

Crunching Numbers: Predicting Consumer Ratings for Breakfast Cereals

Louie S. Venegas

DeVry University

CEIS312 Introduction to Artificial Intelligence and Machine Learning

Instructor: Pete Limon

06/19/2024

Crunching Numbers: Predicting Consumer Ratings for Breakfast Cereals

Introduction to the Problem

Given a dataset of cereal attributes (such as calories, sugar content, protein, etc.), our objective is to build a linear regression model that predicts consumer report ratings for cereals. We aim to understand how specific cereal features impact overall consumer satisfaction and provide actionable insights for cereal manufacturers.

Often cereal manufacturers want to produce new flavors for the market. The problem is that they do not always know what factors influence the consumers' opinion of the cereal. Through polls consumer ratings have been collected about various cereals and paired with the data about that cereal that the consumer can see. The fields are drawn from shelf life, the nutritional label and even the manufacturer. Instead of producing cereals at random to see if the consumers like them or not, we are attempting to create a linear regression model to use relationships between the ratings and the other data. We will use this information to pinpoint what it is about the cereal that makes the consumers give it a higher rating.

The following report is a documentation of the process of building the cereal predictive model. Briefly the steps are expounded on, and pictures are provided to illustrate for the reader each one. Each step required special considerations regarding shaping and engineering the data to produce an accurate predictive model. The process often involves backtracking or reiteration to fine tune the outcome which is why it may seem out of order at times.

A screenshot of a Visualization of the Dataset Cereal.csv in the Studio

Cereal Regression - Louie Venegas > cereal.csv > dataset

rows 77 columns 16

view as  

	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potassium	vitamins	shelf	weight
	100% Bran	N	C	70	4	1	0.13	10	5	6	0.28	25	3	1
	100% Natural Bran	Q	C	120	3	5	0.015	2	8	8	0.135	0	3	1
	All-Bran	K	C	70	4	1	0.26	9	7	5	0.32	25	3	1
	All-Bran with Extra Fiber	K	C	50	4	0	0.14	14	8	0	0.33	25	3	1
	Almond Delight	R	C	110	2	2	0.2	1	14	8	0.33	25	3	1
	Apple Cinnamon Cheerios	G	C	110	2	2	0.18	1.5	10.5	10	0.07	25	1	1
	Apple Jacks	K	C	110	2	0	0.125	1	11	14	0.03	25	2	1
	Basic 4	G	C	130	3	2	0.21	2	18	8	0.1	25	3	1.33
	Bran Chex	R	C	90	2	1	0.2	4	15	6	0.125	25	1	1
	Bran Flakes	P	C	90	3	0	0.21	5	13	5	0.19	25	3	1
	Cap'n Crunch	Q	C	120	1	2	0.22	0	12	12	0.035	25	2	1
	Cheerios	G	C	110	6	2	0.29	2	17	1	0.105	25	1	1
	Cinnamon Toast Crunch	G	C	120	1	3	0.21	0	13	9	0.045	25	2	1
	Clusters	G	C	110	3	2	0.14	2	13	7	0.105	25	3	1
	Cocoa Puffs	G	C	110	1	1	0.18	0	12	13	0.055	25	2	1
	Corn Chex	R	C	110	2	0	0.28	0	22	3	0.025	25	1	1

Upon initial examination it was noted the first three columns would be problematic for this type of model since they are letters and words (strings). First, it was thought to remove the first three columns since they were causing the python code to crash. However, the three string columns may have some predictive value regarding the consumer ratings. So, to make them useful to the regression model being built a python code was used to encode them and give each of them a numerical value.

Python Label Encoder Script

Execute Python Script

Python script

```

1 import pandas as pd
2 from sklearn.preprocessing import LabelEncoder
3
4 def azureml_main(frame1=None):
5     # Choosing the columns to encode
6     columns_to_encode = frame1.columns[:3]
7
8     # Initialize the label encoder
9     label_encoder = LabelEncoder()
10
11    # Fit and transform each column to integer labels
12    for col in columns_to_encode:
13        frame1[col] = label_encoder.fit_transform(frame1[col])
14
15    return frame1
16

```

Python Version

Anaconda 4.0/Python 2.7.11

Encoded the First Three Columns

Cereal Regression - Louie Venegas > Execute Python Script > Results dataset

rows	columns	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potassium	vitamins	shelf	weight	cups	rating
77	16																
		0	3	0	70	4	1	0.13	10	5	6	0.28	25	3	1	0.33	68.402973
		1	5	0	120	3	5	0.015	2	8	8	0.135	0	3	1	1	33.983679
		2	2	0	70	4	1	0.26	9	7	5	0.32	25	3	1	0.33	59.425505
		3	2	0	50	4	0	0.14	14	8	0	0.33	25	3	1	0.5	93.704912
		4	6	0	110	2	2	0.2	1	14	8	0.33	25	3	1	0.75	34.384843
		5	1	0	110	2	2	0.18	1.5	10.5	10	0.07	25	1	1	0.75	29.509541
		6	2	0	110	2	0	0.125	1	11	14	0.03	25	2	1	1	33.174094
		7	1	0	130	3	2	0.21	2	18	8	0.1	25	3	1.33	0.75	37.038562
		8	6	0	90	2	1	0.2	4	15	6	0.125	25	1	1	0.67	49.120253
		9	4	0	90	3	0	0.21	5	13	5	0.19	25	3	1	0.67	53.313813
		10	5	0	120	1	2	0.22	0	12	12	0.035	25	2	1	0.75	18.042851
		11	1	0	110	6	2	0.29	2	17	1	0.105	25	1	1	1.25	50.764999
		12	1	0	120	1	3	0.21	0	13	9	0.045	25	2	1	0.75	19.823573
		13	1	0	110	3	2	0.14	2	13	7	0.105	25	3	1	0.5	40.400208
		14	1	0	110	1	1	0.18	0	12	13	0.055	25	2	1	1	22.736446
		15	6	0	110	2	0	0.28	0	22	3	0.025	25	1	1	1	41.445019
		16	2	0	100	2	0	0.29	1	21	2	0.035	25	1	1	1	45.863324
		17	2	0	110	1	0	0.09	1	13	12	0.02	25	2	1	1	35.782791
		18	1	0	110	1	1	0.18	0	12	13	0.065	25	2	1	1	22.396513
		19	2	0	110	3	3	0.14	4	10	7	0.16	25	3	1	0.5	40.448772
		20	3	1	100	3	0	0.08	1	21	0	0.001	0	2	1	1	64.533816
		21	2	0	110	2	0	0.22	1	21	3	0.03	25	3	1	1	46.895644
		22	1	0	100	2	1	0.14	2	11	10	0.12	25	3	1	0.75	36.176196
		23	6	0	100	2	0	0.19	1	18	5	0.08	25	3	1	0.75	44.330856

Linear regression assumes a linear relationship between the independent variables and the dependent variable.

Real-world data often has non-linear features. Squaring or cubing certain features can help capture these nonlinear

relationships. So, I chose features that are likely to have a direct impact on the target variable and applied the following python script:

```
def azureml_main(frame1):  
  
    # Specify the columns to perform actions on (columns four through six and columns nine and ten)  
  
    sqrList = frame1.columns[3:6] + frame1.columns[8:10]  
  
  
    # Calculate squared values for the specified columns  
  
    sqredList = [col + " Sqred" for col in sqrList]  
  
    frame1[sqredList] = frame1[sqrList] ** 2  
  
  
    # Calculate cubed values for the specified columns  
  
    cubeList = [col + " 3" for col in sqrList]  
  
    frame1[cubeList] = frame1[sqrList] ** 3  
  
  
    return frame1
```

This script resulted in the following modifications to the cereal dataset.

rows
77columns
26

cups	rating	calories Sqred	carbo Sqred	fat Sqred	protein Sqred	sugars Sqred	calories 3	carbo 3	fat 3	protein 3	sugars 3
0.33	68.402973	4900	25	1	16	36	343000	125	1	64	216
1	33.983679	14400	64	25	9	64	1728000	512	125	27	512
0.33	59.425505	4900	49	1	16	25	343000	343	1	64	125
0.5	93.704912	2500	64	0	16	0	125000	512	0	64	0
0.75	34.384843	12100	196	4	4	64	1331000	2744	8	8	512
0.75	29.509541	12100	110.25	4	4	100	1331000	1157.625	8	8	1000
1	33.174094	12100	121	0	4	196	1331000	1331	0	8	2744
0.75	37.038562	16900	324	4	9	64	2197000	5832	8	27	512
0.67	49.120253	8100	225	1	4	36	729000	3375	1	8	216
0.67	53.313813	8100	169	0	9	25	729000	2197	0	27	125
0.75	18.042851	14400	144	4	1	144	1728000	1728	8	1	1728
1.25	50.764999	12100	289	4	36	1	1331000	4913	8	216	1
0.75	19.823573	14400	169	9	1	81	1728000	2197	27	1	729
0.5	40.400208	12100	169	4	9	49	1331000	2197	8	27	343
1	22.736446	12100	144	1	1	169	1331000	1728	1	1	2197
1	41.445019	12100	484	0	4	9	1331000	10648	0	8	27
1	45.863324	10000	441	0	4	4	1000000	9261	0	8	8
1	35.782791	12100	169	0	1	144	1331000	2197	0	1	1728
1	22.396513	12100	144	1	1	169	1331000	1728	1	1	2197
0.5	40.448772	12100	100	9	9	49	1331000	1000	27	27	343
1	64.533816	10000	441	0	9	0	1000000	9261	0	27	0
1	46.895644	12100	441	0	4	9	1331000	9261	0	8	27

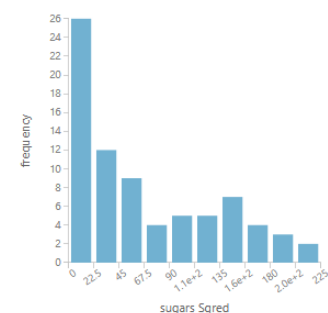
Statistics

Mean	67.4156
Median	49
Min	0
Max	225
Standard Deviation	64.79
Unique Values	16
Missing Values	0
Feature Type	Numeric Feature

Visualizations

sugars Sqred

Histogram

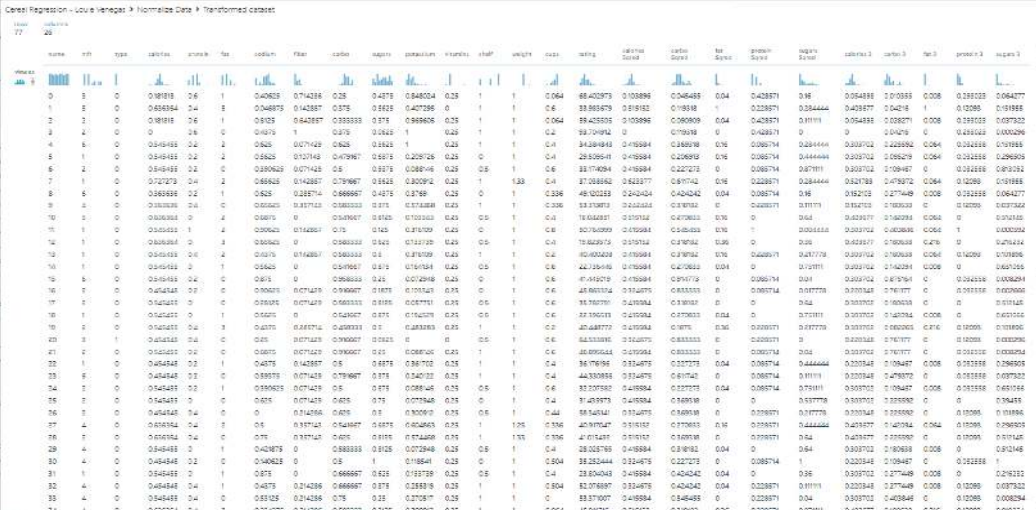
compare to 

To avoid overweighing a feature simply because its values are larger than others, I normalized the data. This process makes the data easier to interpret and understand. If our features have similar ranges, we are more certain that we are capturing the true information in each feature.

