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

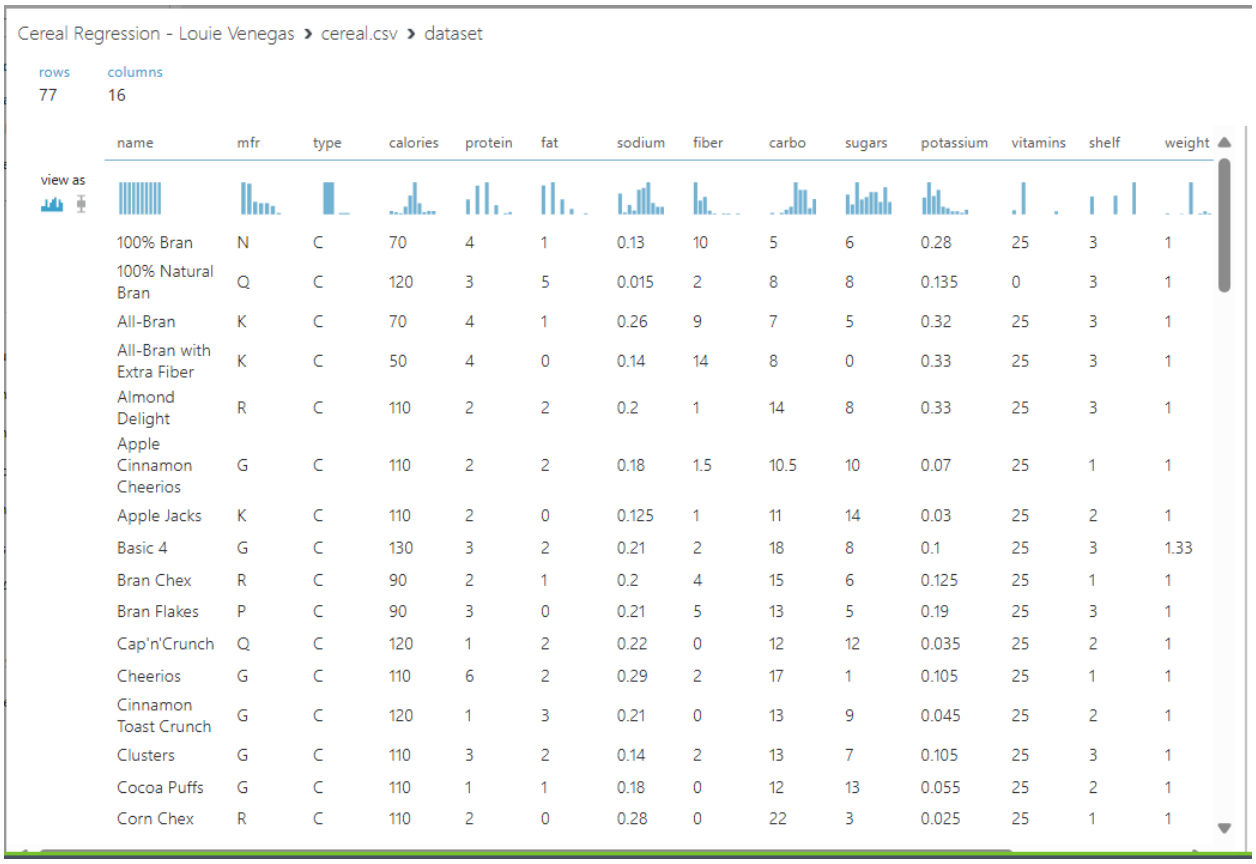
Introduction to the Problem

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

Properties Project

Execute Python Script

Python script

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

def azureml_main(frame1=None):
    # Choosing the colums to encode
    columns_to_encode = frame1.columns[:3]

    # Initialize the label encoder
    label_encoder = LabelEncoder()

    # Fit and transform each column to integer labels
    for col in columns_to_encode:
        frame1[col] = label_encoder.fit_transform(frame1[col])

