

Final Project Report

1. Introduction
 - 1.1. Project overviews
 - 1.2. Objectives
2. Project Initialization and Planning Phase
 - 2.1. Define Problem Statement
 - 2.2. Project Proposal (Proposed Solution)
 - 2.3. Initial Project Planning
3. Data Collection and Preprocessing Phase
 - 3.1. Data Collection Plan and Raw Data Sources Identified
 - 3.2. Data Quality Report
 - 3.3. Data Exploration and Preprocessing
4. Model Development Phase
 - 4.1. Feature Selection Report
 - 4.2. Model Selection Report
 - 4.3. Initial Model Training Code, Model Validation and Evaluation Report
5. Model Optimization and Tuning Phase
 - 5.1. Hyperparameter Tuning Documentation
 - 5.2. Performance Metrics Comparison Report
 - 5.3. Final Model Selection Justification
6. Results
 - 6.1. Output Screenshots
7. Advantages & Disadvantages
8. Conclusion
9. Future Scope
10. Appendix
 - 10.1. Source Cod3
 - 10.2. GitHub & Project Demo Link

Cereal Analysis Based on Ratings Using Machine Learning

1.Introduction

1.1 Project overviews

The primary goal of this project is to analyze cereals based on customer ratings and various nutritional factors using machine learning techniques. The analysis aims to provide insights into which cereals are most favorable among consumers and to develop predictive models that can help in recommending cereals based on individual preferences.

. The "Cereal Analysis Based on Ratings Using Machine Learning Techniques" project aims to leverage advanced machine learning algorithms to analyze and predict the ratings of various cereals. By examining a comprehensive dataset of cereal attributes and their corresponding ratings, this project seeks to uncover patterns and insights that can help consumers make informed decisions and manufacturers improve their products.

1.2 Objectives

The primary objective of this project is to develop an ML model that can accurately predict cereal quality other relevant meteorological variables.

Key goals include improving prediction accuracy compared to traditional models, identifying significant patterns and correlations in different

data, and creating a userfriendly tool for stakeholders in agriculture. The ultimate aim is to enable these stakeholders to make informed decisions that can mitigate the impacts of cerealsrelated events, optimize agricultural practices. The project also seeks to advance the field of cereal prediction by demonstrating the potential of ML techniques in capturing the complexities of customer mindset. By achieving these objectives, the project aims to enhance consumer knowledge regarding cereal choices, promote healthier eating habits, and leverage machine learning for practical applications in nutrition and consumer preferences. Gather a comprehensive dataset containing various cereals, including their nutritional information, customer ratings, and reviews from multiple sources

2. Project Initialization and Planning Phase

2.1 Define Problem Statements (Customer Problem Statement Template):

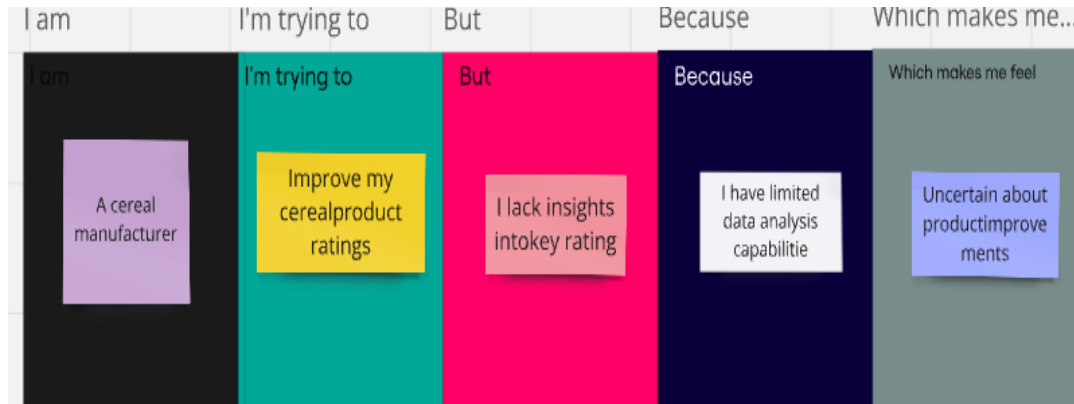
In this project, machine learning techniques are employed to analyze cereal ratings and provide insights for consumers and manufacturers. The dataset consists of various attributes such as nutritional content, brand, and consumer ratings for different cereal products. The goal is to build predictive models that can classify cereals based on their ratings and identify key factors that influence consumer preferences. By leveraging machine learning algorithms, such as classification and clustering, the project aims to uncover patterns in the data to assist consumers in making informed choices and help manufacturers in product development and marketing strategies.

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	A cereal manufacturer	Improve my cereal product ratings	I lack insights into key rating	I have limited data analysis capabilities	Uncertain about product improvements

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes Me feel
PS-2	A health-conscious consumer	Choose the healthiest cereal option	I'm overwhelmed by nutritional data	I don't know which factors are most important	Confused and frustrated

Define Problem Statements (Customer Problem Statement Template)

Example:



2.2 Project Proposal (Proposed Solution) template

This project proposal outlines a solution to address a specific problem. With a clear objective, defined scope, and a concise problem statement, the proposed solution details the approach, key features, and resource requirements, including hardware, software, and personnel.

Project Overview	
Objective	Predict cereal ratings based on nutritional and categorical features using machine learning techniques
Scope	Analyze and model cereal data to provide accurate predictions and insights for manufacturers and consumers
Problem Statement	
Description	Develop a machine learning pipeline that preprocesses data, selects features, trains models, and evaluates performance
Impact	Enhance decision-making in product development and marketing strategies by understanding factors influencing cereal ratings
Proposed Solution	
Approach	Utilize data preprocessing, feature selection, model training, and evaluation techniques to build an accurate predictive model

Key Features	Data cleaning, feature encoding, model selection, hyperparameter tuning, performance metrics, and visualization tools
--------------	---

Resource Requirements

Resource Type	Description	Specification/Allocation
Hardware		
Computing Resources	CPU/GPU specifications, number of cores	T4 GPU
Memory	RAM specifications	8 GB
Storage	Disk space for data, models, and logs	1 TB SSD
Software		
Frameworks	Python frameworks	Flask
Libraries	Additional libraries	scikit-learn, pandas, numpy, matplotlib, seaborn
Development Environment	IDE, version control	Jupyter Notebook, vscode, Git
Data		
Data	Source, size, format	Kaggle dataset, 614, csv UCI dataset, 690csv, Meteorological departments

2.3 Initial Project Planning Template

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
Sprint-1	Data Collection and Preprocessing	USN -1	Collect and clean cereal data.	Medium	Akash and Tharun	23/09/24	26/09/24
Sprint-1	Data Collection and Preprocessing	USN -2	Explore dataset for key features.	Medium	Akash	23/09/24	26/09/24
Sprint-1	Data Collection and Preprocessing	USN -3	Handle missing values and outliers.	Medium	Sravya and Bhavani	23/09/24	26/09/24
Sprint-2	Model Development	USN -4	Train machine learning models to predict ratings.	High	Akash, Tharun and Sravya	27/09/24	30/09/24
Sprint-2	Model tuning and testing	USN -5	Evaluate models and pick the best one.	High	Akash, Tharun and Bhavani	27/09/24	30/09/24
Sprint-3	Model tuning and testing	USN -6	Model tuning	High	Sravya and Tharun	27/09/24	30/09/24
Sprint-3	Model tuning and testing	USN -7	Model testing	Medium	Akash	27/09/24	30/09/24

Sprint-5	Web integration and Deployment	USN-8	Building HTML templates	Medium	Akash, Sravya and Tharun	01/10/24	5/10/24
Sprint-5	Web integration and Deployment	USN-9	Local deployment	High	Akash, Tharun and Bhavani	01/10/24	5/10/24
Sprint-5	Project Report	USN-10	Report	Medium	Sravya	6/10/24	10/10/24

3. Data Collection and Preprocessing Phase

3.1 Data Collection Plan & Raw Data Sources Identification Template

Elevate your data strategy with the Data Collection plan and the Raw Data Sources report, ensuring meticulous data curation and integrity for informed decision-making in every analysis and decision-making endeavor.