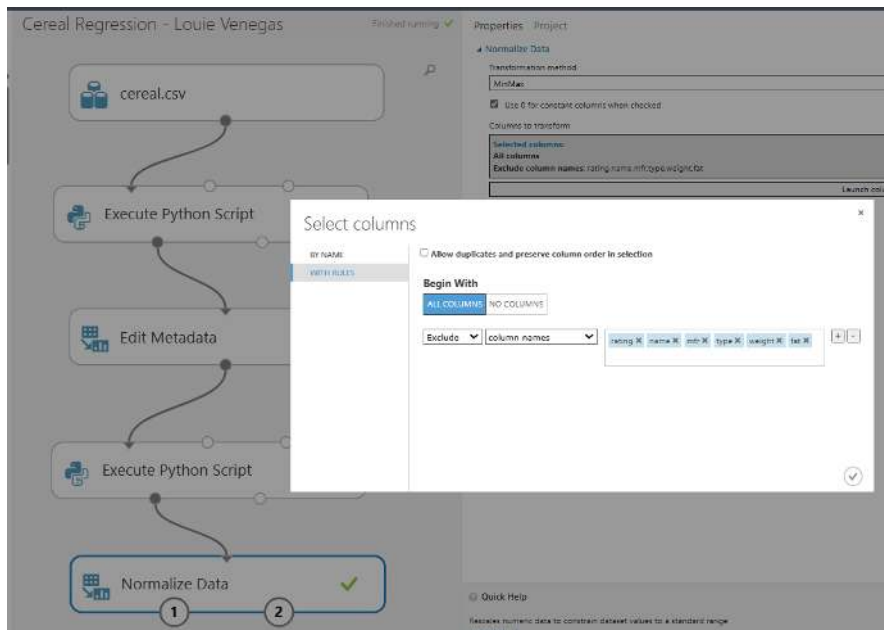
Normalization

Cereal Regression - Louie Venegas > Normalize Data > Transformed dataset

Rows: 77 Columns: 26



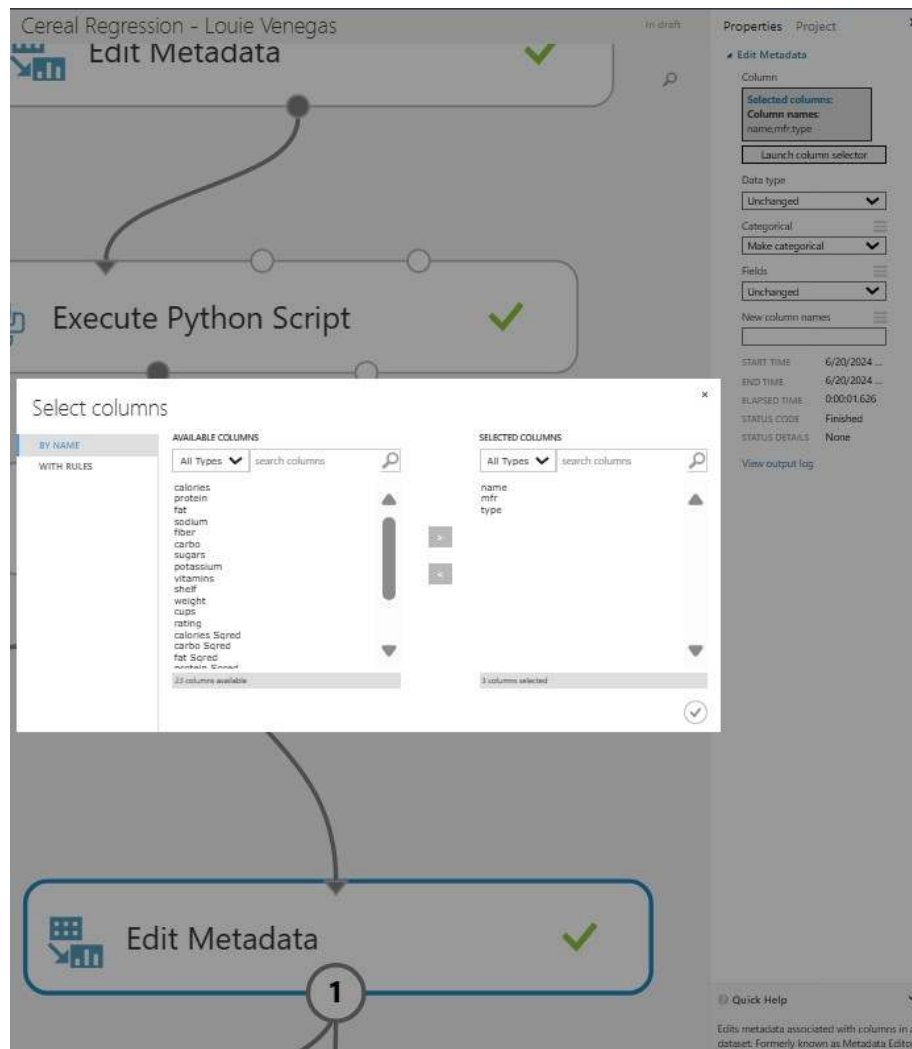
	name	xfl	type	calories	protein	fat	sodium	fiber	sodium	sugar	potassium	vitamins	shelf	weight	cost	rating	cereal brand	cereal type	fat brand	protein brand	sugar brand	sodium brand	vitamin brand	cereal 3		
0	0	0	0	0.01918	0.6	1	0.40635	0.714216	0.28	0.4375	0.848024	0.28	1	1	0.064	88.402973	0.103896	0.046488	0.04	0.428871	0.16	0.054888	0.03588	0.008	0.280203	0.064277
1	1	0	0	0.683634	0.4	1	0.048975	0.142857	0.375	0.3333	0.427295	0	1	1	0.6	33.869479	0.191932	0.19318	0	0.222871	0.334444	0.403877	0.04216	1	0.10288	0.191888
2	2	0	0	0.891818	0.4	1	0.8335	0.642857	0.333333	0.375	0.369605	0.35	1	1	0.064	88.402973	0.103896	0.046488	0.04	0.428871	0.311111	0.054888	0.03588	0.008	0.280203	0.064277
3	3	2	0	0	0.26	0	0.4375	1	0.375	0.05625	1	0.25	1	1	0.2	39.704942	0	0.79918	0	0.428871	0	0	0.04216	0	0.426122	0.000296
4	4	0	0	0.01918	0.2	2	0.625	0.071429	0.625	0.3625	1	0.25	1	1	0.4	34.381843	0.159884	0.069798	0.16	0.085774	0.234444	0.037702	0.222882	0.004	0.102888	0.191888
5	5	1	0	0.01918	0.2	2	0.33625	0.107143	0.479587	0.3375	0.222728	0.25	0	1	0.4	29.05941	0.159884	0.069798	0.16	0.085774	0.444444	0.037702	0.222882	0.004	0.102888	0.191888
6	6	2	0	0.01918	0.2	0	0.339625	0.071429	0.5	0.3375	0.288716	0.25	0.5	1	0.6	33.716284	0.159884	0.069798	0.16	0.085774	0.387778	0.037702	0.222882	0.004	0.102888	0.191888
7	7	1	0	0.712719	0.4	2	0.669625	0.142857	0.799587	0.33625	0.200912	0.25	1	1.33	0.4	27.088662	0.263777	0.397742	0.16	0.228871	0.284444	0.037702	0.222882	0.004	0.102888	0.191888
8	8	0	0	0.891818	0.2	1	0.625	0.228714	0.688687	0.4375	0.3759	0.25	0	1	0.336	48.102253	0.242424	0.424242	0.04	0.085774	0.16	0.182702	0.227449	0.008	0.102888	0.191888
9	9	0	0	0.01918	0.4	0	0.65625	0.337143	0.588333	0.375	0.373888	0.25	1	1	0.336	33.716284	0.242424	0.370161	0	0.428871	0.377778	0.182702	0.227449	0.008	0.102888	0.191888
10	10	0	0	0.01918	0.2	2	0.6875	0	0.599587	0.625	0.101010	0.25	0.5	1	0.6	19.65887	0.101010	0.471818	0.16	0	0.64	0.037702	0.227449	0.008	0.102888	0.191888
11	11	0	0	0.01918	0.2	2	0.309625	0.142857	0.75	0.745	0.371019	0.25	0	1	0.6	30.760899	0.159884	0.069798	0.16	1	0.000161	0.037702	0.227449	0.008	0.102888	0.191888
12	12	1	0	0.01918	0.2	2	0.669625	0	0.588333	0.525	0.137729	0.25	0.5	1	0.4	19.65887	0.101010	0.471818	0.16	0	0.64	0.037702	0.227449	0.008	0.102888	0.191888
13	13	1	0	0.01918	0.2	0	0.55625	0	0.599587	0.525	0.101010	0.25	0.5	1	0.6	42.101046	0.159884	0.069798	0.16	0	0.000161	0.037702	0.227449	0.008	0.102888	0.191888
14	14	1	0	0.01918	0.2	0	0.675	0	0.588333	0.25	0.012918	0.25	0	1	0.6	47.454979	0.159884	0.069798	0.16	0.085774	0.477778	0.037702	0.227449	0.008	0.102888	0.191888
15	15	0	0	0.01918	0.2	0	0.50625	0.071429	0.588333	0.1875	0.101010	0.25	0	1	0.6	48.860154	0.159884	0.069798	0.16	0.085774	0.587778	0.037702	0.227449	0.008	0.102888	0.191888