    return frame1
```

Python Version

Anaconda 4.0/Python 2.7.11

Encoded the First Three Columns

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

rows 77 columns 16

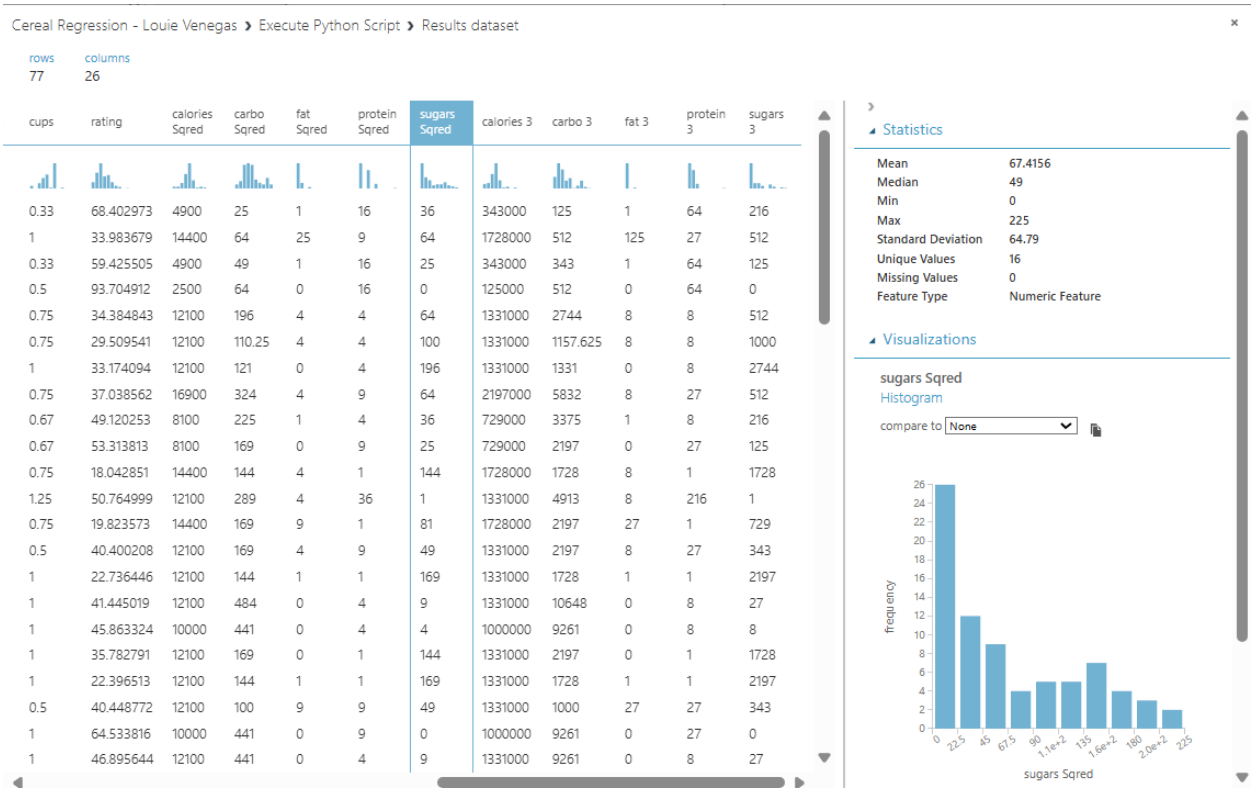
	name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potassium	vitamins	shelf	weight	cups	rating
0	3	0	70	4	1	0.13	10	5	6	0.28	25	3	1	0.33	68.402973
1	5	0	120	3	5	0.015	2	8	8	0.135	0	3	1	1	33.983679
2	2	0	70	4	1	0.26	9	7	5	0.32	25	3	1	0.33	59.425505
3	2	0	50	4	0	0.14	14	8	0	0.33	25	3	1	0.5	93.704912
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:

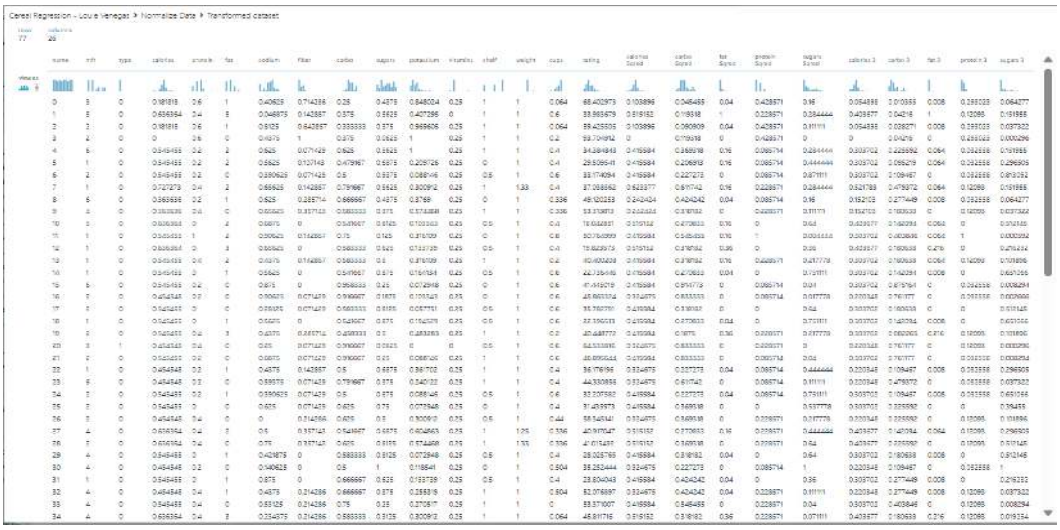
```
def azureml_main(frame1):
    # Specify the columns to perform actions on (columns four through six and columns nine and ten)
    sqrList = frame1.columns[3:6] + frame1.columns[8:10]

