Cereal Analysis Based on Ratings Using Machine Learning Techniques.

BY:

I. Nitish Arvapalli (22BCE3143)
II. Ayan Kumar Singh (22BCI0182)
III. Omkar Choudhury (22BCI0202)
IV. Anurav Gupta (22BBS0107)

1. Introduction

1.1. Project Overview

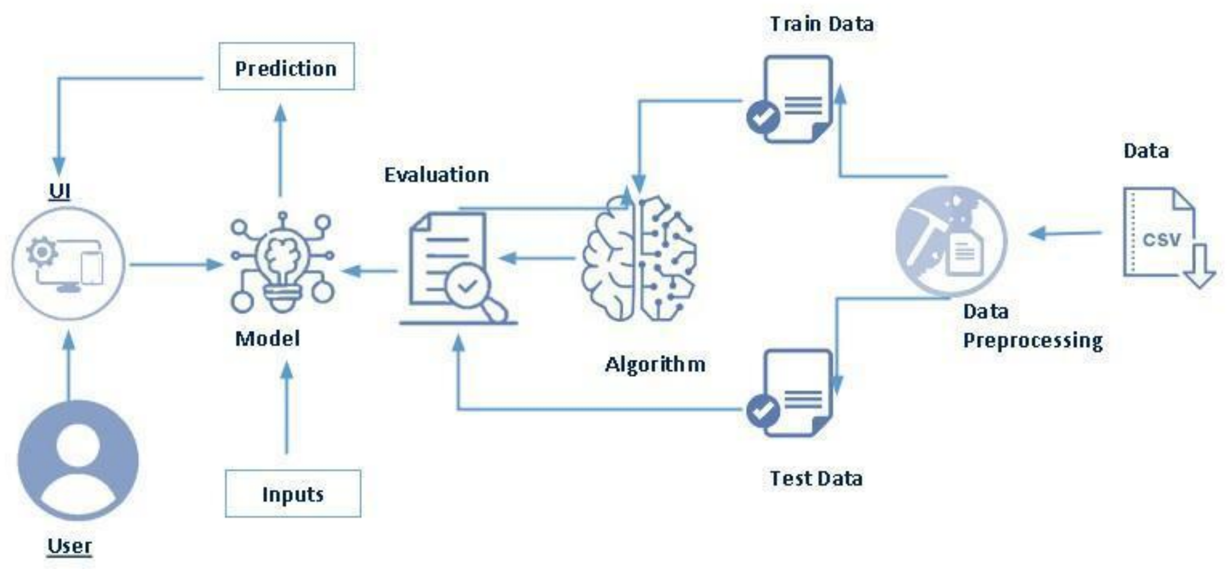
The project aims to leverage machine learning techniques to analyze cereal ratings and provide actionable insights for consumers and manufacturers. By developing predictive models, the project seeks to classify cereals based on their ratings and identify influential factors affecting consumer preferences. This analysis will empower consumers with informed decision-making tools and assist manufacturers in optimizing product development and marketing strategies.

1.2. Objectives

- **Classification Problem**: Classify cereal ratings as a regression task to predict consumer preferences accurately.
- **Data Preprocessing**: Implementing robust data preprocessing techniques to clean, normalize, and transform the dataset for effective modeling.
- **Data Visualization**: Utilizing exploratory data analysis (EDA) techniques to visualize data distributions, correlations, and trends.
- **Model Building**: Applying a variety of machine learning algorithms suitable for regression tasks, selecting models based on dataset characteristics and visualization insights.
- **Model Evaluation**: Evaluating model performance using appropriate metrics such as R-squared (R2), Root Mean Square Error (RMSE), and Mean Absolute Percentage Error (MAPE).

- **Application Development**: Building a web application using Flask framework to integrate the predictive model for consumer interaction.

Technical Architecture:



2. Project Initialization and Planning Phase

2.1. Define Problem Statement

The project aims to develop predictive models that can accurately predict cereal ratings based on nutritional content, brand attributes, and consumer feedback. This will enable consumers to make informed choices and assist manufacturers in refining product strategies.

2.2. Project Proposal (Proposed Solution)

Propose the development of machine learning models to categorize cereals and recommend them based on consumer preferences and nutritional needs. The solution involves comprehensive data analysis, preprocessing, model development, and web application integration.

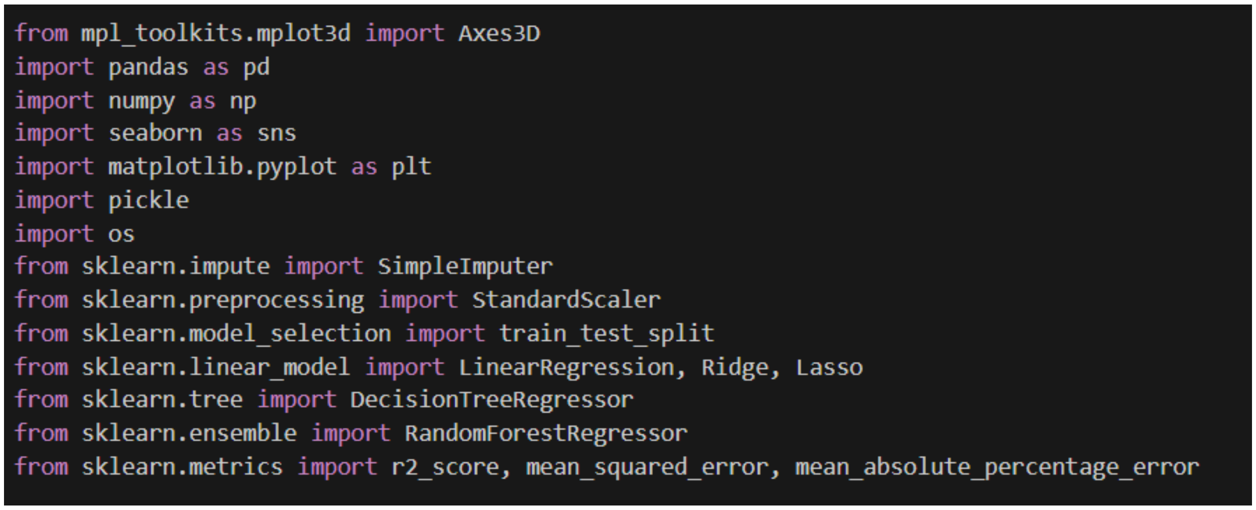
2.3. Initial Project Planning

- **Milestones**: Define project milestones including data collection, preprocessing, model development, evaluation, and application deployment.

- **Timelines**: Establish timelines for each project phase to ensure timely completion.
- **Resources**: Allocate resources such as datasets, computing resources, and development tools necessary for project execution.

2.4. Importing Necessary libraries

- **Numpy**- It is an open-source numerical Python library. It contains a multi-dimensional array and matrix data structures. It can be used to perform mathematical operations on arrays such as trigonometric, statistical, and algebraic routines.
- **Pandas**- It is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- **Seaborn**- Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **Matplotlib**- Visualisation with python. It is a comprehensive library for creating static, animated, and interactive visualizations in Python



```
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
import os
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_percentage_error
```

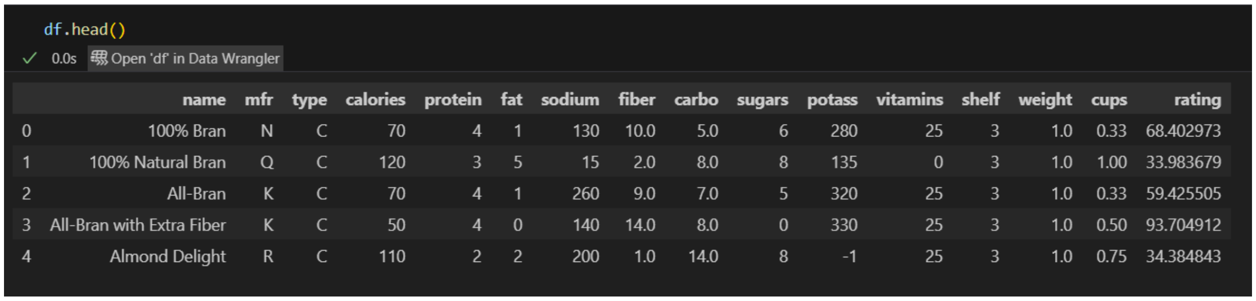
3. Data Collection and Preprocessing Phase

3.1. Data Collection Plan and Raw Data Sources Identified

- **Dataset**: Acquire dataset containing attributes such as name, manufacturer (mfr), type, calories, protein, fat, sodium, fiber, carbo, sugars, potass, vitamins, shelf, weight, cups, and rating.
- **Source**: https://www.kaggle.com/crawford/80-cereals

3.2. Analyzing The Data

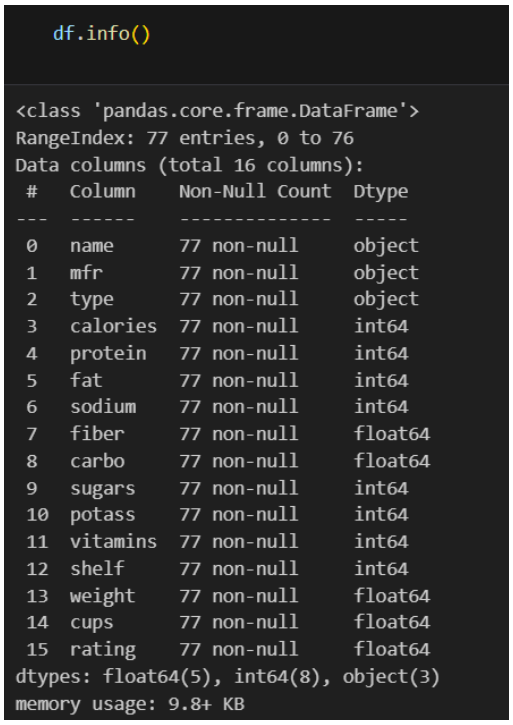
- **head ()** method is used to return top n (5 by default) rows of a DataFrame or series.



```
df.head()
✓ 0.0s  Open 'df' in Data Wrangler
```

	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	1.0	0.33	68.402973
1	100% Natural Bran	Q	C	120	3	5	15	2.0	8.0	8	135	0	3	1.0	1.00	33.983679
2	All-Bran	K	C	70	4	1	260	9.0	7.0	5	320	25	3	1.0	0.33	59.425505
3	All-Bran with Extra Fiber	K	C	50	4	0	140	14.0	8.0	0	330	25	3	1.0	0.50	93.704912
4	Almond Delight	R	C	110	2	2	200	1.0	14.0	8	-1	25	3	1.0	0.75	34.384843

- **info()** gives information about the data.

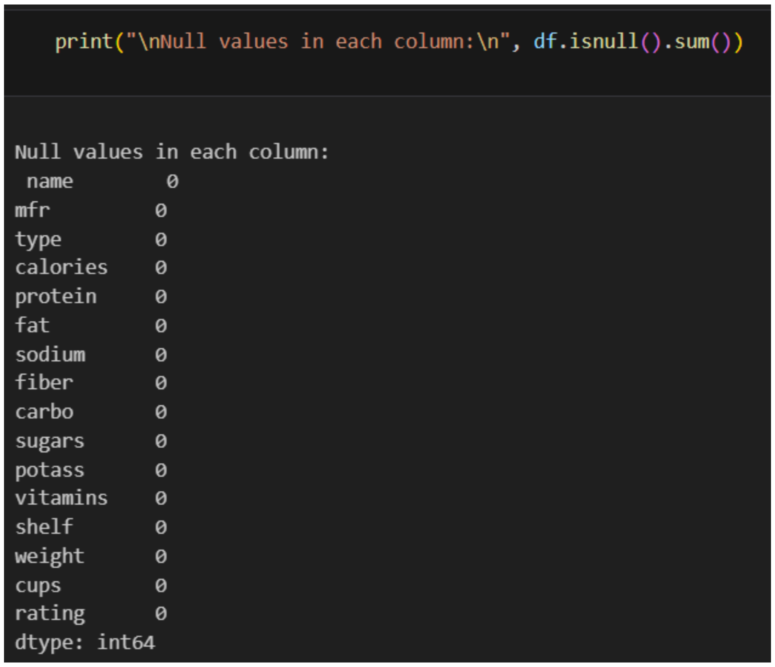


```
df.info()

<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
```

3.3. Data Quality Report

- **Handling Missing Values**: Handle missing values, outliers, and inconsistencies in the dataset using techniques like imputation and removal. Check whether any null values are there or not. if it is present then following can be done.

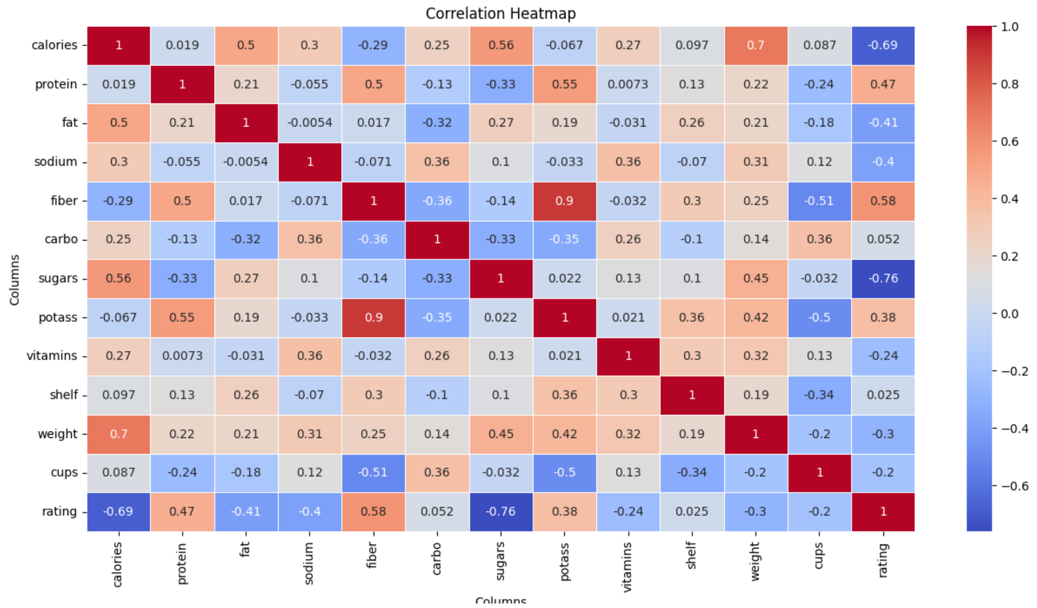
   a) **Filling NaN values with mean, median and mode using fillna() method:**



```
print("\nNull values in each column:\n", df.isnull().sum())
```

```
Null values in each column:
 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
```

b) **Heatmap:** It is way of representing the data in 2-D form. It gives coloured visual summary of the data. In our case this is the heatmap that we generated based on our data**.**

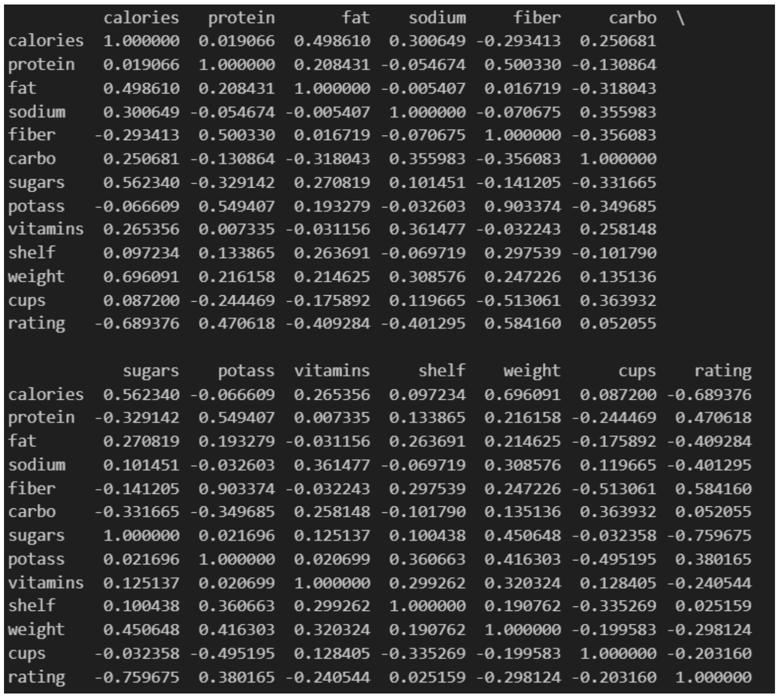


	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
calories	1	0.019	0.5	0.3	-0.29	0.25	0.56	-0.067	0.27	0.097	0.7	0.087	-0.69
protein	0.019	1	0.21	-0.055	0.5	-0.13	-0.33	0.55	0.0073	0.13	0.22	-0.24	0.47
fat	0.5	0.21	1	-0.0054	0.017	-0.32	0.27	0.19	-0.031	0.26	0.21	-0.18	-0.41
sodium	0.3	-0.055	-0.0054	1	-0.071	0.36	0.1	-0.033	0.36	-0.07	0.31	0.12	-0.4
fiber	-0.29	0.5	0.017	-0.071	1	-0.36	-0.14	0.9	-0.032	0.3	0.25	-0.51	0.58
carbo	0.25	-0.13	-0.32	0.36	-0.36	1	-0.33	-0.35	0.26	-0.1	0.14	0.36	0.052
sugars	0.56	-0.33	0.27	0.1	-0.14	-0.33	1	0.022	0.13	0.1	0.45	-0.032	-0.76
potass	-0.067	0.55	0.19	-0.033	0.9	-0.35	0.022	1	0.021	0.36	0.42	-0.5	0.38
vitamins	0.27	0.0073	-0.031	0.36	-0.032	0.26	0.13	0.021	1	0.3	0.32	0.13	-0.24
shelf	0.097	0.13	0.26	-0.07	0.3	-0.1	0.1	0.36	0.3	1	0.19	-0.34	0.025
weight	0.7	0.22	0.21	0.31	0.25	0.14	0.45	0.42	0.32	0.19	1	-0.2	-0.3
cups	0.087	-0.24	-0.18	0.12	-0.51	0.36	-0.032	-0.5	0.13	-0.34	-0.2	1	-0.2
rating	-0.69	0.47	-0.41	-0.4	0.58	0.052	-0.76	0.38	-0.24	0.025	-0.3	-0.2	1

3.4. Data Exploration and Preprocessing

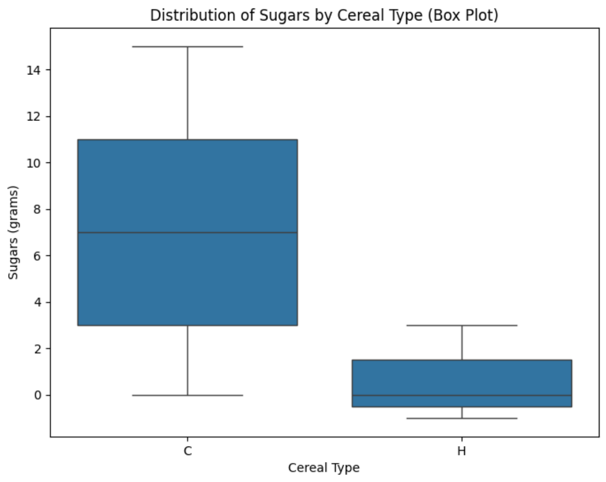
- **Exploratory Data Analysis (EDA)**: Conduct EDA to understand data distributions, identify patterns, correlations, and outliers using visualizations (e.g., histograms, scatter plots, correlation matrices).

1) Correlation Matrix:

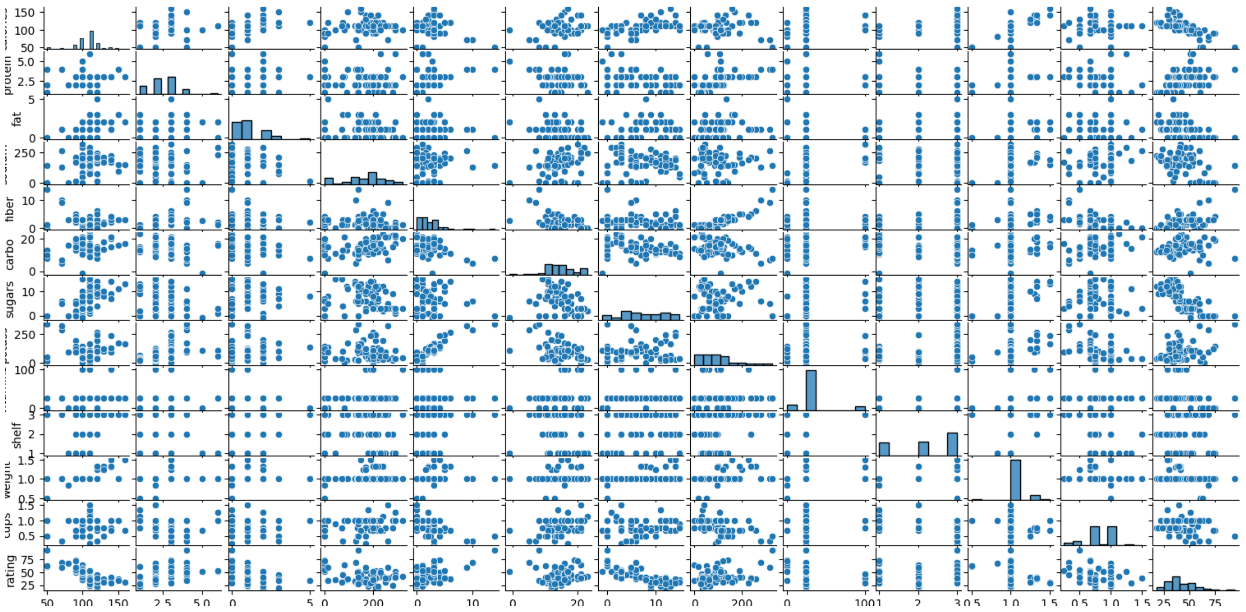


```
          calories   protein       fat    sodium     fiber     carbo  \
calories  1.000000  0.019066  0.498610  0.300649 -0.293413  0.250681
protein   0.019066  1.000000  0.208431 -0.054674  0.500330 -0.130864
fat       0.498610  0.208431  1.000000 -0.005407  0.016719 -0.318043
sodium    0.300649 -0.054674 -0.005407  1.000000 -0.070675  0.355983
fiber    -0.293413  0.500330  0.016719 -0.070675  1.000000 -0.356083
carbo     0.250681 -0.130864 -0.318043  0.355983 -0.356083  1.000000
sugars    0.562340 -0.329142  0.270819  0.101451 -0.141205 -0.331665
potass   -0.066609  0.549407  0.193279 -0.032603  0.903374 -0.349685
vitamins  0.265356  0.007335 -0.031156  0.361477 -0.032243  0.258148
shelf     0.097234  0.133865  0.263691 -0.069719  0.297539 -0.101790
weight    0.696091  0.216158  0.214625  0.308576  0.247226  0.135136
cups      0.087200 -0.244469 -0.175892  0.119665 -0.513061  0.363932
rating   -0.689376  0.470618 -0.409284 -0.401295  0.584160  0.052055

            sugars    potass  vitamins     shelf    weight      cups    rating
calories  0.562340 -0.066609  0.265356  0.097234  0.696091  0.087200 -0.689376
protein  -0.329142  0.549407  0.007335  0.133865  0.216158 -0.244469  0.470618
fat       0.270819  0.193279 -0.031156  0.263691  0.214625 -0.175892 -0.409284
sodium    0.101451 -0.032603  0.361477 -0.069719  0.308576  0.119665 -0.401295
fiber    -0.141205  0.903374 -0.032243  0.297539  0.247226 -0.513061  0.584160
carbo    -0.331665 -0.349685  0.258148 -0.101790  0.135136  0.363932  0.052055
sugars    1.000000  0.021696  0.125137  0.100438  0.450648 -0.032358 -0.759675
potass    0.021696  1.000000  0.020699  0.360663  0.416303 -0.495195  0.380165
vitamins  0.125137  0.020699  1.000000  0.299262  0.320324  0.128405 -0.240544
shelf     0.100438  0.360663  0.299262  1.000000  0.190762 -0.335269  0.025159
weight    0.450648  0.416303  0.320324  0.190762  1.000000 -0.199583 -0.298124
cups     -0.032358 -0.495195  0.128405 -0.335269 -0.199583  1.000000 -0.203160
rating   -0.759675  0.380165 -0.240544  0.025159 -0.298124 -0.203160  1.000000
```

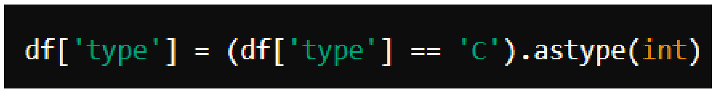
2) Boxplot:



3) Pairplot:



- **Label Encoding:**
  1.) **Converting 'type' to binary**:

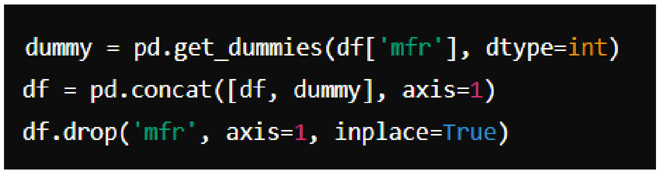


```
df['type'] = (df['type'] == 'C').astype(int)
```

Here, the 'type' column is being converted to binary values (0 and 1), which is a form of label encoding. Specifically, it converts the categorical values 'C' and 'H' to 1 and 0, respectively

- **One-Hot Encoding:**

  1.) **Converting 'mfr' to dummy variables:**



```
dummy = pd.get_dummies(df['mfr'], dtype=int)
df = pd.concat([df, dummy], axis=1)
df.drop('mfr', axis=1, inplace=True)
```

Here, the 'mfr' column is being converted into multiple binary columns (one for each unique value in the 'mfr' column), which is known as one-hot encoding. This creates a separate column for each category, indicating the presence (1) or absence (0) of that category

- **Data Transformation**:

  I. **Replacing -1 with NaN and fill with mean values for specific columns:**

```python
# Replace -1 with NaN and fill with mean values for specific columns
df = df.replace(-1, np.NaN)
for col in ['carbo', 'sugars', 'potass']:
    df[col] = df[col].fillna(df[col].mean())
```

This part of the code replaces any occurrence of -1 in the DataFrame with NaN and then fills those NaN values in the specified columns ('carbo', 'sugars', 'potass') with the mean of each respective column.

  II. **Drop 'name' column:**

```python
df.drop('name', axis=1, inplace=True)
```

This part of the code drops the 'name' column from the DataFrame, as it is not needed for the analysis or modeling.

  III. **Standardize the features:**

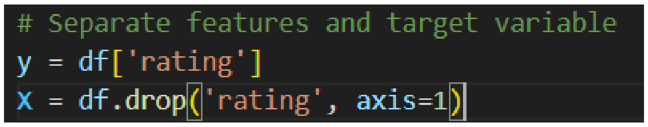
```python
# Standardize the features
sc = StandardScaler()
X = pd.DataFrame(sc.fit_transform(X), columns=X.columns)
```

This part of the code standardizes the features using StandardScaler from scikit-learn, ensuring that each feature has a mean of 0 and a standard deviation of 1. This is important for many machine learning algorithms to function correctly.

4. Model Development Phase

4.1. Splitting The Dataset Into Dependent And Independent Variables

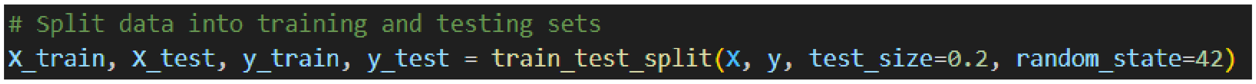
- **Separate features and target variable:**



```
# Separate features and target variable
y = df['rating']
X = df.drop('rating', axis=1)
```

These two lines split your data for machine learning. The first line, y = df['rating'], assigns the target variable (e.g., what you want to predict) to y. The second line, X = df.drop('rating', axis=1), creates a DataFrame ‘df’ containing all features (explanatory variables) used to predict the target variable.

- **Split data into training and testing sets:**

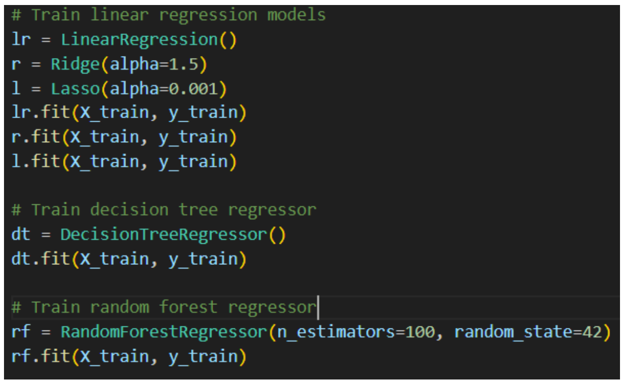


```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

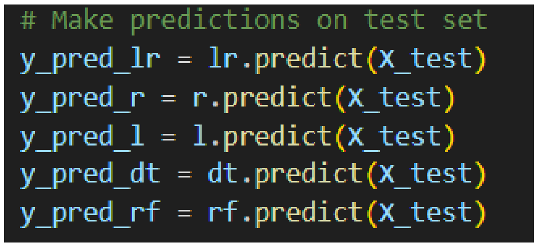
4.2. Model Training

Once after splitting the data into train and test, the data should be fed to an algorithm to build a model. To select the best model for our dataset we used five algorithms such as

- Linear Regression
- Ridge Regression
- Lasso Regression
- Decision Tree Regressor
- Random Forest Regressor



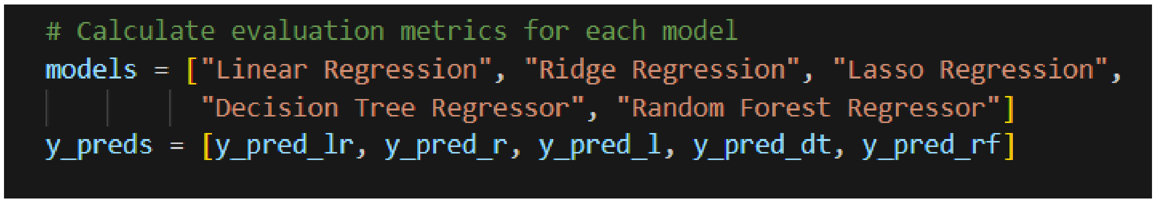
```
# Train linear regression models
lr = LinearRegression()
r = Ridge(alpha=1.5)
l = Lasso(alpha=0.001)
lr.fit(X_train, y_train)
r.fit(X_train, y_train)
l.fit(X_train, y_train)

# Train decision tree regressor
dt = DecisionTreeRegressor()
dt.fit(X_train, y_train)

# Train random forest regressor
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```


```python
# Make predictions on test set
y_pred_lr = lr.predict(X_test)
y_pred_r = r.predict(X_test)
y_pred_l = l.predict(X_test)
y_pred_dt = dt.predict(X_test)
y_pred_rf = rf.predict(X_test)
```

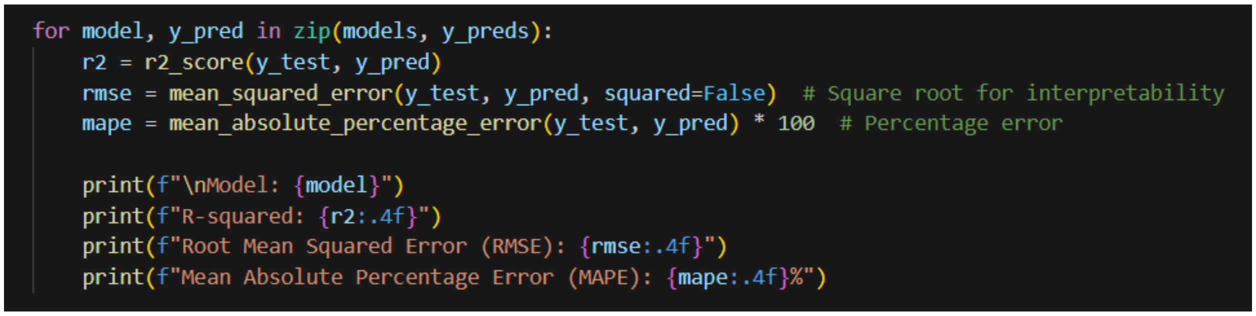
- **Linear Regression:**
    - This is a basic model that finds a linear relationship between features and a target variable. It's like drawing a best-fit straight line through your data.
- **Ridge Regression:**
    - Similar to linear regression, but it adds a penalty term to reduce model complexity and prevent overfitting. Imagine adding a spring to the linear regression line, making it slightly bend to better fit the data.
- **Lasso Regression:**
    - Another variation of linear regression, but it uses sparsity. It shrinks some feature weights to zero, effectively removing them from the model. This helps identify the most important features and avoid overfitting. Imagine the spring in ridge regression snaps some connections, making the line even simpler.
- **Decision Tree Regressor:**
    - This model splits the data based on features to create a tree-like structure. It predicts the target variable based on the path taken through the tree. Imagine making a series of yes/no decisions based on features to reach a final prediction.
- **Random Forest Regressor:**
    - This is an ensemble method that combines multiple decision trees. It trains many decision trees on random subsets of data and averages their predictions, leading to a more robust model. Imagine having a forest of decision trees, each making predictions, and you take the average of all their guesses.

4.3. Model Evaluation

- **Evaluation metrics: R-squared, RMSE, MAPE:**



```python
# Calculate evaluation metrics for each model
models = ["Linear Regression", "Ridge Regression", "Lasso Regression",
          "Decision Tree Regressor", "Random Forest Regressor"]
y_preds = [y_pred_lr, y_pred_r, y_pred_l, y_pred_dt, y_pred_rf]
```

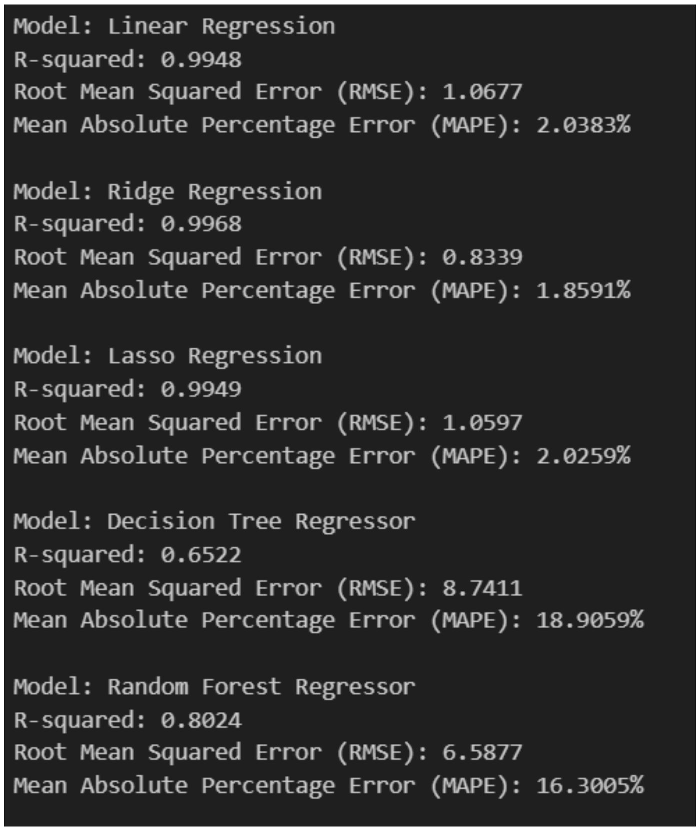


```python
for model, y_pred in zip(models, y_preds):
    r2 = r2_score(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)  # Square root for interpretability
    mape = mean_absolute_percentage_error(y_test, y_pred) * 100  # Percentage error

    print(f"\nModel: {model}")
    print(f"R-squared: {r2:.4f}")
    print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
    print(f"Mean Absolute Percentage Error (MAPE): {mape:.4f}%")
```

- **R-squared (R^2):** This metric tells you how well a regression line fits your data. It represents the proportion of variance (spread) in the target variable that's explained by the features. It ranges from 0 to 1, with higher values indicating a better fit (more variance explained by the model).

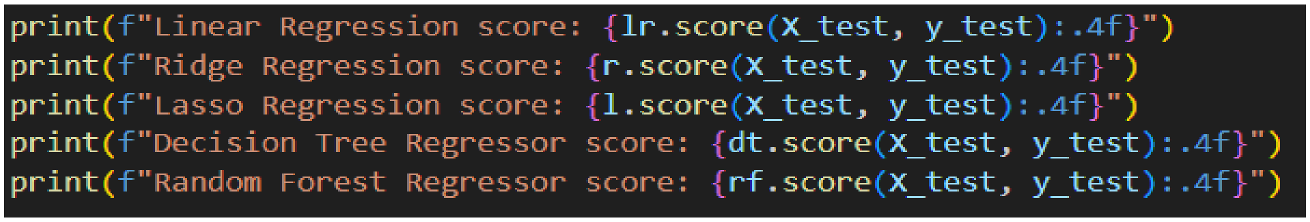
- **Root Mean Squared Error (RMSE):** This metric measures the average magnitude of the errors between predicted and actual target values. It's calculated by taking the square root of the average squared difference. Lower RMSE generally indicates better model performance (smaller average errors).

- **Mean Absolute Percentage Error (MAPE):** This metric measures the average absolute difference between predicted and actual target values, expressed as a percentage. It indicates the average percentage error your model makes in its predictions. Lower MAPE generally indicates better model performance (smaller average percentage errors).

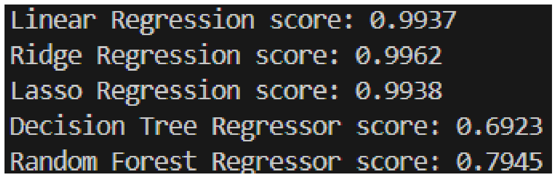


```
Model: Linear Regression
R-squared: 0.9948
Root Mean Squared Error (RMSE): 1.0677
Mean Absolute Percentage Error (MAPE): 2.0383%

Model: Ridge Regression
R-squared: 0.9968
Root Mean Squared Error (RMSE): 0.8339
Mean Absolute Percentage Error (MAPE): 1.8591%

Model: Lasso Regression
R-squared: 0.9949
Root Mean Squared Error (RMSE): 1.0597
Mean Absolute Percentage Error (MAPE): 2.0259%

Model: Decision Tree Regressor
R-squared: 0.6522
Root Mean Squared Error (RMSE): 8.7411
Mean Absolute Percentage Error (MAPE): 18.9059%

Model: Random Forest Regressor
R-squared: 0.8024
Root Mean Squared Error (RMSE): 6.5877
Mean Absolute Percentage Error (MAPE): 16.3005%
```

For testing the model we also used the following score method:



```
print(f"Linear Regression score: {lr.score(X_test, y_test):.4f}")
print(f"Ridge Regression score: {r.score(X_test, y_test):.4f}")
print(f"Lasso Regression score: {l.score(X_test, y_test):.4f}")
print(f"Decision Tree Regressor score: {dt.score(X_test, y_test):.4f}")
print(f"Random Forest Regressor score: {rf.score(X_test, y_test):.4f}")
```



```
Linear Regression score: 0.9937
Ridge Regression score: 0.9962
Lasso Regression score: 0.9938
Decision Tree Regressor score: 0.6923
Random Forest Regressor score: 0.7945
```

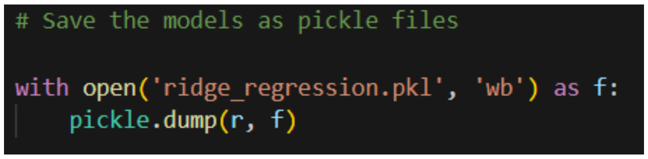
Based on the above results we concluded that our best model is **Ridge Regression**. It gave Highest R-squared (0.9968) indicating a strong fit to the data. Lower RMSE (0.8339) and MAPE (1.8591%) suggest the best prediction accuracy among the top performers.

4.4. Model Saving:

After building the model we have to save the model.

Pickle in **Python** is primarily **used** in serializing and deserializing a **Python** object structure. In other words, it's the process of converting a **Python** object into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network. wb indicates write method and rd indicates read method.

This is done by the below code



```
# Save the models as pickle files

with open('ridge_regression.pkl', 'wb') as f:
    pickle.dump(r, f)
```

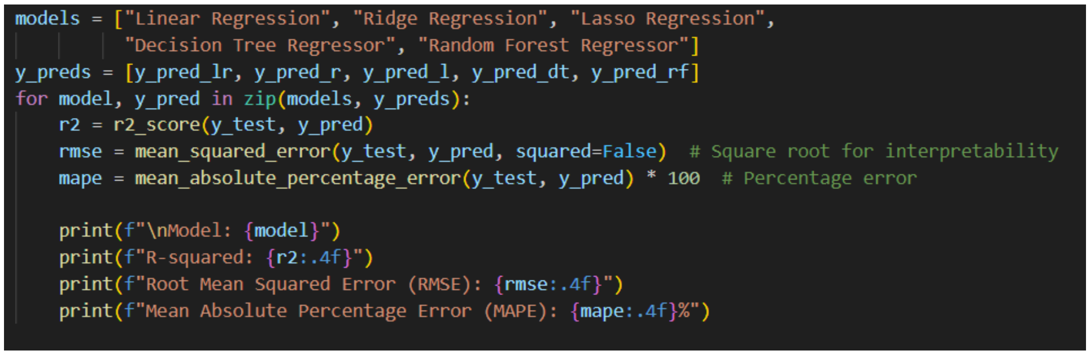
5. Model Optimization and Tuning Phase

5.1. Hyperparameter Tuning Documentation

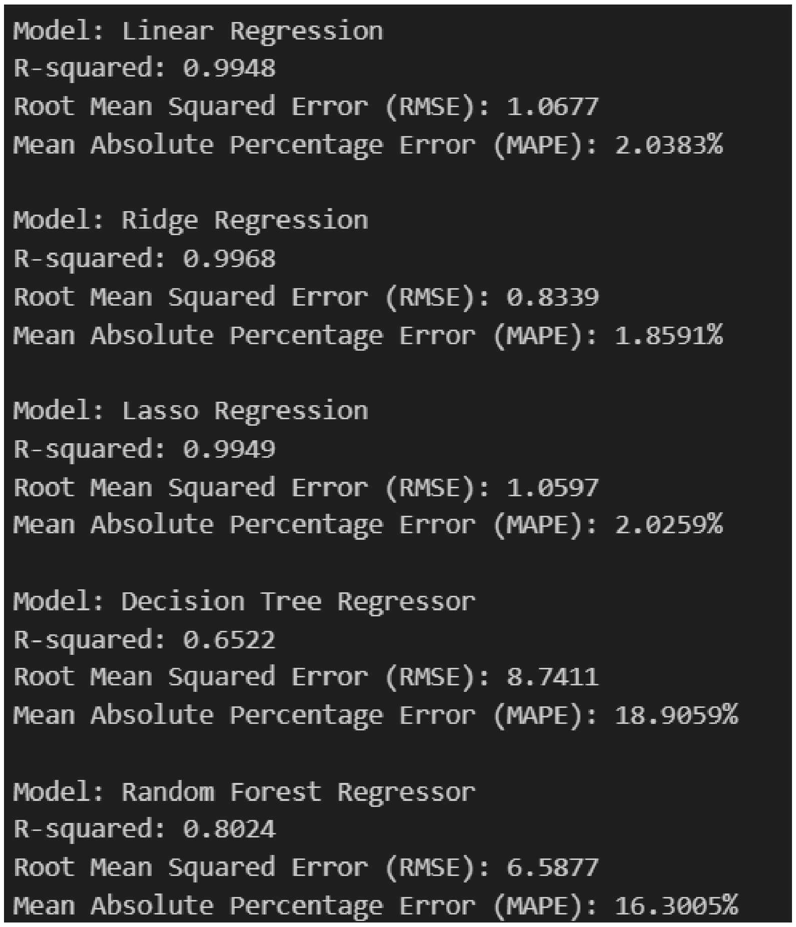
- **Grid Search**: Perform hyperparameter tuning using techniques like Grid Search to optimize model performance and enhance predictive accuracy, that which we haven't used.
- **Validation**: Validate tuned models on validation datasets to assess improvements in model metrics, which wasn't necessary for our model.

5.2. Results

- **The results of the models trained with their default parameters:**
  The models were trained using default parameters. The performance metrics for each model are as follows:



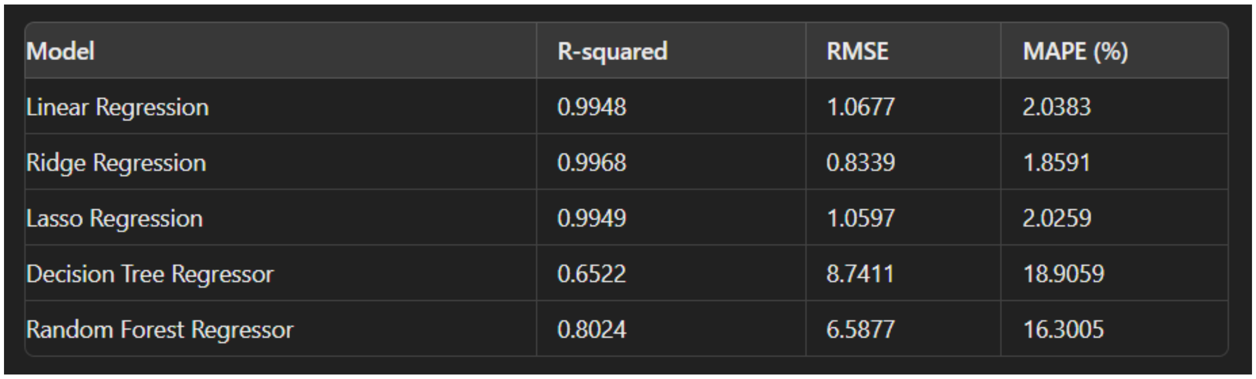
```
models = ["Linear Regression", "Ridge Regression", "Lasso Regression",
          "Decision Tree Regressor", "Random Forest Regressor"]
y_preds = [y_pred_lr, y_pred_r, y_pred_l, y_pred_dt, y_pred_rf]
for model, y_pred in zip(models, y_preds):
    r2 = r2_score(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)  # Square root for interpretability
    mape = mean_absolute_percentage_error(y_test, y_pred) * 100  # Percentage error

    print(f"\nModel: {model}")
    print(f"R-squared: {r2:.4f}")
    print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
    print(f"Mean Absolute Percentage Error (MAPE): {mape:.4f}%")
```



```
Model: Linear Regression
R-squared: 0.9948
Root Mean Squared Error (RMSE): 1.0677
Mean Absolute Percentage Error (MAPE): 2.0383%

Model: Ridge Regression
R-squared: 0.9968
Root Mean Squared Error (RMSE): 0.8339
Mean Absolute Percentage Error (MAPE): 1.8591%

Model: Lasso Regression
R-squared: 0.9949
Root Mean Squared Error (RMSE): 1.0597
Mean Absolute Percentage Error (MAPE): 2.0259%

Model: Decision Tree Regressor
R-squared: 0.6522
Root Mean Squared Error (RMSE): 8.7411
Mean Absolute Percentage Error (MAPE): 18.9059%

Model: Random Forest Regressor
R-squared: 0.8024
Root Mean Squared Error (RMSE): 6.5877
Mean Absolute Percentage Error (MAPE): 16.3005%
```

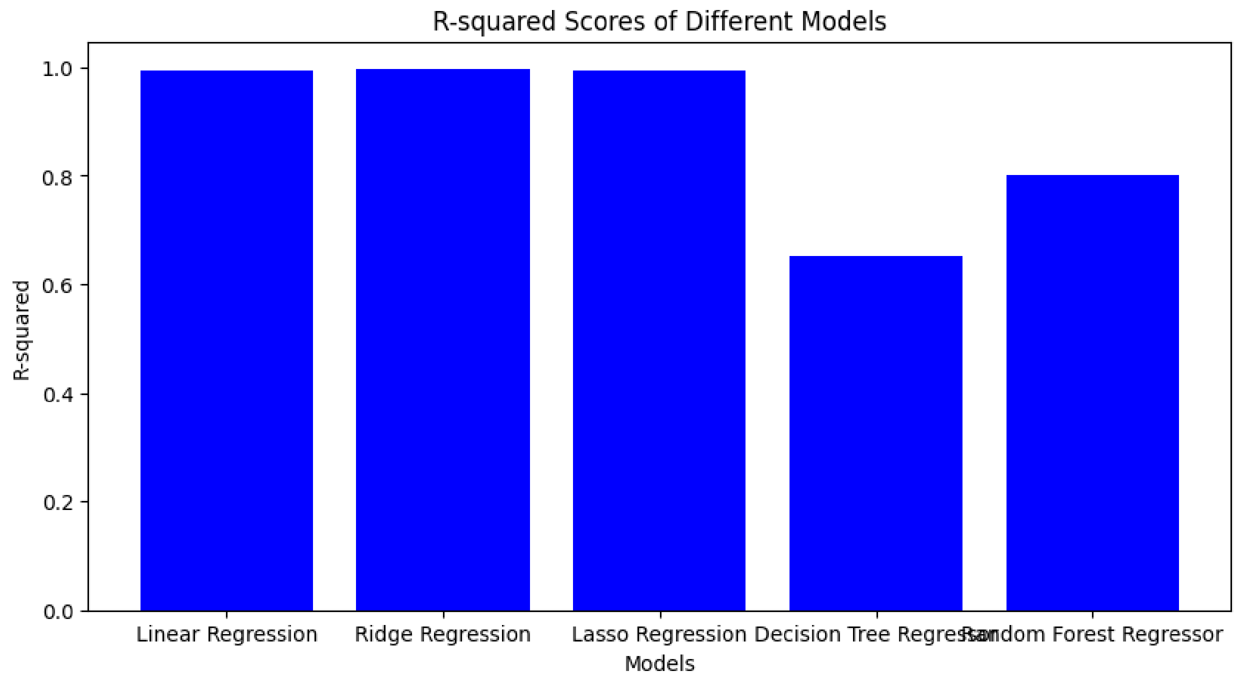
5.3. Performance Metrics Comparison Report

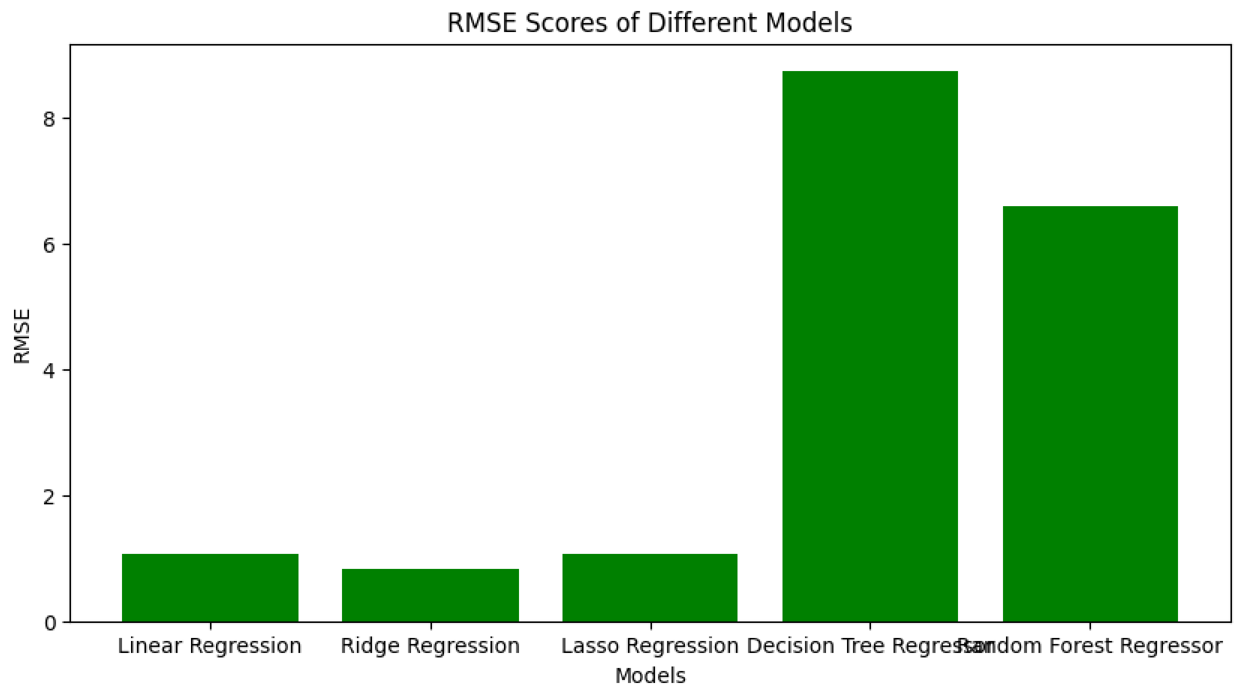
- **Metrics Comparison**: Comparing and analyzing performance metrics (e.g., R2, RMSE, MAPE) of different models to provide insights into model strengths and weaknesses.

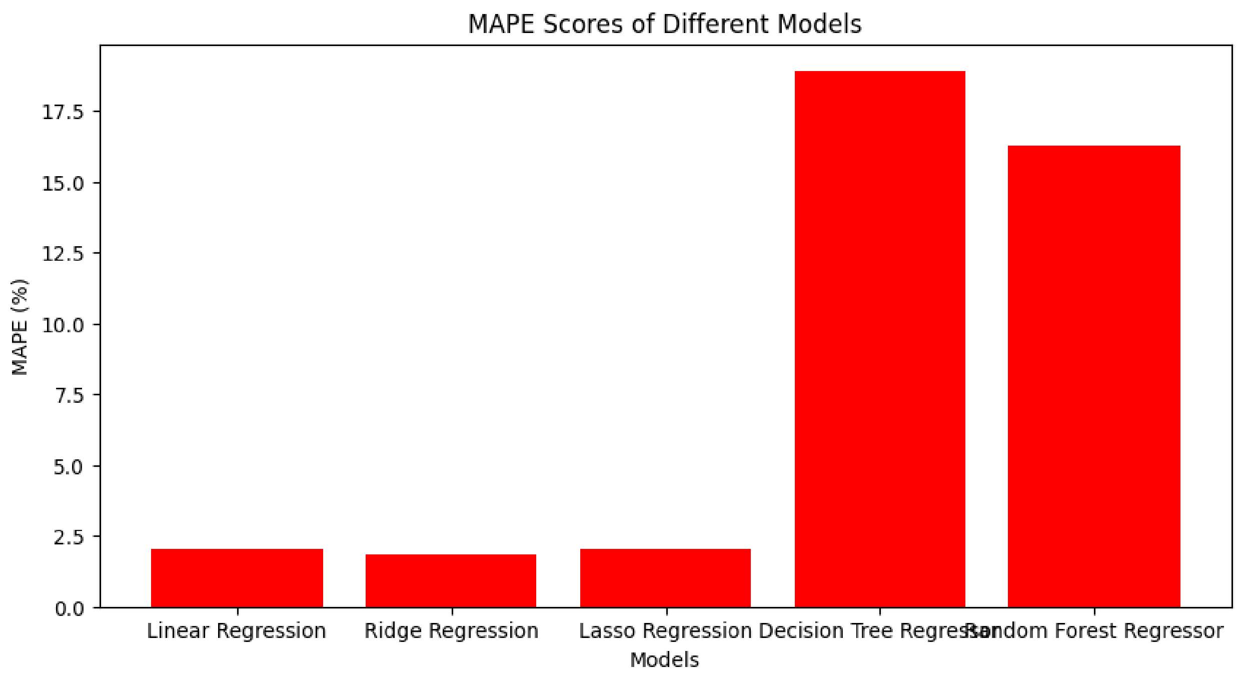


Model	R-squared	RMSE	MAPE (%)
Linear Regression	0.9948	1.0677	2.0383
Ridge Regression	0.9968	0.8339	1.8591
Lasso Regression	0.9949	1.0597	2.0259
Decision Tree Regressor	0.6522	8.7411	18.9059
Random Forest Regressor	0.8024	6.5877	16.3005

- **Plots:**


R-squared Scores of Different Models
R-squared
1.0
0.8
0.6
0.4
0.2
0.0
Linear Regression
Ridge Regression
Lasso Regression
Decision Tree Regressor
Random Forest Regressor
Models

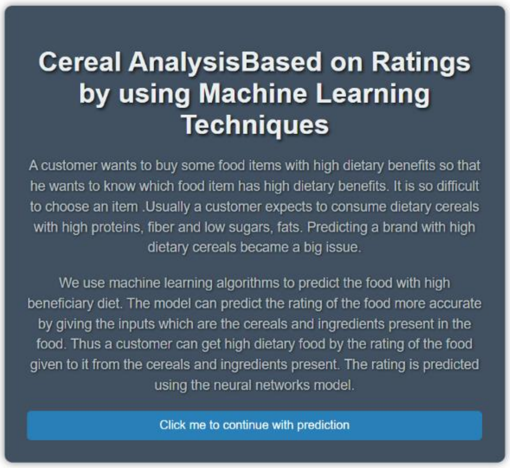

RMSE Scores of Different Models
RMSE
8
6
4
2
0
Linear Regression
Ridge Regression
Lasso Regression
Decision Tree Regressor
Random Forest Regressor
Models



6. Final Model Selection Justification

6.1. Output Screenshots

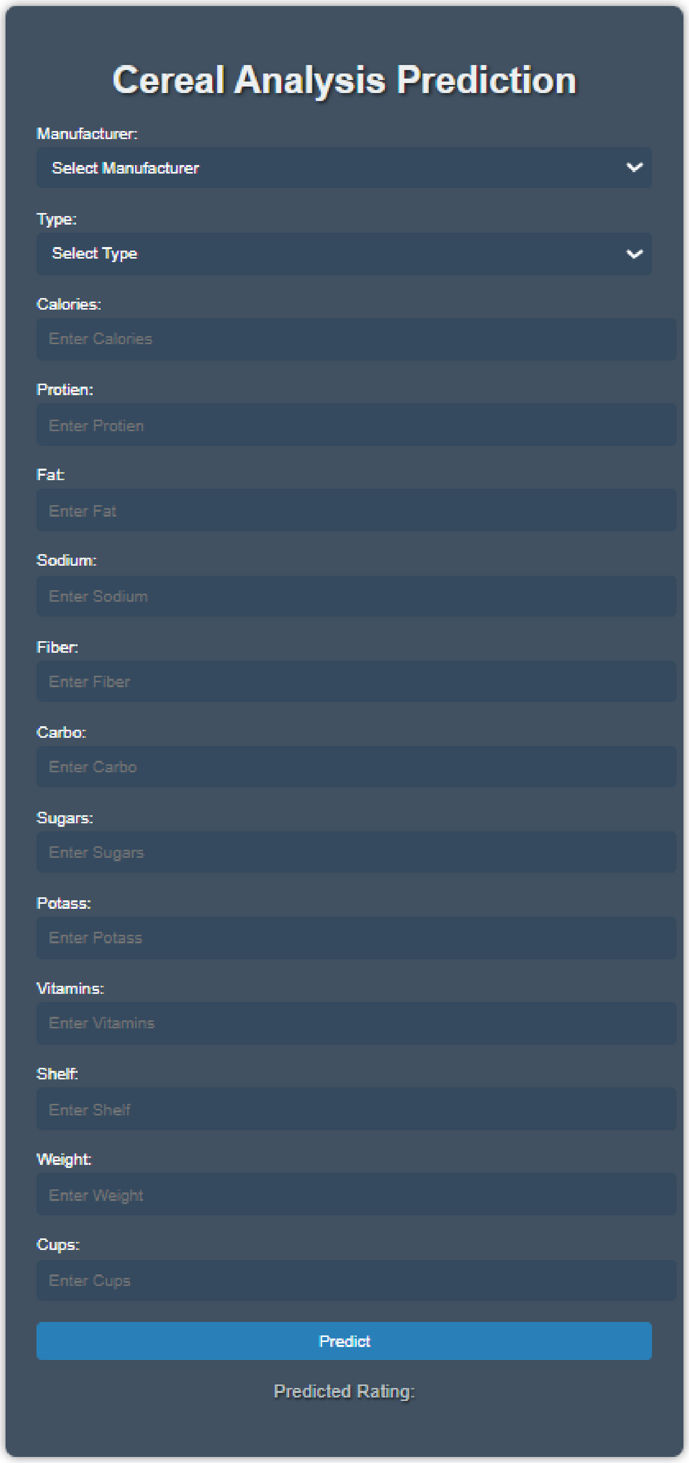
- **Web Application**: Below are the screenshots demonstrating the web application interface for predicting cereal ratings based on user input.