16	16	1	0	0.01918	0.2	0	0.59375	0.071429	0.588333	0.1875	0.101010	0.25	0.5	1	0.6	35.769791	0.241010	0.370161	0	0	0.64	0.037702	0.227449	0.008	0.102888	0.191888
17	17	1	0	0.01918	0.2	0	0.55625	0	0.599587	0.525	0.101010	0.25	0.5	1	0.6	32.166013	0.101010	0.471818	0.16	0	0.000161	0.037702	0.227449	0.008	0.102888	0.191888
18	18	1	0	0.01918	0.2	0	0.675	0.688716	0.588333	0.5	0.283383	0.25	1	1	0.6	40.488772	0.241010	0.370161	0.16	0.085774	0.587778	0.037702	0.227449	0.008	0.102888	0.191888
19	19	0	0	0.01918	0.2	0	0.675	0.071429	0.588333	0.0625	0	0	0.5	1	0.6	34.333333	0.241010	0.370161	0.16	0	0.000161	0.037702	0.227449	0.008	0.102888	0.191888
20	20	3	1	0.01918	0.2	0	0.5875	0.071429	0.588333	0.525	0.087101	0.25	1	1	0.6	48.895544	0.159884	0.069798	0.16	0.085774	0.587778	0.037702	0.227449	0.008	0.102888	0.191888
21	21	0	0	0.01918	0.2	0	0.59375	0.071429	0.588333	0.5	0.087101	0.25	1	1	0.6	36.79794	0.159884	0.069798	0.16	0.085774	0.444444	0.037702	0.227449	0.008	0.102888	0.191888
22	22	1	0	0.01918	0.2	0	0.59375	0.071429	0.799587	0.375	0.240131	0.25	1	1	0.4	44.333333	0.159884	0.069798	0.16	0.085774	0.587778	0.037702	0.227449	0.008	0.102888	0.191888
23	23	0	0	0.01918	0.2	0	0.590625	0.071429	0.5	0.575	0.088748	0.25	0.5	1	0.6	33.37587	0.159884	0.069798	0.16	0.085774	0.587778	0.037702	0.227449	0.008	0.102888	0.191888
24	24	1	0	0.01918	0.2	0	0.625	0.071429	0.625	0.375	0.073828	0.25	0	1	0.4	31.403973	0.159884	0.069798	0.16	0	0.000161	0.037702	0.227449	0.008	0.102888	0.191888
25	25	0	0	0.01918	0.4	0	0	0.142856	0.625	0.5	0.300912	0.25	0.5	1	0.4	36.142141	0.159884	0.069798	0.16	0.085774	0.587778	0.037702	0.227449	0.008	0.102888	0.191888
26	26	0	0	0.01918	0.4	0	0.6	0.187143	0.548887	0.5875	0.404040	0.15	1	1.33	0.136	40.705047	0.101010	0.471818	0.16	0.085774	0.444444	0.037702	0.227449	0.008	0.102888	0.191888
27	27	4	0	0.01918	0.4	0	0.75	0.187143	0.405	0.61875	0.174286	0.15	1	1.33	0.136	40.705047	0.101010	0.471818	0.16	0.085774	0.444444	0.037702	0.227449	0.008	0.102888	0.191888
28	28	0	0	0.01918	0.4	1	0.421875	0	0.888333	0.3125	0.072949	0.25	0.5	1	0.4	28.025765	0.159884	0.069798	0.16	0.085774	0.587778	0.037702	0.227449	0.008	0.102888	0.191888
29	29	4	0	0.01918	0.2	0	0.140625	0	0.5	1	0.18841	0.25	0	1	0.504	33.234444	0.142857	0.227702	0	0.085774	1	0.032344	0.104847	0	0.102888	0.191888
30	30	1	0	0.01918	0.2	1	0.875	0	0.688687	0.525	0.137729	0.25	0.5	1	0.4	23.04043	0.159884	0.069798	0.16	0.085774	0.587778	0.037702	0.227449	0.008	0.102888	0.191888
31	31	4	0	0.01918	0.4	1	0.4375	0.214286	0.688687	0.375	0.255918	0.25	1	1	0.504	32.07697	0.159884	0.069798	0.16	0.085774	0.587778	0.037702	0.227449	0.008	0.102888	0.191888
32	32	4	0	0.01918	0.4	0	0.59125	0.214286	0.75	0.33	0.270571	0.25	1	1	0	33.37007	0.159884	0.069798	0.16	0.085774	0.587778	0.037702	0.227449	0.008	0.102888	0.191888
33	33	1	0	0.01918	0.2	2	0.234375	0.214286	0.888333	0.3125	0.300912	0.25	1	1	0.064	45.81176	0.159884	0.069798	0.16	0.085774	0.587778	0.037702	0.227449	0.008	0.102888	0.191888



