Machine Learning Libraries

Objective

- Solve the below 10-step problem:
    1. Download the 80-cereals dataset from https://www.kaggle.com/crawford/80-cereals (https://www.kaggle.com/crawford/80-cereals)
    2. Load the dataset using pd.read_csv method
    3. a. In the column "mfr", replace the column "K" as "Kellogg's", "G" as "Nestle" and all other values as "Other Brands" b. In the column "type", replace "C" with "Type 1" and "H" with "Type 2"
    4. Visualise the count of above two features "mfr" and "type" with a bar-plot
    5. Describe the five-number summary and boxplots of the features - protien, sugars, fat, carbo
    6. Plot Histograms for the features - fat, carbo, sodium, fiber
    7. Split the datasets into following ratios: 60:40, 70:30, 80:20. Write down what happens when you give "random_state" parameter with a constant value and what happens if you do not mention the parameter at all.
    8. Apply MinMaxScaler() and StandardScaler() to the following features: calories, protien, fat, sodium, fiber, carbo, sugars.
    9. Does the standard or min-max scaling make a difference in value distribution? Support your answers with some visualisations on the above dataset.
    10. As an extension of 7th step, Generate a new Pandas DataFrame with the following columns based on the Training Dataset: Split Ratio | Random State | Total Number of Entries | Count of Kellogg's | Count of Nestle | Count of Other Brands
- Additional Notes:
    - Display the dataset/part of dataset wherever applicable
    - Make use of various visualization methods wherever imperative
    - Try to explore further and see whether there are any additional inferences that you could make using sklearn/pandas/matplotlib/seaborn.

Problem definition

- Works on 80-cereals dataset
- Exploring some code of machine learning libraies
- Visualization some features of this dataset

Approach

- Loading the 80-cereals dataset using pd.read_csv method
- Changing the some columns value of this dataset to another values using replace() function like in the column "mfr", replace the column "K" as "Kellogg's", "G" as "Nestle" and all other

values as "Other Brands" b. In the column "type", replace "C" with "Type 1" and "H" with "Type 2"

- Visualizing the count of two features "mfr" and "type" with a bar-plot using seaborn library
- Describe the five-number summary of the features - protien, sugars, fat, carbo using describe() function
- Visualizing boxplots of the each features - protien, sugars, fat, carbo and together also using seaborn library
- Plotting histograms for the each features - fat, carbo, sodium, fiber using seaborn library
- Plotting histograms for the features together - fat, carbo, sodium, fiber using matplotlib library and also using random.randn() and random.rand() method
- Split this dataset into following ratios: 60:40, 70:30, 80:20 with importing train_test_split from sklearn.model_selection
- Importing MinMaxScaler() and StandardScaler() from sklearn.preprocessing and applying to the following features: calories, protien, fat, sodium, fiber, carbo, sugars
- After making standard or min-max scaling, from visualisation of some features histograms we can able the answer that both have same value distribution or not
- Generating a new Pandas DataFrame using pandas library with the following columns based on the Training Dataset: Split Ratio | Random State | Total Number of Entries | Count of Kellogg's | Count of Nestle | Count of Other Brands
- Again split this dataset into following ratios: 60:40, 70:30, 80:20 and adding all values of Split Ratio | Random State | Total Number of Entries | Count of Kellogg's | Count of Nestle | Count of Other Brands using append() function and print that new dataframe

Code

In [1]:
```
import pandas as pd  # It used to import the panda file
```

In [2]:
```
import jupyterthemes as jt
from jupyterthemes.stylefx import set_nb_theme
```

In [3]:
```
set_nb_theme('chesterish')
```

Out[3]:

2. Load the dataset using pd.read_csv method

In [4]:
```
data = pd.read_csv("C:/Users/HP/Downloads/archive (6)/cereal.csv")
# Loading the dataset using pd.read_csv method
```

In [5]:
```
1 data
```

Out[5]:

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

77 rows × 16 columns

3. a. In the column "mfr", replace the column "K" as "Kellogg's", "G" as "Nestle" and all other values as "Other Brands"

In [6]:
```
1 data['mfr'].replace({'K':"Kellogg's",'G':'Nestle'},inplace=True)
2 # It used to replace the values of 'mfr' column into the values
```

In [7]:
```
data # show the data
```

Out[7]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins
0	100% Bran	N	C	70	4	1	130	10.0	5.0	6	280	25
1	100% Natural Bran	Q	C	120	3	5	15	2.0	8.0	8	135	0
2	All-Bran	Kellogg's	C	70	4	1	260	9.0	7.0	5	320	25
3	All-Bran with Extra Fiber	Kellogg's	C	50	4	0	140	14.0	8.0	0	330	25
4	Almond Delight	R	C	110	2	2	200	1.0	14.0	8	-1	25
...	...	...	...	...	...	...	...	...	...	...	...	..
72	Triples	Nestle	C	110	2	1	250	0.0	21.0	3	60	25
73	Trix	Nestle	C	110	1	1	140	0.0	13.0	12	25	25
74	Wheat Chex	R	C	100	3	1	230	3.0	17.0	3	115	25
75	Wheaties	Nestle	C	100	3	1	200	3.0	17.0	3	110	25
76	Wheaties Honey Gold	Nestle	C	110	2	1	200	1.0	16.0	8	60	25

77 rows × 16 columns

In [8]:
```
for i in data['mfr']: # It Iterating the all values of column 'mfr'
    # And its Checking the value same as kellog's and nestle or not in the d
    if i!="Kellogg's" and i!="Nestle":

        data['mfr'].replace({i:'Other Brands'},inplace=True)
        # And if the values are not same, change it into the Other Brands
```

In [9]:
```
1 data
```

Out[9]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins
0	100% Bran	Other Brands	C	70	4	1	130	10.0	5.0	6	280	25
1	100% Natural Bran	Other Brands	C	120	3	5	15	2.0	8.0	8	135	0
2	All-Bran	Kellogg's	C	70	4	1	260	9.0	7.0	5	320	25
3	All-Bran with Extra Fiber	Kellogg's	C	50	4	0	140	14.0	8.0	0	330	25
4	Almond Delight	Other Brands	C	110	2	2	200	1.0	14.0	8	-1	25
...	...	...	...	...	...	...	...	...	...	...	...	..
72	Triples	Nestle	C	110	2	1	250	0.0	21.0	3	60	25
73	Trix	Nestle	C	110	1	1	140	0.0	13.0	12	25	25
74	Wheat Chex	Other Brands	C	100	3	1	230	3.0	17.0	3	115	25
75	Wheaties	Nestle	C	100	3	1	200	3.0	17.0	3	110	25
76	Wheaties Honey Gold	Nestle	C	110	2	1	200	1.0	16.0	8	60	25

77 rows × 16 columns

3. b. In the column "type", replace "C" with "Type 1" and "H" with "Type 2"

In [10]:
```
1 # Replace the some values of 'type' column into other values
2 data['type'].replace({'C':'Type 1','H':'Type 2'},inplace=True)
```

In [11]:
```
1 data
```

Out[11]:

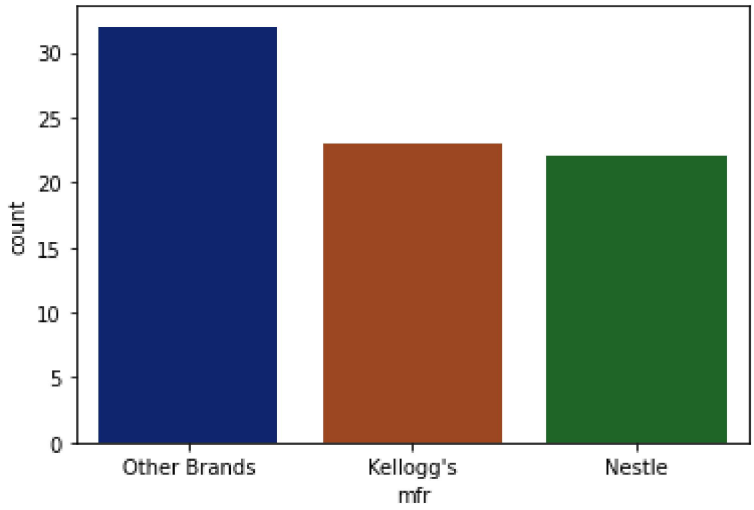
	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins
0	100% Bran	Other Brands	Type 1	70	4	1	130	10.0	5.0	6	280	25
1	100% Natural Bran	Other Brands	Type 1	120	3	5	15	2.0	8.0	8	135	(
2	All-Bran	Kellogg's	Type 1	70	4	1	260	9.0	7.0	5	320	25
3	All-Bran with Extra Fiber	Kellogg's	Type 1	50	4	0	140	14.0	8.0	0	330	25
4	Almond Delight	Other Brands	Type 1	110	2	2	200	1.0	14.0	8	-1	25
...	...	...	...	...	...	...	...	...	...	...	...	..
72	Triples	Nestle	Type 1	110	2	1	250	0.0	21.0	3	60	25
73	Trix	Nestle	Type 1	110	1	1	140	0.0	13.0	12	25	25
74	Wheat Chex	Other Brands	Type 1	100	3	1	230	3.0	17.0	3	115	25
75	Wheaties	Nestle	Type 1	100	3	1	200	3.0	17.0	3	110	25
76	Wheaties Honey Gold	Nestle	Type 1	110	2	1	200	1.0	16.0	8	60	25

77 rows × 16 columns

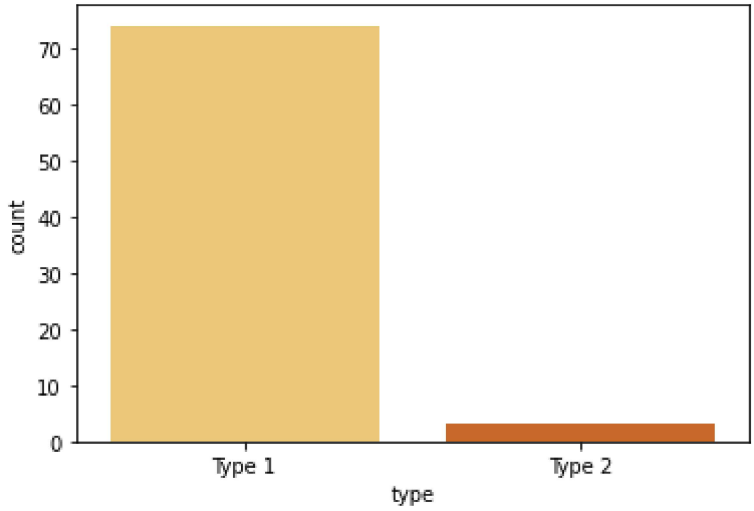
4. Visualise the count of above two features "mfr" and "type" with a bar-plot

In [12]:
```
1 # Import seaborn
2 import seaborn as sns
```

In [86]:
```
# Barplot of counting the total numbers of all values of 'mfr' column
count=sns.countplot(x='mfr',data=data, palette='dark')
```



In [91]:
```
# Barplot of counting the total numbers of all values of 'type' column
count1=sns.countplot(x='type',data=data , palette = "YlOrBr")
```



5. Describe the five-number summary and boxplots of the features - protien, sugars, fat, carbo

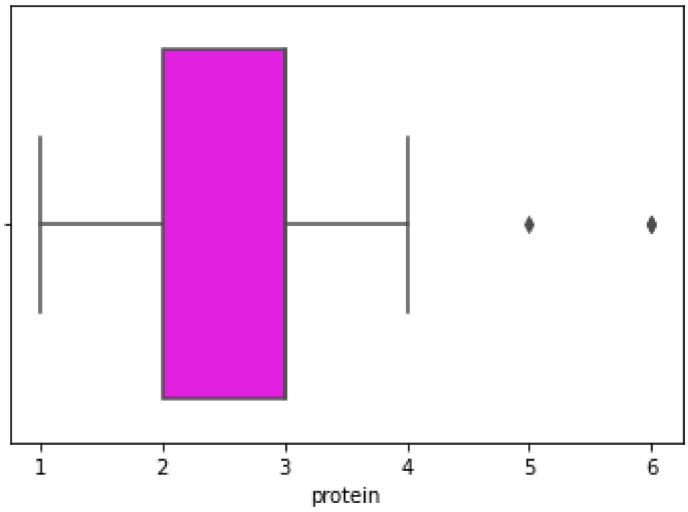
In [15]:
```
# Describing all things of the fatures- protein, sugars, fat, carbo
data[['protein','sugars','fat','carbo']].describe()
```

Out[15]:

	protein	sugars	fat	carbo
count	77.000000	77.000000	77.000000	77.000000
mean	2.545455	6.922078	1.012987	14.597403
std	1.094790	4.444885	1.006473	4.278956
min	1.000000	-1.000000	0.000000	-1.000000
25%	2.000000	3.000000	0.000000	12.000000
50%	3.000000	7.000000	1.000000	14.000000
75%	3.000000	11.000000	2.000000	17.000000
max	6.000000	15.000000	5.000000	23.000000

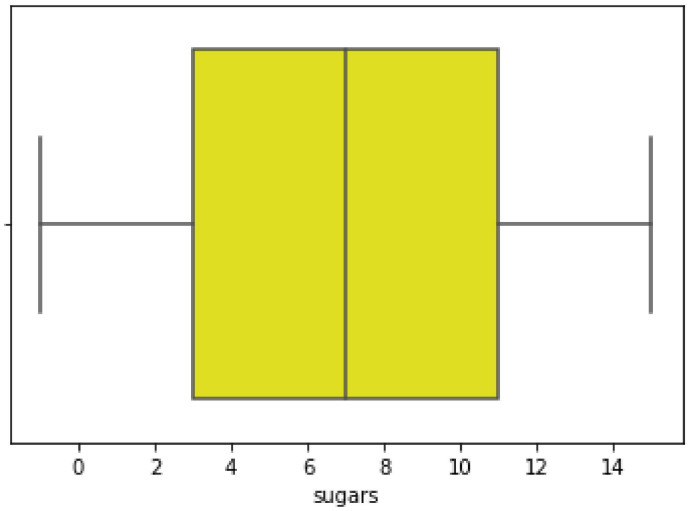
In [16]:
```
sns.boxplot(x='protein',data=data, color = 'magenta') # Boxplot of protein
```

Out[16]: <AxesSubplot:xlabel='protein'>



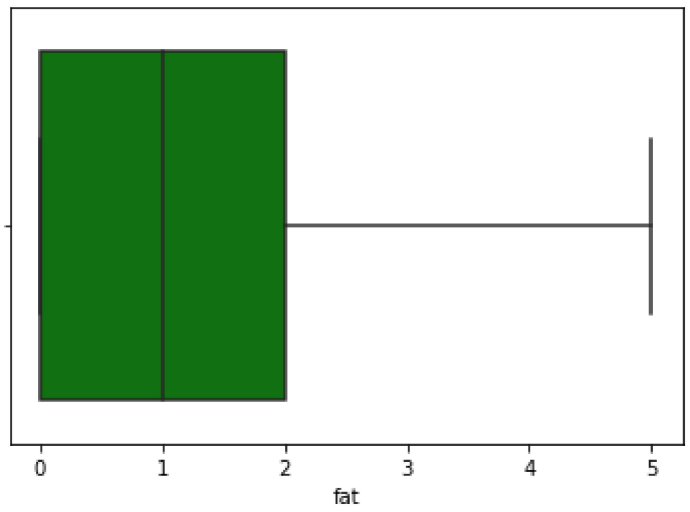
In [17]:
```
# Boxplot of sugars
sns.boxplot(x='sugars',data=data,color='yellow')
```

