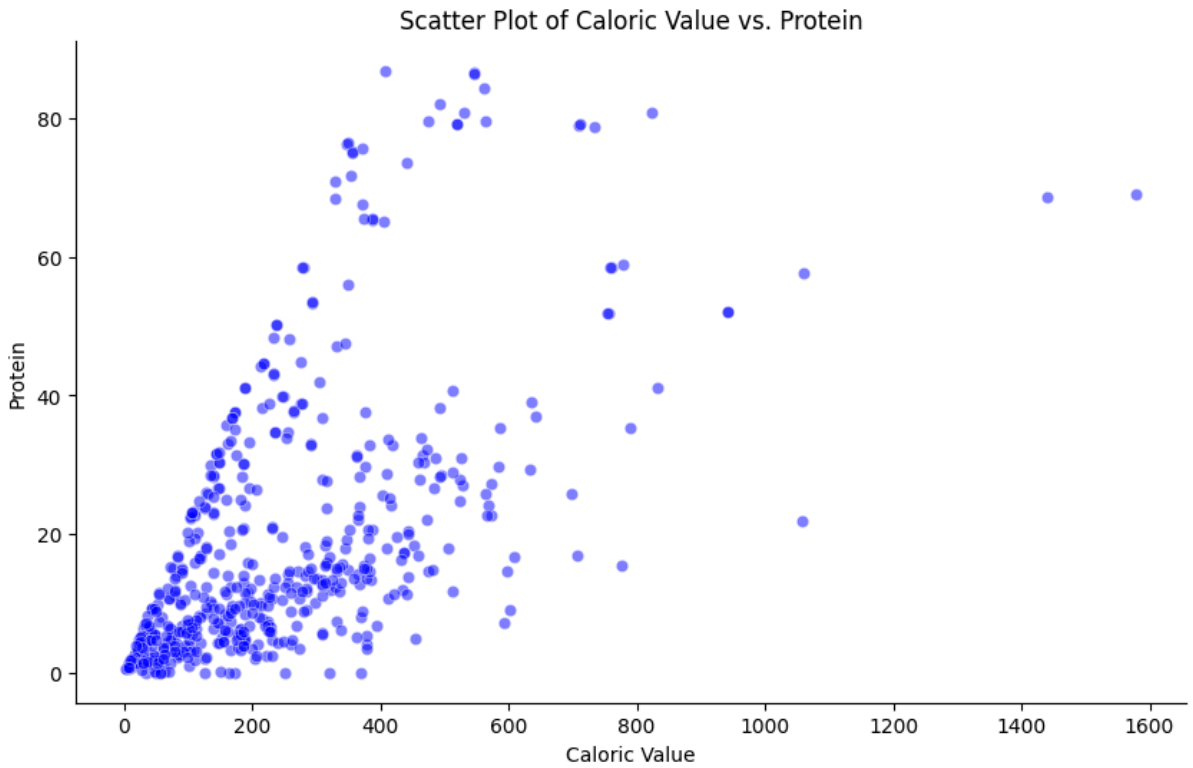
4.Seaborn Barplot Similar to Matplotlib's bar chart, but Seaborn automatically adds statistical estimates like error bars. This plot shows the relationship between 'Protein' and 'Nutrition Density'.The height of each bar represents the mean of 'Nutrition Density' for each 'Protein' value. Error bars show variability (e.g., standard deviation).Seaborn integrates with Pandas and provides a cleaner, more aesthetically pleasing interface, often with less code.

```
[12] import seaborn as sns
     n = 10
     subset_df = df[['Protein', 'Nutrition Density']].head(n)
     plt.figure(figsize=(10, 10))
     sns.barplot(x='Protein', y='Nutrition Density', data=subset_df, palette='viridis')
     plt.title('Bar Plot of Protein vs. Nutrition Density')
     plt.xlabel('Protein')
     plt.ylabel('Nutrition Density')
     plt.xticks(rotation=90)
     sns.despine()
     plt.show()
```