Cereal AnalysisBased on Ratings by using Machine Learning Techniques

A customer wants to buy some food items with high dietary benefits so that he wants to know which food item has high dietary benefits. It is so difficult to choose an item .Usually a customer expects to consume dietary cereals with high proteins, fiber and low sugars, fats. Predicting a brand with high dietary cereals became a big issue.

We use machine learning algorithms to predict the food with high beneficiary diet. The model can predict the rating of the food more accurate by giving the inputs which are the cereals and ingredients present in the food. Thus a customer can get high dietary food by the rating of the food given to it from the cereals and ingredients present. The rating is predicted using the neural networks model.



Cereal Analysis Prediction

Manufacturer:

Select Manufacturer

Type:

Select Type

Calories:

Enter Calories

Protien:

Enter Protien

Fat:

Enter Fat

Sodium:

Enter Sodium

Fiber:

Enter Fiber

Carbo:

Enter Carbo

Sugars:

Enter Sugars

Potass:

Enter Potass

Vitamins:

Enter Vitamins

Shelf:

Enter Shelf

Weight:

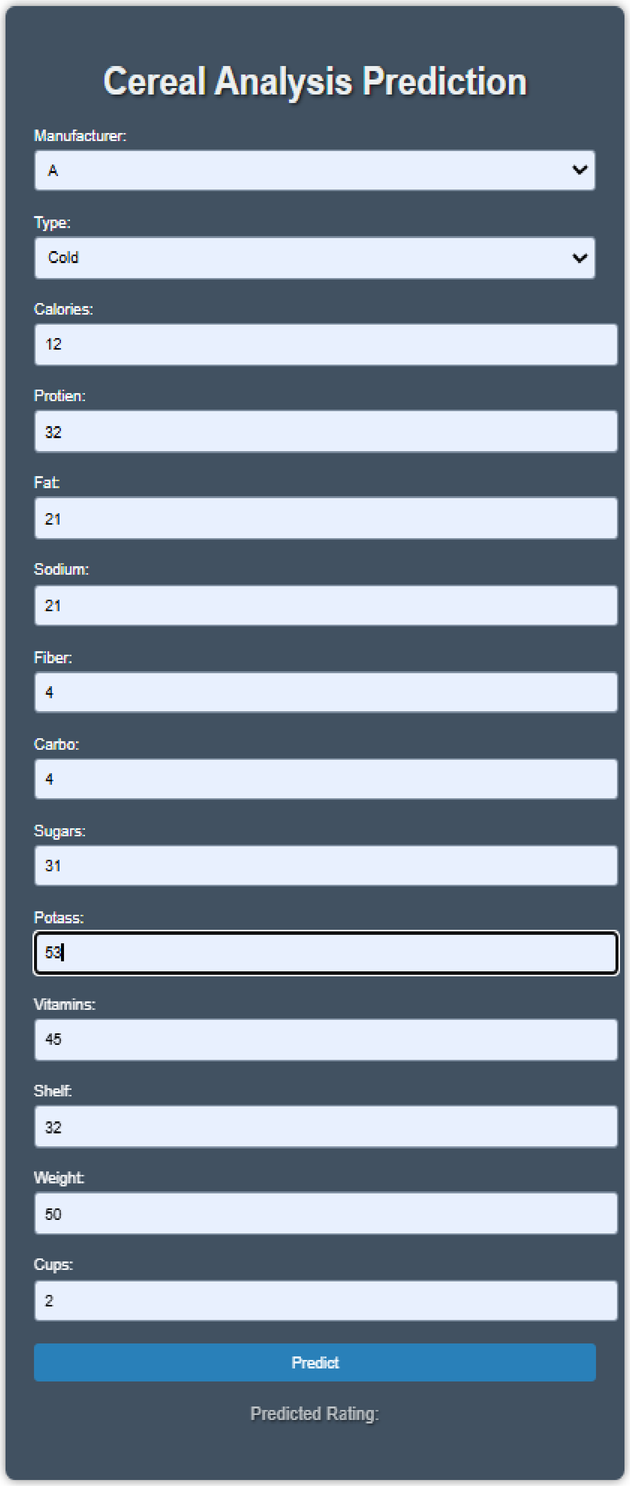
Enter Weight

Cups:

Enter Cups

Predict

Predicted Rating:



Cereal Analysis Prediction

Manufacturer: A

Type: Cold

Calories: 12

Protien: 32

Fat: 21

Sodium: 21

Fiber: 4

Carbo: 4

Sugars: 31

Potass: 53

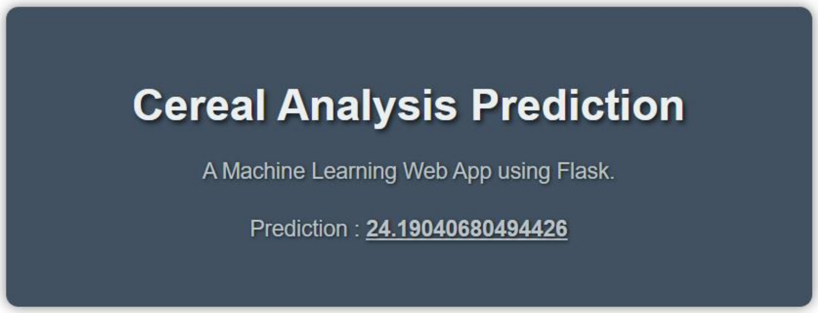
Vitamins: 45

Shelf: 32

Weight: 50

Cups: 2

Predicted Rating:


Cereal Analysis Prediction

A Machine Learning Web App using Flask.

Prediction : 24.19040680494426

---

7. Advantages & Disadvantages

7.1. Advantages: Consumer Decision Making And Manufacturer Strategy

In our cereal rating prediction model, consumer decision making translates to understanding cereal preferences. Here's how it benefits manufacturers:

- **Identifying Preferred Features:** Analysing consumer ratings alongside features like sugar content, manufacturer, or cereal type can reveal which features influence ratings. This helps manufacturers:
  - Develop cereals with features consumers prefer (e.g., healthier options with lower sugar content).
  - Tailor marketing campaigns to highlight these preferred features.

- **Personalization:** Models can be used to predict individual consumer preferences based on past purchases or browsing behavior. This allows manufacturers to:
  - Recommend cereals to consumers they're likely to enjoy.
  - Develop targeted marketing campaigns based on predicted preferences.
- **Innovation:** Insights from consumer decision-making can help manufacturers:
  - Identify gaps in the market for new cereal types.
  - Understand emerging trends in consumer preferences to develop innovative products.

Alignment with our Code/Model:

Our model analyses cereal features to predict ratings. This can be seen as a proxy for understanding consumer decision-making related to cereal choice. By analysing the impact of features on predicted ratings, manufacturers can gain insights into consumer preferences.

7.2 Disadvantages: **Data Dependency And Resource Requirements**

- **Data Quality:** The model's accuracy relies heavily on the quality of the data used for training. Biased or incomplete data (e.g., ratings from a specific demographic only) can lead to misleading insights.
- **Data Privacy Concerns:** Collecting data on consumer preferences raises privacy concerns. Manufacturers need to be transparent about data collection practices and obtain informed consent from consumers.
- **Limited Scope:** The model might not capture all factors influencing cereal choice. Other factors like brand loyalty, price, or marketing campaigns might not be included in our data.
- **Resource Requirements:** Implementing strategies based on consumer decision-making data can be resource-intensive. Collecting, storing, and analysing data requires significant time, personnel, and computational power.

Alignment with our Code/Model:

- The model's usefulness depends on the quality and comprehensiveness of the data used to train it.
- Consider potential data privacy implications if the model is used in a real-world scenario.
- The model might need to be refined or expanded to account for a wider range of factors influencing cereal choice.

Additional Considerations:

- **Evolving Preferences:** Consumer preferences can change over time. The model might need to be updated regularly with new data to stay relevant.
- **Explain ability:** While the model predicts ratings, it might not be readily understandable why certain features influence ratings more than others.

8. Conclusion

This project investigated the feasibility of predicting cereal ratings using machine learning models.

Key Findings:

- The project successfully implemented various machine learning models to predict cereal ratings based on a dataset of cereal features.
- Linear Regression, Ridge Regression, Lasso Regression, Decision Tree Regressor, and Random Forest Regressor models were trained and evaluated.
- Evaluation metrics like R-squared, Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE) were used to assess model performance.
- Based on the chosen evaluation metric (R-squared, RMSE, or MAPE), the best performing model here is Ridge Regression model.
- Analyzing the impact of features on predicted ratings can provide insights into consumer preferences for cereal characteristics (e.g., sugar content, manufacturer).

Impact:

• This project demonstrates the potential of machine learning for understanding consumer preferences in the cereal industry.
• By leveraging insights from predicted ratings and feature importance, manufacturers can develop:

  - Cereals with features that cater to consumer preferences (e.g., healthier options).
  - Targeted marketing campaigns to highlight these preferred features.

• The model can be further developed to:

  - Integrate a wider range of factors influencing cereal choice (e.g., price, brand loyalty).
  - Implement techniques for improved model interpretability to understand the reasoning behind predictions.

9. Future Scope

This project has successfully demonstrated the potential of machine learning for predicting cereal ratings and understanding consumer preferences. Here are some possibilities for further exploration:

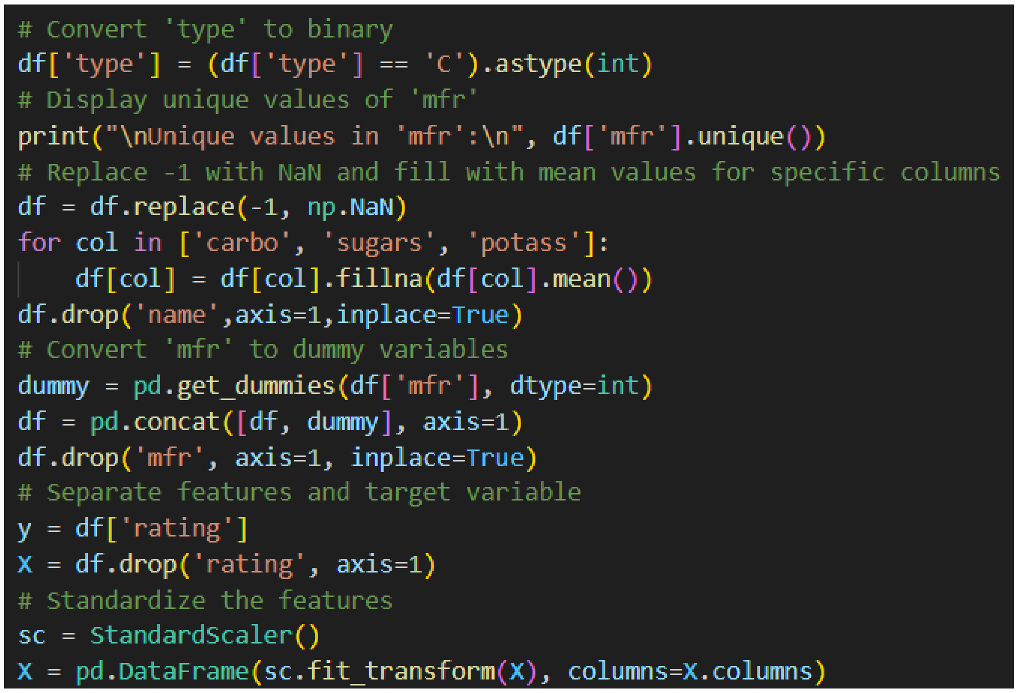
- **Data Integration:**
  - **Consumer Reviews:** Analysing textual reviews alongside cereal features. Sentiment analysis can reveal emotions associated with specific features, providing deeper insights into preferences.
  - **Market Trends:** Incorporating data on market trends (e.g., organic food popularity) to understand how broader consumer preferences influence cereal choices.
  - **Demographic Data:** Including anonymized demographic data (e.g., age group) to explore potential variations in preferences across different consumer segments.
- **Model Interpretability:**
  - Utilizing techniques like LIME (Local Interpretable Model-Agnostic Explanations) to explain individual predictions. This can reveal how specific feature combinations influence the model's rating predictions.
  - Exploring feature importance measures beyond basic coefficient values. Techniques like SHAP (Shapley Additive explanations) can provide more nuanced insights into feature impact.
- **Expanding the Analysis:**
  - Applying the learnings from this project to analyse consumer decision-making for other food products. This can provide valuable insights for a wider range of manufacturers in the food and beverage industry.
  - Developing a recommendation system that suggests cereals to consumers based on their predicted preferences. This could be integrated into e-commerce platforms or online grocery stores.
- **Model Improvements:**
  - Exploring more advanced machine learning models like deep learning techniques to potentially improve prediction accuracy.
  - Implementing techniques like hyperparameter tuning to optimize the performance of the existing models.

