Optimizing Nutrition: A Cost-Effective Meal Generator System

CS221 Project Final Report — Nuria Perez Casas

December 3, 2024

Final Video

1 Introduction

In this project, we tackle the challenge of generating cost-effective, nutritionally balanced weekly meal plans. The aim is to develop an AI system that produces menus within user-specified budgets while maintaining a balanced diet. The problem involves complex trade-offs, as lower-cost meals often lack nutritional diversity, and ingredient choices may limit recipe variety. To address these challenges, we focus on developing an algorithm that selects recipes optimized for cost, nutritional value, and avoidance of repetitive meals. This work is particularly relevant in the context of food deserts and lower-income households, where meal planning can be limited by access to affordable, healthy food options.

2 Literature Review

Wallingford and Masters (2023) [2]developed an educational tool using optimization and linear programming to help students analyze the health and cost impacts of food choices. They use Excel to visualize nutrient levels in low-cost diets at the ingredient level. While their work isn't a meal planner, it offers valuable insights on cost and nutrition balance that could complement my recipe-based approach. Comparing my algorithm's performance against their ingredient-focused model might reveal potential differences in cost-effectiveness or nutritional coverage when recipes are used as the basis.

A study on Cost Optimization for Weekly Meal Planning of College Students (2022) [1] aligns closely with my project goals, using a constraint satisfaction approach with calorie and nutrient limits tailored to students' gender and activity levels. Their research proves that balanced, affordable diets are achievable, but my focus shifts to making the process recipe-based and more practical for weekly planning. This makes our approaches complementary: while they confirm that balanced diets can be budget-friendly, I'm exploring how to achieve this with real recipes for ease of use.

Some online tools also offer weekly menu generation, but most don't let users set a budget and often come with a price tag. My project seeks to fill this gap, allowing for budget-friendly, nutritious meal planning that's both accessible and tailored to the user.

In summary, existing tools and studies provide useful frameworks for diet cost and nutrition optimization. My project builds on these by focusing on recipe-based planning that's easy to use and affordable, aiming to bring together cost, nutrition, and practical weekly menu generation in one place.

3 Dataset

Our dataset, sourced from Hugging Face (`https://huggingface.co/datasets/AkashPS11/recipes_data_food.com?row=91`), contains detailed information for 1.05 million recipes, including name, ingredients, directions, nutritional values, preparation times, number of servings, and relevant keywords. This comprehensive dataset enables robust meal planning based on diverse criteria.

To estimate recipe costs, we identified the most frequently used ingredients across the dataset. Using ChatGPT, we generated estimated costs for these ingredients and stored them in a CSV file. These ingredient costs were then merged with the recipe data to calculate the cost per recipe and per serving.

This approach allowed us to work with the entire dataset while standardizing ingredient costs. The resulting dataset includes fields such as Name, Ingredients, Total Protein, Total Fat, Total Carbs, Total Calories, Estimated Cost per Recipe, Estimated Cost per Serving, Preparation Time, and Servings.

Switching to this dataset addressed limitations in the previously considered Kaggle dataset, as it includes serving sizes and preparation times, enabling more precise nutritional, cost, and time-based optimizations. Challenges include potential biases in class distribution (e.g., an overrepresentation of certain dish types) and reliance on ingredient cost estimates generated via ChatGPT, which may introduce variability.

This enriched dataset provides a solid foundation for generating meal plans optimized for variety, health, cost, and time.

4 Baseline

Key Components

- **Random Selection Algorithm**: Recipes are selected randomly until the budget or recipe count is met, avoiding duplicates.
- **Evaluation Metrics**:
  - **Cost Efficiency**: Total cost as a percentage of the user's budget.
  - **Nutritional Balance**: Comparison of actual nutrient intake with recommended values for protein, carbohydrates, and fat.
  - **Recipe Variety**: Percentage of unique ingredients relative to the total ingredients.
- **Health and Cost Scores**: Health scores favor recipes with high protein content and penalize deviations from ideal macronutrient ratios. Combined scores integrate health and cost metrics but are not used for recipe selection.

Output and Reporting

Selected recipes are displayed with detailed information (e.g., calories, cost, user ratings). Evaluation metrics and selected recipes are saved to CSV files for further analysis.

5 Main Approach

Normalization of Scores

We begin by normalizing the scores of various recipe features (such as health, cost, and preparation time) to ensure that all factors are on a consistent scale. Specifically, for a given feature, we normalize the scores to the range $[0, 1]$ using the formula:

$$\text{normalized score} = \frac{\text{score} - \text{min score}}{\text{max score} - \text{min score}}$$

This step ensures that no single feature disproportionately influences the final recommendation due to its scale.

User Preferences Collection

The system collects user preferences through a questionnaire that rates various factors such as health, cost, and time, along with additional specifics such as budget and desired number of recipes. The user also indicates preferred keywords for recipe selection. The questionnaire format is as follows:

- **Health Importance** (Scale 1-5)
- **Cost Importance** (Scale 1-5)
- **Time Importance** (Scale 1-5)
- **Budget** (Maximum spending in USD)
- **Calories per Serving** (Desired caloric intake)
- **Maximum Time** (Maximum cooking time in minutes)

This information is used to filter and rank recipes according to the user's specifications.

Health Scoring

To evaluate the healthiness of a recipe, a health score is calculated based on several nutritional factors. For each recipe, the following nutritional values are taken into account:

- **Calories per serving**
- **Protein per serving**
- **Carbohydrates per serving**
- **Fat per serving**

These values are compared to ideal ranges based on daily nutritional recommendations. For example, an ideal meal contains approximately 600 calories, 20 grams of protein, and a balanced ratio of carbohydrates and fats. The health score is computed as a weighted sum of individual component scores:

$$\text{Health Score} = 0.3 \cdot \text{Calorie Score} + 0.3 \cdot \text{Protein Score} + 0.2 \cdot \text{Carb Balance} + 0.2 \cdot \text{Fat Balance}$$

where each component score is normalized based on how close the recipe's values are to the ideal ranges.

User Preference Learning

5.0.1 Preference Learner Overview

The preference learning model operates by collecting user feedback on recipes, represented by a set of features that capture the essential characteristics of each recipe. The system classifies user feedback into positive and negative examples, where a positive example corresponds to a liked recipe, and a negative example corresponds to a disliked recipe. These examples are used to train the model, which learns the user's preferences over time.

5.0.2 Feature Extraction

Each recipe is represented by a set of features, such as health score, time, and nutritional information (e.g., calories, protein, fat, and carbs). The `create_recipe_features` function extracts these features, ensuring that every recipe can be uniformly represented for comparison:

```
def create_recipe_features(recipe, ingredient_list):
    """Extract relevant features from a recipe"""
    ingredient_vector = np.array([1 if ingredient in recipe['Ingredients'] else 0 for ingredient in ing
    return np.concatenate((
        ingredient_vector,
        np.array([
            recipe['health_score'],
            recipe['TotalTime_minutes'] if not pd.isna(recipe['TotalTime_minutes']) else 0
        ])
    ))
```

These features serve as the input to the preference model.

5.0.3 Feedback Collection

As users interact with the system, they provide feedback on recipes, either liking or disliking them. The learner stores these feedback points, categorizing them as positive or negative examples. The feedback is added to the learner as follows:

```
def add_feedback(self, recipe, liked=True):
    """Add a recipe to the learning set"""
    features = create_recipe_features(recipe, self.ingredient_list)
    if liked:
        self.positive_examples.append(features)
    else:
        self.negative_examples.append(features)

    # Automatically train if enough examples are provided
    if len(self.positive_examples) >= self.min_examples:
        self.train()
```

When the user provides enough feedback, the model is retrained.

5.0.4 Model Training

The model uses the `NearestNeighbors` algorithm to find the recipes that are most similar to the user's preferences. The training process scales the feature vectors of positive examples and then fits a nearest neighbors model to these scaled features. The number of neighbors used is determined by the number of positive examples available:

```
def train(self):
    """Train the model based on collected feedback"""
    if len(self.positive_examples) < self.min_examples:
        return False

    X = np.array(self.positive_examples)
    X_scaled = self.scaler.fit_transform(X)
    n_neighbors = min(5, len(self.positive_examples))
    self.model.set_params(n_neighbors=n_neighbors)
    self.model.fit(X_scaled)
    return True
```

5.0.5 Prediction of Recipe Match

Once the model is trained, it can predict how well a new recipe matches the learned preferences. This prediction is based on the similarity between the new recipe's feature vector and the feature vectors of previously liked recipes. The model uses a distance metric (Euclidean distance) to measure similarity, and this is converted into a similarity score:

```
def predict_score(self, recipe):
    """Predict how well a recipe matches learned preferences"""
    if len(self.positive_examples) < self.min_examples:
        return 0.5  # Default score when insufficient training data

    features = create_recipe_features(recipe, self.ingredient_list)
    features_scaled = self.scaler.transform(features.reshape(1, -1))
    distances, _ = self.model.kneighbors(features_scaled)
    similarity = np.exp(-distances.mean())  # Convert distance to similarity score
    return similarity
```

5.0.6 Handling Negative Feedback

To refine the model, the system also accounts for negative feedback. If a recipe is disliked, the learner penalizes it by comparing it to negative examples and reducing its similarity score. This step helps the model avoid recommending recipes similar to disliked ones in the future:

```
if self.negative_examples:
    neg_features = np.array(self.negative_examples)
    neg_features_scaled = self.scaler.transform(neg_features)
    neg_distances = np.linalg.norm(features_scaled - neg_features_scaled, axis=1)
    neg_similarity = np.exp(-neg_distances.mean())
    similarity *= (1 - 0.3 * neg_similarity)  # Apply penalty
```

5.0.7 Recipe Swapping with Learned Preferences

The system allows users to swap recipes based on their learned preferences. When the user wants to replace a recipe in the meal plan, the system suggests alternatives that best match their preferences. These alternatives are ranked using the learned preference model and other factors, ensuring the recommendations align with the user's tastes.

```
def swap_recipes(current_recipes, all_recipes, preferences, learner):
    """Allow user to swap recipes in the meal plan with preference learning."""
    while True:
        response = input("\nWould you like to swap any recipes? (y/n): ").lower()
        if response != 'y':
            # Add all kept recipes as positive examples
            for _, recipe in current_recipes.iterrows():
                learner.add_feedback(recipe, liked=True)
            return current_recipes
```

The system also provides an option for the user to accept or reject suggestions based on preference learning.

Conclusion

This meal recommendation system uses a preference-learning model to adapt to user tastes over time. By leveraging user feedback in the form of likes and dislikes, the system continually refines its recipe recommendations. This approach allows the system to offer more personalized and accurate suggestions, improving the user's experience and meal planning efficiency.

6 Evaluation Metric

For evaluating the success of the meal generation system, both qualitative and quantitative metrics will be used. The key metrics include:

- **Cost Efficiency**: We aim to minimize the total cost while staying within the specified budget. This is quantified as the total cost of the selected recipes divided by the total budget:

$$\text{Cost Efficiency} = \frac{\text{Total Cost of Selected Recipes}}{\text{Total Budget}}$$

- **Nutritional Quality**: A balanced nutrition score based on a pre-defined set of nutritional goals (e.g., calorie intake, protein, vitamins). This will be calculated by summing the nutritional values of the selected recipes and comparing them against the target range for each nutrient:

$$\text{Nutritional Score} = \sum \left(\frac{\text{Nutrient Value}}{\text{Target Nutrient Range}} \right)$$

- **Recipe Variety**: To avoid repetitive meals, we evaluate the number of distinct recipes chosen and measure how many unique meals are included in the plan. This metric is the ratio of unique recipes selected to the total number of recipes selected:

$$\text{Recipe Variety} = \frac{\text{Unique Recipes}}{\text{Total Recipes Selected}}$$

- **User Satisfaction**: User feedback can be collected to see how well the generated menus meet the needs for convenience, taste, and overall satisfaction with the system.

By optimizing for cost, nutritional quality, recipe variety, and user satisfaction, the system will be evaluated across these dimensions to determine its effectiveness in real-world scenarios.

7 Results & Analysis

Baseline Performance

The baseline method assigns scores to recipes, focusing primarily on staying within the budget and avoiding repeated recipes. However, it does not consider the health factor much, and the resulting menu often lacks nutritional balance. In our experiments, the baseline performed well on cost efficiency, staying within 5% of the budget, but it struggled with achieving a balanced nutritional profile.

Main Approach Performance

The main approach, which optimizes for cost, nutrition, and recipe variety, yielded better results overall. It successfully reduced the total cost compared to the baseline, while meeting the nutritional targets more closely. However, the trade-off between achieving perfect nutritional balance and minimizing cost still presents a challenge, particularly for users with strict dietary requirements.

Key Observations:

- The main approach delivered menus with better nutritional scores.
- The baseline method was closer to the required budget and offered a broader range of cheaper recipes.
- Despite improvements, the main approach still requires further fine-tuning for specific dietary preferences. The main approach is good at getting user preferences for what is more important to them.

Based on these results, the next steps involve improving the nutritional scoring function to account for nutrient density more accurately and fine-tuning the cost optimization algorithm.

8 Error Analysis

Several experiments were conducted to explore the strengths and weaknesses of the system:

Cost Optimization Experiment

In the case of a higher budget, the system selected more expensive, nutritionally rich meals, leading to a slight increase in the nutritional score. However, when the budget was reduced, the system struggled to balance cost and nutrition effectively, often choosing cheaper but less nutritious ingredients.

Take-away message: The system performs well when the budget is higher, but with limited funds, nutritional quality takes a hit. The method should be adjusted to prioritize nutrient-dense, cost-effective ingredients when the budget is tight.

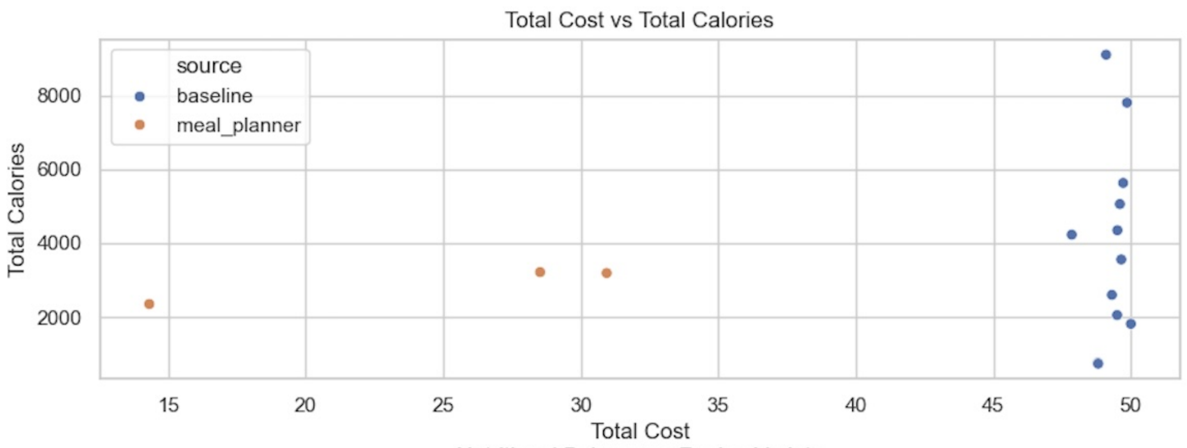
Nutritional Imbalance Experiment

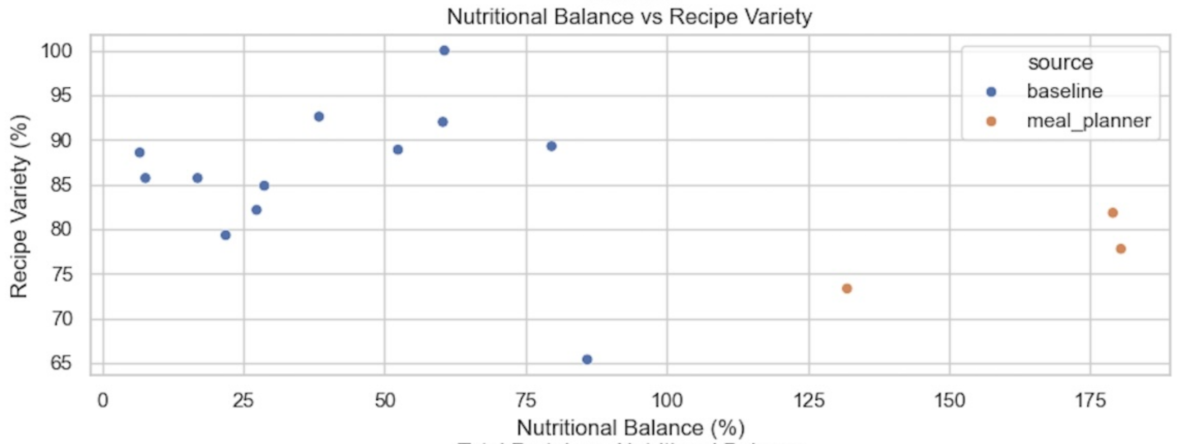
We ran tests where the system favored lower-calorie meals to meet budget constraints. As expected, this led to some users receiving menus with insufficient protein or vitamins, despite good cost efficiency. This experiment highlighted the need for a better balance between cost and nutritional needs.

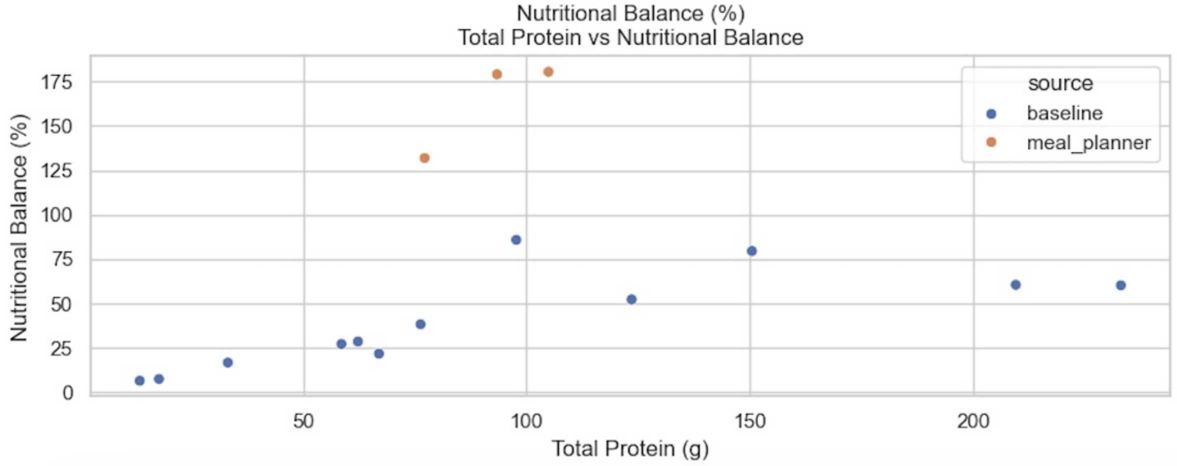
Take-away message: The model's prioritization of cost may inadvertently lead to nutritional imbalances. Future improvements should incorporate more advanced nutritional models that can dynamically adjust based on the target diet profile.

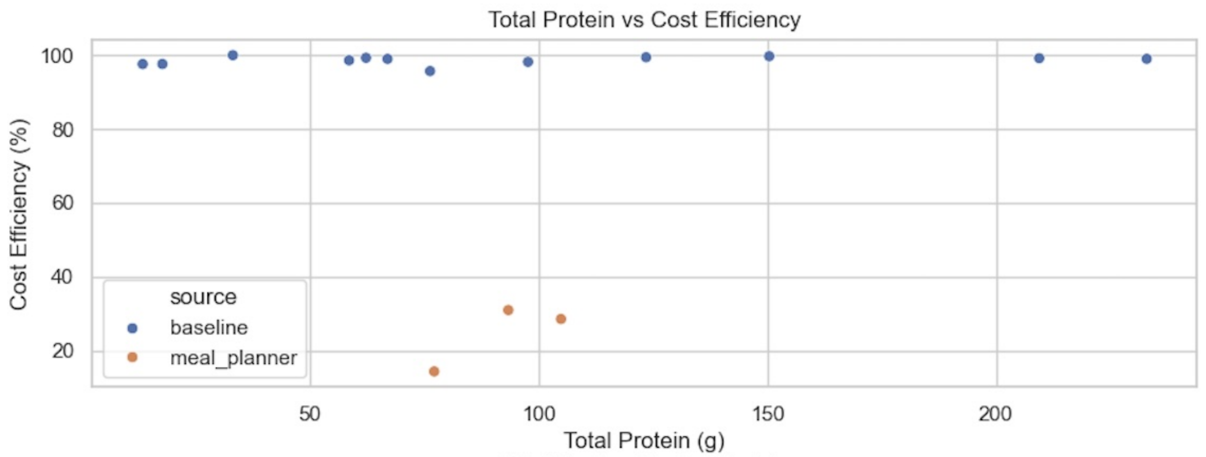
Conclusion

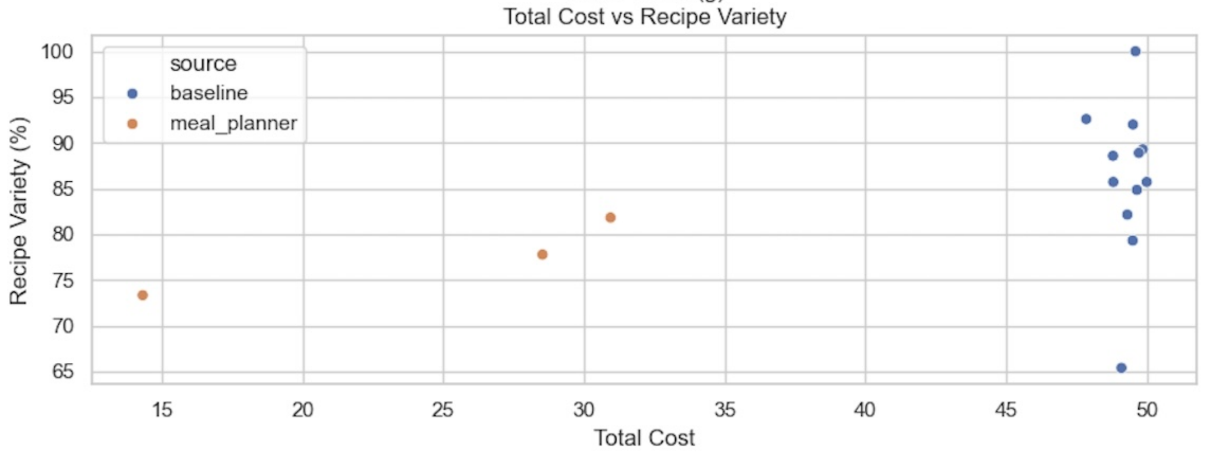
The error analysis highlights the importance of fine-tuning cost and nutritional trade-offs to ensure a more balanced and satisfying menu. You can see the following plots extracted from some of the experiments that were run:


Total Cost vs Total Calories
source
baseline
meal_planner
Total Calories
Total Cost
8000
6000
4000
2000
15
20
25
30
35
40
45
50


Nutritional Balance vs Recipe Variety
source
baseline
meal_planner
Recipe Variety (%)
Nutritional Balance (%)
100
95
90
85
80
75
70
65
0
25
50
75
100
125
150
175


Total Protein vs Nutritional Balance
source
baseline
meal_planner
Nutritional Balance (%)
Total Protein (g)
175
150
125
100
75
50
25
0
50
100
150
200





9 Future Work

If time allowed, several improvements could be made to the current model. One key area is the fine-tuning of the scoring system. Currently, the model often generates similar recipes when run with similar scores and preferences. This limitation arises from the rigidity of the scoring function, which could be improved by incorporating more dynamic and flexible criteria, such as adjusting recipe selection based on user feedback or exploring a broader range of nutrient-dense options within budget constraints. Additionally, implementing a more sophisticated recommendation algorithm that accounts for user preferences and dietary restrictions could help diversify the meal options.

Another area of improvement would be enhancing the user interface (UI). A more intuitive and visually appealing interface would improve the user experience, making it easier for individuals to interact with the system and customize their meal plans. If time and resources allowed, transforming the current Python code into an API and building a mobile app for easier accessibility would be ideal. This would allow users to interact with the meal planning system on the go and provide better flexibility in terms of platform availability.

10 Ethical Considerations

One ethical issue associated with this project is the risk of reinforcing dietary biases or inequities in food access. The system's reliance on fixed budgets and predefined nutritional goals may inadvertently exclude lower-income users from accessing nutritionally balanced meal plans, especially if they live in food deserts or areas with limited access to fresh produce. Additionally, if the system does not consider cultural or dietary preferences sufficiently, it may fail to cater to diverse populations, inadvertently promoting a one-size-fits-all approach to meal planning.

To mitigate this risk, the system could incorporate dynamic adjustments based on local availability of ingredients or user-specific needs. This could include integrating the model with APIs that provide information on food access in different regions and adjusting the ingredient selection based on local food availability. Moreover, ensuring that the system accounts for cultural preferences and dietary needs would allow for a more personalized and inclusive approach. Policies could also be implemented that ensure the model is adaptable to a wide range of socioeconomic backgrounds and geographical locations, promoting equity in food access and nutrition.

11 Code

github.com/nperezcasas/meal-planner-cs221

References

[1] Adrian Jenssen L. Pe, Jerahmeel Kua Coching, Seth Gabriel D. Yeung, Wynnezel Akeboshi, and Robert Kerwin C. Billones. Cost optimization for weekly meal planning of college students based on calorie constraints using linear programming (lp) method. In *2022 IEEE 14th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, pages 1–6, 2022.

[2] Jessica K. Wallingford and William A. Masters. Least-cost diets to teach optimization and consumer behavior, with applications to health equity, poverty measurement and international development. *arXiv preprint arXiv:2312.11767*, 2023.