Out[17]: <AxesSubplot:xlabel='sugars'>



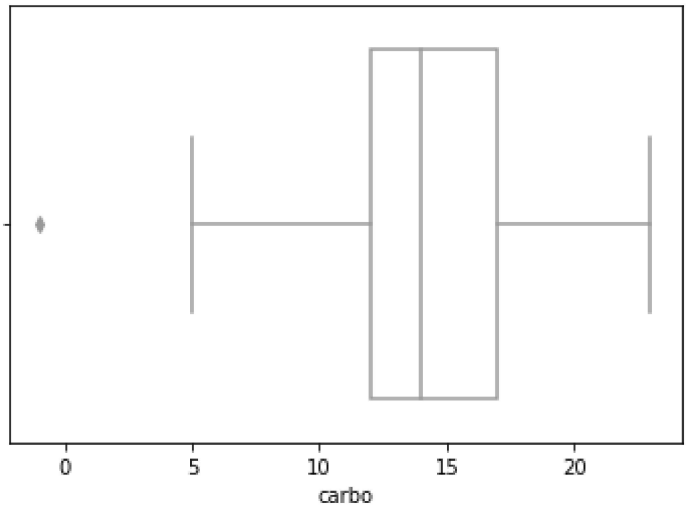
In [18]:
```
# Boxplot of fat
sns.boxplot(x='fat',data=data,color='green')
```

Out[18]: <AxesSubplot:xlabel='fat'>



In [19]:
```
# Boxplot of carbo
sns.boxplot(x='carbo',data=data,color='white')
```

Out[19]: <AxesSubplot:xlabel='carbo'>

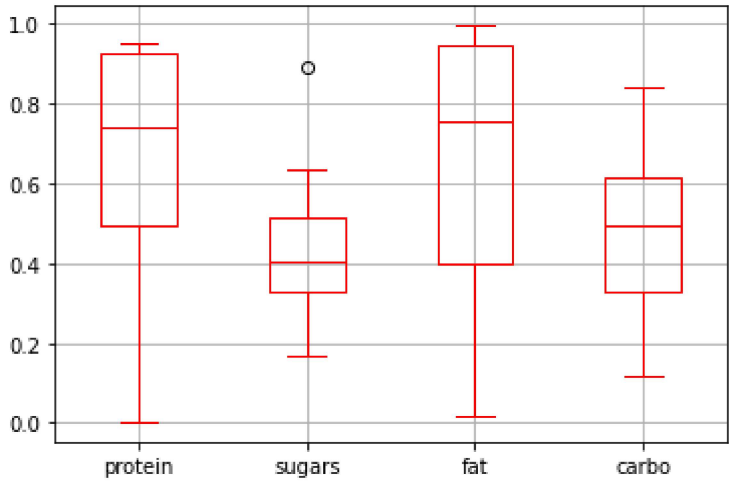


In [20]:
```
# Import numpy
import numpy as np
```

In [21]:
```
# Creating a new dataframe of protein, sugars, fat and carbo
data1=pd.DataFrame(data=np.random.random(size=(10,4)),columns=['protein','su
```

In [22]:
```
# Boxplot of that new dataframe which consist of protein, sugars, fat and ca
data1.boxplot(color='red')
```

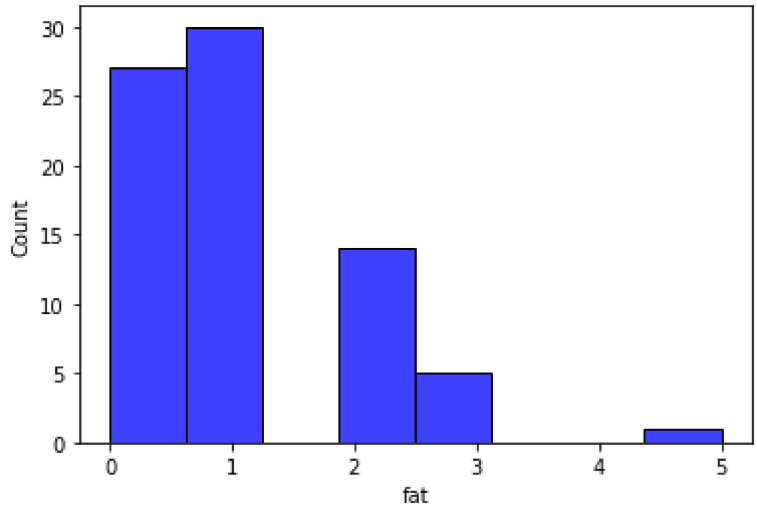
Out[22]: <AxesSubplot:>



6. Plot Histograms for the features - fat, carbo, sodium, fiber

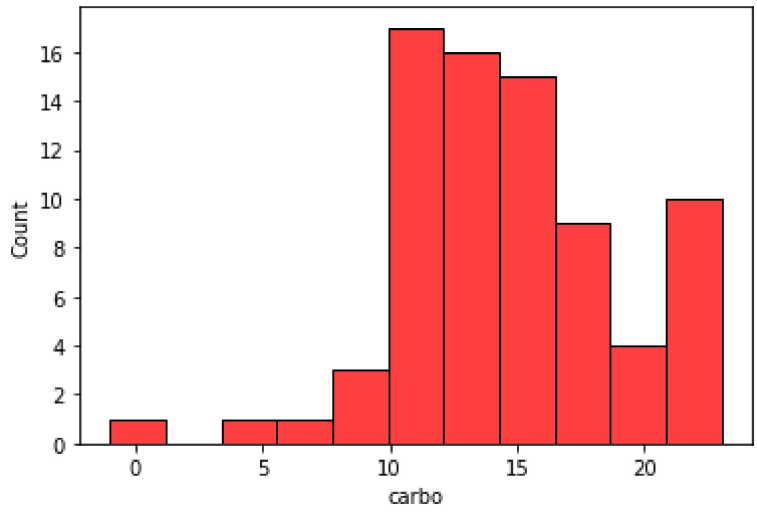
In [23]:
```
# Histogram of fat
sns.histplot(x='fat',data=data, color = "blue")
```

Out[23]: <AxesSubplot:xlabel='fat', ylabel='Count'>



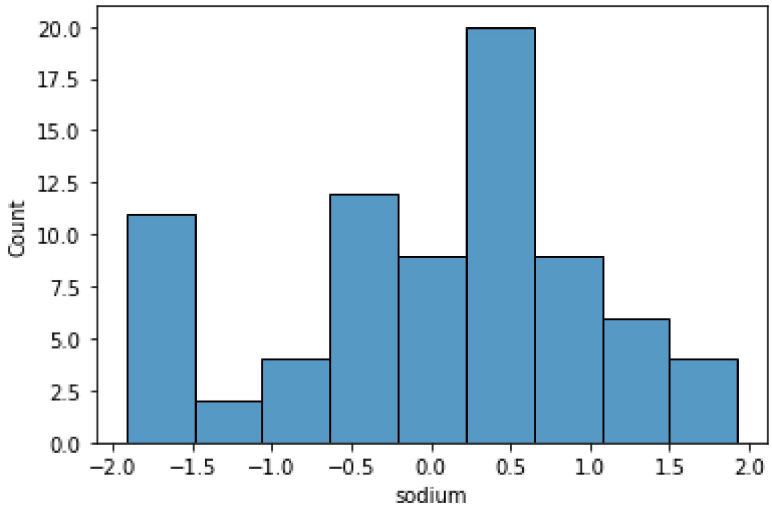
In [24]:
```
# Histogram of carbo
sns.histplot(x='carbo',data=data,color='red')
```

Out[24]: <AxesSubplot:xlabel='carbo', ylabel='Count'>



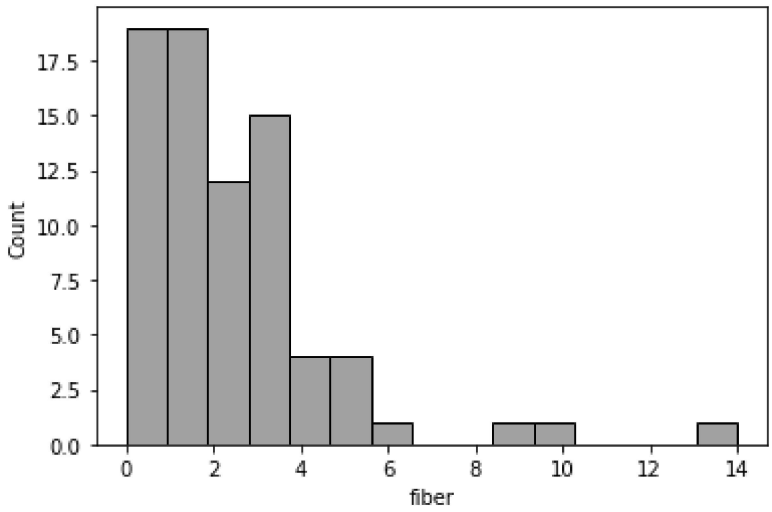
In [94]:
```
# Histogram of sodium
sns.histplot(x='sodium',data=data,palette='vlag')
```

Out[94]: <AxesSubplot:xlabel='sodium', ylabel='Count'>



In [26]:
```
# Histogram of fiber
sns.histplot(x='fiber',data=data,color='grey')
```

Out[26]: <AxesSubplot:xlabel='fiber', ylabel='Count'>

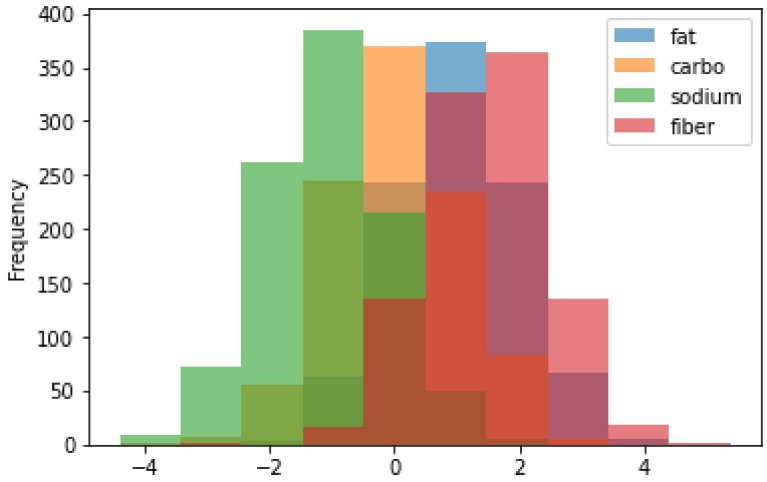


In [27]:
```
# Import matplotlib
import matplotlib.pyplot as plt
```

In [28]:
```
# Create a new dataframe with randomly standard normalising the main dataset
# using np.random.randn
df = pd.DataFrame({'fat': np.random.randn(1000) + 1, 'carbo': np.random.rand
                   'sodium': np.random.randn(1000) - 1,'fiber':np.random.ra
# Plotting the figure
plt.figure()
# Histogram of these features together
df.plot(kind='hist', alpha=0.6)
```

Out[28]:
```
<AxesSubplot:ylabel='Frequency'>

<Figure size 432x288 with 0 Axes>
```



In [29]:
```
df
```

Out[29]:

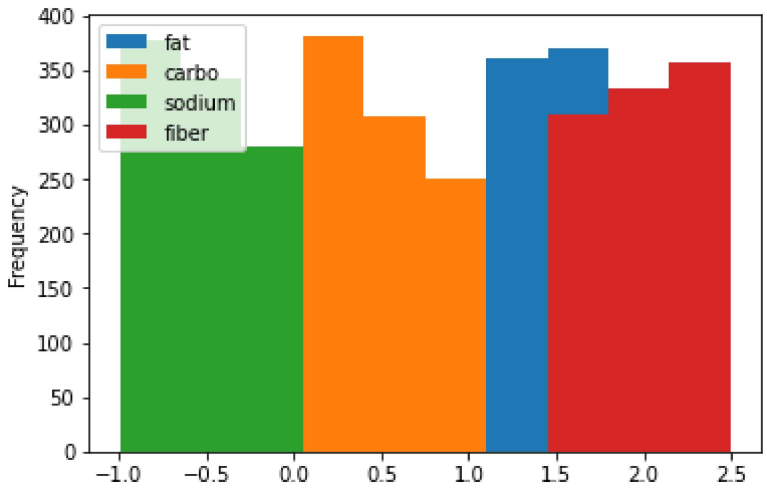
	fat	carbo	sodium	fiber
0	0.665593	0.780557	0.256517	2.298659
1	0.892994	0.049351	-1.254048	1.242300
2	0.240562	-0.972699	-0.609947	0.968830
3	0.796159	-0.721420	-0.592978	0.909012
4	0.701182	1.084950	-1.833453	1.584564
...	...	...	...	...
995	3.434756	0.594539	-0.721986	1.103442
996	2.166907	-1.033963	-1.495269	0.859825
997	0.889365	-0.207003	-1.971563	0.650245
998	1.106985	0.209013	-0.786063	2.218969
999	0.662700	-0.474909	-1.809491	0.370744

1000 rows × 4 columns

In [30]:
```
# Create a new dataframe with randomly normalising the main dataset of these
# using np.random.rand
df1 = pd.DataFrame({'fat': np.random.rand(1000) + 1, 'carbo': np.random.rand
                    'sodium': np.random.rand(1000) - 1,'fiber':np.random.ran
# Plotting the figure
plt.figure()
# Histogram of these features together
df1.plot(kind='hist', alpha=1)
```

Out[30]: <AxesSubplot:ylabel='Frequency'>

<Figure size 432x288 with 0 Axes>



In [31]:
```
df1
```

Out[31]:

	fat	carbo	sodium	fiber
0	1.873943	0.347757	-0.782043	1.614591
1	1.454679	0.344190	-0.897019	2.049527
2	1.342835	0.951539	-0.298935	2.111306
3	1.053201	0.375632	-0.206982	2.480823
4	1.423575	0.197365	-0.636385	2.137017
...	...	...	...	...
995	1.577191	0.107811	-0.107700	1.929592
996	1.149186	0.438619	-0.043836	2.080322
997	1.328675	0.423688	-0.085815	2.448182
998	1.163813	0.194059	-0.298962	1.975136
999	1.107119	0.990548	-0.397664	1.545682

1000 rows × 4 columns

- np.random.randn returns a random numpy array or scalar of sample(s), drawn randomly from

the standard normal distribution.

- np.random.rand returns a random numpy array or scalar whose element(s) are drawn randomly from the normal distribution over [0,1).

7. Split the datasets into following ratios: 60:40, 70:30, 80:20. Write down what happens when you give "random_state" parameter with a constant value and what happens if you do not mention the parameter at all.

In [32]:
```
# Import train_test_split
from sklearn.model_selection import train_test_split
```

60:40

In [33]:
```
# Spliting the main dataset into training and testing dataset with 60:40 rat
train_data,test_data = train_test_split(data, test_size=0.4)
```

In [34]:
```
train_data
```

Out[34]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vit
2	All-Bran	Kellogg's	Type 1	70	4	1	260	9.0	7.0	5	320	
0	100% Bran	Other Brands	Type 1	70	4	1	130	10.0	5.0	6	280	
38	Just Right Crunchy Nuggets	Kellogg's	Type 1	110	2	1	170	1.0	17.0	6	60	
17	Corn Pops	Kellogg's	Type 1	110	1	0	90	1.0	13.0	12	20	
49	Nutri-Grain Almond-Raisin	Kellogg's	Type 1	140	3	2	220	3.0	21.0	7	130	
19	Cracklin' Oat Bran	Kellogg's	Type 1	110	3	3	140	4.0	10.0	7	160	
54	Puffed Rice	Other	Type	50	1	0	0	0.0	13.0	0	15	

In [35]:
```
1 test_data
```

Out[35]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitar
68	Strawberry Fruit Wheats	Other Brands	Type 1	90	2	0	15	3.0	15.0	5	90	
26	Frosted Mini-Wheats	Kellogg's	Type 1	100	3	0	0	3.0	14.0	7	100	
6	Apple Jacks	Kellogg's	Type 1	110	2	0	125	1.0	11.0	14	30	
63	Shredded Wheat	Other Brands	Type 1	80	2	0	0	3.0	16.0	0	95	
60	Raisin Squares	Kellogg's	Type 1	90	2	0	0	2.0	15.0	6	110	
71	Total Whole Grain	Nestle	Type 1	100	3	1	200	3.0	16.0	3	110	
33	Grape-Nuts	Other Brands	Type 1	110	3	0	170	3.0	17.0	3	90	
39	Just Right Fruit & Nut	Kellogg's	Type 1	140	3	1	170	2.0	20.0	9	95	
10	Cap'n'Crunch	Other Brands	Type 1	120	1	2	220	0.0	12.0	12	35	
21	Crispix	Kellogg's	Type 1	110	2	0	220	1.0	21.0	3	30	
28	Fruitful Bran	Kellogg's	Type 1	120	3	0	240	5.0	14.0	12	190	
76	Wheaties Honey Gold	Nestle	Type 1	110	2	1	200	1.0	16.0	8	60	
56	Quaker Oat Squares	Other Brands	Type 1	100	4	1	135	2.0	14.0	6	110	
44	Muesli Raisins; Dates; & Almonds	Other Brands	Type 1	150	4	3	95	3.0	16.0	11	170	
36	Honey Nut Cheerios	Nestle	Type 1	110	3	1	250	1.5	11.5	10	90	
45	Muesli Raisins; Peaches; & Pecans	Other Brands	Type 1	150	4	3	150	3.0	16.0	11	170	
9	Bran Flakes	Other Brands	Type 1	90	3	0	210	5.0	13.0	5	190	
61	Rice Chex	Other Brands	Type 1	110	1	0	240	0.0	23.0	2	30	
57	Quaker Oatmeal	Other Brands	Type 2	100	5	2	0	2.7	-1.0	-1	110	
46	Mueslix Crispy Blend	Kellogg's	Type 1	160	3	2	150	3.0	17.0	13	160	
72	Triples	Nestle	Type 1	110	2	1	250	0.0	21.0	3	60	

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitar
1	100% Natural Bran	Other Brands	Type 1	120	3	5	15	2.0	8.0	8	135	
74	Wheat Chex	Other Brands	Type 1	100	3	1	230	3.0	17.0	3	115	
40	Kix	Nestle	Type 1	110	2	1	260	0.0	21.0	3	40	
66	Smacks	Kellogg's	Type 1	110	2	1	70	1.0	9.0	15	40	
5	Apple Cinnamon Cheerios	Nestle	Type 1	110	2	2	180	1.5	10.5	10	70	
15	Corn Chex	Other Brands	Type 1	110	2	0	280	0.0	22.0	3	25	
34	Great Grains Pecan	Other Brands	Type 1	120	3	3	75	3.0	13.0	4	100	
23	Double Chex	Other Brands	Type 1	100	2	0	190	1.0	18.0	5	80	
51	Oatmeal Raisin Crisp	Nestle	Type 1	130	3	2	170	1.5	13.5	10	120	
50	Nutri-grain Wheat	Kellogg's	Type 1	90	3	0	170	3.0	18.0	2	90	

In [36]:
```
print("The length of training data",len(train_data))
print("The length of testing data",len(test_data))
```

```
The length of training data 46
The length of testing data 31
```

70:30

In [37]:
```
# Spliting the main dataset into training and testing dataset with 70:30 rat
train_data1,test_data1 = train_test_split(data, test_size=0.3,random_state=4
```

In [38]:
```
1 train_data1
```

Out[38]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vita
65	Shredded Wheat spoon size	Other Brands	Type 1	90	3	0	0	3.0	20.0	0	120	
54	Puffed Rice	Other Brands	Type 1	50	1	0	0	0.0	13.0	0	15	
31	Golden Grahams	Nestle	Type 1	110	1	1	280	0.0	15.0	9	45	
7	Basic 4	Nestle	Type 1	130	3	2	210	2.0	18.0	8	100	
62	Rice Krispies	Kellogg's	Type 1	110	2	0	290	0.0	22.0	3	35	
44	Muesli Raisins; Dates; & Almonds	Other Brands	Type 1	150	4	3	95	3.0	16.0	11	170	

In [39]:
```
1 test_data1
```

Out[39]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	v
4	Almond Delight	Other Brands	Type 1	110	2	2	200	1.0	14.0	8	-1	
35	Honey Graham Ohs	Other Brands	Type 1	120	1	2	220	1.0	12.0	11	45	
10	Cap'n'Crunch	Other Brands	Type 1	120	1	2	220	0.0	12.0	12	35	
0	100% Bran	Other Brands	Type 1	70	4	1	130	10.0	5.0	6	280	
45	Muesli Raisins; Peaches; & Pecans	Other Brands	Type 1	150	4	3	150	3.0	16.0	11	170	
47	Multi-Grain Cheerios	Nestle	Type 1	100	2	1	220	2.0	15.0	6	90	
66	Smacks	Kellogg's	Type 1	110	2	1	70	1.0	9.0	15	40	
53	Product 19	Kellogg's	Type 1	100	3	0	320	1.0	20.0	3	45	
50	Nutri-grain Wheat	Kellogg's	Type 1	90	3	0	170	3.0	18.0	2	90	
28	Fruitful Bran	Kellogg's	Type 1	120	3	0	240	5.0	14.0	12	190	
68	Strawberry Fruit Wheats	Other Brands	Type 1	90	2	0	15	3.0	15.0	5	90	
74	Wheat Chex	Other Brands	Type 1	100	3	1	230	3.0	17.0	3	115	
18	Count Chocula	Nestle	Type 1	110	1	1	180	0.0	12.0	13	65	
12	Cinnamon Toast Crunch	Nestle	Type 1	120	1	3	210	0.0	13.0	9	45	
58	Raisin Bran	Kellogg's	Type 1	120	3	1	210	5.0	14.0	12	240	
33	Grape-Nuts	Other Brands	Type 1	110	3	0	170	3.0	17.0	3	90	
9	Bran Flakes	Other Brands	Type 1	90	3	0	210	5.0	13.0	5	190	
5	Apple Cinnamon Cheerios	Nestle	Type 1	110	2	2	180	1.5	10.5	10	70	
34	Great Grains Pecan	Other Brands	Type 1	120	3	3	75	3.0	13.0	4	100	
22	Crispy Wheat & Raisins	Nestle	Type 1	100	2	1	140	2.0	11.0	10	120	
30	Golden Crisp	Other Brands	Type 1	100	2	0	45	0.0	11.0	15	40	

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	v
40	Kix	Nestle	Type 1	110	2	1	260	0.0	21.0	3	40	
39	Just Right Fruit & Nut	Kellogg's	Type 1	140	3	1	170	2.0	20.0	9	95	
16	Corn Flakes	Kellogg's	Type 1	100	2	0	290	1.0	21.0	2	35	

In [40]:
```
print("The length of training data",len(train_data1))
print("The length of testing data",len(test_data1))
```

```
The length of training data 53
The length of testing data 24
```

80:20

In [41]:
```
# Spliting the main dataset into training and testing dataset with 80:20 rat
train_data2,test_data2 = train_test_split(data, test_size=0.2,random_state=4
```

In [42]:
```
1 train_data2
```

Out[42]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vita
9	Bran Flakes	Other Brands	Type 1	90	3	0	210	5.0	13.0	5	190	
5	Apple Cinnamon Cheerios	Nestle	Type 1	110	2	2	180	1.5	10.5	10	70	
34	Great Grains Pecan	Other Brands	Type 1	120	3	3	75	3.0	13.0	4	100	
22	Crispy Wheat & Raisins	Nestle	Type 1	100	2	1	140	2.0	11.0	10	120	
30	Golden Crisp	Other Brands	Type 1	100	2	0	45	0.0	11.0	15	40	
...	...	...	...	...	...	...	...	...	...	...	...	
20	Cream of Wheat (Quick)	Other Brands	Type 2	100	3	0	80	1.0	21.0	0	-1	
60	Raisin Squares	Kellogg's	Type 1	90	2	0	0	2.0	15.0	6	110	
71	Total Whole Grain	Nestle	Type 1	100	3	1	200	3.0	16.0	3	110	
14	Cocoa Puffs	Nestle	Type 1	110	1	1	180	0.0	12.0	13	55	
51	Oatmeal Raisin Crisp	Nestle	Type 1	130	3	2	170	1.5	13.5	10	120	

61 rows × 16 columns

In [43]:
```
1 test_data2
```

Out[43]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitar
4	Almond Delight	Other Brands	Type 1	110	2	2	200	1.0	14.0	8	-1	
35	Honey Graham Ohs	Other Brands	Type 1	120	1	2	220	1.0	12.0	11	45	
10	Cap'n'Crunch	Other Brands	Type 1	120	1	2	220	0.0	12.0	12	35	
0	100% Bran	Other Brands	Type 1	70	4	1	130	10.0	5.0	6	280	
45	Muesli Raisins; Peaches; & Pecans	Other Brands	Type 1	150	4	3	150	3.0	16.0	11	170	
47	Multi-Grain Cheerios	Nestle	Type 1	100	2	1	220	2.0	15.0	6	90	
66	Smacks	Kellogg's	Type 1	110	2	1	70	1.0	9.0	15	40	
53	Product 19	Kellogg's	Type 1	100	3	0	320	1.0	20.0	3	45	
50	Nutri-grain Wheat	Kellogg's	Type 1	90	3	0	170	3.0	18.0	2	90	
28	Fruitful Bran	Kellogg's	Type 1	120	3	0	240	5.0	14.0	12	190	
68	Strawberry Fruit Wheats	Other Brands	Type 1	90	2	0	15	3.0	15.0	5	90	
74	Wheat Chex	Other Brands	Type 1	100	3	1	230	3.0	17.0	3	115	
18	Count Chocula	Nestle	Type 1	110	1	1	180	0.0	12.0	13	65	
12	Cinnamon Toast Crunch	Nestle	Type 1	120	1	3	210	0.0	13.0	9	45	
58	Raisin Bran	Kellogg's	Type 1	120	3	1	210	5.0	14.0	12	240	
33	Grape-Nuts	Other Brands	Type 1	110	3	0	170	3.0	17.0	3	90	

In [44]:
```
1 print("The length of training data",len(train_data2))
2 print("The length of testing data",len(test_data2))
```

```
The length of training data 61
The length of testing data 16
```

- If we don't use random_state parameter then our dataset splits into training and testing

dataset randomly. For every time when it is run we get shuffled split dataset. But if we use random_state parameter then every time we get same split dataset.

8. Apply MinMaxScaler() and StandardScaler() to the following features: calories, protien, fat, sodium, fiber, carbo, sugars.