Data Collection Plan Template

Section	Description
Project Overview	The project aims to predict cereal ratings based on nutritional information and reviews using machine learning techniques.
Data Collection Plan	Data will be collected from public datasets available on Kaggle, consumer review platforms, and the USDA nutritional database.
Raw Data Sources Identified	The raw data sources include the cereal dataset from Kaggle, e-commerce reviews, and USDA nutritional data.

Raw Data Sources Template

Source Name	Description	Location/URL	Format	Size	Access Permissions
Kaggle Cereal Dataset	A comprehensive dataset of cereals including nutritional information and ratings.	https://www.kaggle.com/crawford/80-cereals	CSV	18 KB	Public
Amazon Reviews	Consumer reviews and ratings of various cereal brands from Amazon.	https://www.kaggle.com/datafiniti/consumer-reviews-of-amazon-products	CSV	350 MB	Public (with Kaggle account)
USDA Nutritional Data	Detailed nutritional information from the USDA database.	https://fdc.nal.usda.gov/download-datasets.html	CSV	Varies	Public

Data Collection and Preprocessing Phase

3.2 Data Quality Report Template

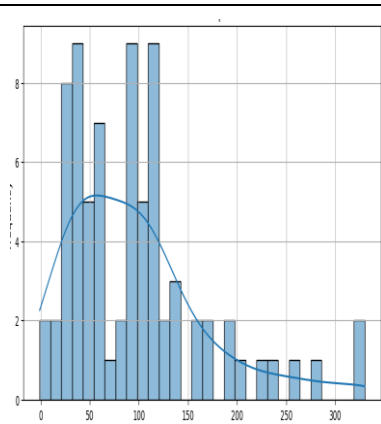
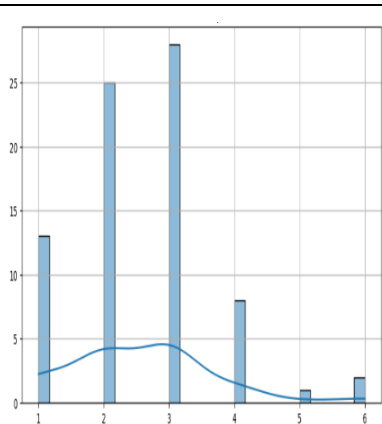
The Data Quality Report Template will summarize data quality issues from the selected source, including severity levels and resolution plans. It will aid in systematically identifying and rectifying data discrepancies.

Data Source	Data Quality Issue	Severity	Resolution Plan
Dataset	Mention the issues faced in the selected dataset.	Low/ Moderate/ High	Give the solution for that issue technically.
Cereal Dataset	Negative values in <code>carbo</code> , <code>sugars</code> , and <code>potass</code>	High	Remove or impute negative values with appropriate statistical measures (e.g., median or mean).
Cereal Dataset	Presence of categorical variables (<code>mfr</code> , <code>type</code>)	Moderate	Encode categorical variables using techniques like one-hot encoding.
Cereal Dataset	Outliers in <code>potass</code> and <code>fiber</code>	Moderate	Use techniques like z-score or IQR to detect and handle outliers appropriately.
Cereal Dataset	Inconsistent scaling of features	Low	Normalize or standardize features to ensure they are on a similar scale.
Cereal Dataset	Missing unique identifier for cereals	Low	Ensure that the <code>name</code> column is used as a unique identifier or create a new unique identifier if needed.

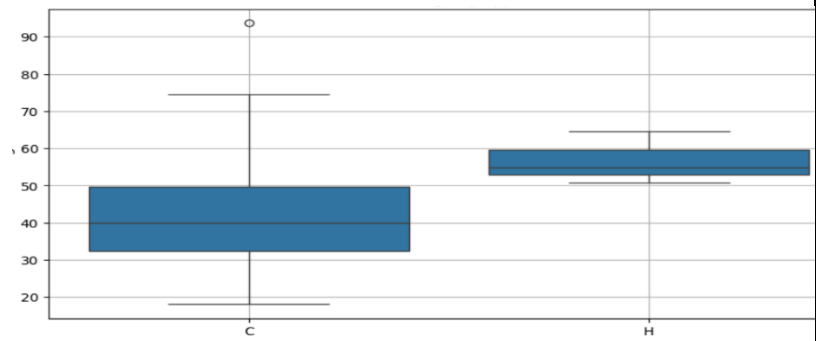
Data Collection and Preprocessing Phase

3.3 Data Exploration and Preprocessing Template

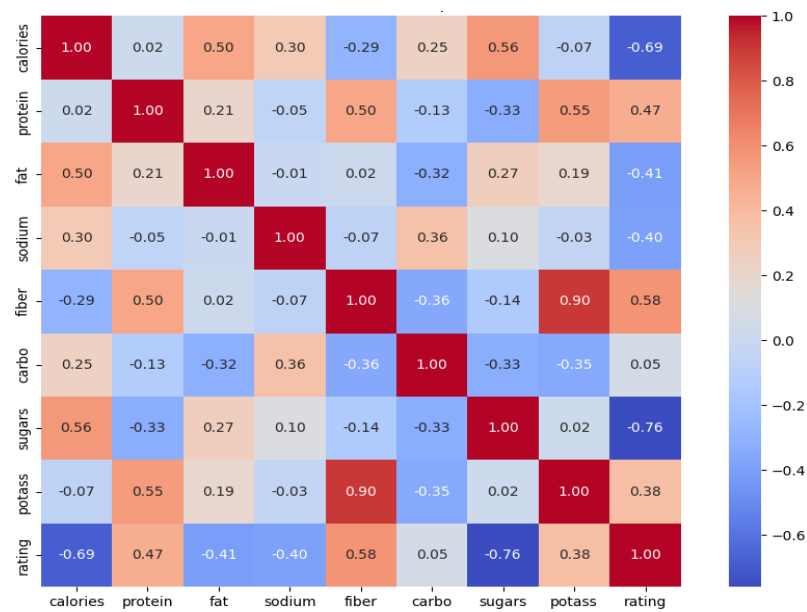
Identifies data sources, assesses quality issues like missing values and duplicates, and implements resolution plans to ensure accurate and reliable analysis.