    # 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
```

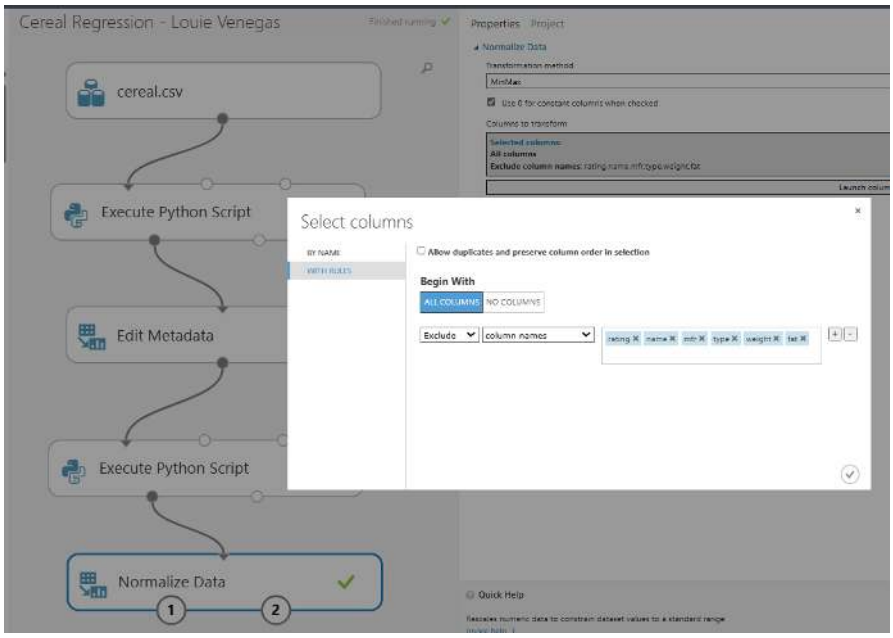
This script resulted in the following modifications to the cereal dataset.



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

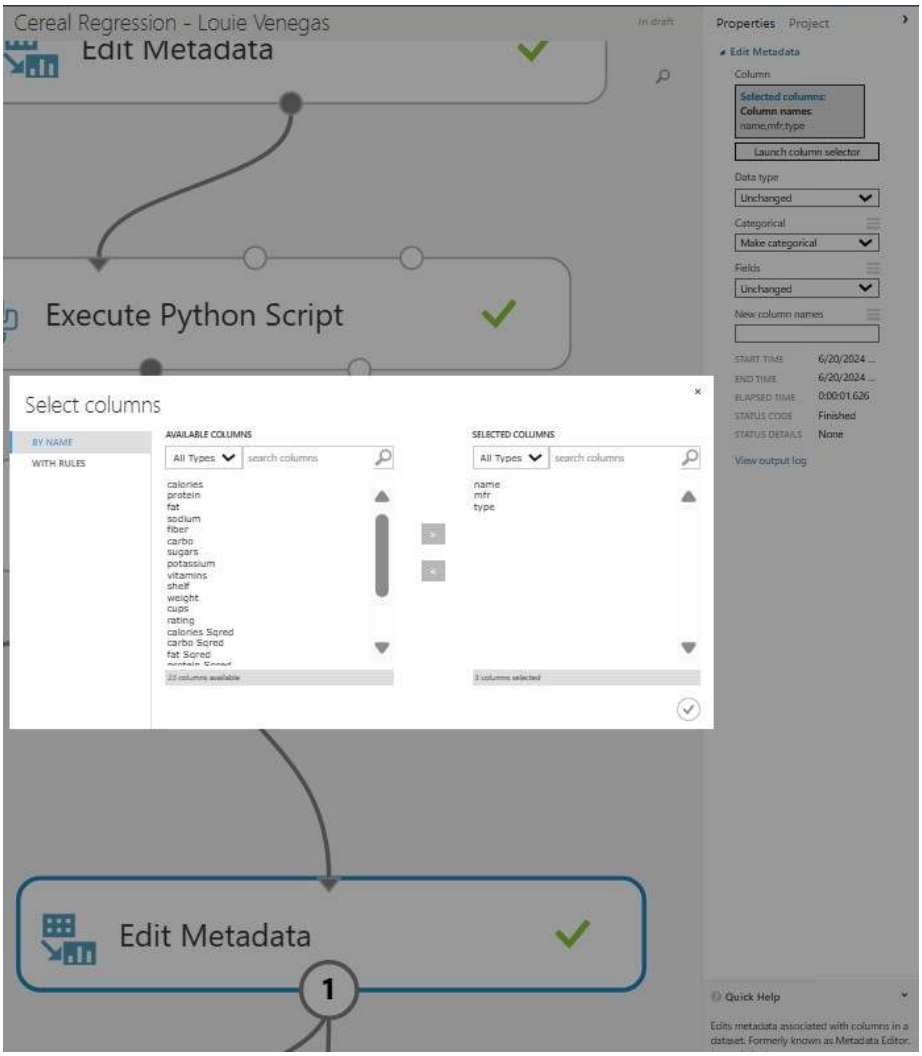




In the beginning of our experiment, the first three fields name, mfr, and type