Initialize

In [1]:
```
# Libraries
import pandas as pd
```

In [2]:
```
# Load data
df = pd.read_csv("cereal.csv")
```

EDA

We inspected the dataset and learned that it consists of 77 total cereal entries. There were 3 categorical vriables and thirteen numerical variables. Immediately, we knew we would need to either drop the categorical variables or convert them to numerical format in order to use them in our K-Means algorithm. We also spotted four data entries that contained '-1' values.

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77 entries, 0 to 76
Data columns (total 16 columns):
name        77 non-null object
mfr         77 non-null object
type        77 non-null object
calories    77 non-null int64
protein     77 non-null int64
fat         77 non-null int64
sodium      77 non-null int64
fiber       77 non-null float64
carbo       77 non-null float64
sugars      77 non-null int64
potass      77 non-null int64
vitamins    77 non-null int64
shelf       77 non-null int64
weight      77 non-null float64
cups        77 non-null float64
rating      77 non-null float64
dtypes: float64(5), int64(8), object(3)
memory usage: 9.7+ KB
```

In [4]:
```
df.head()
```

Out[4]:

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

Data cleaning

After the initial inspection, we looked for areas that we could clean up in order to make our analysis clearer, and easier to carry out. This included replacing missing values, renaming features and row entries, and dealing with the '-1' values.

(1) Check for missing values

In [5]:
```
# 1 - check for missing values
df.isnull().sum()
```

Out[5]:
```
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
```

(2) Rename 'mfr' column

Most of the column names were easily understood, but 'mfr' was not immediately apparent in it's representation so we decided to rename it for clarity.

In [6]:
```
colNames = list(df)

for col in colNames:
    if col == "mfr":
        df.rename({'mfr': 'manufacturer'}, axis = 'columns', inplace=True)

df.columns.values
```

Out[6]:
```
array(['name', 'manufacturer', 'type', 'calories', 'protein', 'fat',
       'sodium', 'fiber', 'carbo', 'sugars', 'potass', 'vitamins',
       'shelf', 'weight', 'cups', 'rating'], dtype=object)
```

(3) Replace letters with brand names

The manufacturer names were listed as single letters, which could cause confusion. It's also not future-proof since adding new manufacturers with the same first letter would break the data. We decided to replace the single letters with the full brand names instead.

In [7]:
```
# print all unique values in the 'manufacturer' columns
df.manufacturer.unique()
```

Out[7]:
```
array(['N', 'Q', 'K', 'R', 'G', 'P', 'A'], dtype=object)
```

In [8]:
```
# Replace 'manufacturer' letters with brand names
df['manufacturer'] = df['manufacturer'].map({'N': 'Nabisco',
                                             'Q': 'Quaker',
                                             'K': 'Kellogs',
                                             'R': 'Ralston Purina',
                                             'G': 'General Mills',
                                             'P': 'Post',
                                             'A': 'American Home Food Products'})

# show new brand list
df.manufacturer.unique()
```

Out[8]:
```
array(['Nabisco', 'Quaker', 'Kellogs', 'Ralston Purina', 'General Mills',
       'Post', 'American Home Food Products'], dtype=object)
```

(4) Replace values in 'type' column

The 'type' column contained single letter entries which was not immediately apparent. Referring to the codebook, we learned this column meant "hot" or "cold" cereal. We decided to translate the values to their literal meaning.

In [9]:
```
# print all unique values in the 'type' columns
df.type.unique()
```

Out[9]:
```
array(['C', 'H'], dtype=object)
```

In [10]:
```
# Replace 'type' letters with words
df['type'] = df['type'].map({'H': 'Hot',
                             'C': 'Cold'})

# show new brand list
df.type.unique()
```

Out[10]:
```
array(['Cold', 'Hot'], dtype=object)
```

(5) Handle *impossible* '-1' values

We saw that four rows in the data contained '-1' values for the potass feature. We knew a product couldn't have '-1' grams of potassium. This stood out to us as something symbolic of another meaning and needed to be addressed prior to continuing the analysis or else it may cause skew or bias. After some research, we've concluded that a value of '-1' typically refers to infinity after a number has been divided by zero. This usually indicates that the value was supposed to be 'null' in the case that it is so small it is negligent, or it simply wasn't recorded. In this case, we decided to fill those values with '0'. Originally we had tried to average out the values with the rest of the data so that the average value of that feature wouldn't change, but we quickly realized that this would cause bias in our data if potassium ended up being a major indicator of a cereal's rating. Instead, we chose '0' for two reasons: (1) Since the data only contained 77 cereals for us to work with, we didn't want to drop any of them and (2) K-Means requires numerical values in order to cluster the data so we couldn't use 'NA'.

In [11]:
```
# count the number of '-1' values in the data
df.isin([-1]).sum(axis=0)
```

Out[11]:
```
name            0
manufacturer    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 [12]:
```
# Replace all '-1' values in carbo', 'sugars', and 'potass'.
df.replace(-1, 0, inplace=True)
```

In [13]:
```
# Verify there are no '-1' values left.
df.isin([-1]).sum(axis=0)
```

Out[13]:
```
name           0
manufacturer   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
```

When reviewing the codebook, we noticed that both the sodium and potassium features were listed in milligrams, while the other features were all listed in grams. In order to use similar units, and avoid biasing our data, we converted milligrams to grams for both features.

In [14]:
```
df["sodium"] = df["sodium"]/1000
df["potass"] = df["potass"]/1000
```

In [15]:
```
# Review of cleaned data
df.describe()
```

Out[15]:

	calories	protein	fat	sodium	fiber	carbo	sugars	potass
count	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000
mean	106.883117	2.545455	1.012987	0.159675	2.151948	14.610390	6.935065	0.096104
std	19.484119	1.094790	1.006473	0.083832	2.383364	4.232257	4.422840	0.071251
min	50.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	100.000000	2.000000	0.000000	0.130000	1.000000	12.000000	3.000000	0.040000
50%	110.000000	3.000000	1.000000	0.180000	2.000000	14.000000	7.000000	0.090000
75%	110.000000	3.000000	2.000000	0.210000	3.000000	17.000000	11.000000	0.120000
max	160.000000	6.000000	5.000000	0.320000	14.000000	23.000000	15.000000	0.330000

In [16]:
```
df.head()
```

Out[16]:

	name	manufacturer	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitami
0	100% Bran	Nabisco	Cold	70	4	1	0.130	10.0	5.0	6	0.280	
1	100% Natural Bran	Quaker	Cold	120	3	5	0.015	2.0	8.0	8	0.135	
2	All-Bran	Kellogs	Cold	70	4	1	0.260	9.0	7.0	5	0.320	
3	All-Bran with Extra Fiber	Kellogs	Cold	50	4	0	0.140	14.0	8.0	0	0.330	
4	Almond Delight	Ralston Purina	Cold	110	2	2	0.200	1.0	14.0	8	0.000	

Export cleaned data file

In [17]:
```
df.to_csv("cereal_cleaned.csv")
```