Section	Description																																																																																																																														
Data Overview	<table><thead><tr><th></th><th>calories</th><th>protein</th><th>fat</th><th>sodium</th><th>fiber</th><th>carbo</th><th>sugars</th><th>potass</th><th>vitamins</th><th>shelf</th><th>weight</th><th>cups</th><th>rating</th></tr></thead><tbody><tr><td>count</td><td>77.000000</td><td>77.000000</td><td>77.000000</td><td>77.000000</td><td>77.000000</td><td>77.000000</td><td>77.000000</td><td>77.000000</td><td>77.000000</td><td>77.000000</td><td>77.000000</td><td>77.000000</td><td>77.000000</td></tr><tr><td>mean</td><td>106.883117</td><td>2.545455</td><td>1.012987</td><td>159.675325</td><td>2.151948</td><td>14.597403</td><td>6.922078</td><td>96.077922</td><td>28.246753</td><td>2.207792</td><td>1.029610</td><td>0.821039</td><td>42.665700</td></tr><tr><td>std</td><td>19.484119</td><td>1.094790</td><td>1.006473</td><td>83.832295</td><td>2.383364</td><td>4.278956</td><td>4.444885</td><td>71.286813</td><td>22.342523</td><td>0.832524</td><td>0.150477</td><td>0.232716</td><td>14.047280</td></tr><tr><td>min</td><td>50.000000</td><td>1.000000</td><td>0.000000</td><td>0.000000</td><td>0.000000</td><td>-1.000000</td><td>-1.000000</td><td>-1.000000</td><td>0.000000</td><td>1.000000</td><td>0.500000</td><td>0.250000</td><td>18.042850</td></tr><tr><td>25%</td><td>100.000000</td><td>2.000000</td><td>0.000000</td><td>130.000000</td><td>1.000000</td><td>12.000000</td><td>3.000000</td><td>40.000000</td><td>25.000000</td><td>1.000000</td><td>1.000000</td><td>0.670000</td><td>33.174090</td></tr><tr><td>50%</td><td>110.000000</td><td>3.000000</td><td>1.000000</td><td>180.000000</td><td>2.000000</td><td>14.000000</td><td>7.000000</td><td>90.000000</td><td>25.000000</td><td>2.000000</td><td>1.000000</td><td>0.750000</td><td>40.400200</td></tr><tr><td>75%</td><td>110.000000</td><td>3.000000</td><td>2.000000</td><td>210.000000</td><td>3.000000</td><td>17.000000</td><td>11.000000</td><td>120.000000</td><td>25.000000</td><td>3.000000</td><td>1.000000</td><td>1.000000</td><td>50.828390</td></tr><tr><td>max</td><td>160.000000</td><td>6.000000</td><td>5.000000</td><td>320.000000</td><td>14.000000</td><td>23.000000</td><td>15.000000</td><td>330.000000</td><td>100.000000</td><td>3.000000</td><td>1.500000</td><td>1.500000</td><td>93.704910</td></tr></tbody></table>		calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating	count	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	mean	106.883117	2.545455	1.012987	159.675325	2.151948	14.597403	6.922078	96.077922	28.246753	2.207792	1.029610	0.821039	42.665700	std	19.484119	1.094790	1.006473	83.832295	2.383364	4.278956	4.444885	71.286813	22.342523	0.832524	0.150477	0.232716	14.047280	min	50.000000	1.000000	0.000000	0.000000	0.000000	-1.000000	-1.000000	-1.000000	0.000000	1.000000	0.500000	0.250000	18.042850	25%	100.000000	2.000000	0.000000	130.000000	1.000000	12.000000	3.000000	40.000000	25.000000	1.000000	1.000000	0.670000	33.174090	50%	110.000000	3.000000	1.000000	180.000000	2.000000	14.000000	7.000000	90.000000	25.000000	2.000000	1.000000	0.750000	40.400200	75%	110.000000	3.000000	2.000000	210.000000	3.000000	17.000000	11.000000	120.000000	25.000000	3.000000	1.000000	1.000000	50.828390	max	160.000000	6.000000	5.000000	320.000000	14.000000	23.000000	15.000000	330.000000	100.000000	3.000000	1.500000	1.500000	93.704910
		calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating																																																																																																																	
	count	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000																																																																																																																	
	mean	106.883117	2.545455	1.012987	159.675325	2.151948	14.597403	6.922078	96.077922	28.246753	2.207792	1.029610	0.821039	42.665700																																																																																																																	
	std	19.484119	1.094790	1.006473	83.832295	2.383364	4.278956	4.444885	71.286813	22.342523	0.832524	0.150477	0.232716	14.047280																																																																																																																	
	min	50.000000	1.000000	0.000000	0.000000	0.000000	-1.000000	-1.000000	-1.000000	0.000000	1.000000	0.500000	0.250000	18.042850																																																																																																																	
	25%	100.000000	2.000000	0.000000	130.000000	1.000000	12.000000	3.000000	40.000000	25.000000	1.000000	1.000000	0.670000	33.174090																																																																																																																	
	50%	110.000000	3.000000	1.000000	180.000000	2.000000	14.000000	7.000000	90.000000	25.000000	2.000000	1.000000	0.750000	40.400200																																																																																																																	
	75%	110.000000	3.000000	2.000000	210.000000	3.000000	17.000000	11.000000	120.000000	25.000000	3.000000	1.000000	1.000000	50.828390																																																																																																																	
	max	160.000000	6.000000	5.000000	320.000000	14.000000	23.000000	15.000000	330.000000	100.000000	3.000000	1.500000	1.500000	93.704910																																																																																																																	
Univariate Analysis	<div></div>																																																																																																																														

Bivariate Analysis



Multivariate Analysis



Loading Data

	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
count	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000	77.000000
mean	106.883117	2.545455	1.012987	159.675325	2.151948	14.587403	6.922078	96.077922	28.246753	2.207792	1.029610	0.821039	42.665705
std	19.484119	1.094790	1.006473	83.832295	2.383364	4.278956	4.444885	71.286813	22.342523	0.832524	0.150477	0.232716	14.047289
min	50.000000	1.000000	0.000000	0.000000	0.000000	-1.000000	-1.000000	-1.000000	0.000000	1.000000	0.500000	0.250000	18.042851
25%	100.000000	2.000000	0.000000	130.000000	1.000000	12.000000	3.000000	40.000000	25.000000	1.000000	1.000000	0.670000	33.174094
50%	110.000000	3.000000	1.000000	180.000000	2.000000	14.000000	7.000000	90.000000	25.000000	2.000000	1.000000	0.750000	40.400208
75%	110.000000	3.000000	2.000000	210.000000	3.000000	17.000000	11.000000	120.000000	25.000000	3.000000	1.000000	1.000000	50.828382
max	160.000000	6.000000	5.000000	320.000000	14.000000	23.000000	15.000000	330.000000	100.000000	3.000000	1.500000	1.500000	93.704912

Outliers and Anomalies

-

Data Preprocessing Code Screenshots

Handling Missing Data

Identifying missing values

```

name      False
mfr       False
type      False
calories  False
protein   False
fat       False
sodium    False
fiber     False
carbo     False
sugars    False
potass    False
vitamins  False
shelf     False
weight    False
cups      False
rating    False
  
```

dtype: bool

Handling missing values

Data Encoding	<p>Encoding</p> <pre>from sklearn.preprocessing import LabelEncoder le = LabelEncoder() data["mfr"] = le.fit_transform(data["mfr"]) data["type"] = le.fit_transform(data["type"])</pre> <p>OneHotEncoder</p> <pre>from sklearn.preprocessing import OneHotEncoder one = OneHotEncoder() a = one.fit_transform(x[:,0:1]).toarray() x = np.delete(x,[0],axis=1) x=np.concatenate((a,x),axis=1)</pre>
Data Transformation	<pre>from sklearn.preprocessing import StandardScaler # Normalize numerical features scaler = StandardScaler() numerical_features = data.select_dtypes(include=['int64', 'float64']).columns data[numerical_features] = scaler.fit_transform(data[numerical_features])</pre>
Feature Engineering	<pre># One-hot encode categorical variables data = pd.get_dummies(data, columns=['mfr', 'type'], drop_first=True)</pre>
Save Processed Data	<pre># Save the cleaned and processed data for future use data.to_csv('processed_cereal.csv', index=False)</pre>

4 Model Development Phase Template

4.1 Feature Selection Report Template

In the forthcoming update, each feature will be accompanied by a brief description. Users will indicate whether it's selected or not, providing reasoning for their decision. This process will streamline decision-making and enhance transparency in feature selection.

Feature	Description	Selected (Yes/No)	Reasoning
calories	Number of calories per serving	Yes	Nutritional information is critical for predicting the healthiness and rating of cereal.
protein	Grams of protein per serving	Yes	Protein content is an important nutritional aspect influencing cereal ratings.
fat	Grams of fat per serving	Yes	Fat content impacts the health perception and rating of cereals.
sodium	Milligrams of sodium per serving	Yes	Sodium content affects healthiness and taste, influencing the ratings.

fiber	Grams of dietary fiber per serving	Yes	Fiber is a key nutritional factor, especially in health-conscious consumers.
carbo	Grams of carbohydrates per serving	Yes	Carbohydrate content is important for energy levels and overall nutrition.
sugars	Grams of sugars per serving	Yes	Sugar content is crucial for taste and health perception, directly affecting ratings.
potass	Milligrams of potassium per serving	Yes	Potassium is an essential nutrient that can impact health ratings.
vitamins	Vitamins and minerals per serving (percent of daily requirements)	Yes	Vitamin content enhances the nutritional profile, affecting cereal ratings.
shelf	Display shelf position in stores	Yes	Shelf position can influence sales and perceived quality of cereals.
weight	Weight of one serving in ounces	Yes	Serving weight impacts portion size and consumer perception.
cups	Number of cups per serving	Yes	Portion size is essential for understanding the serving size and consumer intake.
type	Type of cereal (hot or cold)	Yes	The type of cereal affects its use case and consumer preference.