were encoded. Since these were categories encoded into numbers, they had to be made categorical. The first couple of iterations through the process this step was overlooked but the models mean absolute error and root mean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

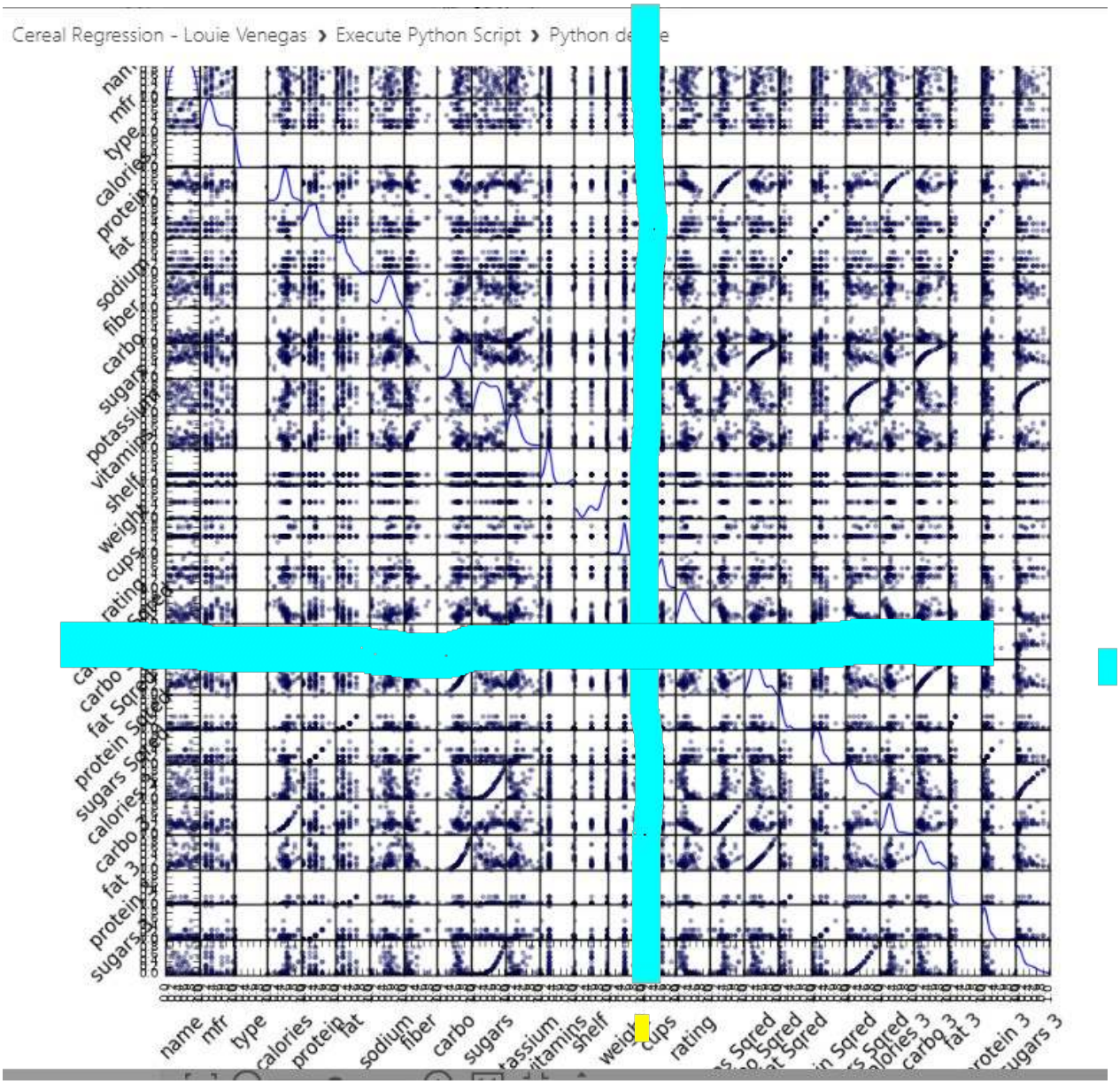