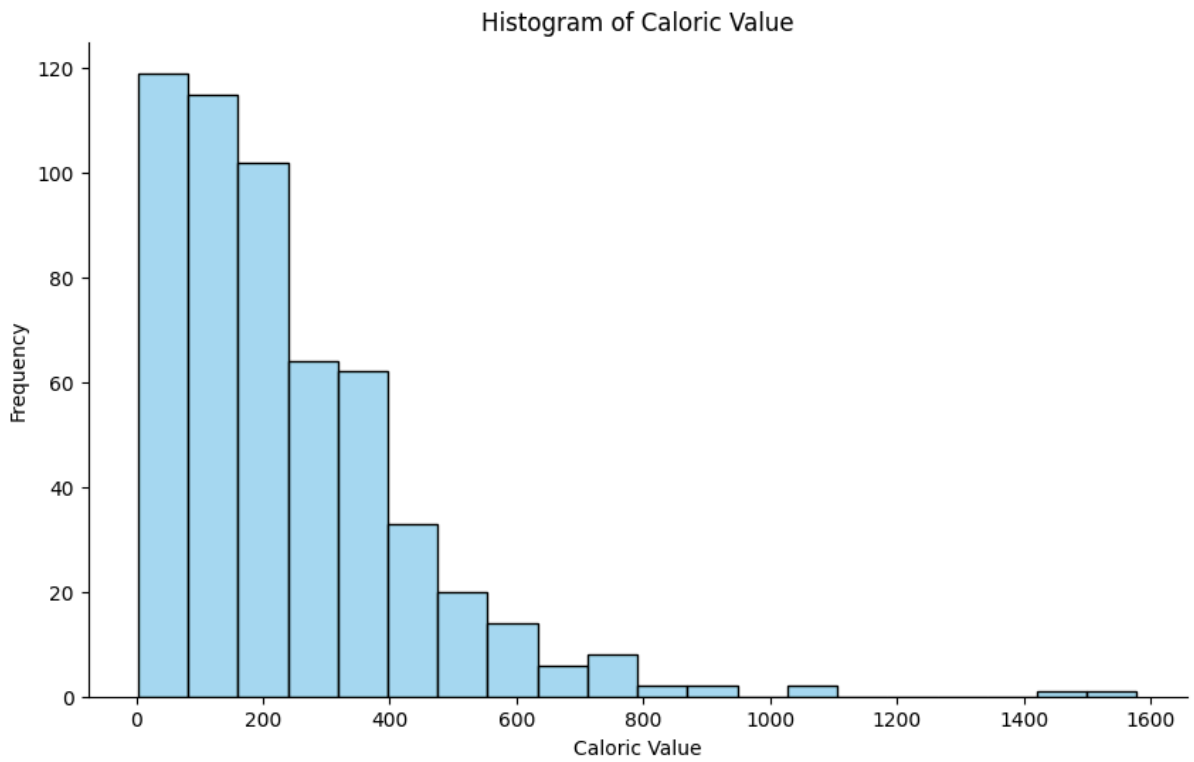
5. Seaborn Scatter Plot Similar to Matplotlib's scatter plot but with more features like grouping by color, adding regression lines, etc. It shows the relationship between 'Caloric Value' and 'Protein'. Like a Matplotlib scatter plot, look for trends. Seaborn also allows easy visualization of groups (via color or size).Seaborn provides better support for categorical or grouped data and aesthetics with minimal code.

```
[16] # Create a scatter plot
     plt.figure(figsize=(10, 6))
     sns.scatterplot(x='Caloric Value', y='Protein', data=df, color='blue', alpha=0.5)
     # Set title and labels
     plt.title('Scatter Plot of Caloric Value vs. Protein')
     plt.xlabel('Caloric Value')
     plt.ylabel('Protein')
     # Show the plot
     sns.despine()
     plt.show()
```



