In [126]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [127]:
```
df = pd.read_csv('/content/cereal.csv')
df
```

Out [127]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
0	100% Bran	N	C	70	4	1	130	10.0	5.0	6	280	25	3	1.0	0.33	68.402973
1	100% Natural Bran	Q	C	120	3	5	15	2.0	8.0	8	135	0	3	1.0	1.00	33.983679
2	All-Bran	K	C	70	4	1	260	9.0	7.0	5	320	25	3	1.0	0.33	59.425505
3	All-Bran with Extra Fiber	K	C	50	4	0	140	14.0	8.0	0	330	25	3	1.0	0.50	93.704912
4	Almond Delight	R	C	110	2	2	200	1.0	14.0	8	-1	25	3	1.0	0.75	34.384843
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
72	Triples	G	C	110	2	1	250	0.0	21.0	3	60	25	3	1.0	0.75	39.106174
73	Trix	G	C	110	1	1	140	0.0	13.0	12	25	25	2	1.0	1.00	27.753301
74	Wheat Chex	R	C	100	3	1	230	3.0	17.0	3	115	25	1	1.0	0.67	49.787445
75	Wheaties	G	C	100	3	1	200	3.0	17.0	3	110	25	1	1.0	1.00	51.592193
76	Wheaties Honey Gold	G	C	110	2	1	200	1.0	16.0	8	60	25	1	1.0	0.75	36.187559

77 rows × 16 columns

In [128]:
```
df.head()
```

Out [128]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
0	100% Bran	N	C	70	4	1	130	10.0	5.0	6	280	25	3	1.0	0.33	68.402973
1	100% Natural Bran	Q	C	120	3	5	15	2.0	8.0	8	135	0	3	1.0	1.00	33.983679
2	All-Bran	K	C	70	4	1	260	9.0	7.0	5	320	25	3	1.0	0.33	59.425505
3	All-Bran with Extra Fiber	K	C	50	4	0	140	14.0	8.0	0	330	25	3	1.0	0.50	93.704912
4	Almond Delight	R	C	110	2	2	200	1.0	14.0	8	-1	25	3	1.0	0.75	34.384843

In [129]:
```
df.tail()
```

Out [129]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
72	Triples	G	C	110	2	1	250	0.0	21.0	3	60	25	3	1.0	0.75	39.106174
73	Trix	G	C	110	1	1	140	0.0	13.0	12	25	25	2	1.0	1.00	27.753301
74	Wheat Chex	R	C	100	3	1	230	3.0	17.0	3	115	25	1	1.0	0.67	49.787445
75	Wheaties	G	C	100	3	1	200	3.0	17.0	3	110	25	1	1.0	1.00	51.592193
76	Wheaties Honey Gold	G	C	110	2	1	200	1.0	16.0	8	60	25	1	1.0	0.75	36.187559

In [130]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77 entries, 0 to 76
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   name      77 non-null     object 
 1   mfr       77 non-null     object 
 2   type      77 non-null     object 
 3   calories  77 non-null     int64  
 4   protein   77 non-null     int64  
 5   fat       77 non-null     int64  
 6   sodium    77 non-null     int64  
 7   fiber     77 non-null     float64
 8   carbo     77 non-null     float64
 9   sugars    77 non-null     int64  
 10  potass    77 non-null     int64  
 11  vitamins  77 non-null     int64  
 12  shelf     77 non-null     int64  
 13  weight    77 non-null     float64
 14  cups      77 non-null     float64
 15  rating    77 non-null     float64
dtypes: float64(5), int64(8), object(3)
memory usage: 9.8+ KB
```

In [131]:
```
df.describe
```

Out [131]:
```
<bound method NDFrame.describe of                           name mfr type  calories  protein  fat  sodium  fiber  \
0                    100% Bran   N    C        70        4    1     130   10.0  
```

```
1          100% Natural Bran   Q    C       120        3    5       15    2.0
2                    All-Bran   K    C        70        4    1      260    9.0
3   All-Bran with Extra Fiber   K    C        50        4    0      140   14.0
4              Almond Delight   R    C       110        2    2      200    1.0
..                        ...  ..  ...       ...      ...  ...      ...    ...
72                    Triples   G    C       110        2    1      250    0.0
73                       Trix   G    C       110        1    1      140    0.0
74                 Wheat Chex   R    C       100        3    1      230    3.0
75                   Wheaties   G    C       100        3    1      200    3.0
76        Wheaties Honey Gold   G    C       110        2    1      200    1.0

    carbo  sugars  potass  vitamins  shelf  weight  cups     rating
0     5.0       6     280        25      3     1.0  0.33  68.402973
1     8.0       8     135         0      3     1.0  1.00  33.983679
2     7.0       5     320        25      3     1.0  0.33  59.425505
3     8.0       0     330        25      3     1.0  0.50  93.704912
4    14.0       8      -1        25      3     1.0  0.75  34.384843
..    ...     ...     ...       ...    ...     ...   ...        ...
72   21.0       3      60        25      3     1.0  0.75  39.106174
73   13.0      12      25        25      2     1.0  1.00  27.753301
74   17.0       3     115        25      1     1.0  0.67  49.787445
75   17.0       3     110        25      1     1.0  1.00  51.592193
76   16.0       8      60        25      1     1.0  0.75  36.187559

[77 rows x 16 columns]>
```

In [132]:
```
df.shape
```

Out [132]: (77, 16)

In [133]:
```
df.isna().sum()
```

Out [133]:
```
name        0
mfr         0
type        0
calories    0
protein     0
fat         0
sodium      0
fiber       0
carbo       0
sugars      0
potass      0
vitamins    0
shelf       0
weight      0
cups        0
rating      0
dtype: int64
```

In [134]:
```
df = df.drop_duplicates()
df.shape
```

Out [134]: (77, 16)

The columns "sugar" and "potass"(for potassium) contain a few values that are negative. Next section will remove them.

In [135]:
```
df = df[(df['sugars']>0) & (df['potass']>0)]
df
```

Out [135]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
0	100% Bran	N	C	70	4	1	130	10.0	5.0	6	280	25	3	1.0	0.33	68.402973
1	100% Natural Bran	Q	C	120	3	5	15	2.0	8.0	8	135	0	3	1.0	1.00	33.983679
2	All-Bran	K	C	70	4	1	260	9.0	7.0	5	320	25	3	1.0	0.33	59.425505
5	Apple Cinnamon Cheerios	G	C	110	2	2	180	1.5	10.5	10	70	25	1	1.0	0.75	29.509541
6	Apple Jacks	K	C	110	2	0	125	1.0	11.0	14	30	25	2	1.0	1.00	33.174094
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
72	Triples	G	C	110	2	1	250	0.0	21.0	3	60	25	3	1.0	0.75	39.106174
73	Trix	G	C	110	1	1	140	0.0	13.0	12	25	25	2	1.0	1.00	27.753301
74	Wheat Chex	R	C	100	3	1	230	3.0	17.0	3	115	25	1	1.0	0.67	49.787445
75	Wheaties	G	C	100	3	1	200	3.0	17.0	3	110	25	1	1.0	1.00	51.592193
76	Wheaties Honey Gold	G	C	110	2	1	200	1.0	16.0	8	60	25	1	1.0	0.75	36.187559

68 rows × 16 columns

In [136]:
```
# creating a dictionary to map the manufacturer codes to their full names
manufacturer_mapping = {
    'A': 'American Home Food Products',
    'G': 'General Mills',
    'K': 'Kelloggs',
    'N': 'Nabisco',
    'P': 'Post',
    'Q': 'Quaker Oats',
    'R': 'Ralston Purina'
}
```

```
# replacing the manufacturer codes with their full names in the 'mfr' column
df['mfr'] = df['mfr'].map(manufacturer_mapping)
```

In [137]:

