Task cereal rating prediction given data about different breakfast cereals , let's try to predict the rating of given cereal

We will use a linear regression model to make our predictions

Import

In [59]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,Ridge,Lasso
```

In [60]:
```
data=pd.read_csv('cereal.csv')
data.head(20)
```

Out[60]:

	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	1.00	0.33	68.402973
1	100% Natural Bran	Q	C	120	3	5	15	2.0	8.0	8	135	0	3	1.00	1.00	33.983679
2	All-Bran	K	C	70	4	1	260	9.0	7.0	5	320	25	3	1.00	0.33	59.425505
3	All-Bran with Extra Fiber	K	C	50	4	0	140	14.0	8.0	0	330	25	3	1.00	0.50	93.704912
4	Almond Delight	R	C	110	2	2	200	1.0	14.0	8	-1	25	3	1.00	0.75	34.384843
5	Apple Cinnamon Cheerios	G	C	110	2	2	180	1.5	10.5	10	70	25	1	1.00	0.75	29.509541
6	Apple Jacks	K	C	110	2	0	125	1.0	11.0	14	30	25	2	1.00	1.00	33.174094
7	Basic 4	G	C	130	3	2	210	2.0	18.0	8	100	25	3	1.33	0.75	37.038562
8	Bran Chex	R	C	90	2	1	200	4.0	15.0	6	125	25	1	1.00	0.67	49.120253
9	Bran Flakes	P	C	90	3	0	210	5.0	13.0	5	190	25	3	1.00	0.67	53.313813
10	Cap'n'Crunch	Q	C	120	1	2	220	0.0	12.0	12	35	25	2	1.00	0.75	18.042851
11	Cheerios	G	C	110	6	2	290	2.0	17.0	1	105	25	1	1.00	1.25	50.764999
12	Cinnamon Toast Crunch	G	C	120	1	3	210	0.0	13.0	9	45	25	2	1.00	0.75	19.823573
13	Clusters	G	C	110	3	2	140	2.0	13.0	7	105	25	3	1.00	0.50	40.400208
14	Cocoa Puffs	G	C	110	1	1	180	0.0	12.0	13	55	25	2	1.00	1.00	22.736446
15	Corn Chex	R	C	110	2	0	280	0.0	22.0	3	25	25	1	1.00	1.00	41.445019
16	Corn Flakes	K	C	100	2	0	290	1.0	21.0	2	35	25	1	1.00	1.00	45.863324
17	Corn Pops	K	C	110	1	0	90	1.0	13.0	12	20	25	2	1.00	1.00	35.782791
18	Count Chocula	G	C	110	1	1	180	0.0	12.0	13	65	25	2	1.00	1.00	22.396513
19	Cracklin' Oat Bran	K	C	110	3	3	140	4.0	10.0	7	160	25	3	1.00	0.50	40.448772

In [61]:
```
data.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
```

Preprocessing

In [62]:
```
data= data.drop('name',axis=1)
```

MISSING VALUES

In [63]:
```
(data == -1).sum()
```

Out[63]:
```
mfr         0
type        0
calories    0
protein     0
fat         0
sodium      0
fiber       0
carbo       1
sugars      1
potass      2
vitamins    0
shelf       0
weight      0
cups        0
rating      0
dtype: int64
```

In [64]:
```
data=data.replace(-1,np.NaN)
```

In [65]:
```
data.isna().sum()
```

Out[65]:
```
mfr         0
type        0
calories    0
protein     0
fat         0
sodium      0
fiber       0
carbo       1
sugars      1
potass      2
vitamins    0
shelf       0
weight      0
cups        0
rating      0
dtype: int64
```

In [66]:
```
for column in ['carbo','sugars','potass']:
    data[column]=data[column].fillna(data[column].mean())
```

In [67]:
```
data.isna().sum()
```

Out[67]:
```
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
```

Encoding

In [68]:
```
{column : list(data[column].unique()) for column in ['mfr','type']}
```

Out[68]:
```
{'mfr': ['N', 'Q', 'K', 'R', 'G', 'P', 'A'], 'type': ['C', 'H']}
```

In [69]:
```
data['type']=data['type'].apply(lambda x: 1 if x=='H' else 0)
```

In [70]:
```
#One-Hot Encoder the 'mfr' column