In the beginning of our experiment, the first three fields name, mfr, and type were encoded. Since these were categories encoded into numbers, they had to be made categorical. The first couple of iterations through the process this step was overlooked but the models mean absolute error and root mean squared error were ranging above five closer to

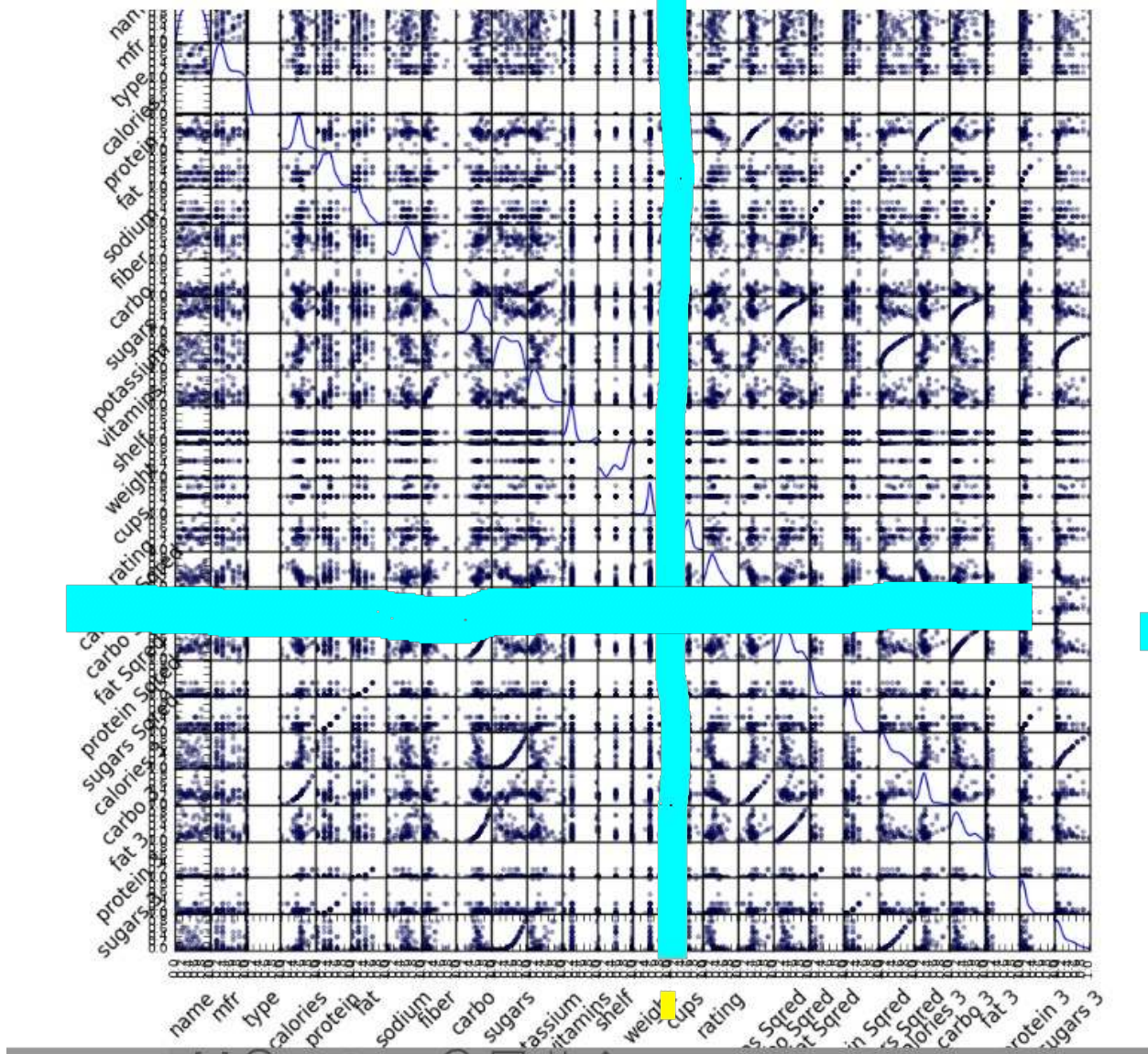
ten (which was unacceptable). As part of the process of data engineering It was quickly realized these fields needed to be made categorical. The following is a screenshot of the process of making the fields name, mfr, and type categorical.

Categorical



Data visualization is an important part of the feature engineering process. One way to visualize this is with a pairwise scatter plot. A pairwise scatter plot overlays pairs of data to help the engineer visualize relationships that cannot be seen just by reading numbers. In order to obtain this graph a python code was executed.