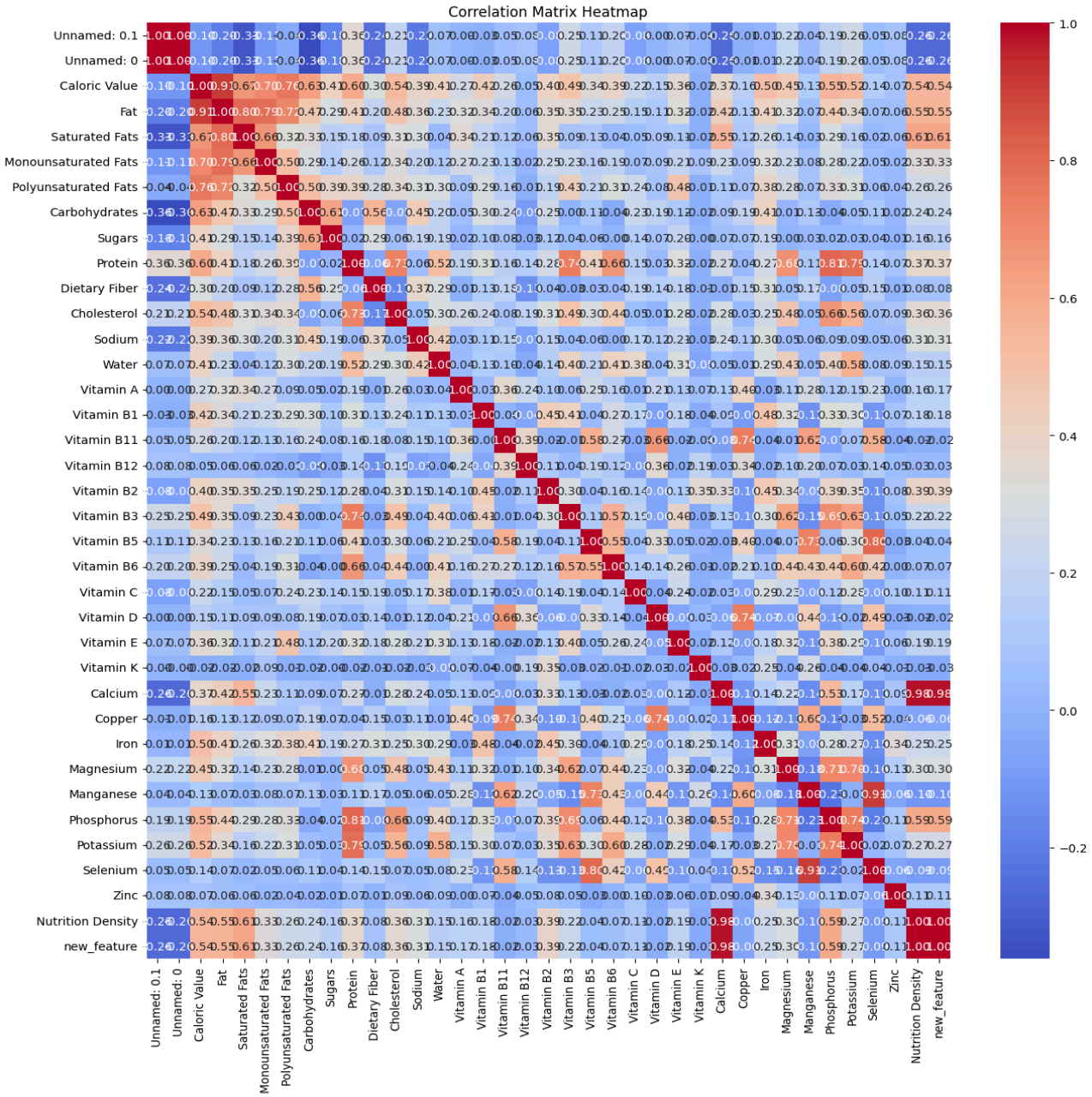
6. Seaborn Histogram showing the distribution of 'Caloric Value'. Like the Matplotlib histogram, it shows the frequency of data points across bins.Identify the shape of the distribution and look for outliers or patterns.Seaborn provides an easy-to-use function with default styling, such as kernel density estimation (KDE) overlays and aesthetic improvements.

```
# Create a histogram
plt.figure(figsize=(10, 6))
sns.histplot(df['Caloric Value'], bins=20, kde=False, color='skyblue')
# Set title and labels
plt.title('Histogram of Caloric Value')
plt.xlabel('Caloric Value')
plt.ylabel('Frequency')
# Show the plot
sns.despine()
plt.show()
```