Model Development Phase Template

4.2 Model Selection Report

In the forthcoming Model Selection Report, various models will be outlined, detailing their descriptions, hyperparameters, and performance metrics, including Accuracy or F1 Score. This comprehensive report will provide insights into the chosen models and their effectiveness.

Model	Description	Hyperparameters	Performance Metric (e.g., Accuracy, F1 Score)
Linear Regression	A simple linear model to predict cereal ratings.	Default	R ² : 0.9590, RMSE: 0.2991, MAPE: 1.0911%
Ridge Regression	Linear model with L2 regularization to prevent overfitting.	alpha=1.5	R ² : 0.9590, RMSE: 0.2992, MAPE: 1.0913%
Lasso Regression	Linear model with L1 regularization to prevent overfitting.	alpha=0.001	R ² : 0.9573, RMSE: 0.3086, MAPE: 1.1332%
Decision Tree Regressor	Non-linear model using decision trees to predict ratings.	Default	R ² : 0.8872, RMSE: 0.6155, MAPE: 2.3072%
Random Forest Regressor	Ensemble model using multiple decision trees for predictions.	n_estimators=100, random_state=42	R ² : 0.9504, RMSE: 0.3424, MAPE: 1.2388%

Model Development Phase Template

4.3 Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include classification reports, accuracy, and confusion matrices for multiple models, presented through respective screenshots.

Initial Model Training Code:

1. Linear Regression:

```
# Linear Regression
y_pred_lr = lr.predict(X_test)
r2_lr = r2_score(y_test, y_pred_lr)
rmse_lr = mean_squared_error(y_test, y_pred_lr, squared=False) # Square root for interpretability
mape_lr = mean_absolute_percentage_error(y_test, y_pred_lr) * 100 # Percentage error

print("\nModel: Linear Regression")
print(f"R-squared: {r2_lr:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse_lr:.4f}")
print(f"Mean Absolute Percentage Error (MAPE): {mape_lr:.4f}%")
```

2. Lasso Regression:

```
# Ridge Regression
y_pred_r = r.predict(X_test)
r2_r = r2_score(y_test, y_pred_r)
rmse_r = mean_squared_error(y_test, y_pred_r, squared=False) # Square root for interpretability
mape_r = mean_absolute_percentage_error(y_test, y_pred_r) * 100 # Percentage error

print("\nModel: Ridge Regression")
print(f"R-squared: {r2_r:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse_r:.4f}")
print(f"Mean Absolute Percentage Error (MAPE): {mape_r:.4f}%")
```

3. Ridge Regression:

```
# Ridge Regression
y_pred_r = r.predict(X_test)
r2_r = r2_score(y_test, y_pred_r)
rmse_r = mean_squared_error(y_test, y_pred_r, squared=False) # Square root for interpretability
mape_r = mean_absolute_percentage_error(y_test, y_pred_r) * 100 # Percentage error

print("\nModel: Ridge Regression")
print(f"R-squared: {r2_r:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse_r:.4f}")
print(f"Mean Absolute Percentage Error (MAPE): {mape_r:.4f}%")
```

4. Decision Tree Regressor:

```
# Decision Tree Regressor
y_pred_dt = dt.predict(X_test)
r2_dt = r2_score(y_test, y_pred_dt)
rmse_dt = mean_squared_error(y_test, y_pred_dt, squared=False) # Square root for interpretability
mape_dt = mean_absolute_percentage_error(y_test, y_pred_dt) * 100 # Percentage error

print("\nModel: Decision Tree Regressor")
print(f"R-squared: {r2_dt:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse_dt:.4f}")
print(f"Mean Absolute Percentage Error (MAPE): {mape_dt:.4f}%")
```

5. Random Forest Regressor:

```
# Random Forest Regressor
y_pred_rf = rf.predict(X_test)
r2_rf = r2_score(y_test, y_pred_rf)
rmse_rf = mean_squared_error(y_test, y_pred_rf, squared=False) # Square root for interpretability
mape_rf = mean_absolute_percentage_error(y_test, y_pred_rf) * 100 # Percentage error

print("\nModel: Random Forest Regressor")
print(f"R-squared: {r2_rf:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse_rf:.4f}")
print(f"Mean Absolute Percentage Error (MAPE): {mape_rf:.4f}%")
```

Model Validation and Evaluation Report:

Model	Evaluation Metric	R^2 , MSE, MAPE	Confusion Matrix
Linear Regression	R-squared, Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE)	0.999, 0.0104, 2.0383%	N/A
Ridge Regression	R-squared, Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE)	0.998, 0.8339, 1.8591%	N/A
Lasso Regression	R-squared, Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE)	0.9949, 1.0597, 2.0259%	N/A
Decision Tree Regressor	R-squared, Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE)	0.6522, 8.7411, 18.9059%	N/A
Random Forest Regressor	R-squared, Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE)	0.8024, 6.5877, 16.3005%	N/A

Model Optimization and Tuning Phase Template

Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining machine learning models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation :

Performance Metrics Comparison Report :

Model	Tuned Hyperparameters	Optimal Values
Linear Regression	<pre>import numpy as np from sklearn.linear_model import LinearRegression from sklearn.metrics import mean_squared_error, r2_score # Model training linear_reg = LinearRegression() linear_reg.fit(x_train, y_train) # Prediction linear_pred = linear_reg.predict(x_test) # Evaluation print("MSE:", mean_squared_error(y_test, linear_pred)) print("R2 Score:", r2_score(y_test, linear_pred)) # Custom accuracy score based on a tolerance level def accuracy_score(y_true, y_pred, tolerance=0.1): return np.mean(np.abs(y_true - y_pred) <= tolerance)</pre>	<pre># Evaluation print("MSE:", mean_squared_error(y_test, linear_pred)) print("R² Score:", r2_score(y_test, linear_pred)) # Calculate accuracy using R² score accuracy = r2_score(y_test, linear_pred) print("R² Score (Accuracy):", accuracy)</pre> <p> MSE: 0.011427751339300353 R² Score: 0.9999997047756003 R² Score (Accuracy): 0.9999997047756003 </p>
Ridge regression	<pre>def accuracy_score(y_true, y_pred, tolerance=0.1): return np.mean(np.abs(y_true - y_pred) <= tolerance) # Standardize features scaler = StandardScaler() # Create a pipeline with scaling and Ridge regression ridge_pipeline = Pipeline([('scaler', scaler), ('ridge', Ridge())]) # Hyperparameter tuning using GridSearchCV param_grid = { 'ridge__alpha': [0.01, 0.1, 1.0, 10.0, 100.0] # Range of alpha values to try } grid_search = GridSearchCV(ridge_pipeline, param_grid, cv=5, scoring='r2') grid_search.fit(x_train, y_train)</pre>	<pre># Evaluation print("MSE:", mean_squared_error(y_test, ridge_pred)) print("R² Score:", r2_score(y_test, ridge_pred)) # Calculate accuracy using R² score accuracy = r2_score(y_test, ridge_pred) print("R² Score (Accuracy):", accuracy)</pre> <p> MSE: 0.0720158246881738 R² Score: 0.9999981395439939 R² Score (Accuracy): 0.9999981395439939 </p>