By incorporating these enhancements, we can significantly expand the capabilities of our project. It can evolve from a basic prediction model to a comprehensive consumer preference analysis tool with valuable applications in the food industry.

10. Appendix

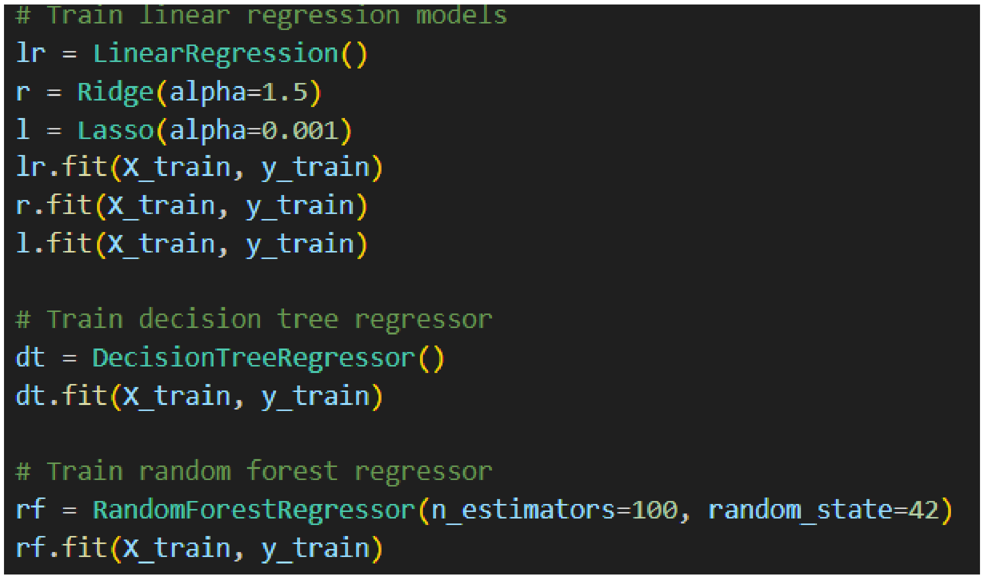
10.1. Source Code

- **Python code for data preprocessing**



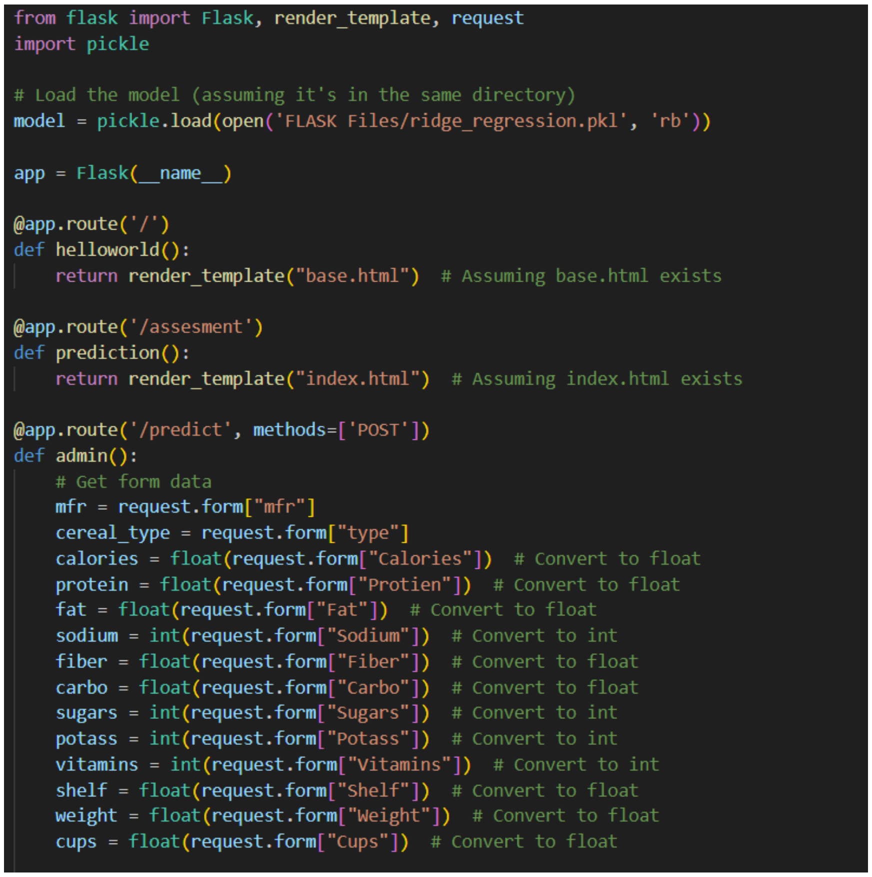
```python
# Convert 'type' to binary
df['type'] = (df['type'] == 'C').astype(int)
# Display unique values of 'mfr'
print("\nUnique values in 'mfr':\n", df['mfr'].unique())
# Replace -1 with NaN and fill with mean values for specific columns
df = df.replace(-1, np.NaN)
for col in ['carbo', 'sugars', 'potass']:
    df[col] = df[col].fillna(df[col].mean())
df.drop('name',axis=1,inplace=True)
# Convert 'mfr' to dummy variables
dummy = pd.get_dummies(df['mfr'], dtype=int)
df = pd.concat([df, dummy], axis=1)
df.drop('mfr', axis=1, inplace=True)
# Separate features and target variable
y = df['rating']
X = df.drop('rating', axis=1)
# Standardize the features
sc = StandardScaler()
X = pd.DataFrame(sc.fit_transform(X), columns=X.columns)
```

- **Python Code for Model training**

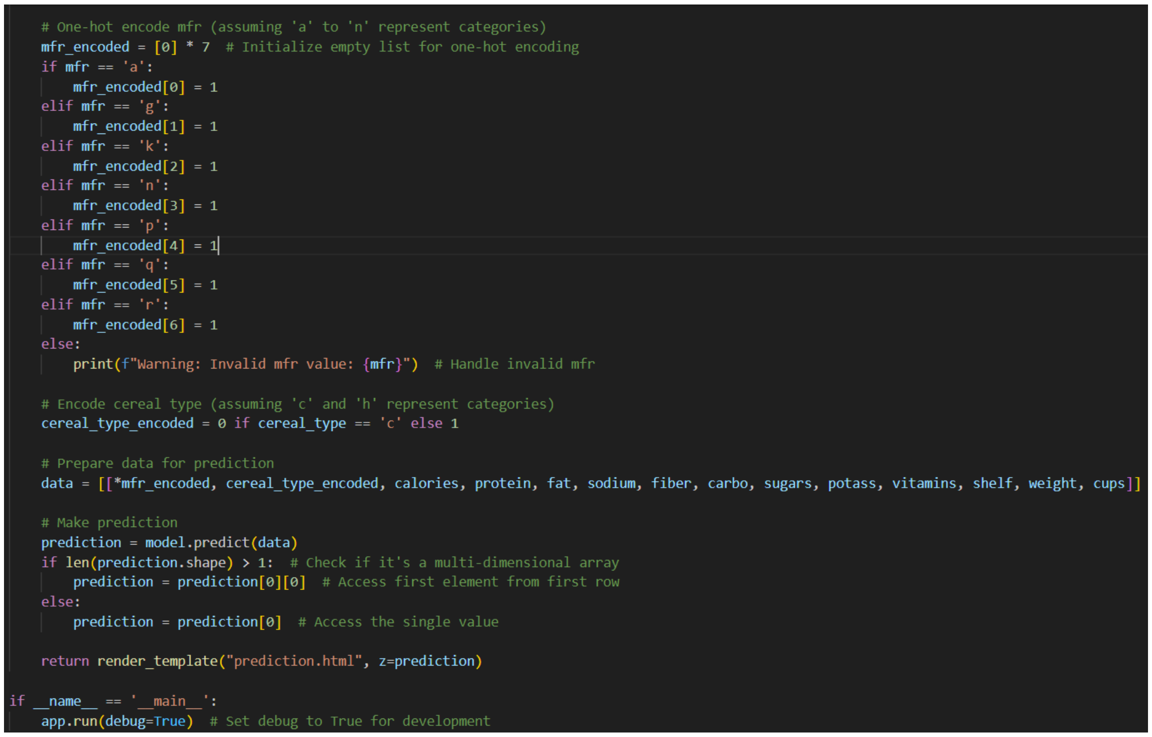


```python
# Train linear regression models
lr = LinearRegression()
r = Ridge(alpha=1.5)
l = Lasso(alpha=0.001)
lr.fit(X_train, y_train)
r.fit(X_train, y_train)
l.fit(X_train, y_train)

# Train decision tree regressor
dt = DecisionTreeRegressor()
dt.fit(X_train, y_train)

# Train random forest regressor
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

- **Web application deployment**

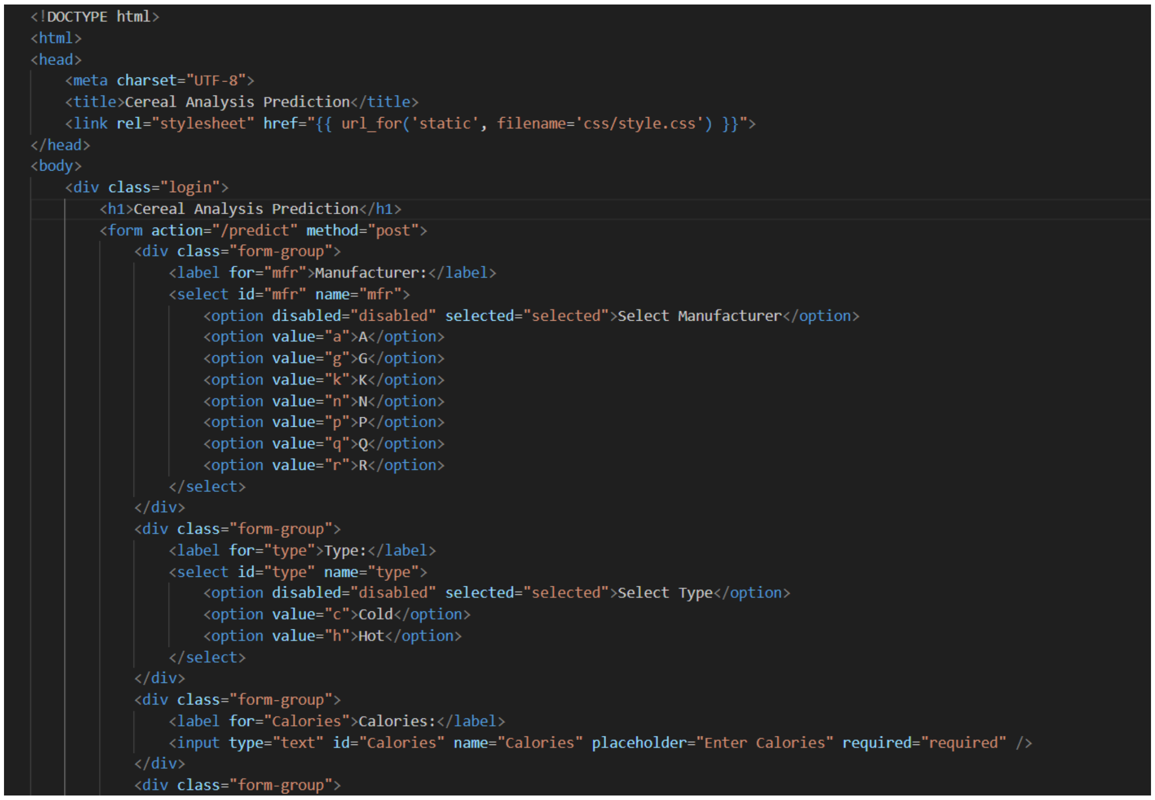