```
df
```

Out [137]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
0	100% Bran	Nabisco	C	70	4	1	130	10.0	5.0	6	280	25	3	1.0	0.33	68.402973
1	100% Natural Bran	Quaker Oats	C	120	3	5	15	2.0	8.0	8	135	0	3	1.0	1.00	33.983679
2	All-Bran	Kelloggs	C	70	4	1	260	9.0	7.0	5	320	25	3	1.0	0.33	59.425505
5	Apple Cinnamon Cheerios	General Mills	C	110	2	2	180	1.5	10.5	10	70	25	1	1.0	0.75	29.509541
6	Apple Jacks	Kelloggs	C	110	2	0	125	1.0	11.0	14	30	25	2	1.0	1.00	33.174094
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
72	Triples	General Mills	C	110	2	1	250	0.0	21.0	3	60	25	3	1.0	0.75	39.106174
73	Trix	General Mills	C	110	1	1	140	0.0	13.0	12	25	25	2	1.0	1.00	27.753301
74	Wheat Chex	Ralston Purina	C	100	3	1	230	3.0	17.0	3	115	25	1	1.0	0.67	49.787445
75	Wheaties	General Mills	C	100	3	1	200	3.0	17.0	3	110	25	1	1.0	1.00	51.592193
76	Wheaties Honey Gold	General Mills	C	110	2	1	200	1.0	16.0	8	60	25	1	1.0	0.75	36.187559

68 rows × 16 columns

In [138]:

```
df['mfr'].unique()
```

Out [138]:

```
array(['Nabisco', 'Quaker Oats', 'Kelloggs', 'General Mills',
       'Ralston Purina', 'Post', 'American Home Food Products'],
      dtype=object)
```

In [139]:

```
# exploring the distribution of cereal types (cold vs. hot)
df['type'].value_counts()
```

Out [139]:

```
C    67
H     1
Name: type, dtype: int64
```

In [140]:

```
df['name'].unique()
```

Out [140]:

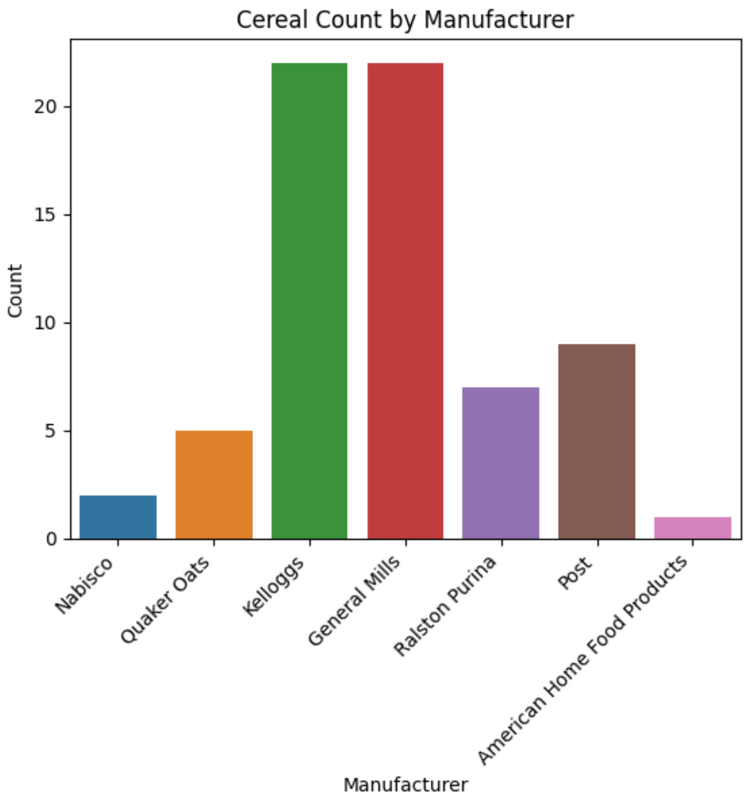
```
array(['100% Bran', '100% Natural Bran', 'All-Bran',
       'Apple Cinnamon Cheerios', 'Apple Jacks', 'Basic 4', 'Bran Chex',
       'Bran Flakes', "Cap'n'Crunch", 'Cheerios', 'Cinnamon Toast Crunch',
       'Clusters', 'Cocoa Puffs', 'Corn Chex', 'Corn Flakes', 'Corn Pops',
       'Count Chocula', "Cracklin' Oat Bran", 'Crispix',
       'Crispy Wheat & Raisins', 'Double Chex', 'Froot Loops',
       'Frosted Flakes', 'Frosted Mini-Wheats',
       'Fruit & Fibre Dates; Walnuts; and Oats', 'Fruitful Bran',
       'Fruity Pebbles', 'Golden Crisp', 'Golden Grahams',
       'Grape Nuts Flakes', 'Grape-Nuts', 'Great Grains Pecan',
       'Honey Graham Ohs', 'Honey Nut Cheerios', 'Honey-comb',
       'Just Right Crunchy  Nuggets', 'Just Right Fruit & Nut', 'Kix',
       'Life', 'Lucky Charms', 'Maypo',
       'Muesli Raisins; Dates; & Almonds',
       'Muesli Raisins; Peaches; & Pecans', 'Mueslix Crispy Blend',
       'Multi-Grain Cheerios', 'Nut&Honey Crunch',
       'Nutri-Grain Almond-Raisin', 'Nutri-grain Wheat',
       'Oatmeal Raisin Crisp', 'Post Nat. Raisin Bran', 'Product 19',
       'Quaker Oat Squares', 'Raisin Bran', 'Raisin Nut Bran',
       'Raisin Squares', 'Rice Chex', 'Rice Krispies', 'Smacks',
       'Special K', 'Strawberry Fruit Wheats', 'Total Corn Flakes',
       'Total Raisin Bran', 'Total Whole Grain', 'Triples', 'Trix',
       'Wheat Chex', 'Wheaties', 'Wheaties Honey Gold'], dtype=object)
```

In [141]:

```
sns.countplot(x = 'mfr',data =df)
plt.xlabel('Manufacturer')
plt.ylabel('Count')
plt.title('Cereal Count by Manufacturer')
plt.xticks(rotation=45, ha='right')
```

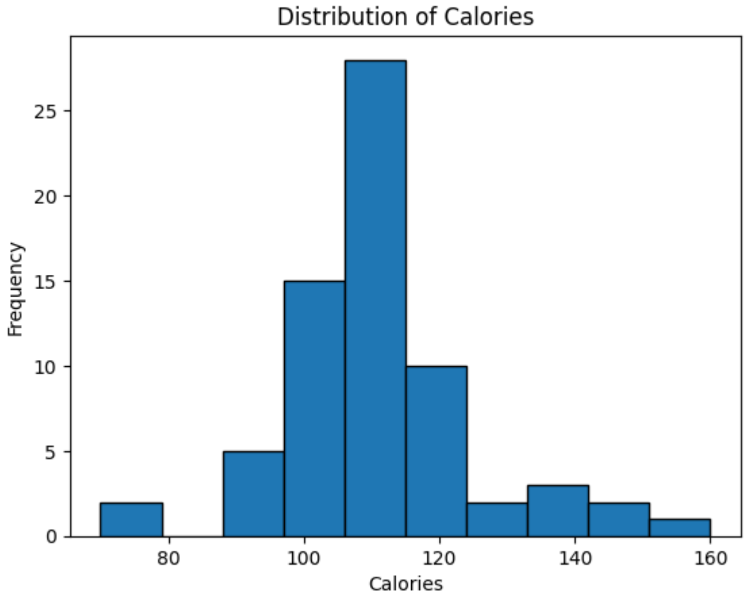
Out [141]:

```
(array([0, 1, 2, 3, 4, 5, 6]),
 [Text(0, 0, 'Nabisco'),
  Text(1, 0, 'Quaker Oats'),
  Text(2, 0, 'Kelloggs'),
  Text(3, 0, 'General Mills'),
  Text(4, 0, 'Ralston Purina'),
  Text(5, 0, 'Post'),
  Text(6, 0, 'American Home Food Products')])
```



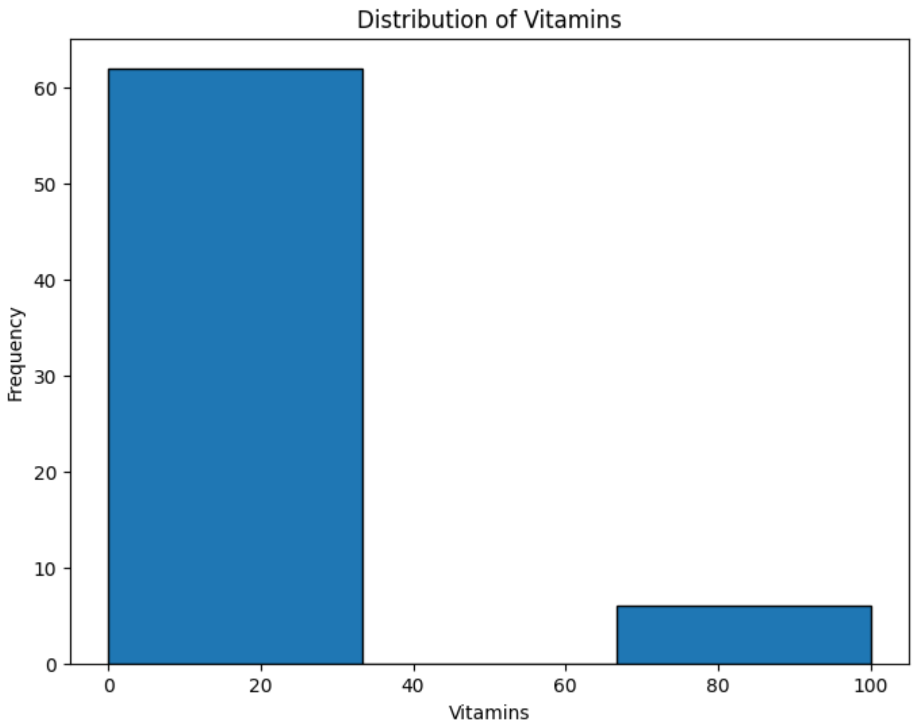
In [142]:
```
# Histogram for the distribution of calories
plt.hist(df['calories'], bins=10, edgecolor='black')
plt.xlabel('Calories')
plt.ylabel('Frequency')
plt.title('Distribution of Calories')
```

Out [142]: Text(0.5, 1.0, 'Distribution of Calories')



In [143]:
```
plt.figure(figsize=(8, 6))
plt.hist(df['vitamins'], bins=3, edgecolor='black')
plt.xlabel('Vitamins')
plt.ylabel('Frequency')
plt.title('Distribution of Vitamins')
```

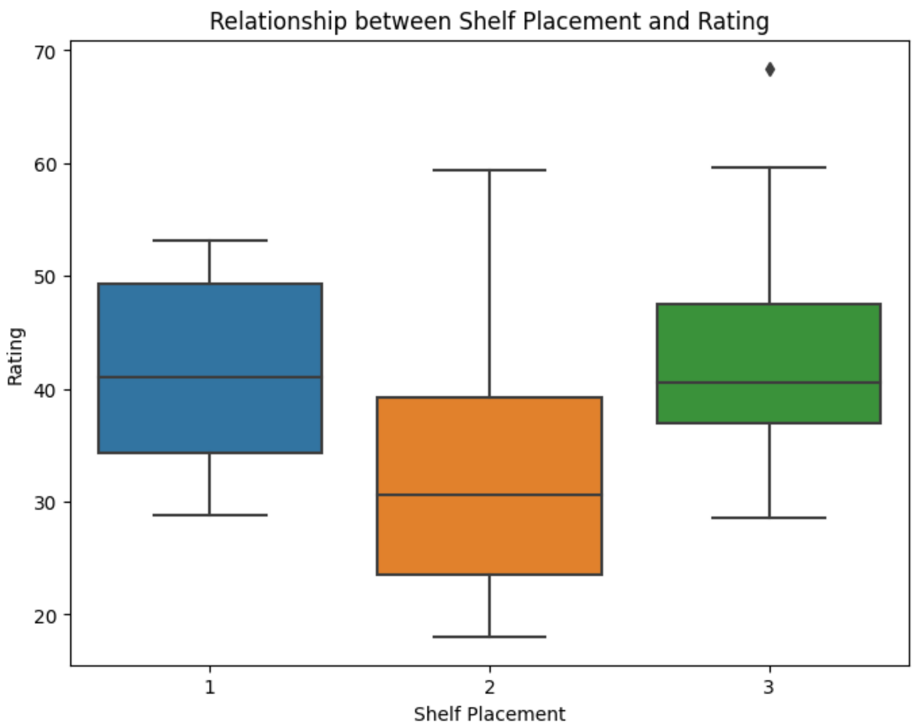
Out [143]: Text(0.5, 1.0, 'Distribution of Vitamins')



In [144]:

```
# summary statistics of rating grouped by shelf placement
rating_stats = df.groupby('shelf')['rating'].describe()
# box plot of rating by shelf placement
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, x='shelf', y='rating')
plt.xlabel('Shelf Placement')
plt.ylabel('Rating')
plt.title('Relationship between Shelf Placement and Rating')
```

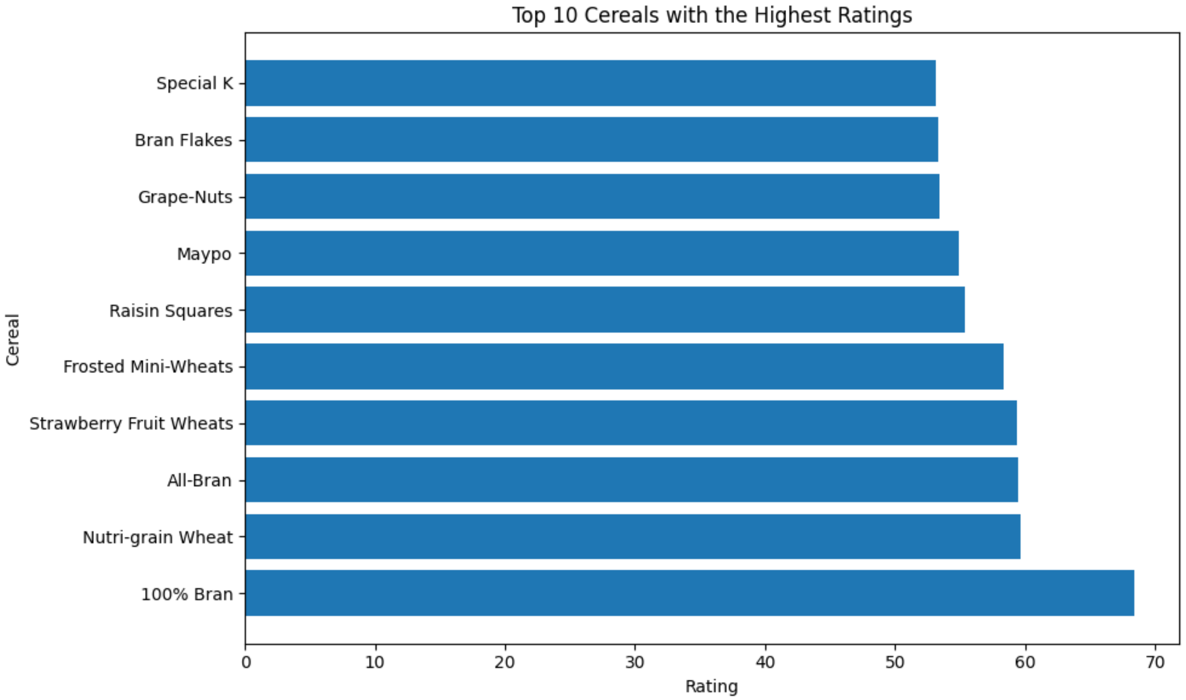
Out [144]: Text(0.5, 1.0, 'Relationship between Shelf Placement and Rating')



In [145]:

```
# Select the top 10 cereals with the highest ratings
top_10_cereals = df.nlargest(10, 'rating')