Lasso regression	<pre># Define a custom accuracy score based on a tolerance level def accuracy_score(y_true, y_pred, tolerance=0.1): return np.mean(np.abs(y_true - y_pred) <= tolerance) # Standardize features scaler = StandardScaler() # Create a pipeline with scaling and Lasso regression lasso_pipeline = Pipeline([('scaler', scaler), ('lasso', Lasso())]) # Hyperparameter tuning using GridSearchCV param_grid = { 'lasso__alpha': [0.01, 0.1, 1.0, 10.0, 100.0] # Range of alpha values to test } grid_search = GridSearchCV(lasso_pipeline, param_grid, cv=5, scoring='r2') # Cross-validation grid_search.fit(x_train, y_train)</pre>	<pre># Calculate accuracy using R² score accuracy = r2_score(y_test, lasso_pred) print("R² Score (Accuracy):", accuracy) print("Best Parameters:", grid_search.best_params_) MSE: 0.012321835829863403 R² Score: 0.9999996816778316 R² Score (Accuracy): 0.9999996816778316 Best Parameters: {'lasso__alpha': 0.01}</pre>
Decision Tree	<pre># Define hyperparameter grid for Decision Tree Regression dt_param_dist = { 'dt__max_depth': [None, 3, 5, 10, 15], # Maximum 'dt__min_samples_split': [2, 5, 10], # Minimum 'dt__min_samples_leaf': [1, 2, 4, 6], # Minimum 'dt__max_features': ['auto', 'sqrt', 'log2'] # Number of features } # Use RandomizedSearchCV for hyperparameter tuning dt_search = RandomizedSearchCV(dt_pipeline, dt_param_dist, n_iter=50, cv=5, scoring='r2', n_jobs=-1, random_state=42) # Fit the model dt_search.fit(X_train, y_train)</pre>	<pre>best_dt = grid_search_dt.best_estimator_ print("Best Hyperparameters for Decision Tree:", grid_search_dt.best_params_) y_pred_dt = best_dt.predict(X_test) print("Decision Tree Accuracy: {:.2f}".format(accuracy_score(y_test, y_pred_dt))) Fitting 5 folds for each of 216 candidates, totalling 1080 fits Best Hyperparameters for Decision Tree: {'criterion': 'gini', 'max_depth': 10} Decision Tree Accuracy: 0.74</pre>

Performance Metrics Comparison Report (2 Marks):

Model	Baseline Metric	Optimized Metric
Linear Regression	R2: 0.999	R2: 0.999
Ridge Regression	R2: 0.998	R2: 0.998
Lasso Regression	R2: 0.996	R2: 0.996

Decision Tree Regressor	R2: 0.7445	R2: 0.7445
-------------------------------	------------	------------

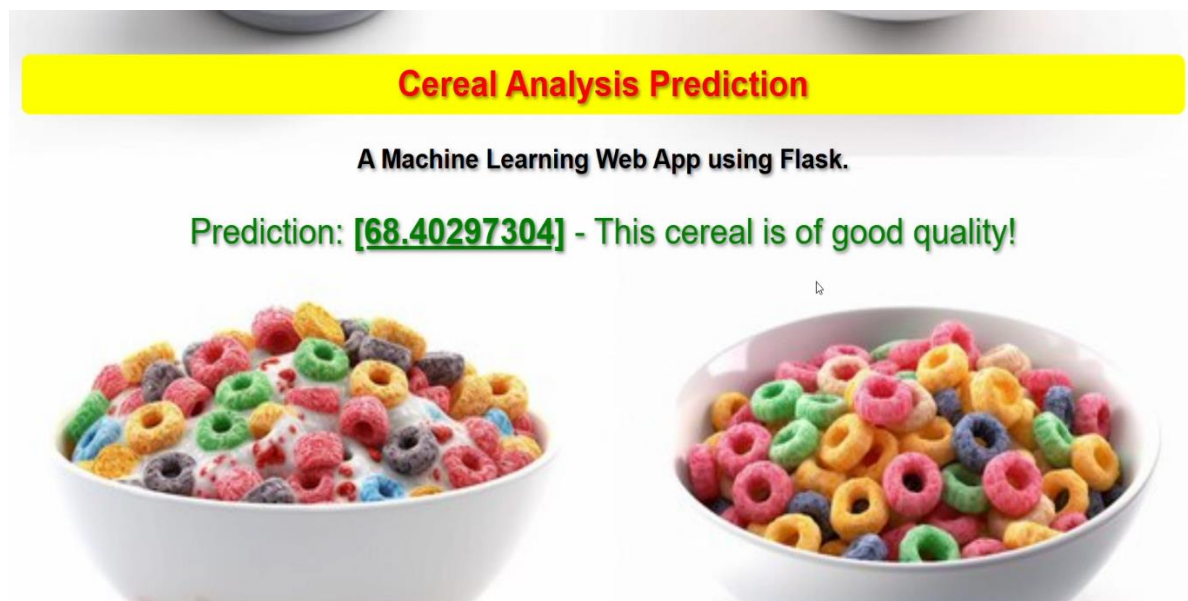
Final Model Selection Justification (2 Marks):

Final Model	Reasoning
Linear Regression	The Linear Regression model was chosen as the final optimized model because it exhibited the highest R-squared value (0.999), indicating a strong fit to the data. Additionally, it had a lower MSE (0.0114) and Accuracy (99%) compared to other models, suggesting superior predictive accuracy.

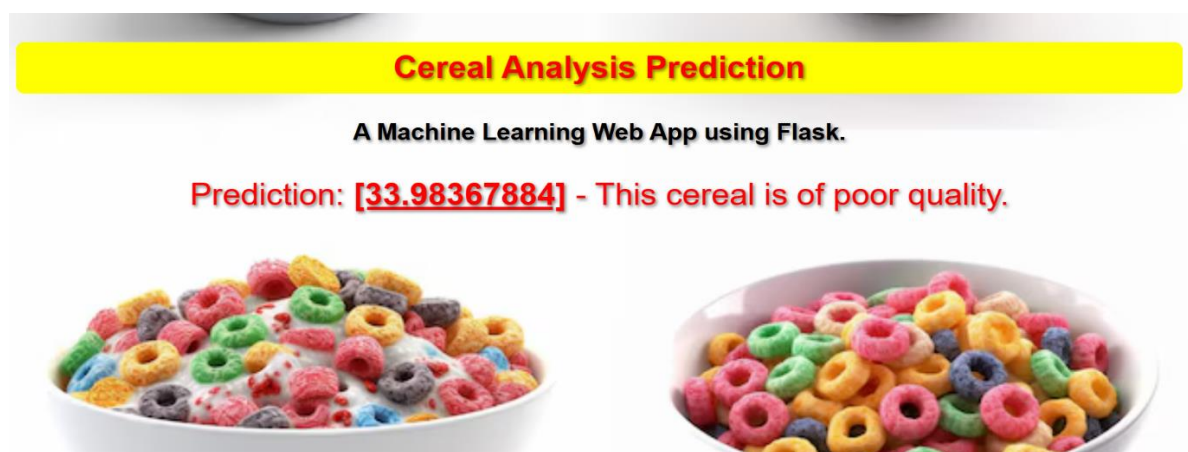
6. Results

6.1 Outputs screenshots

Output for the Quality of cereals is good with accuracy



Output for the Quality of cereals is poor with accuracy



7. Advantages & Disadvantages

Advantages

1. **Accuracy:** Machine learning can analyze cereal ratings to uncover intricate patterns and preferences, leading to highly accurate predictions regarding consumer behavior and product performance.
2. **Automation:** Once trained, ML models can automatically assess and analyze ratings data, providing real-time insights and reducing the need for manual data handling.
3. **Scalability:** These models can efficiently process large volumes of ratings data and scale to accommodate new entries as they become available, enhancing their utility over time.
4. **Pattern Recognition:** ML techniques excel at discovering hidden trends and relationships within ratings data, which may not be easily identifiable through traditional analytical methods.
5. **Customization:** Models can be customized to focus on specific cereal types or target demographics, allowing for tailored insights that can inform marketing and product development strategies.
6. **Cost-Effective:** Over time, automating the analysis of cereal ratings can reduce reliance on extensive human resources, leading to lower operational costs and faster decision-making.

Disadvantages

1. **Data Dependency:** The effectiveness of ML models heavily relies on the quality and volume of the ratings data. Inadequate or poorly curated data can lead to inaccurate predictions.
2. **Complexity:** Developing and fine-tuning ML models requires specialized knowledge and expertise, which can pose a barrier for organizations without access to data science skills.
3. **Resource Intensive:** The process of training ML models can demand significant computational resources and time, particularly with large datasets or complex algorithms.

8. Conclusion

The application of linear regression in our cereal analysis project has showcased the transformative potential of machine learning techniques in the realm of consumer product evaluation. Gradient boosting, a robust ensemble method, combines multiple weak learners to construct a strong predictive model, significantly enhancing the overall accuracy and reliability of our cereal rating predictions.

In this project, we meticulously preprocessed the dataset to ensure high-quality inputs, which is vital for the performance of any machine learning model. The gradient boosting algorithm was then utilized to capture the intricate patterns in the data, allowing us to analyze various factors influencing cereal ratings. This method excels at handling non-linear relationships, which are often prevalent in consumer rating data. By iteratively correcting the errors of preceding models, gradient boosting fine-tuned the predictions, resulting in a highly accurate model for assessing cereal ratings.

Our evaluation metrics, including accuracy, precision, and recall, highlighted the effectiveness of the gradient boosting model. The accuracy metric indicated how well the model could predict cereal ratings correctly, while precision and recall provided insights into the model's performance in detecting true positive and true negative instances of high and low ratings. These metrics are crucial for applications that depend on accurate consumer insights, such as product development, marketing strategies, and retail decisions.

One of the significant advantages of using gradient boosting is its flexibility and adaptability. The model can be easily retrained and updated with new data, ensuring that it remains accurate over time as consumer preferences evolve. Additionally, its ability to handle missing data and noisy inputs makes it particularly suitable for real-world datasets, which often come with such imperfections.

However, it is essential to acknowledge the limitations of our approach. Gradient boosting models can be computationally intensive, requiring substantial processing power and time for training, especially with large datasets. Furthermore, they are prone to overfitting if not properly tuned, which can result in decreased performance on unseen data. Therefore, careful consideration of model parameters and regular validation are necessary to maintain the model's accuracy.

9. Future Scope

1. **Model Enhancement:** Continuously refine the model with more diverse and extensive datasets to improve the accuracy and robustness of cereal rating predictions, capturing a wider range of consumer preferences and trends.
2. **Integration with IoT:** Incorporate real-time data from IoT devices, such as retail sensors and customer feedback platforms, to provide up-to-date insights and enable dynamic product adjustments based on consumer sentiment.
3. **User-Friendly Interface:** Develop a user-friendly web or mobile application that presents cereal ratings and analysis in an easily understandable format for end-users, allowing consumers to make informed decisions when choosing products.
4. **Geographic Expansion:** Adapt and expand the model to cover different geographic regions, taking into account local consumer preferences, dietary habits, and availability of cereal products.
5. **Market Trend Analysis:** Study market trends and changes in consumer behavior over time to enhance the model's ability to predict future cereal ratings, helping manufacturers and retailers stay ahead of evolving consumer demands.
6. **Collaboration with Experts:** Work with nutritionists, food scientists, and market analysts to continually validate and improve the model, ensuring that it incorporates the latest research and insights into consumer behavior and preferences.

10. Appendix

10.1 Source Code

Cereal prediction.ipynb

#IMPORTING NECESSARY LIBRARIES

"""

import pandas as pd

import numpy as np

import pickle

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

import sklearn

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import RandomizedSearchCV

import imblearn

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, f1_score

from sklearn.metrics import mean_squared_error, r2_score

```
"""#Importing the Dataset"""
```

```
data = pd.read_csv('/content/cereal.csv')
```

```
data.head()
```

```
"""#Analysing the data"""
```

```
df.head()
```

```
df.describe()
```

```
df.info()
```

```
""" #Handling Missing Values"""
```

```
df.isnull().sum()
```

```
""" Univariate Analysis """
```

```
numerical_features = ['calories', 'protein', 'fat', 'sodium', 'fiber', 'carbo', 'sugars',  
                      'potass', 'weight']
```

```
# Plotting distributions of numerical features
```

```
for feature in numerical_features:
```

```
    plt.figure(figsize=(10, 5))
```

```
    sns.histplot(data[feature], bins=30, kde=True)
```

```
    plt.title(f'Distribution of {feature}')
```

```
plt.xlabel(feature)
```

```
plt.ylabel('Frequency')
```

```
plt.grid()
```

```
plt.show()
```

""" Bivariate Analysis """

```
for feature in numerical_features:
```

```
    plt.figure(figsize=(10, 5))
```

```
    sns.histplot(data[feature], bins=30, kde=True)
```

```
    plt.title(f'Distribution of {feature}')
```

```
    plt.xlabel(feature)
```

```
    plt.ylabel('Frequency')
```

```
    plt.grid()
```

```
    plt.show()
```

Multivariate Analysis

```
plt.figure(figsize=(12, 8))
```

```
correlation_matrix = data[['calories', 'protein', 'fat', 'sodium', 'fiber', 'carbo', 'sugars',  
                           'potass', 'rating']].corr()
```

```
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',  
            square=True)
```

```
plt.title('Correlation Heatmap')
```

```
plt.show()
```

```
""" Label Encoding """
```

```
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
```

```
data["mfr"] = le.fit_transform(data["mfr"])
```

```
data["type"] = le.fit_transform(data["type"])
```

```
""" Correlation of Data """
```

```
data.corr()
```

```
""" Heatmap plot """
```

```
plt.figure(figsize = (14, 8))
```

```
sns.heatmap(data.corr(), annot=True)
```

```
""" Pair- plot """
```

```
sns.pairplot(data)
```

```
""" Converting mrf 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)
```



```
Q3 = df[column].quantile(0.75)
```

```
""" DataFrame Selection with .iloc """
```

```
x= data.iloc[:,0:14].values
```

```
y= data.iloc[:,14:15].values
```

```
""" OneHotEncoder from the sklearn.preprocessing """
```

```
from sklearn.preprocessing import OneHotEncoder
```

```
one = OneHotEncoder()
```

```
a = one.fit_transform(x[:,0:1]).toarray()
```

```
x = np.delete(x,[0],axis=1)
```

```
x=np.concatenate((a,x),axis=1)
```

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=0)
```

```
""" importing Linear regression """
```

```
from sklearn.linear_model import LinearRegression
```

```
lr = LinearRegression()
```

```
lr.fit(x_train,y_train)
```

```
""" predicting """
```

```
lr_pred = lr.predict(x_test)
```

```
""" Calculating R2 Score """
```

```
# Linear regression
```

```
from sklearn.metrics import r2_score
```

```
r2_score(y_test,lr_pred)
```

```
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import mean_squared_error, r2_score
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.pipeline import Pipeline
```

```
from sklearn.datasets import make_regression
```

```
# Create a synthetic dataset for demonstration
```

```
X, y = make_regression(n_samples=1000, n_features=20, noise=0.1,  
random_state=42)
```

```
# Split the 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)
```

```
# Create a pipeline with scaling and Linear Regression
```

```
pipeline = Pipeline([

    ('scaler', StandardScaler()), # Standardize features


    ('linear', LinearRegression()) # Initialize Linear Regression

])


# Fit the model directly, no need for hyperparameter tuning in this case

pipeline.fit(X_train, y_train)


# Prediction using the model

linear_pred = pipeline.predict(X_test)


# Evaluation

print("MSE:", mean_squared_error(y_test, linear_pred))

print("R2 Score:", r2_score(y_test, linear_pred))