7. Correlation Heatmap (Seaborn) visualizes the correlation matrix, where each cell shows the correlation coefficient between two variables (e.g., 'Caloric Value' and 'Protein').The color intensity represents the strength of the correlation. Strong positive correlations are close to 1 (Red), and strong negative correlations are close to -1 (Blue). Zero correlation (no relationship) is shown in neutral colors.Seaborn is excellent for making complex plots like heatmaps visually appealing with default color schemes and annotations.

```
# Compute the correlation matrix
corr_matrix = df.corr()
# Display the correlation matrix
print(corr_matrix)
# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(15, 15))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```



8. Normalization rescales the data to a range of [0,1][0, 1][0,1], where the minimum value of the dataset becomes 0 and the maximum value becomes 1.Useful when you know the distribution of the data is not Gaussian and when the algorithm assumes a bounded input, like in Min-Max Scaler.The 'Caloric Value' column will now have values between 0 and 1.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
# Scale the 'Caloric Value' column
df[['Caloric Value']] = scaler.fit_transform(df[['Caloric Value']])
```

9.Standardization centers the data by subtracting the mean and scaling to unit variance (standard deviation of 1).The 'Caloric Value' column will now have a mean of 0 and a standard deviation of 1.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[['Caloric Value']] = scaler.fit_transform(df[['Caloric Value']])
```

7. Dropping the 'food' Column removes the 'food' column from the DataFrame.The DataFrame now no longer contains the 'food' column.

```
# Drop the 'food' column
df = df.drop(columns='food')
# Verify the column has been dropped
print(df.head())
print(df.columns)
```

```
   Unnamed: 0.1  Unnamed: 0  Caloric Value   Fat  Saturated Fats  \
0             0           0       0.030476   5.0             2.9
1             1           1       0.134603  19.4            10.9
2             2           2       0.029206   3.6             2.3
3             3           3       0.017143   2.0             1.3
4             4           4       0.017143   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

   ...  Copper   Iron  Magnesium  Manganese  Phosphorus  Potassium  Selenium  \
0  ...  14.100  0.082      0.027      1.300       0.091       15.5    19.100
1  ...   0.034  0.100      8.500      0.088     117.300      129.2     0.054
2  ...   0.000  0.000      0.000      0.000       0.000        0.0     0.000
3  ...  41.200  0.097      0.096      4.000       0.024       30.8    43.800
4  ...   0.072  0.008      1.200      0.098      22.800       37.1     0.034

    Zinc  Nutrition Density  new_feature
0  0.039              7.070     7.100476
1  0.700            130.100   130.234603
2  0.000              5.400     5.429206
3  0.035              5.196     5.213143
4  0.053             27.007    27.024143

[5 rows x 37 columns]
Index(['Unnamed: 0.1', 'Unnamed: 0', 'Caloric Value', 'Fat', 'Saturated Fats',
       'Monounsaturated Fats', 'Polyunsaturated Fats', 'Carbohydrates',
       'Sugars', 'Protein', 'Dietary Fiber', 'Cholesterol', 'Sodium', 'Water',
       'Vitamin A', 'Vitamin B1', 'Vitamin B11', 'Vitamin B12', 'Vitamin B2',
       'Vitamin B3', 'Vitamin B5', 'Vitamin B6', 'Vitamin C', 'Vitamin D',
       'Vitamin E', 'Vitamin K', 'Calcium', 'Copper', 'Iron', 'Magnesium',
       'Manganese', 'Phosphorus', 'Potassium', 'Selenium', 'Zinc',
       'Nutrition Density', 'new_feature'],
      dtype='object')
```

10. Dropping Columns drops the specified columns from the DataFrame. The DataFrame without the dropped columns.