Pairwise Scatter Plot



The python script used to obtain the preceding data visualization:

```
def azureml_main(frame1):
```

```
    ## import libraries
```

```
    import matplotlib
```

```

matplotlib.use('agg') # Set backend

from pandas.tools.plotting import scatter_matrix
import pandas.tools.rplot as rplot
import matplotlib.pyplot as plt
import numpy as np

## Create a pair-wise scatter plot

Azure = True

if(Azure == False):

    frame1 = eeframe

fig1 = plt.figure(1, figsize=(10, 10))
ax = fig1.gca()

sm=scatter_matrix(frame1, alpha=0.3,
                  diagonal='kde', ax = ax)

[s.xaxis.label.set_rotation(45) for s in sm.reshape(-1)]

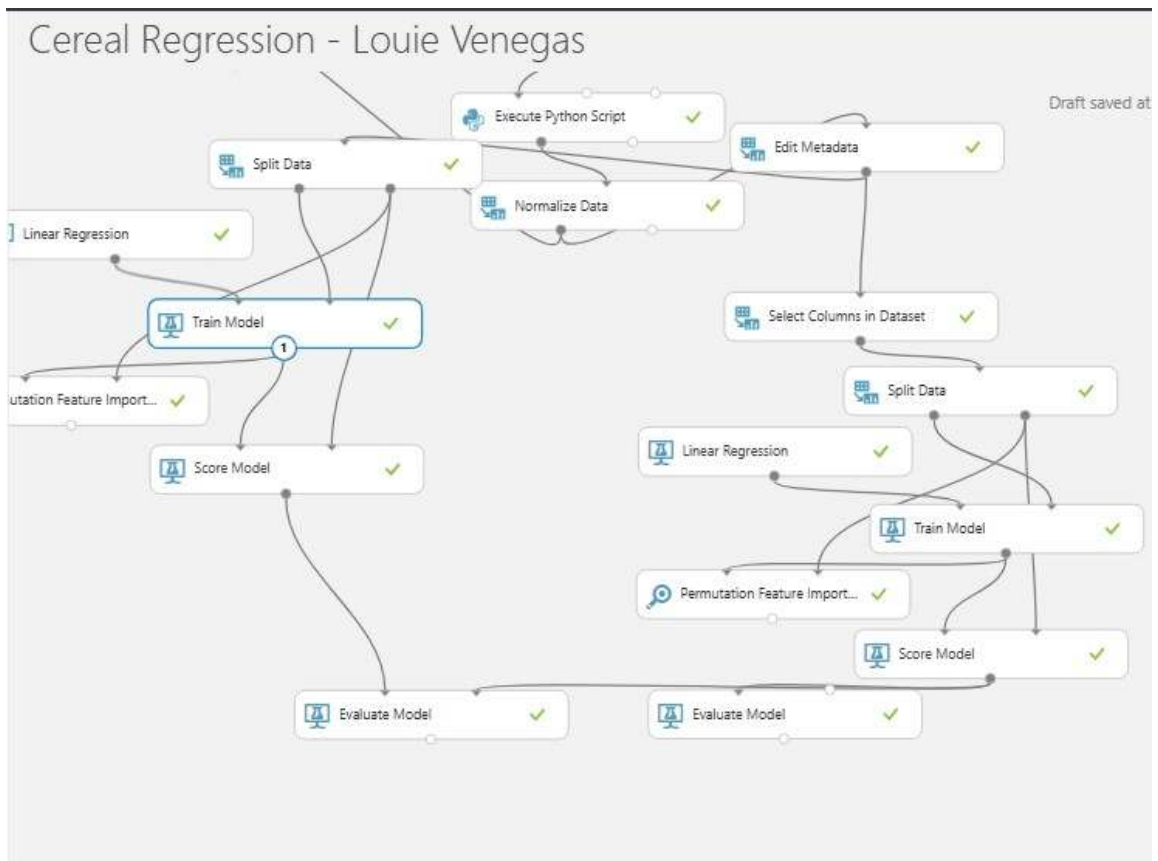
[s.yaxis.label.set_rotation(45) for s in sm.reshape(-1)]

plt.show()

if(Azure == True): fig1.savefig('scatter1.png')

```

**Splitting Data, Trained Model, Linear Regression Module, Score Module, Evaluate model,
Permutation Feature Importance**

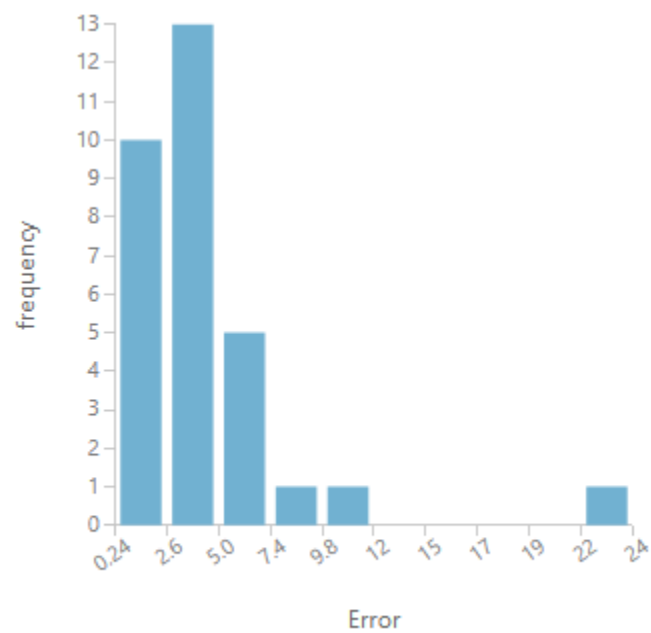


To gain a handle on how the model would score before removing features, the model was run as is. The following is a screenshot of the evaluation results and the permutation feature importance results.