# Create a horizontal bar chart
```

```
plt.figure(figsize=(10, 6))
plt.barh(top_10_cereals['name'], top_10_cereals['rating'])
plt.xlabel('Rating')
plt.ylabel('Cereal')
plt.title('Top 10 Cereals with the Highest Ratings')
plt.tight_layout()
```



In [169]:

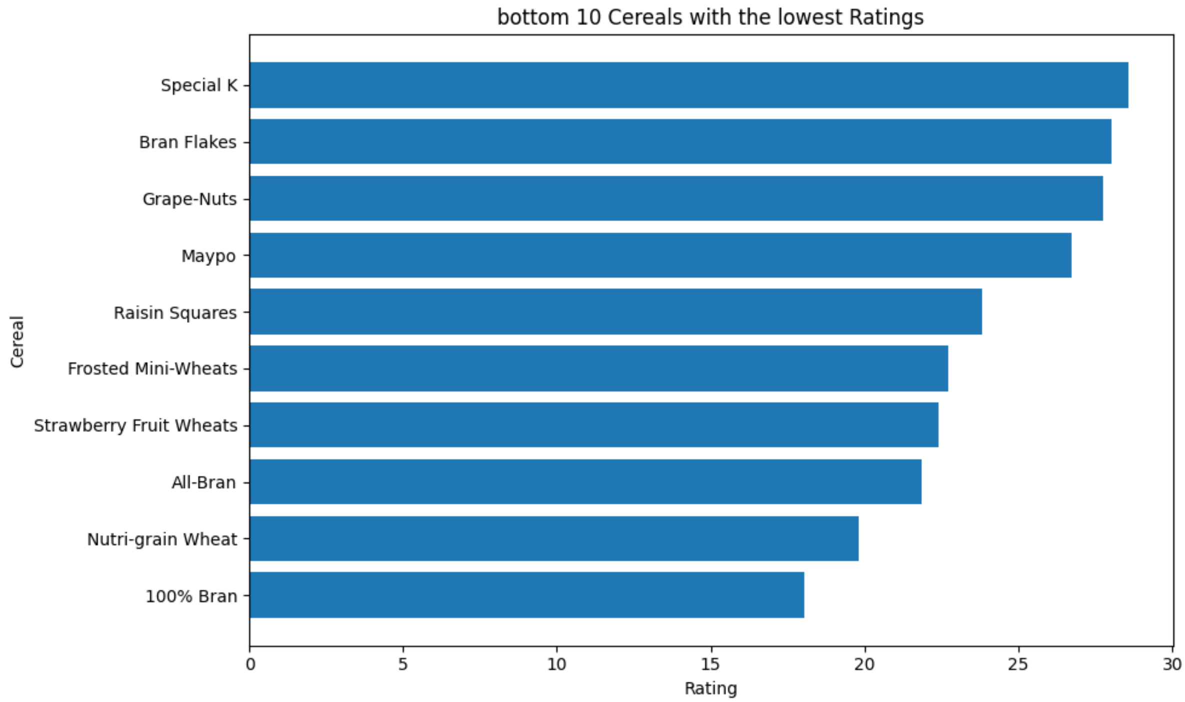
```
#bottom 10 cereals with the lowest thing
lowest_cereals= df.nsmallest(10,'rating')
lowest_cereals
```

Out [169]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
10	8	5	0	120	1	2	220	0.0	12.0	12	35	25	2	1.0	0.75	18.042851
12	10	1	0	120	1	3	210	0.0	13.0	9	45	25	2	1.0	0.75	19.823573
35	32	5	0	120	1	2	220	1.0	12.0	11	45	25	2	1.0	1.00	21.871292
18	16	1	0	110	1	1	180	0.0	12.0	13	65	25	2	1.0	1.00	22.396513
14	12	1	0	110	1	1	180	0.0	12.0	13	55	25	2	1.0	1.00	22.736446
31	28	1	0	110	1	1	280	0.0	15.0	9	45	25	2	1.0	0.75	23.804043
42	39	1	0	110	2	1	180	0.0	12.0	12	55	25	2	1.0	1.00	26.734515
73	64	1	0	110	1	1	140	0.0	13.0	12	25	25	2	1.0	1.00	27.753301
29	26	4	0	110	1	1	135	0.0	13.0	12	25	25	2	1.0	0.75	28.025765
70	61	1	0	140	3	1	190	4.0	15.0	14	230	100	3	1.5	1.00	28.592785

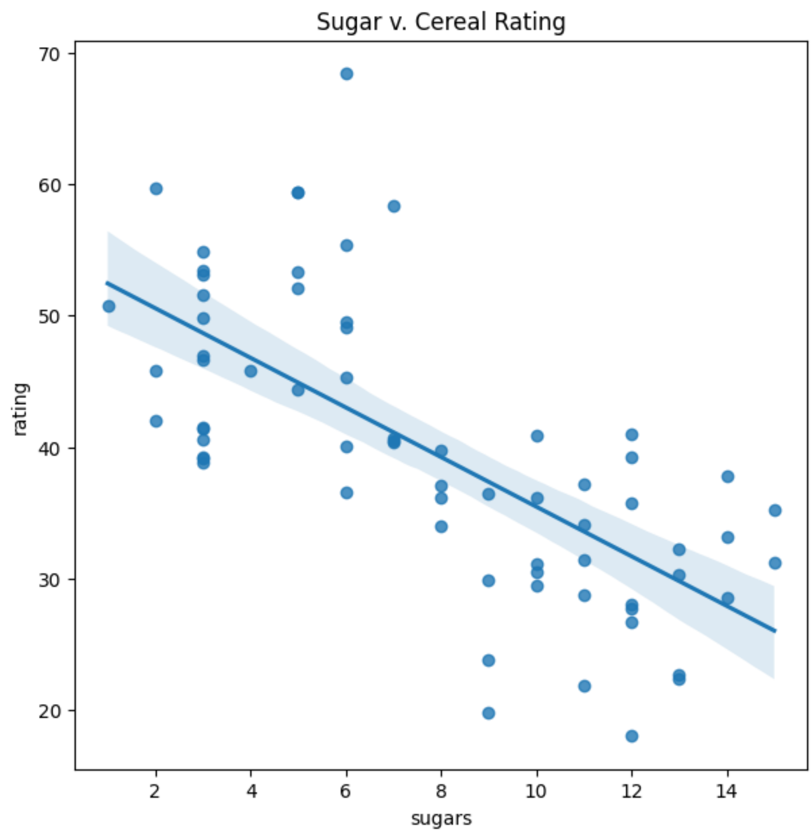
In [170]:

```
# Create a horizontal bar chart
plt.figure(figsize=(10, 6))
plt.barh(top_10_cereals['name'], lowest_cereals['rating'])
plt.xlabel('Rating')
plt.ylabel('Cereal')
plt.title('bottom 10 Cereals with the lowest Ratings')
plt.tight_layout()
```



In [146]:
```
#We visualize the relation between Sugar & Rating
y_rating=df["rating"]
x_sugar=df["sugars"]
plt.figure(figsize=(7,7))
sns.regplot(x=x_sugar,y=y_rating) #regression best fit line command
plt.title('Sugar v. Cereal Rating')
```

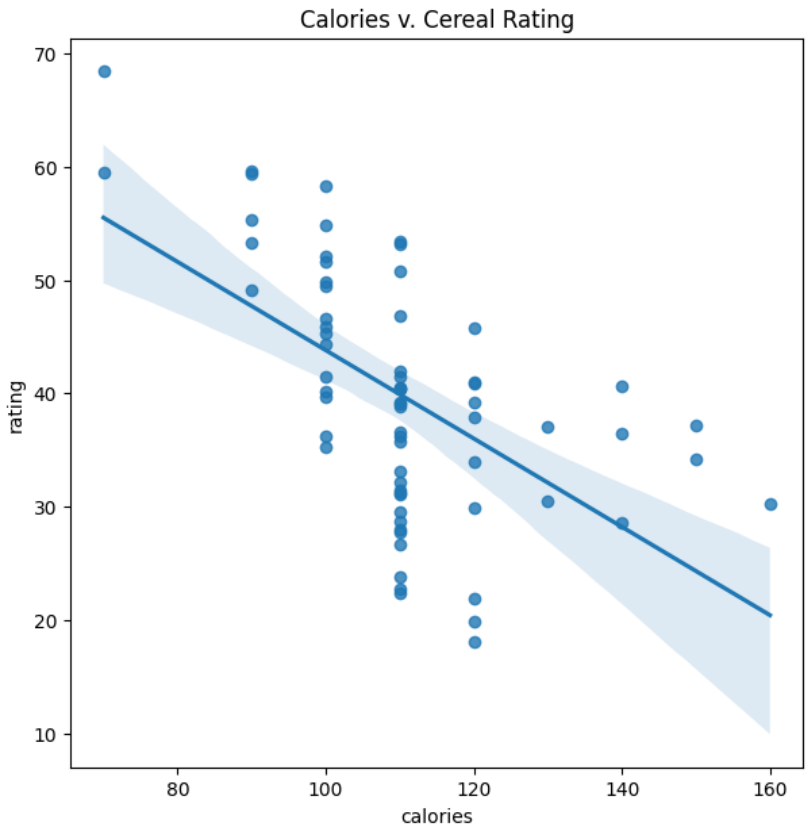
Out [146]: Text(0.5, 1.0, 'Sugar v. Cereal Rating')



In [147]:
```
#We visualize the relation between Calories & Rating
x_calories=df["calories"]
```

```
plt.figure(figsize=(7,7))
sns.regplot(x=x_calories,y=y_rating)
plt.title('Calories v. Cereal Rating')
```

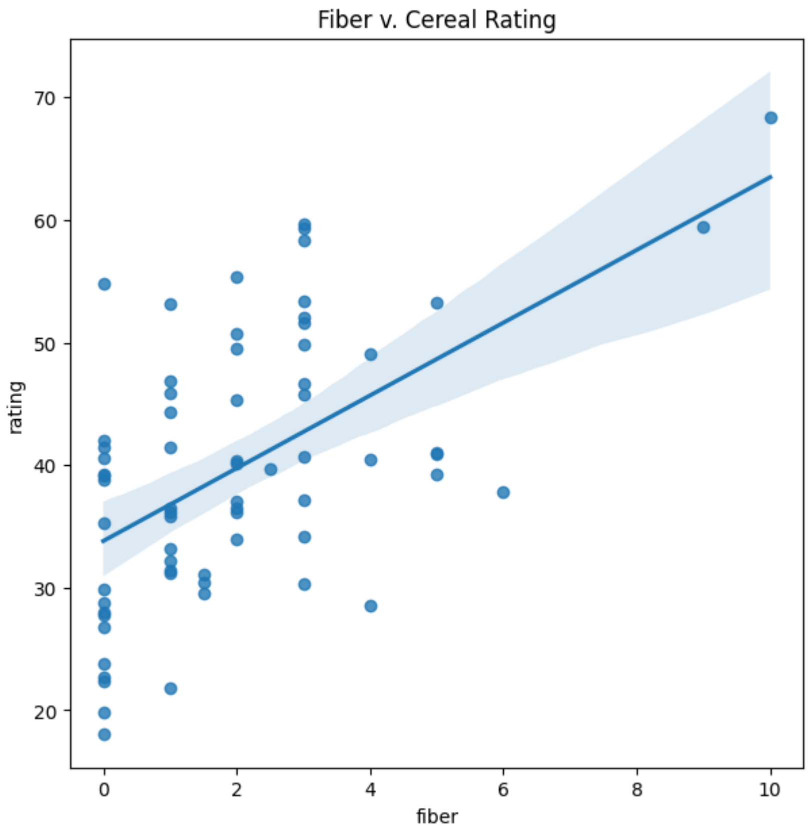
Out [147]: Text(0.5, 1.0, 'Calories v. Cereal Rating')



In [148]:

```
#Let's check for relationship between fiber & ratings
x_fiber=df["fiber"]
plt.figure(figsize=(7,7))
sns.regplot(x=x_fiber,y=y_rating)
plt.title('Fiber v. Cereal Rating')
```

Out [148]: Text(0.5, 1.0, 'Fiber v. Cereal Rating')

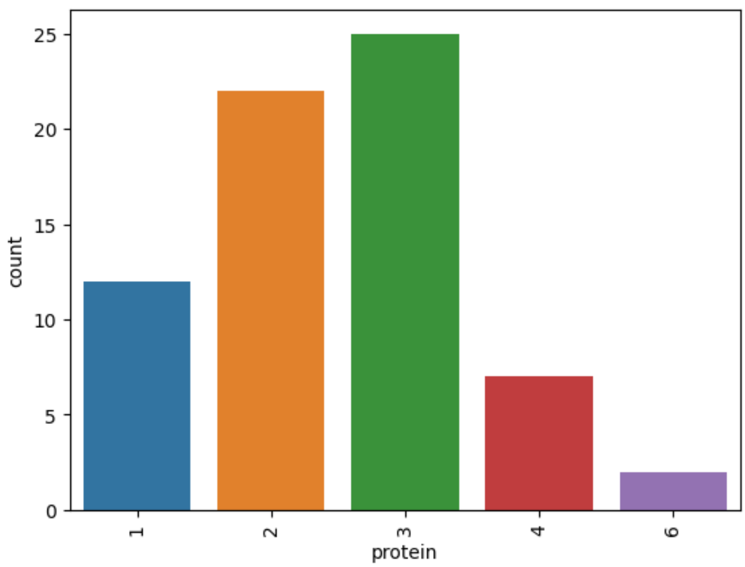


In [149]:

```
sns.countplot(x='protein',data=df)
plt.xticks(rotation=90)
```

Out [149]:

```
(array([0, 1, 2, 3, 4]),
 [Text(0, 0, '1'),
  Text(1, 0, '2'),
  Text(2, 0, '3'),
  Text(3, 0, '4'),
  Text(4, 0, '6')])
```



In [150]:

```
df.corr()
```

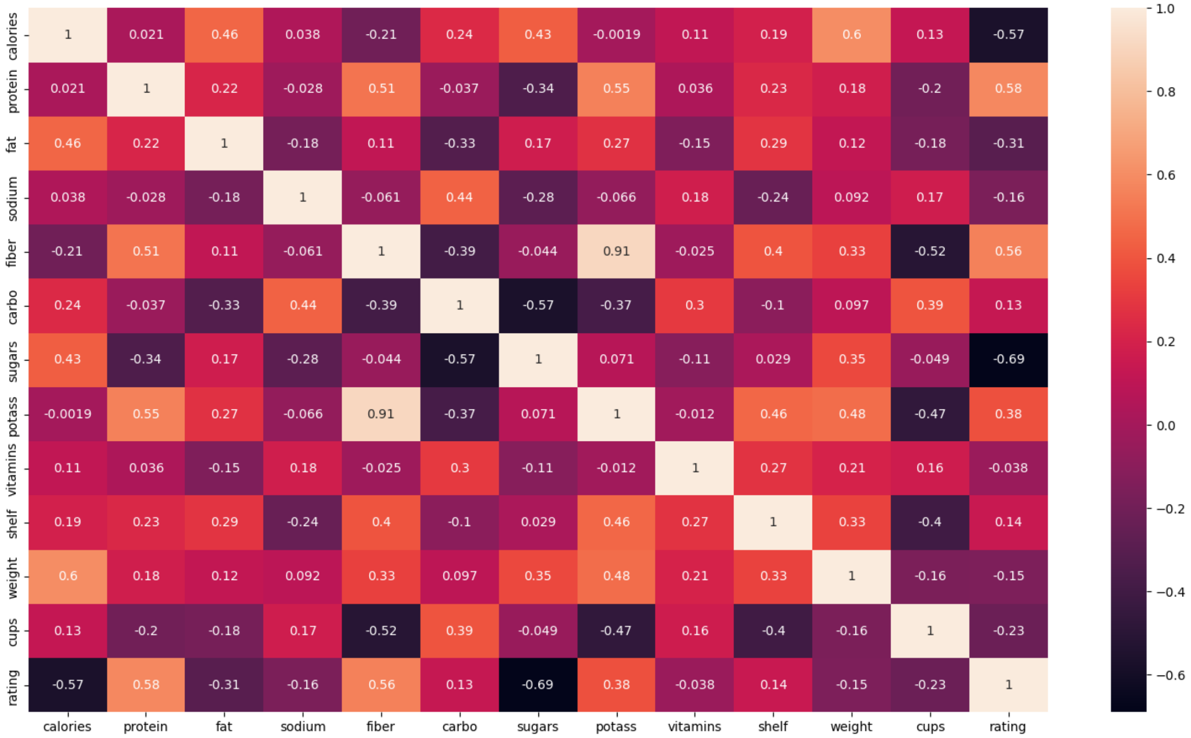
Out [150]:

	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
calories	1.000000	0.021265	0.456576	0.038106	-0.206126	0.235623	0.425517	-0.001897	0.110722	0.188527	0.601348	0.125399	-0.570107
protein	0.021265	1.000000	0.217966	-0.027686	0.511351	-0.036759	-0.341675	0.552241	0.035538	0.232868	0.180902	-0.196138	0.583036
fat	0.456576	0.217966	1.000000	-0.183803	0.113585	-0.325812	0.170691	0.269039	-0.145259	0.285335	0.119897	-0.176339	-0.306151

	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
sodium	0.038106	-0.027686	-0.183803	1.000000	-0.060901	0.439055	-0.281675	-0.065647	0.179877	-0.238648	0.092207	0.171050	-0.159142
fiber	-0.206126	0.511351	0.113585	-0.060901	1.000000	-0.389852	-0.044429	0.914078	-0.025348	0.398211	0.332647	-0.516464	0.555233
carbo	0.235623	-0.036759	-0.325812	0.439055	-0.389852	1.000000	-0.565990	-0.373787	0.303795	-0.101616	0.097088	0.394486	0.134003
sugars	0.425517	-0.341675	0.170691	-0.281675	-0.044429	-0.565990	1.000000	0.070668	-0.110087	0.028833	0.354491	-0.049264	-0.690449
potass	-0.001897	0.552241	0.269039	-0.065647	0.914078	-0.373787	0.070668	1.000000	-0.011509	0.464642	0.481634	-0.469893	0.380286
vitamins	0.110722	0.035538	-0.145259	0.179877	-0.025348	0.303795	-0.110087	-0.011509	1.000000	0.274373	0.207248	0.162506	-0.037703
shelf	0.188527	0.232868	0.285335	-0.238648	0.398211	-0.101616	0.028833	0.464642	0.274373	1.000000	0.333264	-0.401040	0.141588
weight	0.601348	0.180902	0.119897	0.092207	0.332647	0.097088	0.354491	0.481634	0.207248	0.333264	1.000000	-0.161512	-0.154102
cups	0.125399	-0.196138	-0.176339	0.171050	-0.516464	0.394486	-0.049264	-0.469893	0.162506	-0.401040	-0.161512	1.000000	-0.225457
rating	-0.570107	0.583036	-0.306151	-0.159142	0.555233	0.134003	-0.690449	0.380286	-0.037703	0.141588	-0.154102	-0.225457	1.000000

In [151]:
```
plt.figure(figsize=(18,10))
sns.heatmap(df.corr(),annot=True)
```

Out [151]: <Axes: >



In [152]:
```
df.dtypes
```

Out [152]:
```
name          object
mfr           object
type          object
calories       int64
protein        int64
fat            int64
sodium         int64
fiber        float64
carbo        float64
sugars         int64
potass         int64
vitamins       int64
shelf          int64
weight       float64
cups         float64
rating       float64
dtype: object
```

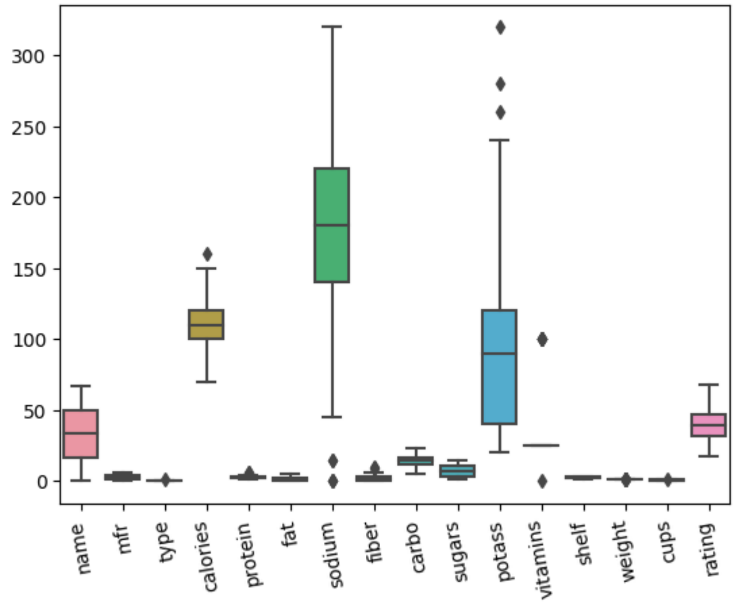
In [153]:
```
#label encoding to the categorical label into numerical label
#each unique category is  assigned a unique integer
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['name']=le.fit_transform(df['name'])
df['mfr']=le.fit_transform(df['mfr'])
df['type']=le.fit_transform(df['type'])
```

In [154]:
```
df.dtypes
```

Out [154]:
```
name          int64
mfr           int64
type          int64
calories      int64
protein       int64
fat           int64
sodium        int64
fiber       float64
carbo       float64
sugars        int64
potass        int64
vitamins      int64
shelf         int64
weight      float64
cups        float64
rating      float64
dtype: object
```

In [155]:
```
# to find outlayers
sns.boxplot(df)
plt.xticks(rotation=100)
```

Out [155]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, 'name'),
  Text(1, 0, 'mfr'),
  Text(2, 0, 'type'),
  Text(3, 0, 'calories'),
  Text(4, 0, 'protein'),
  Text(5, 0, 'fat'),
  Text(6, 0, 'sodium'),
  Text(7, 0, 'fiber'),
  Text(8, 0, 'carbo'),
  Text(9, 0, 'sugars'),
  Text(10, 0, 'potass'),
  Text(11, 0, 'vitamins'),
  Text(12, 0, 'shelf'),
  Text(13, 0, 'weight'),
  Text(14, 0, 'cups'),
  Text(15, 0, 'rating')])
```



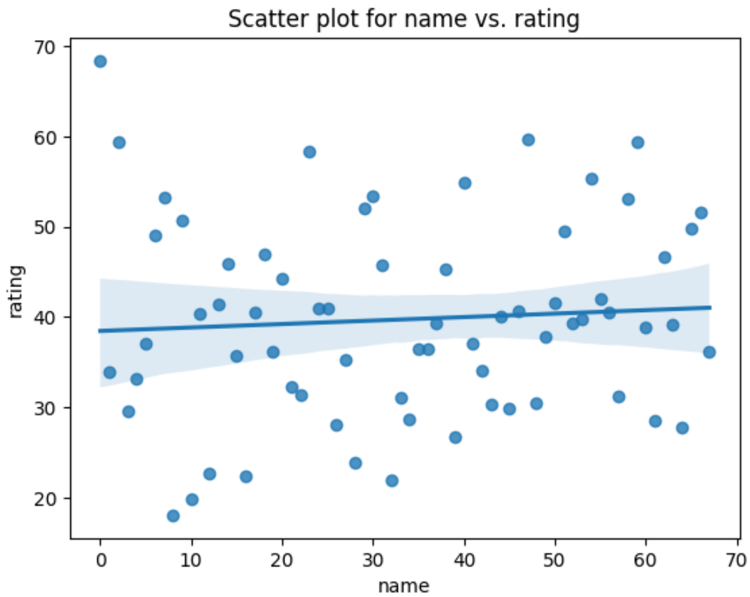
Feature selection using chi_square

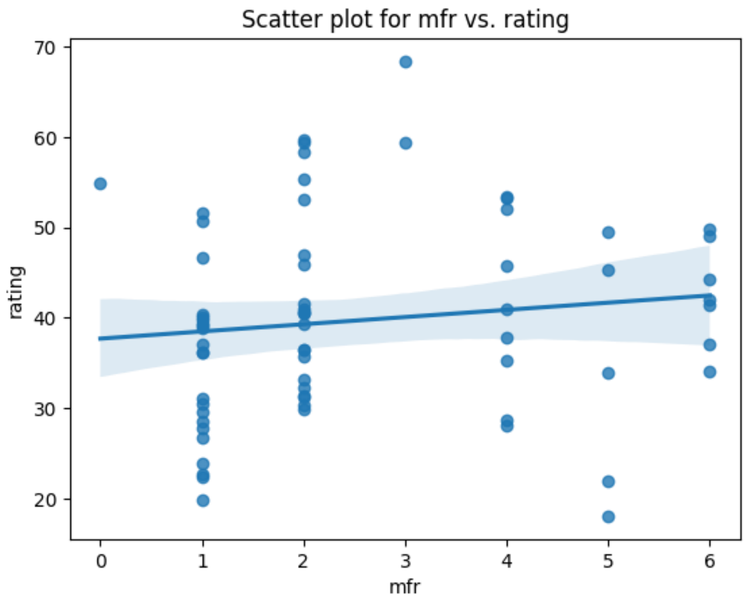
In [156]:
```
#feature slection is used to choose most relevent and informative features from the dataset

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
x=df.drop(columns=['rating']).astype(int) #input feature
y=df['rating'].astype(int)  #class label
k=10

selector=SelectKBest(chi2, k=k)

bst=selector.fit_transform(x, y)
bst

selected_feature_indices = selector.get_support(indices=True)

# Print the names of the selected features
selected_features = x.columns[selected_feature_indices]
print("Selected Features:", selected_features.tolist())
```