ten (which was unacceptable). As part of the process of data engineering It was quickly realized these fields needed to be made categorical. The following is a screenshot of the process of making the fields name, mfr, and type categorical.

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

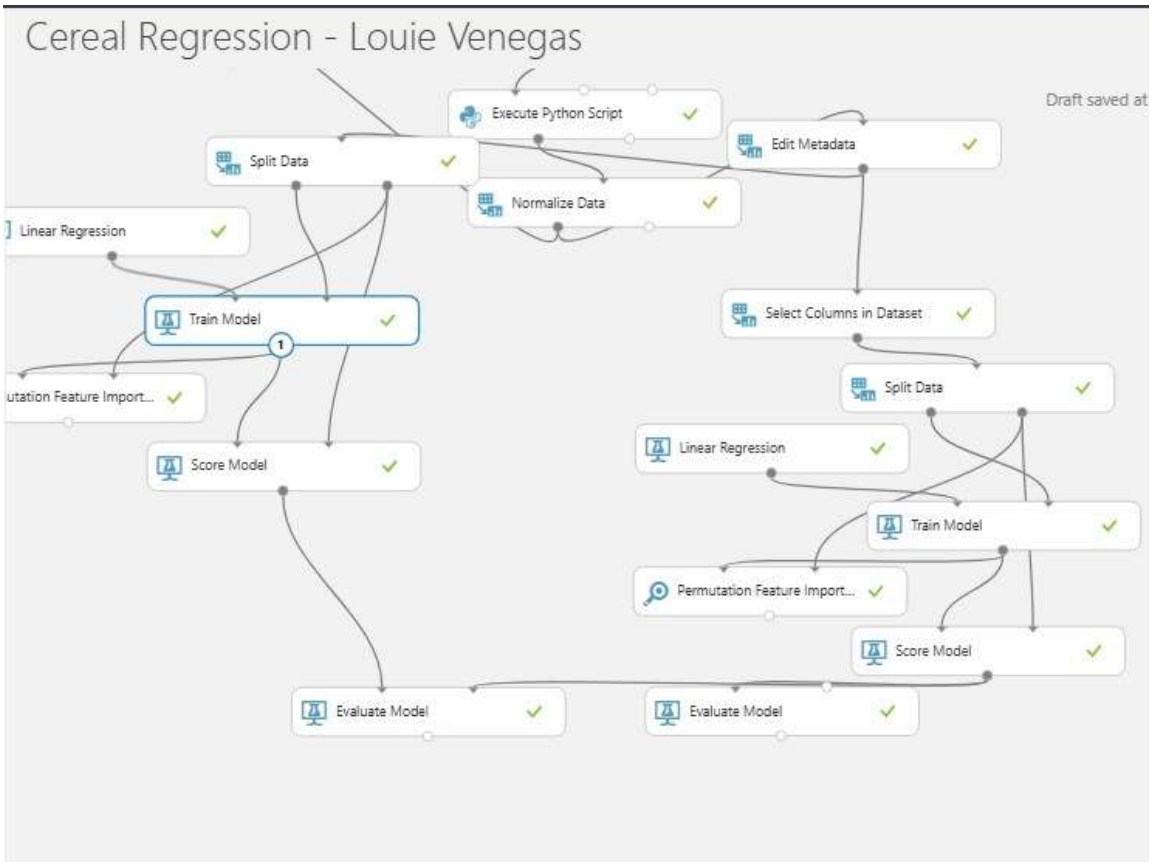


The python script used to obtain the preceding data visualization:

```
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

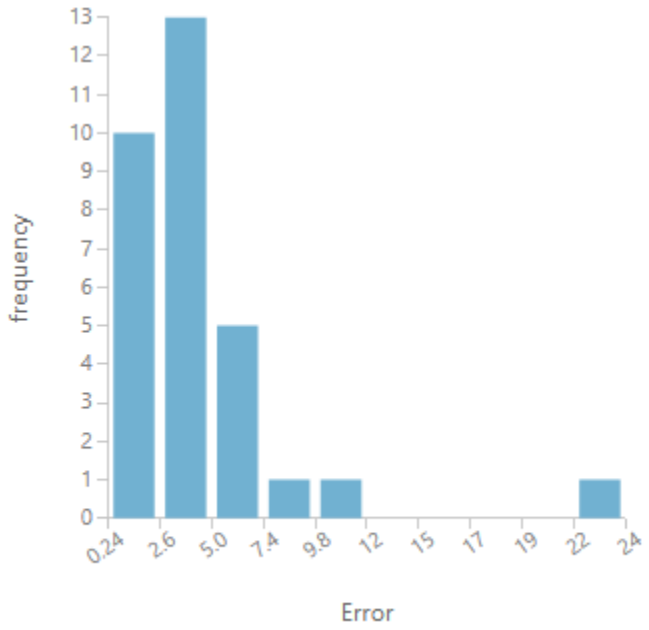


To gain a handle on how the model would score before removing features, the model was run as is. The following is a screenshot of the evaluation results and the permutation feature importance results.

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

Cereal Regression - Louie Venegas › Permutation Feature Importance › Feature importance