dummies = pd.get_dummies(data['mfr'])
dummies
```

Out[70]:

	A	G	K	N	P	Q	R
0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0
2	0	0	1	0	0	0	0
3	0	0	1	0	0	0	0
4	0	0	0	0	0	0	1
...	...	...	...	...	...	...	...
72	0	1	0	0	0	0	0
73	0	1	0	0	0	0	0
74	0	0	0	0	0	0	1
75	0	1	0	0	0	0	0
76	0	1	0	0	0	0	0

77 rows × 7 columns

In [71]:
```
#concat
data=pd.concat([data,dummies],axis=1)
data=data.drop('mfr',axis=1)
```

In [72]:
```
data
```

Out[72]:

	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	...	weight	cups	rating	A	G	K	N	P	Q	R
0	0	70	4	1	130	10.0	5.0	6.0	280.000000	25	...	1.0	0.33	68.402973	0	0	0	1	0	0	0
1	0	120	3	5	15	2.0	8.0	8.0	135.000000	0	...	1.0	1.00	33.983679	0	0	0	0	0	1	0
2	0	70	4	1	260	9.0	7.0	5.0	320.000000	25	...	1.0	0.33	59.425505	0	0	1	0	0	0	0
3	0	50	4	0	140	14.0	8.0	0.0	330.000000	25	...	1.0	0.50	93.704912	0	0	1	0	0	0	0
4	0	110	2	2	200	1.0	14.0	8.0	98.666667	25	...	1.0	0.75	34.384843	0	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
72	0	110	2	1	250	0.0	21.0	3.0	60.000000	25	...	1.0	0.75	39.106174	0	1	0	0	0	0	0
73	0	110	1	1	140	0.0	13.0	12.0	25.000000	25	...	1.0	1.00	27.753301	0	1	0	0	0	0	0
74	0	100	3	1	230	3.0	17.0	3.0	115.000000	25	...	1.0	0.67	49.787445	0	0	0	0	0	0	1
75	0	100	3	1	200	3.0	17.0	3.0	110.000000	25	...	1.0	1.00	51.592193	0	1	0	0	0	0	0
76	0	110	2	1	200	1.0	16.0	8.0	60.000000	25	...	1.0	0.75	36.187559	0	1	0	0	0	0	0

77 rows × 21 columns

Splitting and Scaling

In [73]:
```
y=data.loc[:,'rating']
X=data.drop('rating',axis=1)
```

In [74]:
```
X
```

Out[74]:

	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	A	G	K	N	P	Q	R
0	0	70	4	1	130	10.0	5.0	6.0	280.000000	25	3	1.0	0.33	0	0	0	1	0	0	0
1	0	120	3	5	15	2.0	8.0	8.0	135.000000	0	3	1.0	1.00	0	0	0	0	0	1	0
2	0	70	4	1	260	9.0	7.0	5.0	320.000000	25	3	1.0	0.33	0	0	1	0	0	0	0
3	0	50	4	0	140	14.0	8.0	0.0	330.000000	25	3	1.0	0.50	0	0	1	0	0	0	0
4	0	110	2	2	200	1.0	14.0	8.0	98.666667	25	3	1.0	0.75	0	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
72	0	110	2	1	250	0.0	21.0	3.0	60.000000	25	3	1.0	0.75	0	1	0	0	0	0	0
73	0	110	1	1	140	0.0	13.0	12.0	25.000000	25	2	1.0	1.00	0	1	0	0	0	0	0
74	0	100	3	1	230	3.0	17.0	3.0	115.000000	25	1	1.0	0.67	0	0	0	0	0	0	1
75	0	100	3	1	200	3.0	17.0	3.0	110.000000	25	1	1.0	1.00	0	1	0	0	0	0	0
76	0	110	2	1	200	1.0	16.0	8.0	60.000000	25	1	1.0	0.75	0	1	0	0	0	0	0

77 rows × 20 columns

In [75]:
```
scaler=StandardScaler()
X=pd.DataFrame(scaler.fit_transform(X),columns=X.columns)
```

In [76]:
```
X
```

Out[76]:

	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	A
0	-0.201347	-1.905397	1.337319	-0.012988	-0.356306	3.314439	-2.542013	-0.237495	2.627053	-0.14627	0.957813	-0.198067	-2.123870	-0.114708
1	-0.201347	0.677623	0.417912	3.987349	-1.737087	-0.064172	-1.764055	0.225316	0.526376	-1.27255	0.957813	-0.198067	0.774053	-0.114708
2	-0.201347	-1.905397	1.337319	-0.012988	1.204578	2.892113	-2.023374	-0.468901	3.206550	-0.14627	0.957813	-0.198067	-2.123870	-0.114708
3	-0.201347	-2.938605	1.337319	-1.013072	-0.236238	5.003745	-1.764055	-1.625929	3.351425	-0.14627	0.957813	-0.198067	-1.388576	-0.114708
4	-0.201347	0.161019	-0.501495	0.987096	0.484170	-0.486498	-0.208138	0.225316	0.000000	-0.14627	0.957813	-0.198067	-0.307262	-0.114708
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
72	-0.201347	0.161019	-0.501495	-0.012988	1.084510	-0.908824	1.607098	-0.931712	-0.560180	-0.14627	0.957813	-0.198067	-0.307262	-0.114708
73	-0.201347	0.161019	-1.420902	-0.012988	-0.236238	-0.908824	-0.467457	1.150938	-1.067240	-0.14627	-0.251230	-0.198067	0.774053	-0.114708
74	-0.201347	-0.355585	0.417912	-0.012988	0.844374	0.358155	0.569820	-0.931712	0.236628	-0.14627	-1.460273	-0.198067	-0.653283	-0.114708
75	-0.201347	-0.355585	0.417912	-0.012988	0.484170	0.358155	0.569820	-0.931712	0.164191	-0.14627	-1.460273	-0.198067	0.774053	-0.114708
76	-0.201347	0.161019	-0.501495	-0.012988	0.484170	-0.486498	0.310501	0.225316	-0.560180	-0.14627	-1.460273	-0.198067	-0.307262	-0.114708

77 rows × 20 columns

In [78]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,train_size=0.70,random_state=42)
```

Training

In [79]:
```
model=LinearRegression()
l2_model=Ridge(alpha=1.0)
l1_model=Lasso(alpha=1.0)
```

In [82]:
```
model.fit(X_train,y_train)
l2_model.fit(X_train,y_train)
l1_model.fit(X_train,y_train)

print("Models Trained.")
```

```
Models Trained.
```

In [83]:
```
model_r2=model.score(X_test,y_test)
l2_model_r2=l2_model.score(X_test,y_test)
l1_model_r2=l1_model.score(X_test,y_test)
```

In [85]:
```
print('R^2 Score\n' + "*" * 10)
print("    Without Regularization :",model_r2)
print("with L2 (Ridge) Regularization", l2_model_r2)
print("with L1 (lasso) Regularization", l1_model_r2)
```

```
R^2 Score
**********
    Without Regularization : 0.9914187913729915
with L2 (Ridge) Regularization 0.9937426752780412
with L1 (lasso) Regularization 0.9539168386702219
```

In [ ]: