# ML_Scaling_Practice

March 26, 2025

[ ]:

```
# Importing the libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder
```

[ ]:

```
from google.colab import auth
auth.authenticate_user()

import gspread
from google.auth import default
creds, _ = default()
gc = gspread.authorize(creds)

worksheet = gc.open('cereal-kaggle').sheet1

# get_all_values gives a list of rows.
rows = worksheet.get_all_values()
# print(rows)

# Convert to a DataFrame and render.
import pandas as pd
df = pd.DataFrame.from_records(rows)
```

[ ]:

```
# setting first row as headers
# resetting index
# didplaying first 5 rows
df.columns = df.iloc[0]
df = df.iloc[1:].reset_index(drop=True)
df.head()
```

[ ]:

```
0                       name mfr type calories protein fat sodium fiber carbo  \
0                  100% Bran   N    C                4   1    130    10     5
1          100% Natural Bran   Q    C      120       3   5     15     2     8
2                   All-Bran   K    C       70       4   1    260     9     7
```

```
3  All-Bran with Extra Fiber   K    C       50       4   0    140    14     8
4             Almond Delight   R    C                2   2    200     1    14

0 sugars potass vitamins shelf weight  cups     rating
0      6    280       25   top      1  0.33  68.402973
1      8    135        0   top      1     1  33.983679
2      5    320       25   top      1  0.33  59.425505
3      0    330       25   top      1   0.5  93.704912
4      8     -1       25            1  0.75  34.384843
```

### 0.0.1 Scale the numeric features

[ ]:
```python
# mfr, type, calories, protein, fat, fiber, sugars, shelf
df_list = ['mfr', 'type', 'calories', 'protein', 'fat', 'fiber', 'sugars', 'shelf']
df_new = df[df_list]
df_new.head(2)
```

[ ]:
```
0 mfr type calories protein fat fiber sugars shelf
0   N    C                4   1    10      6   top
1   Q    C      120       3   5     2      8   top
```

[ ]:
```python
# Instantiate a StandardScaler for the numeric features.
scaler = StandardScaler()
```

[ ]:
```python
# Fit the scaler on X_train_num_imputed
x_num_imputed_list = ['calories', 'protein', 'fat', 'fiber', 'sugars']
x_num_imputed = df_new[x_num_imputed_list]
```

[ ]:
```python
# Checking the data types
x_num_imputed.dtypes

# Replacing the empty tsrings with np.nan
x_num_imputed = x_num_imputed.replace('', np.nan)

# Changing to float data type
x_num_imputed = x_num_imputed.astype(float)
```

[ ]:
```python
# Spliting into x_train_num_umputed and x_test_num_imputed
X_train_num_imputed, X_test_num_imputed = train_test_split(x_num_imputed, test_size=0.2, random_state=42)
```

[ ]:
```python
# Fit the scaler on X_train_num_imputed
scaler.fit(X_train_num_imputed)
```

[ ]:
```
StandardScaler()
```

```
[ ]: # save the output as "X_train_num_scaled"
     X_train_num_scaled = scaler.transform(X_train_num_imputed)
```

```
[ ]: # Transform the numeric test data (X_test_num_imputed) and save the transformed␣
      ↪data as "X_test_num_scaled"
     X_test_num_scaled = scaler.transform(X_test_num_imputed)
```

```
[ ]:  # Preview the first 5 rows of X_test_num_scaled.
      X_test_num_scaled[:5]
```

```
[ ]: array([[        nan, -0.53311399,  0.95545914, -0.43881613,  0.33525006],
            [        nan, -1.43644603,  0.95545914, -0.43881613,  1.00983861],
            [        nan, -1.43644603,  0.95545914, -0.85709841,  1.23470145],
            [        nan,  1.27355009, -0.01802753,  3.32572435, -0.11447563],
            [        nan,  1.27355009,  1.92894582,  0.39774842,  1.00983861]])
```

```
[ ]: # Describe() of the X_train_num_imputed
     X_train_num_imputed.describe()
```

```
[ ]: 0       calories  protein   fat  fiber  sugars
     count      61.00    61.00 54.00  55.00   55.00
     mean      106.39     2.59  1.02   2.05    6.51
     std        20.00     1.12  1.04   2.41    4.49
     min        50.00     1.00  0.00   0.00   -1.00
     25%       100.00     2.00  0.00   0.00    3.00
     50%       110.00     3.00  1.00   1.50    6.00
     75%       110.00     3.00  1.75   3.00   10.00
     max       160.00     6.00  5.00  14.00   15.00
```

```
[ ]: # Set pandas display option to avoid scientific notation
     pd.set_option('display.float_format', '{:.2f}'.format)

     # converting X_train_num_scaled into a df
     col = X_train_num_imputed.columns.tolist()
     X_train_num_scaled_df = pd.DataFrame(X_train_num_scaled, columns=col)
     X_train_num_scaled_df.describe()
```

```
[ ]:         calories  protein   fat  fiber  sugars
     count      61.00    61.00 54.00  55.00   55.00
     mean        0.00    -0.00 -0.00  -0.00    0.00
     std         1.01     1.01  1.01   1.01    1.01
     min        -2.84    -1.44 -0.99  -0.86   -1.69
     25%        -0.32    -0.53 -0.99  -0.86   -0.79
     50%         0.18     0.37 -0.02  -0.23   -0.11
     75%         0.18     0.37  0.71   0.40    0.78
     max         2.70     3.08  3.88   5.00    1.91
```

**Did any of the following statistics change after scaling?:Min, Max, Mean, Std** Yes, all the above mentioned statistics changed.

This is because when Scaling the mean is set to 0 and standard deviation is set to 1.

During scaling the formular: x_scaled = (x - mean)/standard deviation is applied to all the orginal data points (x).

This transformation alters the distribution of the data, which consequently changes the Min and Max values as well.

### 0.0.2 Scale the ordinal features

[ ]:
```
#ordinal_col = x_train['shelf']
#print(ordinal_col)
ordinal_col = ['shelf']
```

[ ]:
```
df_new[ordinal_col].head()
```

[ ]:
```
0 shelf
0   top
1   top
2   top
3   top
4
```

[ ]:
```
# # One Hot Encoding x_ord
x_ord_encoded = pd.get_dummies(df_new[ordinal_col])
x_ord_encoded = x_ord_encoded.astype(int)
x_ord_encoded.head()
```

[ ]:
```
   shelf_  shelf_bottom  shelf_middle  shelf_top
0       0             0             0          1
1       0             0             0          1
2       0             0             0          1
3       0             0             0          1
4       1             0             0          0
```

[ ]:
```
# Spliting x_ord_encoded
x_train_ord_encoded, x_test_ord_encoded = train_test_split(x_ord_encoded,␣
 ↪test_size=0.2, random_state=42)
```

[ ]:
```
# Scaling x_train_ord_encoded
ss = StandardScaler()
x_train_ord_scaled = scaler.fit_transform(x_train_ord_encoded)
```

[ ]:
```
# Transform the ordinal test data (X_test_ord_encoded) and save the transformed␣
 ↪data as "X_test_ord_scaled"
x_test_ord_scaled = scaler.transform(x_test_ord_encoded)
```

[ ]:

```
# Preview the first 5 rows of X_train_ord_encoded.
x_train_ord_encoded.head()
```

[ ]:

```
    shelf_  shelf_bottom  shelf_middle  shelf_top
9        0             0             0          1
5        0             1             0          0
34       0             0             0          1
22       0             0             0          1
30       0             1             0          0
```

[ ]:

```
# .describe() of X_train_ord_encoded
x_train_ord_encoded.describe()
```

[ ]:

```
       shelf_  shelf_bottom  shelf_middle  shelf_top
count   61.00         61.00         61.00      61.00
mean     0.00          0.30          0.23       0.48
std      0.00          0.46          0.42       0.50
min      0.00          0.00          0.00       0.00
25%      0.00          0.00          0.00       0.00
50%      0.00          0.00          0.00       0.00
75%      0.00          1.00          0.00       1.00
max      0.00          1.00          1.00       1.00
```

[ ]:

```
# vs. X_train_ord_scaled
x_train_ord_scaled_df = pd.DataFrame(x_train_ord_scaled,␣
 ↪columns=x_train_ord_encoded.columns)
x_train_ord_scaled_df.describe()
```

[ ]:

```
       shelf_  shelf_bottom  shelf_middle  shelf_top
count   61.00         61.00         61.00      61.00
mean     0.00          0.00         -0.00       0.00
std      0.00          1.01          1.01       1.01
min      0.00         -0.65         -0.55      -0.95
25%      0.00         -0.65         -0.55      -0.95
50%      0.00         -0.65         -0.55      -0.95
75%      0.00          1.55         -0.55       1.05
max      0.00          1.55          1.83       1.05
```

**Answer the question(s) in a Markdown cell:Did any of the following statistics change after scaling?:Min, Max, Mean, Std?** After scaling, the shelf features (which were originally ordinal) are transformed into continuous values with a mean of 0 and a standard deviation of 1, as expected from the StandardScaler. The values are no longer restricted to 0 and 1 but instead spread across a range of negative and positive values.

### Cat features

[ ]:

```
x_cat = df_new[['mfr', 'type']]
x_cat.head()
```

[ ]:

```
0 mfr type
0   N    C
1   Q    C
2   K    C
3   K    C
4   R    C
```

[ ]:

```
x_cat_encoded = pd.get_dummies(x_cat, columns=['mfr', 'type'])
x_cat_encoded = x_cat_encoded.astype(int)
x_cat_encoded.head()
```

[ ]:

```
    mfr_  mfr_A  mfr_G  mfr_K  mfr_N  mfr_P  mfr_Q  mfr_R  type_C  type_H
0     0      0      0      0      1      0      0      0       1       0
1     0      0      0      0      0      0      1      0       1       0
2     0      0      0      1      0      0      0      0       1       0
3     0      0      0      1      0      0      0      0       1       0
4     0      0      0      0      0      0      0      1       1       0
```

[ ]:

```
# Spliting x_cat_encoded
x_train_cat_encoded, x_test_cat_encoded = train_test_split(x_cat_encoded,␣
 ↪test_size=0.2, random_state=42)
```

### Bringing it together

[ ]:

```
# Create the final training data, "X_train_processed" by concatenating:
 ↪X_train_num_scaled X_train_ord_scaled X_train_cat_encoded
X_train_processed = pd.
 ↪concat([X_train_num_scaled_df,x_train_ord_scaled_df,x_train_cat_encoded],axis=1)
X_train_processed.head()
```

[ ]:

```
   calories  protein   fat  fiber  sugars  shelf_  shelf_bottom  shelf_middle  \
0     -0.83     0.37   NaN    NaN     NaN    0.00         -0.65         -0.55
1      0.18    -0.53  0.96  -0.23    0.78    0.00          1.55         -0.55
2      0.69     0.37  1.93   0.40   -0.56    0.00         -0.65         -0.55
3     -0.32    -0.53   NaN  -0.02    0.78    0.00         -0.65         -0.55
4     -0.32    -0.53 -0.99  -0.86    1.91    0.00          1.55         -0.55

   shelf_top  mfr_  mfr_A  mfr_G  mfr_K  mfr_N  mfr_P  mfr_Q  mfr_R  type_C  \
0       1.05   NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN     NaN
1      -0.95  0.00   0.00   0.00   0.00   0.00   0.00   1.00   0.00    1.00
2       1.05  0.00   0.00   0.00   1.00   0.00   0.00   0.00   0.00    1.00
3       1.05  0.00   0.00   0.00   1.00   0.00   0.00   0.00   0.00    1.00
4      -0.95   NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN     NaN
```

```
   type_H
0     NaN
1    0.00
2    0.00
3    0.00
4     NaN
```

[ ]:
```
# creating x_test_num_scaled_df
col = X_train_num_imputed.columns.tolist()
X_test_num_scaled_df = pd.DataFrame(X_test_num_scaled, columns=col)

#creating x_test_ord_scaled_df
x_test_ord_scaled_df = pd.DataFrame(x_test_ord_scaled,␣
 ↪columns=x_train_ord_encoded.columns)


# Create the final test data, "X_test_processed" by concatenating:
 ↪X_test_num_scaled X_test_ord_scaled X_test_cat_encoded
X_test_processed = pd.
 ↪concat([X_test_num_scaled_df,x_test_ord_scaled_df,x_test_cat_encoded],axis=1)
X_test_processed.head()
```

[ ]:
```
   calories  protein   fat  fiber  sugars  shelf_  shelf_bottom  shelf_middle  \
0       NaN    -0.53  0.96  -0.44    0.34    1.00         -0.65         -0.55
1       NaN    -1.44  0.96  -0.44    1.01    1.00         -0.65         -0.55
2       NaN    -1.44  0.96  -0.86    1.23    0.00         -0.65          1.83
3       NaN     1.27 -0.02   3.33   -0.11    0.00         -0.65         -0.55
4       NaN     1.27  1.93   0.40    1.01    0.00         -0.65         -0.55

   shelf_top  mfr_  mfr_A  mfr_G  mfr_K  mfr_N  mfr_P  mfr_Q  mfr_R  type_C  \
0      -0.95  0.00   0.00   0.00   0.00   1.00   0.00   0.00   0.00    1.00
1      -0.95   NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN     NaN
2      -0.95   NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN     NaN
3       1.05   NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN     NaN
4       1.05  0.00   0.00   0.00   0.00   0.00   0.00   0.00   1.00    1.00

   type_H
0    0.00
1     NaN
2     NaN
3     NaN
4    0.00
```

[ ]:
```
X_test_processed.shape
```

[ ]:
```
(28, 19)
```

```
[ ]: X_train_processed.shape
```

```
[ ]: (74, 19)
```

```
[ ]:
```