```
[30] df.columns
```

```
Index(['Unnamed: 0.1', 'Unnamed: 0', 'Caloric Value', 'Fat', 'Saturated Fats',
       'Monounsaturated Fats', 'Polyunsaturated Fats', 'Carbohydrates',
       'Sugars', 'Protein', 'Dietary Fiber', 'Cholesterol', 'Sodium', 'Water',
       'Vitamin A', 'Vitamin B1', 'Vitamin B11', 'Vitamin B12', 'Vitamin B2',
       'Vitamin B3', 'Vitamin B5', 'Vitamin B6', 'Vitamin C', 'Vitamin D',
       'Vitamin E', 'Vitamin K', 'Calcium', 'Copper', 'Iron', 'Magnesium',
       'Manganese', 'Phosphorus', 'Potassium', 'Selenium', 'Zinc',
       'Nutrition Density', 'new_feature'],
      dtype='object')
```

```
[31] # Drop the columns
     df = df.drop(columns=[ 'Polyunsaturated Fats', 'Monounsaturated Fats', 'Vitamin B6', 'Manganese', 'Selenium'])
     # Resulting DataFrame to ensure the columns were dropped
     print(df.head())
     print(df.columns)
```

```
[31]
   Unnamed: 0.1  Unnamed: 0  Caloric Value   Fat  Saturated Fats  \
0             0           0       0.030476   5.0             2.9
1             1           1       0.134603  19.4            10.9
2             2           2       0.029206   3.6             2.3
3             3           3       0.017143   2.0             1.3
4             4           4       0.017143   2.3             1.4

   Carbohydrates  Sugars  Protein  Dietary Fiber  Cholesterol  ...  Vitamin K  \
0            0.8   0.500      0.9            0.0         14.6  ...      0.100
1            3.1   2.700      7.8            0.0         62.9  ...      0.045
2            0.9   3.400      0.8            0.1          0.0  ...      0.000
3            1.5   0.091      1.5            0.0          9.8  ...      0.011
4            1.2   0.900      1.2            0.0          8.1  ...      0.019

   Calcium  Copper   Iron  Magnesium  Phosphorus  Potassium   Zinc  \
0    0.008  14.100  0.082      0.027       0.091       15.5  0.039
1   99.500   0.034  0.100      8.500     117.300      129.2  0.700
2    0.000   0.000  0.000      0.000       0.000        0.0  0.000
3    0.097  41.200  0.097      0.096       0.024       30.8  0.035
4   22.200   0.072  0.008      1.200      22.800       37.1  0.053

   Nutrition Density  new_feature
0              7.070     7.100476
1            130.100   130.234603
2              5.400     5.429206
3              5.196     5.213143
4             27.007    27.024143

[5 rows x 32 columns]
Index(['Unnamed: 0.1', 'Unnamed: 0', 'Caloric Value', 'Fat', 'Saturated Fats',
       'Carbohydrates', 'Sugars', 'Protein', 'Dietary Fiber', 'Cholesterol',
       'Sodium', 'Water', 'Vitamin A', 'Vitamin B1', 'Vitamin B11',
       'Vitamin B12', 'Vitamin B2', 'Vitamin B3', 'Vitamin B5', 'Vitamin C',
       'Vitamin D', 'Vitamin E', 'Vitamin K', 'Calcium', 'Copper', 'Iron',
       'Magnesium', 'Phosphorus', 'Potassium', 'Zinc', 'Nutrition Density',
       'new_feature'],
      dtype='object')
```

11. Final Correlation Heatmap recomputes and visualizes the correlation matrix after certain columns have been dropped. A new heatmap showing updated correlations between the remaining features in the dataset.

```
[32] # Compute the correlation matrix
     corr_matrix = df.corr()
     # Display the correlation matrix
     print(corr_matrix)
     # Visualize the correlation matrix using a heatmap
     plt.figure(figsize=(15, 15))  # Adjust the figure size to fit your columns
     sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
     plt.title('Correlation Matrix Heatmap')
     plt.show()
```