Metrics

Mean Absolute Error	4.336294
Root Mean Squared Error	6.087806
Relative Absolute Error	0.411197
Relative Squared Error	0.164444
Coefficient of Determination	0.835556

Error Histogram



Feature Importance

rows

25

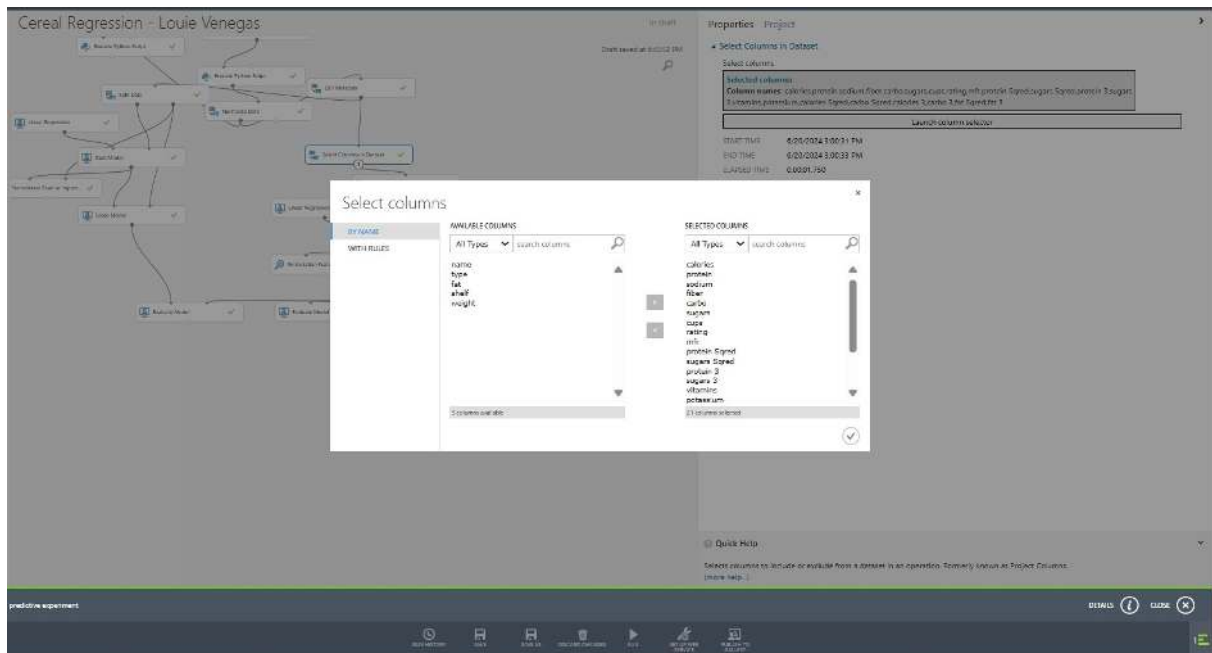
columns

2

view as	Feature		Score	
				
		fat	2.601766	
		mfr	1.558653	
		sodium	1.413459	
		sugars Sqred	1.282498	
		shelf	1.05486	
		fiber	1.04704	
		sugars	1.028833	
		protein	1.005332	
		sugars 3	0.810629	
		calories 3	0.692128	
		calories Sqred	0.627878	
		potassium	0.577505	
		calories	0.453409	
		vitamins	0.426074	
		protein Sqred	0.389917	
		carbo	0.141652	
		protein 3	0.139584	
		carbo Sqred	0.045777	
		carbo 3	0.004587	
		fat Sqred	0.003949	
		fat 3	0.001577	
		name	0	
		type	0	

The numbers in the evaluate model were unacceptable and would not produce an accurate model. Specifically Mean Absolute Error and Root Mean Square Error were too far from zero even though the coefficient was good. Some features were removed based on a range of factors from the feature importance module and pairwise scatterplot.

Pruned Features

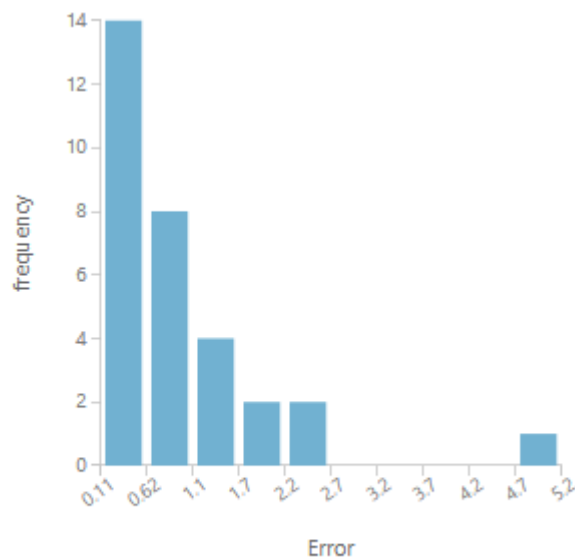


After pruning the above features, after much trial and error an acceptable evaluation was produced.

Metrics

Mean Absolute Error	1.064199
Root Mean Squared Error	1.463008
Relative Absolute Error	0.100915
Relative Squared Error	0.009497
Coefficient of Determination	0.990503

Error Histogram



Majority of the data does lie between 0.11 and 0.62 but the mean absolute error and root mean squared error are both within acceptable range and the coefficient is within an acceptable range.

Conclusion

After encoding the three string columns, engineering squared and cubed terms, normalizing the numeric fields, and pruning features flagged by the permutation feature importance module and the pairwise scatter plot, the model reached a mean absolute error and root mean squared error within an acceptable range with a coefficient of determination to match.

The process mattered as much as the result. The first iterations failed because the encoded columns were still being treated as continuous values rather than categories, and the error rates sat near ten until that was corrected. Feature engineering and data typing, not model selection, drove the accuracy here.

For a manufacturer the practical value is direction. A model built this way indicates which attributes on the nutrition label move consumer ratings, so recipe decisions can be tested against data before a product is produced rather than after.