In [45]:
```
# Import MinMaxScaler and StandardScaler from sklearn.preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

- StandardScaler follows Standard Normal Distribution. Therefore, it makes mean = 0 and scales the data to unit variance.
- MinMaxScaler scales all the data features in the range [0, 1] or else in the range [-1, 1] if there are negative values in the dataset.

In [46]:
```
trans = MinMaxScaler()
# Creating a dataframe of these features: 'calories', 'protein','fat','sodiu
# and transforming all value of these all features using MinMaxScaler
data[['calories', 'protein','fat','sodium','fiber','carbo','sugars']] = tran
```

In [47]:
```
1 data
```

Out[47]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass
0	100% Bran	Other Brands	Type 1	0.181818	0.6	0.2	0.406250	0.714286	0.250000	0.4375	280
1	100% Natural Bran	Other Brands	Type 1	0.636364	0.4	1.0	0.046875	0.142857	0.375000	0.5625	135
2	All-Bran	Kellogg's	Type 1	0.181818	0.6	0.2	0.812500	0.642857	0.333333	0.3750	320
3	All-Bran with Extra Fiber	Kellogg's	Type 1	0.000000	0.6	0.0	0.437500	1.000000	0.375000	0.0625	330
4	Almond Delight	Other Brands	Type 1	0.545455	0.2	0.4	0.625000	0.071429	0.625000	0.5625	-1
...	...	...	...	...	...	...	...	...	...	...	...
72	Triples	Nestle	Type 1	0.545455	0.2	0.2	0.781250	0.000000	0.916667	0.2500	60
73	Trix	Nestle	Type 1	0.545455	0.0	0.2	0.437500	0.000000	0.583333	0.8125	25
74	Wheat Chex	Other Brands	Type 1	0.454545	0.4	0.2	0.718750	0.214286	0.750000	0.2500	115
75	Wheaties	Nestle	Type 1	0.454545	0.4	0.2	0.625000	0.214286	0.750000	0.2500	110
76	Wheaties Honey Gold	Nestle	Type 1	0.545455	0.2	0.2	0.625000	0.071429	0.708333	0.5625	60

77 rows × 16 columns

In [48]:
```
1 trans1 = StandardScaler()
2 # Creating a dataframe of these features: 'calories', 'protein','fat','sodiu
3 # and transforming all value of these all features using StandardScaler
4 data[['calories', 'protein','fat','sodium','fiber','carbo','sugars']]= trans
```

In [49]:
```
1 data
```

Out[49]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	s
0	100% Bran	Other Brands	Type 1	-1.905397	1.337319	-0.012988	-0.356306	3.314439	-2.257639	-0.2
1	100% Natural Bran	Other Brands	Type 1	0.677623	0.417912	3.987349	-1.737087	-0.064172	-1.551936	0.2
2	All-Bran	Kellogg's	Type 1	-1.905397	1.337319	-0.012988	1.204578	2.892113	-1.787170	-0.4
3	All-Bran with Extra Fiber	Kellogg's	Type 1	-2.938605	1.337319	-1.013072	-0.236238	5.003745	-1.551936	-1.5
4	Almond Delight	Other Brands	Type 1	0.161019	-0.501495	0.987096	0.484170	-0.486498	-0.140530	0.2
...	...	...	...	...	...	...	...	...	...	...
72	Triples	Nestle	Type 1	0.161019	-0.501495	-0.012988	1.084510	-0.908824	1.506111	-0.8
73	Trix	Nestle	Type 1	0.161019	-1.420902	-0.012988	-0.236238	-0.908824	-0.375764	1.1
74	Wheat Chex	Other Brands	Type 1	-0.355585	0.417912	-0.012988	0.844374	0.358155	0.565173	-0.8
75	Wheaties	Nestle	Type 1	-0.355585	0.417912	-0.012988	0.484170	0.358155	0.565173	-0.8
76	Wheaties Honey Gold	Nestle	Type 1	0.161019	-0.501495	-0.012988	0.484170	-0.486498	0.329939	0.2

77 rows × 16 columns

9. Does the standard or min-max scaling make a difference in value distribution? Support your answers with some visualisations on the above dataset.

- Yes, the standard or min-max scaling make a difference in value distribution

In [50]:
```
1 trans = MinMaxScaler()
2 data[['calories', 'protein','fat','sodium','fiber','carbo','sugars']] = tran
```

In [51]:
```
1 data
```

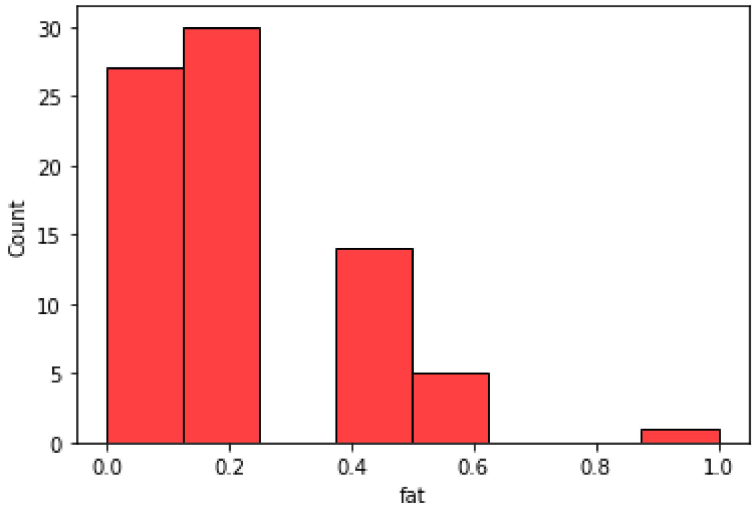
Out[51]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass
0	100% Bran	Other Brands	Type 1	0.181818	0.6	0.2	0.406250	0.714286	0.250000	0.4375	280
1	100% Natural Bran	Other Brands	Type 1	0.636364	0.4	1.0	0.046875	0.142857	0.375000	0.5625	135
2	All-Bran	Kellogg's	Type 1	0.181818	0.6	0.2	0.812500	0.642857	0.333333	0.3750	320
3	All-Bran with Extra Fiber	Kellogg's	Type 1	0.000000	0.6	0.0	0.437500	1.000000	0.375000	0.0625	330
4	Almond Delight	Other Brands	Type 1	0.545455	0.2	0.4	0.625000	0.071429	0.625000	0.5625	-1
...	...	...	...	...	...	...	...	...	...	...	...
72	Triples	Nestle	Type 1	0.545455	0.2	0.2	0.781250	0.000000	0.916667	0.2500	60
73	Trix	Nestle	Type 1	0.545455	0.0	0.2	0.437500	0.000000	0.583333	0.8125	25
74	Wheat Chex	Other Brands	Type 1	0.454545	0.4	0.2	0.718750	0.214286	0.750000	0.2500	115
75	Wheaties	Nestle	Type 1	0.454545	0.4	0.2	0.625000	0.214286	0.750000	0.2500	110
76	Wheaties Honey Gold	Nestle	Type 1	0.545455	0.2	0.2	0.625000	0.071429	0.708333	0.5625	60

77 rows × 16 columns

In [52]:
```
# Histogram of fat features using seaborn library
sns.histplot(x='fat',data=data,color='red')
```

Out[52]: <AxesSubplot:xlabel='fat', ylabel='Count'>



In [53]:
```
trans1 = StandardScaler()
data[['calories', 'protein','fat','sodium','fiber','carbo','sugars']]= trans
```

In [54]:
```
1 data
```

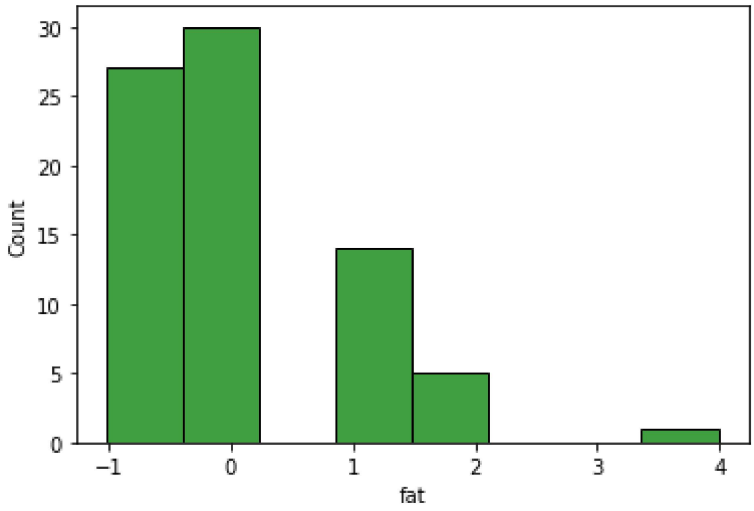
Out[54]:

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	s
0	100% Bran	Other Brands	Type 1	-1.905397	1.337319	-0.012988	-0.356306	3.314439	-2.257639	-0.2
1	100% Natural Bran	Other Brands	Type 1	0.677623	0.417912	3.987349	-1.737087	-0.064172	-1.551936	0.2
2	All-Bran	Kellogg's	Type 1	-1.905397	1.337319	-0.012988	1.204578	2.892113	-1.787170	-0.4
3	All-Bran with Extra Fiber	Kellogg's	Type 1	-2.938605	1.337319	-1.013072	-0.236238	5.003745	-1.551936	-1.5
4	Almond Delight	Other Brands	Type 1	0.161019	-0.501495	0.987096	0.484170	-0.486498	-0.140530	0.2
...	...	...	...	...	...	...	...	...	...	
72	Triples	Nestle	Type 1	0.161019	-0.501495	-0.012988	1.084510	-0.908824	1.506111	-0.8
73	Trix	Nestle	Type 1	0.161019	-1.420902	-0.012988	-0.236238	-0.908824	-0.375764	1.1
74	Wheat Chex	Other Brands	Type 1	-0.355585	0.417912	-0.012988	0.844374	0.358155	0.565173	-0.8
75	Wheaties	Nestle	Type 1	-0.355585	0.417912	-0.012988	0.484170	0.358155	0.565173	-0.8
76	Wheaties Honey Gold	Nestle	Type 1	0.161019	-0.501495	-0.012988	0.484170	-0.486498	0.329939	0.2

77 rows × 16 columns

In [55]:
```
# Histogram of fat features using seaborn library
sns.histplot(x='fat',data=data,color='green')
```

Out[55]: <AxesSubplot:xlabel='fat', ylabel='Count'>



- We can use MinMaxScaler for directly normalizing the input variables
- We can use StandardScaler for directly standardizing the input variables

10. As an extension of 7th step, Generate a new Pandas DataFrame with the following columns based on the Training Dataset: Split Ratio | Random State | Total Number of Entries | Count of Kellogg's | Count of Nestle | Count of Other Brands

In [56]:
```
# Create a dataframe of the following columns based on the Training Dataset:
# Split Ratio | Random State | Total Number of Entries | Count of Kellogg's
# using pandas library
data1 = pd.DataFrame(columns = ["Split Ratio", "Random state", "Total Number
```

60:40

In [57]:
```
# Spliting the main dataset into training and testing dataset with 60:40 rat
train_data3, test_data3 = train_test_split(data, test_size = 0.4, random_sta
print("Train data set count",len(train_data3))
print("Test data set count",len(test_data3))
```

```
Train data set count 46
Test data set count 31
```

In [58]:
```
# Count total number of values of 'mfr' column using count() function
train_data3['mfr'].count()
```

Out[58]:
```
46
```

In [59]:
```
# Count 'mfr' columns all values how many times repeated using value_counts(
train_data3['mfr'].value_counts()
```

Out[59]:
```
Other Brands    18
Kellogg's       15
Nestle          13
Name: mfr, dtype: int64
```

In [60]:
```
# Creating a dictionary of these features of the training dataset as keys an
d1 = {"Split Ratio": "60:40", "Random state": 42, "Total Number of Entries":
# Add all these and create a new dataframe using append() function
data1 = data1.append(d1, ignore_index=True)
```

In [61]:
```
data1
```

Out[61]:

	Split Ratio	Random state	Total Number of Entries	Count of Kellogg's	Count of Nestle	Count of Other Brands
0	60:40	42	46	15	13	18

70:30

In [62]:
```
# Spliting the main dataset into training and testing dataset with 70:30 rat
train_data4, test_data4 = train_test_split(data, test_size = 0.3, random_sta
print("Train data set count",len(train_data4))
print("Test data set count",len(test_data4))
```

```
Train data set count 53
Test data set count 24
```

In [63]:
```
# Count total number of values of 'mfr' column using count() function
train_data4['mfr'].count()
```

Out[63]:
```
53
```

In [64]:
```
# Count 'mfr' columns all values how many times repeated using value_counts(
train_data4['mfr'].value_counts()
```

Out[64]:
```
Other Brands    21
Kellogg's       16
Nestle          16
Name: mfr, dtype: int64
```

In [65]:
```
# Creating a dictionary of these features of the training dataset as keys an
d2 = {"Split Ratio": "70:30", "Random state": 42, "Total Number of Entries":
# Add all these and previous dataframe using append() function
data1 = data1.append(d2, ignore_index=True)
```

In [66]:
```
data1
```

Out[66]:

	Split Ratio	Random state	Total Number of Entries	Count of Kellogg's	Count of Nestle	Count of Other Brands
0	60:40	42	46	15	13	18
1	70:30	42	53	16	16	21

80:20

In [67]:
```
# Spliting the main dataset into training and testing dataset with 80:20 rat
train_data5, test_data5 = train_test_split(data, test_size = 0.2, random_sta
print("Train data set count",len(train_data5))
print("Test data set count",len(test_data5))
```

```
Train data set count 61
Test data set count 16
```

In [68]:
```
# Count total number of values of 'mfr' column using count() function
train_data5['mfr'].count()
```

Out[68]:
```
61
```

In [69]:
```
# Count 'mfr' columns all values how many times repeated using value_counts(
train_data5['mfr'].value_counts()
```

Out[69]:
```
Other Brands    24
Nestle          19
Kellogg's       18
Name: mfr, dtype: int64
```

In [70]:
```
# Creating a dictionary of these features of the training dataset as keys an
d3 = {"Split Ratio": "80:20", "Random state": 42, "Total Number of Entries":
# Add all these and previous two dataframe using append() function
data1 = data1.append(d3, ignore_index=True)
```

In [71]:
```
1 data1
```

Out[71]:

	Split Ratio	Random state	Total Number of Entries	Count of Kellogg's	Count of Nestle	Count of Other Brands
0	60:40	42	46	15	13	18
1	70:30	42	53	16	16	21
2	80:20	42	61	18	19	24

Conclusion

- The libraries discussed here widely, one can do ML using python.
- And applying these libraries in a dataset discussed here widely.
- We can notice that using replace function we can easily change the values of column.
- Using sns.countplot() we can visualize the count of how many times repeated of all values of a particular column.
- Describe() function tells us columns count, mean, sd, min, max etc.
- Using seaborn boxplot we can visualize that a box plot is a way to show the spread and centers of a data set.
- Using seaborn histplot we can visualize that histogram represent the data distribution by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin.
- Using pd.dataframe we can create a new dataframe of some features.
- Using train_test_split we can split the main dataset into training and testing dataset.
- We can transform all values of some columns using MinMaxScaler which scales all the data features in the range [0, 1] and StandardScaler which makes mean = 0 and scales the data to unit variance.
- Using seaborn histplot we can visualize that we use MinMaxScaler for directly normalizing the input variables and use StandardScaler for directly standardizing the input variables.
- Using count() function we count total number of values of particular column.
- Using value_counts() function we count column's all values how many times repeated.
- Using append() function we can add dataframe.
- Seaborn is a siimple graphical tool.
- Seaborn can be used for both exploration and presentation of findings.
- Basically here we learned about some machine learning libraries and we praticed on train_test split and normalisation.

References

- https://www.kite.com/python/answers/how-to-replace-column-values-in-a-pandas-dataframe-in-python (https://www.kite.com/python/answers/how-to-replace-column-values-in-a-pandas-dataframe-in-python)
- https://www.geeksforgeeks.org/seaborn-barplot-method-in-python/ (https://www.geeksforgeeks.org/seaborn-barplot-method-in-python/)

- https://www.javatpoint.com/pandas-dataframe-describe (https://www.javatpoint.com/pandas-dataframe-describe), https://www.geeksforgeeks.org/box-plot-in-python-using-matplotlib/,https://www.geeksforgeeks.org/box-plot-visualization-with-pandas-and-seaborn/ (https://www.geeksforgeeks.org/box-plot-in-python-using-matplotlib/,https://www.geeksforgeeks.org/box-plot-visualization-with-pandas-and-seaborn/)
- https://www.geeksforgeeks.org/matplotlib-pyplot-hist-in-python (https://www.geeksforgeeks.org/matplotlib-pyplot-hist-in-python), https://www.geeksforgeeks.org/how-to-make-histograms-with-density-plots-with-seaborn-histplot/ (https://www.geeksforgeeks.org/how-to-make-histograms-with-density-plots-with-seaborn-histplot/)
- https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/ (https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/)
- https://www.geeksforgeeks.org/count-values-in-pandas-dataframe/ (https://www.geeksforgeeks.org/count-values-in-pandas-dataframe/)
- https://www.machinelearningplus.com/pandas/how-to-create-pandas-dataframe-python/ (https://www.machinelearningplus.com/pandas/how-to-create-pandas-dataframe-python/)
- https://www.kite.com/python/answers/how-to-clear-a-pandas-dataframe-in-python (https://www.kite.com/python/answers/how-to-clear-a-pandas-dataframe-in-python)
- https://stackoverflow.com/questions/25539195/multiple-histograms-in-pandas (https://stackoverflow.com/questions/25539195/multiple-histograms-in-pandas)