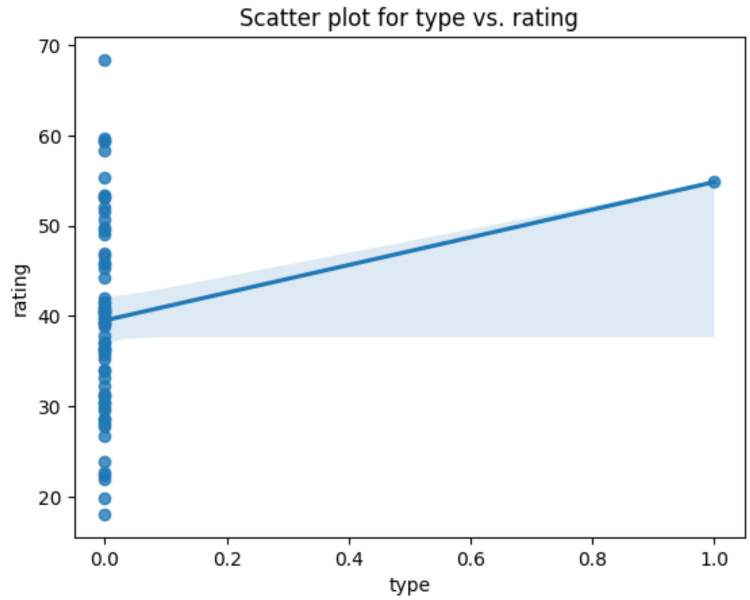
```
Selected Features: ['name', 'mfr', 'type', 'calories', 'fat', 'sodium', 'fiber', 'sugars', 'potass', 'vitamins']
```

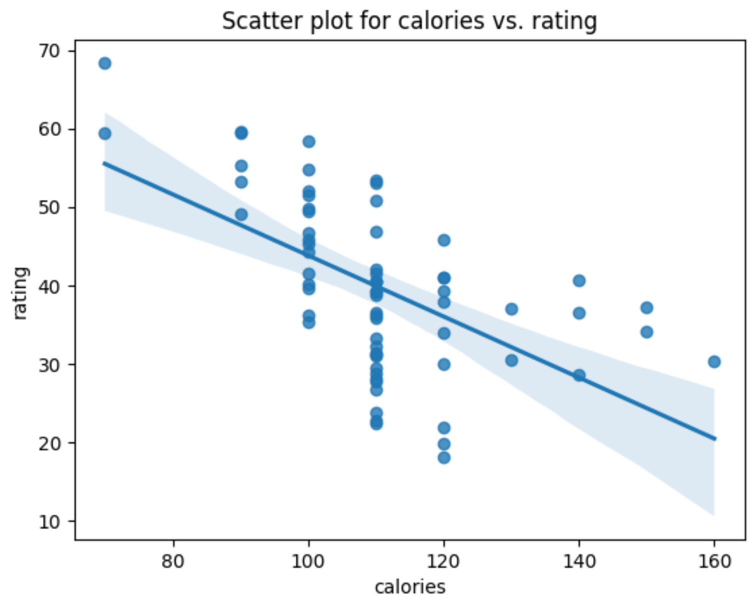
In [157]:

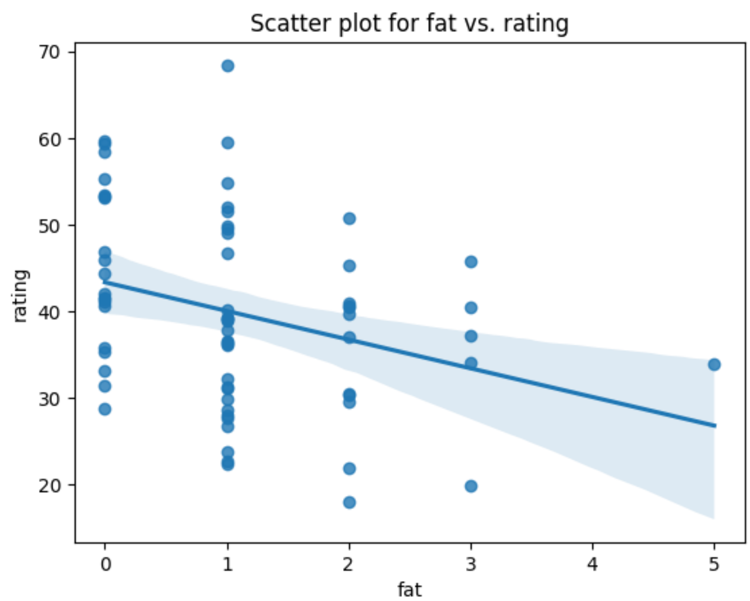
```
# it will generate scatter plots to visualize the relationship between each feature
x_axis=['name','mfr','type','calories','fat','sodium','fiber','sugars','potass','vitamins']
y_axis=df['rating']

for i in x_axis:
    sns.regplot(x=df[i], y=y_axis)
    plt.title(f'Scatter plot for {i} vs. rating')   #
    plt.xlabel(i)
    plt.ylabel('rating')
    plt.show()
```

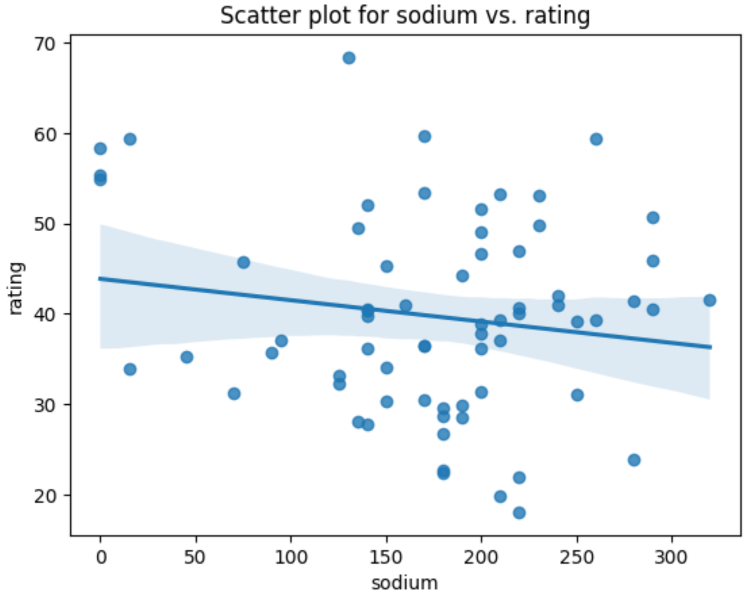


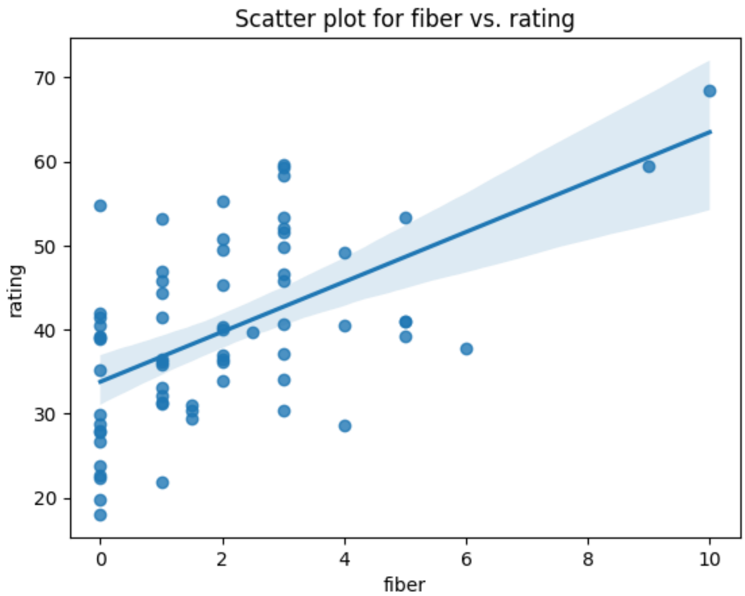



Scatter plot for type vs. rating
rating
70
60
50
40
30
20
0.0
0.2
0.4
0.6
0.8
1.0
type