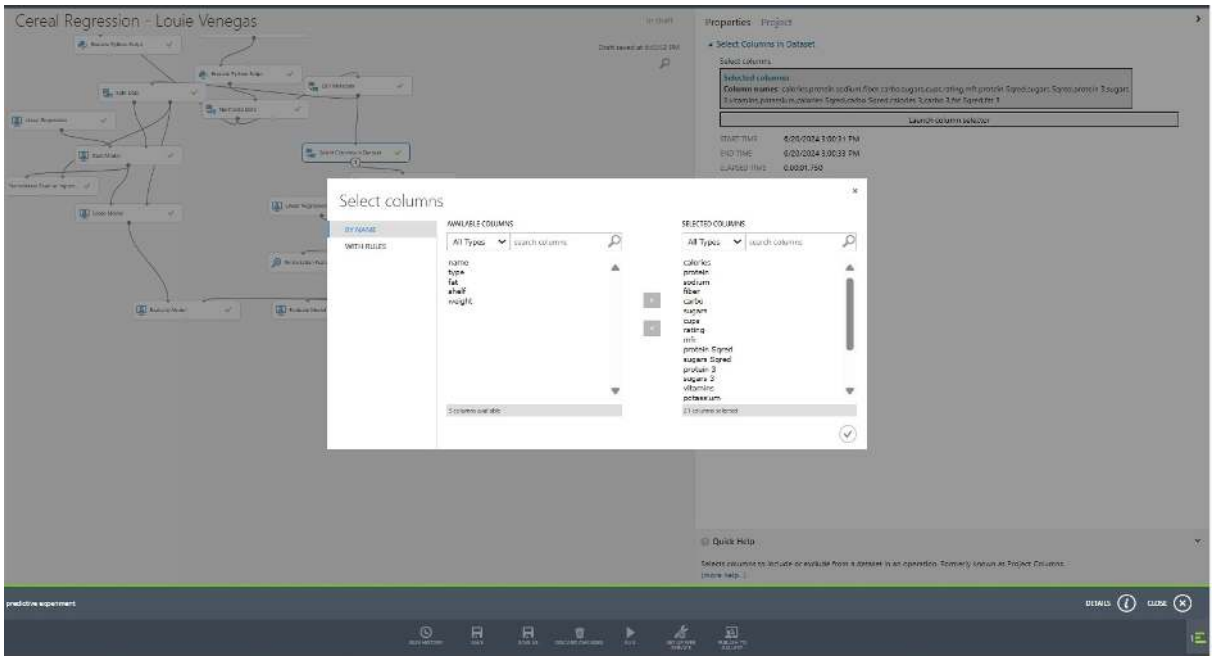
rows 25 columns 2

view as

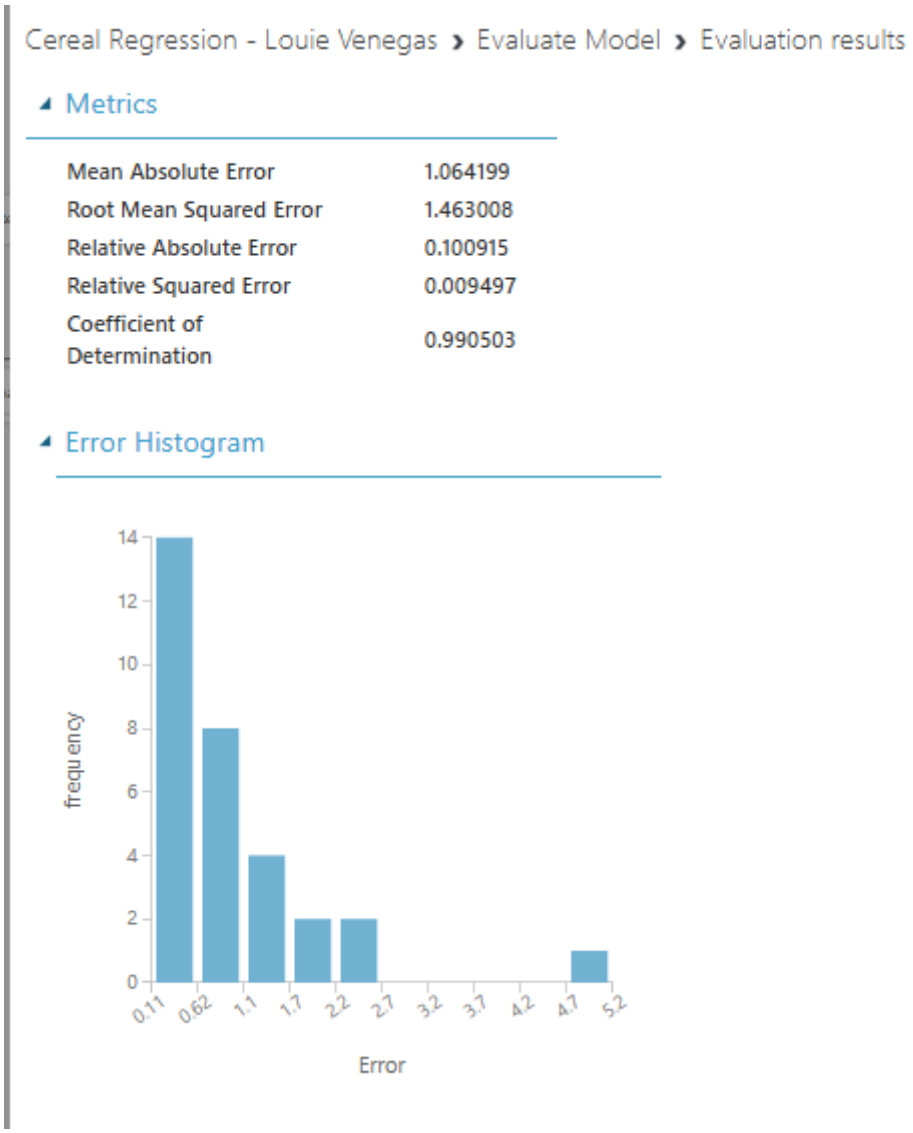
Feature	Score
fat	2.601766
mfr	1.558653
sodium	1.413459
sugars Sqred	1.282498
shelf	1.05486
fiber	1.04704
sugars	1.028833
protein	1.005332
sugars 3	0.810629
calories 3	0.692128
calories Sqred	0.627878
potassium	0.577505
calories	0.453409
vitamins	0.426074
protein Sqred	0.389917
carbo	0.141652
protein 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.



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.