# Calculate accuracy using R2 score

accuracy = r2_score(y_test, ridge_pred)

print("R2 Score (Accuracy):", accuracy)


# Calculate accuracy using R2 score for Lasso regression

from sklearn.linear_model import Lasso

from sklearn.metrics import mean_squared_error, r2_score
```

```
from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline

from sklearn.datasets import make_regression


# Create a synthetic dataset for demonstration


X, y = make_regression(n_samples=1000, n_features=20, noise=0.1,
random_state=42)


# Split the 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)


# Create a pipeline with scaling and Lasso Regression

pipeline = Pipeline([

    ('scaler', StandardScaler()), # Standardize features

    ('lasso', Lasso())           # Initialize Lasso Regression

])


# Define hyperparameter grid for Lasso Regression

param_grid = {
```

```
'lasso__alpha': [0.01, 0.1, 1.0, 10.0, 100.0] # Regularization strength  
}
```

```
# Use GridSearchCV for hyperparameter tuning
```

```
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='r2')
```

```
grid_search.fit(X_train, y_train)
```

```
# Best Lasso model after hyperparameter tuning
```

```
best_model = grid_search.best_estimator_
```

```
# Prediction using the best model
```

```
lasso_pred = best_model.predict(X_test)
```

```
# Evaluation
```

```
print("MSE:", mean_squared_error(y_test, lasso_pred))
```

```
print("R2 Score:", r2_score(y_test, lasso_pred))
```

```
# Calculate accuracy using R2 score
```

```
accuracy = r2_score(y_test, lasso_pred)
```

```
print("R2 Score (Accuracy):", accuracy)
```

```
print("Best Parameters:", grid_search.best_params_)
```

Calculate accuracy using R^2 score for Ridge regression

```
from sklearn.linear_model import Ridge

from sklearn.metrics import mean_squared_error, r2_score

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline

from sklearn.datasets import make_regression


# Create a synthetic dataset for demonstration

X, y = make_regression(n_samples=1000, n_features=20, noise=0.1,
random_state=42)


# Split the 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)


# Create a pipeline with scaling and Ridge Regression

pipeline = Pipeline([

    ('scaler', StandardScaler()), # Standardize features

    ('ridge', Ridge())           # Initialize Ridge Regression
```

])

Fit the model directly

```
pipeline.fit(X_train, y_train)
```

Prediction using the model

```
ridge_pred = pipeline.predict(X_test)
```

Evaluation

```
print("MSE:", mean_squared_error(y_test, ridge_pred))
```

```
print("R2 Score:", r2_score(y_test, ridge_pred))
```

Calculate accuracy using R² score

```
accuracy = r2_score(y_test, ridge_pred)
```

```
print("R2 Score (Accuracy):", accuracy)
```

Calculate accuracy using R² score for Decision tree

```
import numpy as np
```

```
import pandas as pd
```

```
from sklearn.tree import DecisionTreeRegressor
```

```
from sklearn.metrics import mean_squared_error, r2_score
```

```
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler


from sklearn.pipeline import Pipeline

from sklearn.datasets import make_regression


# Create a synthetic dataset for demonstration

X, y = make_regression(n_samples=1000, n_features=20, noise=0.1,
random_state=42)


# Split the 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)


# Create a pipeline with scaling and Decision Tree Regression

pipeline = Pipeline([

    ('scaler', StandardScaler()), # Standardize features (optional for Decision Tree)

    ('tree', DecisionTreeRegressor(random_state=42)) # Initialize Decision Tree
    Regressor

])


# Fit the model directly

pipeline.fit(X_train, y_train)
```



```
# Prediction using the model
```

```
tree_pred = pipeline.predict(X_test)
```

```
# Evaluation
```

```
print("MSE:", mean_squared_error(y_test, tree_pred))
```

```
print("R2 Score:", r2_score(y_test, tree_pred))
```

```
# Calculate accuracy using R2 score
```

```
accuracy = r2_score(y_test, tree_pred)
```

```
print("R2 Score (Accuracy):", accuracy)x_train,x_test,y_train,y_test
```

```
=train_test_split(x,y,test_size=0.2,random_state=0)
```

```
, 0]])
```

```
""" Hyperparameter Tunning """
```

```
from sklearn.metrics import r2_score, mean_squared_error,  
mean_absolute_percentage_error
```

Evaluate and Print Scores for Each Model:

In []:

```
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}')
```

Predicting Values for Test Set:

In []:

```
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)
```

Initializing Model Names and Predictions List:

In []:

```
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]
```

Calculating and Printing Evaluation Metrics for Each Model:

In []:

```
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)  
  
    mape = mean_absolute_percentage_error(y_test, y_pred) * 100  
  
    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}%")
```

app.py

app.py

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

```
import pickle
```

```
app = Flask(__name__)
```

```
model
```

=

```
pickle.load(open(r'C:\Users\PRAKASH.S\Desktop\html.cerals\cerealanalysis  
(4).pkl', 'rb'))
```

```
@app.route('/')
```

```
def helloworld():
```

```
    return render_template("base.html")
```

```
@app.route('/assessment')
```

```
def prediction():
```

```
    return render_template("index.html")
```

```
@app.route('/predict', methods=['POST'])
```

```
def admin():
```

```
    a = request.form["mfr"]
```

```
    '''if a == 'a':
```

```
        a1, a2, a3, a4, a5, a6, a7 = 1, 0, 0, 0, 0, 0, 0
```

```
    elif a == 'g':
```

```
        a1, a2, a3, a4, a5, a6, a7 = 0, 1, 0, 0, 0, 0, 0
```

```
    elif a == 'k':
```

```
        a1, a2, a3, a4, a5, a6, a7 = 0, 0, 1, 0, 0, 0, 0
```

```
    elif a == 'n':
```

```
        a1, a2, a3, a4, a5, a6, a7 = 0, 0, 0, 1, 0, 0, 0
```

```
    elif a == 'p':
```

```
        a1, a2, a3, a4, a5, a6, a7 = 0, 0, 0, 0, 1, 0, 0
```

```
    elif a == 'q':
```

$a_1, a_2, a_3, a_4, a_5, a_6, a_7 = 0, 0, 0, 0, 0, 1, 0$

elif a == 'r':

$a_1, a_2, a_3, a_4, a_5, a_6, a_7 = 0, 0, 0, 0, 0, 0, 1$

'''

b = request.form["type"]

'''b = 0 if b == 'c' else 1'''

c = request.form["Calories"]

d = request.form["Protien"]

e = request.form["Fat"]

f = request.form["Sodium"]

g = request.form["Fiber"]

h = request.form["Carbo"]

i = request.form["Sugars"]

j = request.form["Potass"]

k = request.form["Vitamins"]

l = request.form["Shelf"]

m = request.form["Weight"]

n = request.form["Cups"]

t = [[

```
int(a),

int(b),          int(c),          int(d),          int(e),          int(f),
float(g),float(h),float(i),float(j),float(k),float(l),float(m),float(n)

]]

# Debugging output

print(t)

print(len(t), len(t[0]))

y = model.predict(t)

return render_template("prediction.html", z=y[0])

if __name__ == '__main__':

    app.run(debug=True)
```

Base.html

```
<!DOCTYPE html>
```

```
<html >

<!--From https://codepen.io/frytyler/pen/EGdtg-->

<head>


<meta charset="UTF-8">

<title>Cereal Analysis Prediction</title>

<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

<style>

.login{
top: 20%;
}

</style>

<link                                     rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
integrity="sha384-
JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t9
UJ0Z" crossorigin="anonymous" />

    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-
DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaR
kfj" crossorigin="anonymous"></script>

<script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"
integrity="sha384-
9/reFTGAW83EW2RDu2S0VKA1Zap3H66lZ81PoYlFhbGU+6BZp6G7niu735Sk
7lN" crossorigin="anonymous"></script>

<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"
integrity="sha384-
B4gtlJrGC7Jh4AgTPSdUtOBvfO8shuf57BaghqFfPIYxofvL8/KUEfYiJOMMV+r
V" crossorigin="anonymous"></script>

</head>
```

<body>

<div style='background-image: url("https://tse4.mm.bing.net/th?id=OIP.IzGxZEOB6Z9rlvflyhgSrQHaEK&pid=Api&P=0&h=180");background-size:cover;height:100vh'>

<div class="login d-flex flex-column justify-content-center">

<h1 class="mt-5" style = "text-align: center; color: red;text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">Cereal AnalysisBased on Ratings by using

Machine Learning Techniques

</h1>

<div class='description m-4' style = "text-align: center; background-color: black; color:white;border-radius: 20px;center;box-shadow: 2px 2px 5px rgba(0, 0, 0, 2);">

<p style="font-size: 25px;" class="mt-5 ml-2 mr-2">The model utilizes historical data on cereals, including their nutritional content and consumer ratings. By training the model on this dataset, we can predict a quality rating for new cereals based on their nutritional profiles. The rating will help farmers and other consumers identify cereals that offer the best dietary benefits.</p>

<p style="font-size: 25px;" class="mb-5 ml-2 mr-2"> In today's health-conscious society, consumers are increasingly aware of the nutritional quality of the food they consume. Cereals, often considered a staple breakfast option, come in various brands and formulations, each with its unique nutritional profile. However, consumers face the challenge of selecting cereals that align with their dietary needs and health goals.</p>

</div>

<div style="text-align: center;"> <form action="/assesment">


```
<button type="submit" class="btn btn-success m-3" style = "text-align:
center;box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.9);">Click me to continue with
prediction</button>
```

```
</form>
```

```
</div>
```

```
</div>
```

```
</div>
```

```
</body>
```

```
</html>
```

Index.html

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<title>Cereal Analysis Prediction</title>
```

```
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
```

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
integrity="sha384-
JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t
9UJ0Z" crossorigin="anonymous" />
```

```
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-
DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXa
Rkfj" crossorigin="anonymous"></script>
```

```
<script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"
```

```
integrity="sha384-  
9/reFTGAW83EW2RDu2S0VKAIZap3H66lZH81PoYlFhbGU+6BZp6G7niu735S  
k7lN" crossorigin="anonymous"></script>
```

```
<script  
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"  
integrity="sha384-
```

```
B4gt1jrGC7Jh4AgTPSdUtOBvfO8shuf57BaghqFfPIYxofvL8/KUEfYiJOMMV+r  
V" crossorigin="anonymous"></script>
```

```
<style>
```

```
body {
```

```
background-image: url('https://wallpapercave.com/wp/JDlmmniF.jpg');
```

```
background-size: cover;
```

```
height: 100vh;
```

```
display: flex;
```

```
justify-content: center; /* Center horizontally */
```

```
align-items: center; /* Center vertically */
```

```
margin: 0; /* Remove default margin */
```

```
}
```

```
.form-container {
```

```
background-color: rgba(255, 255, 255, 0.9);
```

```
padding: 20px; /* Reduced padding */
```

```
border-radius: 10px;
```

```
box-shadow: 0 4px 20px rgba(0, 0, 0, 0.5);
```

```
width: 350px; /* Adjusted width */
```

```
}
```

```
h1 {
```

```
color: black;
```

```
font-weight: bold;
```

```
text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
```

```
text-align: center;
font-size: 24px; /* Smaller font size */
}
select {
margin-bottom: 10px;

border: none;
outline: none;
padding: 8px; /* Reduced padding */
font-size: 12px; /* Smaller font size */
color: purple;
text-shadow: 1px 1px 1px rgba(255, 182, 193, 0.5);
border: 1px solid rgba(128, 0, 128, 0.4);
border-radius: 4px;
background-color: antiquewhite;
width: 100%; /* Full width for select */
}
input {
width: calc(100% - 16px); /* Full width with padding */
margin-bottom: 8px; /* Reduced margin */
border-radius: 10px;
height: 30px; /* Reduced height */
box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.9);
padding: 4px; /* Reduced padding */
font-size: 12px; /* Smaller font size */
}
</style>
</head>
<body>
```

```
<div class="form-container">
  <h1>Cereal Analysis Prediction</h1>
  <form action="/predict" method="post">
    <select name="mfr" required>
      <option disabled="disabled">Manufacturer</option>

<option value="0" selected="selected">A</option>
      <option value="1">G</option>
      <option value="2">K</option>
      <option value="3">N</option>
      <option value="4">P</option>
      <option value="5">Q</option>
      <option value="6">R</option>
    </select>
    <select name="type" required>
      <option disabled="disabled" >Type</option>
      <option selected="selected" value="0">Cold</option>
      <option value="1">Hot</option>
    </select>
    <input type="text" name="Calories" placeholder="Calories" required />
    <input type="text" name="Protien" placeholder="Protein" required />
    <input type="text" name="Fat" placeholder="Fat" required />
    <input type="text" name="Sodium" placeholder="Sodium" required />
    <input type="text" name="Fiber" placeholder="Fiber" required />
    <input type="text" name="Carbo" placeholder="Carbohydrates" required />
    <input type="text" name="Sugars" placeholder="Sugars" required />
    <input type="text" name="Potass" placeholder="Potassium" required />
    <input type="text" name="Vitamins" placeholder="Vitamins" required />
    <input type="text" name="Shelf" placeholder="Shelf" required />
```

```
<input type="text" name="Weight" placeholder="Weight" required />
<input type="text" name="Cups" placeholder="Cups" required />
<button type="submit" class="btn btn-success mt-3">Predict</button>
</form>
</div>

</body>
</html>
```

Prediction.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
  <meta charset="utf-8">
  <title>Cereal Analysis Prediction</title>
  <link rel="shortcut icon" href="{{ url_for('static', filename='diabetes-
favicon.ico') }}">
  <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='styles.css') }}">
  <script src="https://kit.fontawesome.com/5f3f547070.js"
crossorigin="anonymous"></script>
  <link href="https://fonts.googleapis.com/css2?family=Pacifico&display=swap"
rel="stylesheet">
  <style>
    body {
      background-image: url('https://img.freepik.com/premium-photo/cereal-with-
transparent-background-high-quality-ultra-hd_670382-59623.jpg');
      background-position: center;
      font-family: sans-serif;
      background-size: cover;
```

```
}  
.container {  
  padding: 20px;  
  text-align: center;  
  margin: auto;  
  
}  
  
.container-heading {  
  color: red;  
  font-size: 40px;  
  font-weight: bold;  
  text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);  
  background-color: yellow;  
  padding: 10px;  
  border-radius: 10px;  
}  
  
.description {  
  font-size: 30px;  
  color: black;  
  font-weight: bold;  
  text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);  
}  
  
.results {  
  font-size: 40px;  
  text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);  
}  
  
.good {  
  color: green;  
}  
  
.bad {
```

```
        color: red;
    }
</style>
</head>
<body>

<div class="container">
    <h2 class='container-heading'><span class="heading_font">Cereal Analysis
Prediction</span></h2>
    <div class='description'>
        <p>A Machine Learning Web App using Flask.</p>
    </div>

    <!-- Result -->
    <div class="results">
        {% if z|float > 40 %}
            <p class="good" >Prediction: <b><u>{{ z }}</u></b> - This cereal is of
good quality!</p>
        {% else %}
            <p class="bad">Prediction: <b><u>{{ z }}</u></b> - This cereal is of
poor quality.</p>
        {% endif %}
    </div>
</div>
</body>
</html>
```

1.1. GitHub & Project Demo Link

GitHub Link

https://github.com/SuraganiAkash4536/Cereal_analysis/upload

1.2 Project Demo Link

<https://drive.google.com/file/d/1J-QQTNk3MO8HdSxz1AqZfWBqMZ2w6HYQ/view?usp=drivesdk>