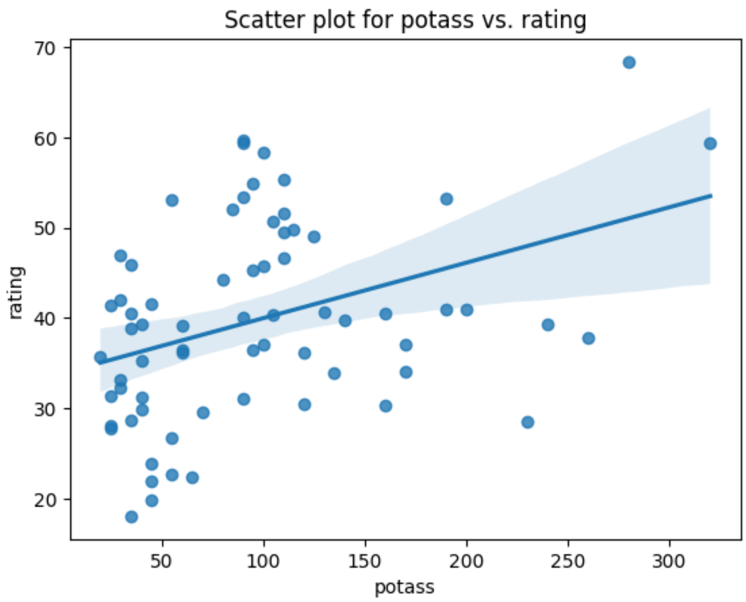

Scatter plot for calories vs. rating
rating
70
60
50
40
30
20
10
80
100
120
140
160
calories

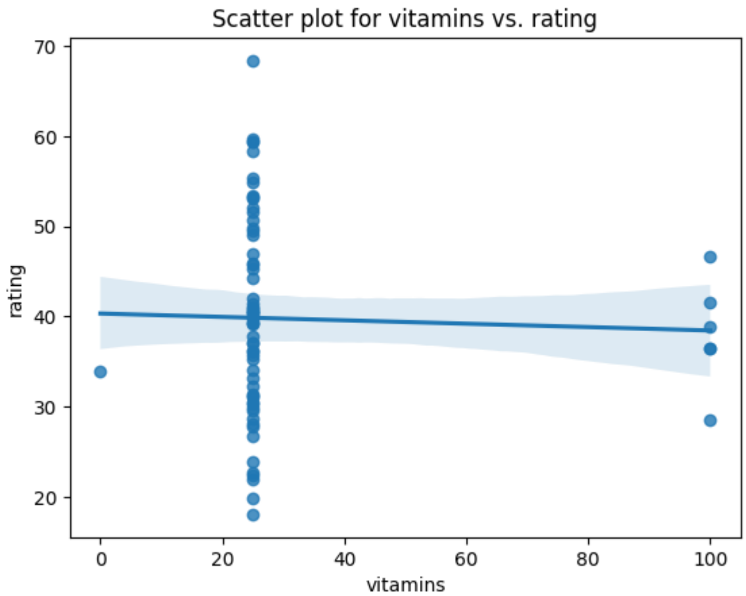

Scatter plot for fat vs. rating
rating
70
60
50
40
30
20
0
1
2
3
4
5
fat


Scatter plot for sodium vs. rating
rating
sodium
70
60
50
40
30
20
0
50
100
150
200
250
300


Scatter plot for fiber vs. rating
rating
fiber
70
60
50
40
30
20
0
2
4
6
8
10


Scatter plot for sugars vs. rating
rating
sugars
70
60
50
40
30
20
2
4
6
8
10
12
14





In [158]:

```
#convert training and testing data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(bst,y,test_size=0.30,random_state=42)
```

In [159]:

```
#multiple linear regression
from sklearn.linear_model import LinearRegression
model=LinearRegression()

# to get the default values for the parameters
model.get_params()
```

Out [159]: {'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'positive': False}

In [160]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [161]:

```
# hyperparmeter tuning to improve its performance

from sklearn.model_selection import GridSearchCV
# Define hyperparameter grid
parameter={'copy_X': [True,False], 'fit_intercept': [True,False], 'n_jobs': [None,1,5,7,6], 'positive':[True, False]}

gsv=GridSearchCV(model,parameter,cv=10,scoring='accuracy')
```

```
gsv.fit(x_train,y_train)
# Access the best hyperparameters
gsv.best_params_
```

Out [161]: {'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'positive': True}

In [162]:
```
# multiple linear regression model creation
model1=LinearRegression(positive=True)
model1.fit(x_train,y_train)
y_pred=model1.predict(x_test)
```

In [163]:
```
df3=pd.DataFrame({'actual_value':y_test,'predicted_value':y_pred,'difference':y_test-y_pred})
df3
```

Out [163]:

	actual_value	predicted_value	difference
49	40	43.448465	-3.448465
18	22	28.408823	-6.408823
6	33	31.755861	1.244139
11	50	34.810669	15.189331
31	23	29.704519	-6.704519
44	37	47.164192	-10.164192
67	53	37.586493	15.413507
7	37	34.378770	2.621230
70	28	50.900011	-22.900011
14	22	27.976924	-5.976924
28	41	48.338666	-7.338666
74	49	49.755584	-0.755584
50	59	43.556440	15.443560
0	68	64.597371	3.402629
60	55	40.733428	14.266572
61	41	37.939334	3.060666
52	37	56.636692	-19.636692
9	53	48.522922	4.477078
45	34	47.272166	-13.272166
34	45	43.956645	1.043355
39	36	42.106875	-6.106875

In [164]:
```
print('slpe is')
list(zip(x,model1.coef_))
```

```
slpe is
```

Out [164]:
```
[('name', 0.10797465879653796),
 ('mfr', 1.0638999261430828),
 ('type', 24.063685260670347),
 ('calories', 0.0),
 ('protein', 0.0),
 ('fat', 0.0),
 ('sodium', 3.578834283994973),
 ('fiber', 0.0),
 ('carbo', 0.0),
 ('sugars', 0.04422654546184028)]
```

In [165]:
```
#performance evaluation
#mean absolute error
from sklearn.metrics import mean_absolute_error,mean_absolute_percentage_error,mean_squared_error,r2_score
r=r2_score(y_test,y_pred)
print('r2 score',r2_score(y_test,y_pred))
print('MAE',mean_absolute_error(y_test,y_pred))
print('error percentage is',mean_absolute_percentage_error(y_test,y_pred))
```

```
r2 score 0.23361128760308214
MAE 8.517856348368419
error percentage is 0.22738290686542625
```

Decision tree algorithm

In [166]:
```
# decision tree algorithm
from sklearn.tree import DecisionTreeRegressor
dec=DecisionTreeRegressor()
```

```
dec.fit(x_train,y_train)
y_pred1=dec.predict(x_test)
r1=r2_score(y_test,y_pred1)
print('r2 score',r2_score(y_test,y_pred1))
print('mean_absolute_percentage_error',mean_absolute_percentage_error(y_test,y_pred1))
```

```
r2 score 0.6499784933021998
mean_absolute_percentage_error 0.13888803699403063
```

Random forest algorithm

In [167]:

```
 #random forest regressor
from sklearn.ensemble import RandomForestRegressor
random=RandomForestRegressor()
random.fit(x_train,y_train)
y_pred2=random.predict(x_test)
r2=r2_score(y_test,y_pred2)
print('r2 score',r2_score(y_test,y_pred2))
print('mean_absolute_percentage_error ',mean_absolute_percentage_error(y_pred2,y_test))
```

```
r2 score 0.754869002703699
mean_absolute_percentage_error  0.13058947070930427
```