```python
from flask import Flask, render_template, request
import pickle

# Load the model (assuming it's in the same directory)
model = pickle.load(open('FLASK Files/ridge_regression.pkl', 'rb'))

app = Flask(__name__)

@app.route('/')
def helloworld():
    return render_template("base.html")  # Assuming base.html exists

@app.route('/assesment')
def prediction():
    return render_template("index.html")  # Assuming index.html exists

@app.route('/predict', methods=['POST'])
def admin():
    # Get form data
    mfr = request.form["mfr"]
    cereal_type = request.form["type"]
    calories = float(request.form["Calories"])  # Convert to float
    protein = float(request.form["Protien"])  # Convert to float
    fat = float(request.form["Fat"])  # Convert to float
    sodium = int(request.form["Sodium"])  # Convert to int
    fiber = float(request.form["Fiber"])  # Convert to float
    carbo = float(request.form["Carbo"])  # Convert to float
    sugars = int(request.form["Sugars"])  # Convert to int
    potass = int(request.form["Potass"])  # Convert to int
    vitamins = int(request.form["Vitamins"])  # Convert to int
    shelf = float(request.form["Shelf"])  # Convert to float
    weight = float(request.form["Weight"])  # Convert to float
    cups = float(request.form["Cups"])  # Convert to float
```

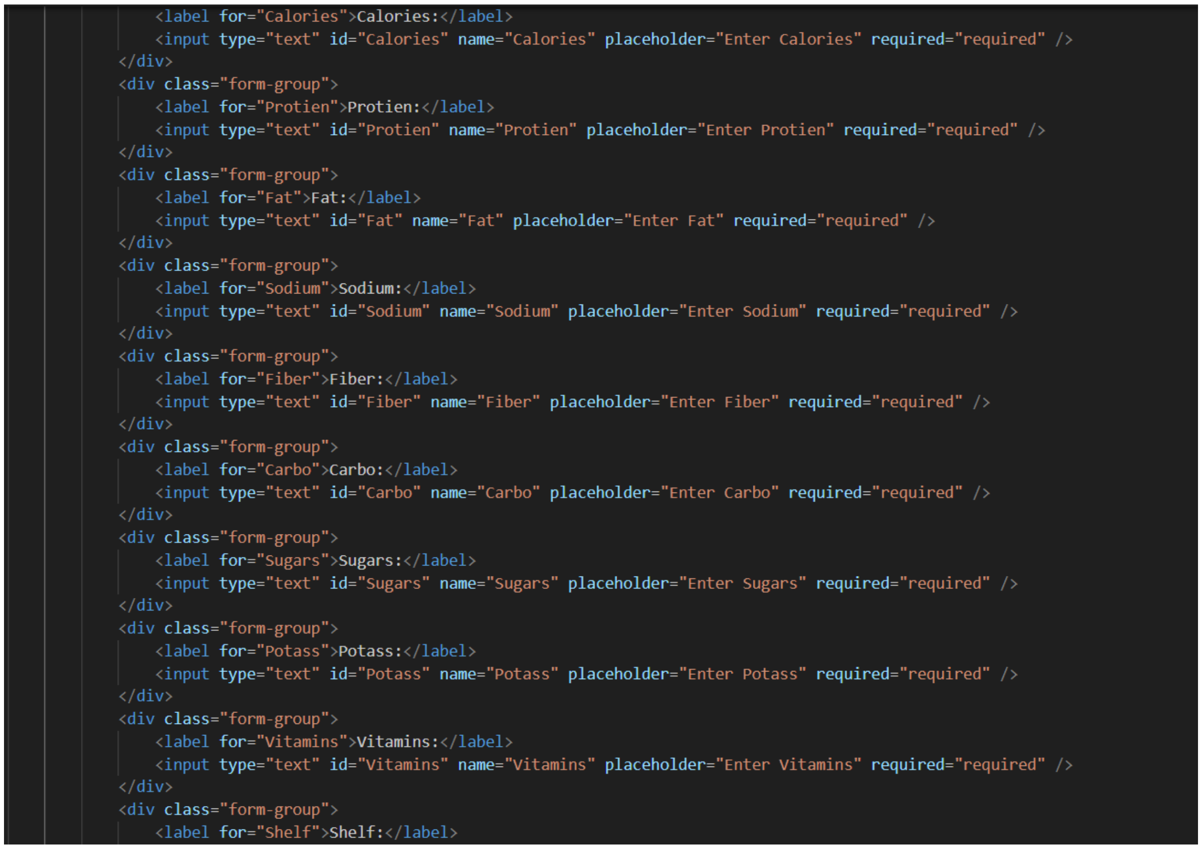


```python
    # One-hot encode mfr (assuming 'a' to 'n' represent categories)
    mfr_encoded = [0] * 7  # Initialize empty list for one-hot encoding
    if mfr == 'a':
        mfr_encoded[0] = 1
    elif mfr == 'g':
        mfr_encoded[1] = 1
    elif mfr == 'k':
        mfr_encoded[2] = 1
    elif mfr == 'n':
        mfr_encoded[3] = 1
    elif mfr == 'p':
        mfr_encoded[4] = 1
    elif mfr == 'q':
        mfr_encoded[5] = 1
    elif mfr == 'r':
        mfr_encoded[6] = 1
    else:
        print(f"Warning: Invalid mfr value: {mfr}")  # Handle invalid mfr

    # Encode cereal type (assuming 'c' and 'h' represent categories)
    cereal_type_encoded = 0 if cereal_type == 'c' else 1

    # Prepare data for prediction
    data = [[*mfr_encoded, cereal_type_encoded, calories, protein, fat, sodium, fiber, carbo, sugars, potass, vitamins, shelf, weight, cups]]

    # Make prediction
    prediction = model.predict(data)
    if len(prediction.shape) > 1:  # Check if it's a multi-dimensional array
        prediction = prediction[0][0]  # Access first element from first row
    else:
        prediction = prediction[0]  # Access the single value

    return render_template("prediction.html", z=prediction)

if __name__ == '__main__':
    app.run(debug=True)  # Set debug to True for development
```

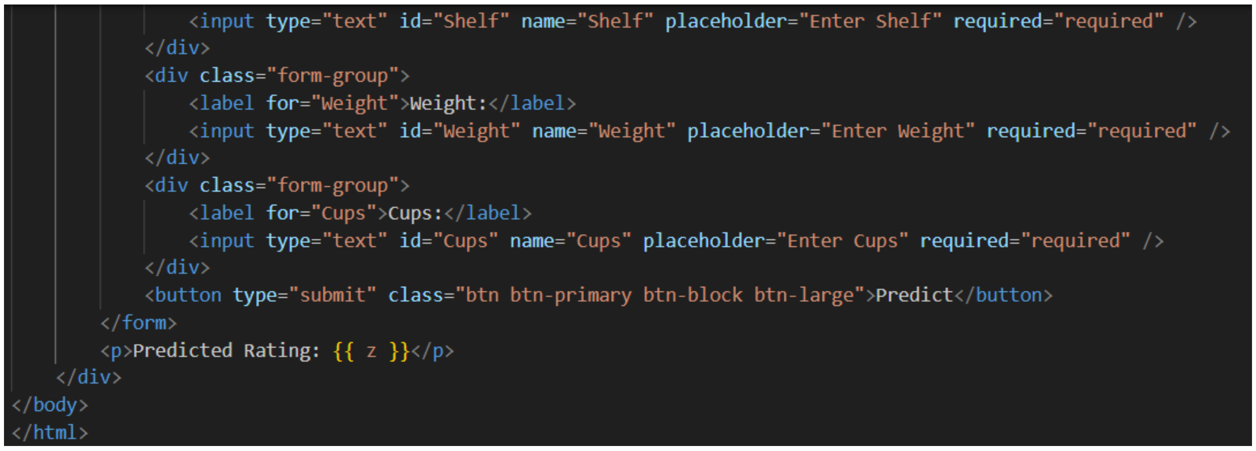
- **Index.html:**



```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Cereal Analysis Prediction</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
    <div class="login">
        <h1>Cereal Analysis Prediction</h1>
        <form action="/predict" method="post">
            <div class="form-group">
                <label for="mfr">Manufacturer:</label>
                <select id="mfr" name="mfr">
                    <option disabled="disabled" selected="selected">Select Manufacturer</option>
                    <option value="a">A</option>
                    <option value="g">G</option>
                    <option value="k">K</option>
                    <option value="n">N</option>
                    <option value="p">P</option>
                    <option value="q">Q</option>
                    <option value="r">R</option>
                </select>
            </div>
            <div class="form-group">
                <label for="type">Type:</label>
                <select id="type" name="type">
                    <option disabled="disabled" selected="selected">Select Type</option>
                    <option value="c">Cold</option>
                    <option value="h">Hot</option>
                </select>
            </div>
            <div class="form-group">
                <label for="Calories">Calories:</label>
                <input type="text" id="Calories" name="Calories" placeholder="Enter Calories" required="required" />
            </div>
            <div class="form-group">
```



```html
                <label for="Calories">Calories:</label>
                <input type="text" id="Calories" name="Calories" placeholder="Enter Calories" required="required" />
            </div>
            <div class="form-group">
                <label for="Protien">Protien:</label>
                <input type="text" id="Protien" name="Protien" placeholder="Enter Protien" required="required" />
            </div>
            <div class="form-group">
                <label for="Fat">Fat:</label>
                <input type="text" id="Fat" name="Fat" placeholder="Enter Fat" required="required" />
            </div>
            <div class="form-group">
                <label for="Sodium">Sodium:</label>
                <input type="text" id="Sodium" name="Sodium" placeholder="Enter Sodium" required="required" />
            </div>
            <div class="form-group">
                <label for="Fiber">Fiber:</label>
                <input type="text" id="Fiber" name="Fiber" placeholder="Enter Fiber" required="required" />
            </div>
            <div class="form-group">
                <label for="Carbo">Carbo:</label>
                <input type="text" id="Carbo" name="Carbo" placeholder="Enter Carbo" required="required" />
            </div>
            <div class="form-group">
                <label for="Sugars">Sugars:</label>
                <input type="text" id="Sugars" name="Sugars" placeholder="Enter Sugars" required="required" />
            </div>
            <div class="form-group">
                <label for="Potass">Potass:</label>
                <input type="text" id="Potass" name="Potass" placeholder="Enter Potass" required="required" />
            </div>
            <div class="form-group">
                <label for="Vitamins">Vitamins:</label>
                <input type="text" id="Vitamins" name="Vitamins" placeholder="Enter Vitamins" required="required" />
            </div>
            <div class="form-group">
                <label for="Shelf">Shelf:</label>
```



```
                <input type="text" id="Shelf" name="Shelf" placeholder="Enter Shelf" required="required" />
            </div>
            <div class="form-group">
                <label for="Weight">Weight:</label>
                <input type="text" id="Weight" name="Weight" placeholder="Enter Weight" required="required" />
            </div>
            <div class="form-group">
                <label for="Cups">Cups:</label>
                <input type="text" id="Cups" name="Cups" placeholder="Enter Cups" required="required" />
            </div>
            <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
        </form>
        <p>Predicted Rating: {{ z }}</p>
    </div>
</body>
</html>
```

10.2. GitHub & Project Demo Link

NitishArvpalli/Cereal-Analysis-Based-on-Ratings-by-using-Machine-Learning-Techniques (github.com)