1)
Choosing what to eat for lunch every day can lead to decision fatigue and repetitive eating habits. This project addresses that problem by designing an adaptive algorithm that recommends meals based on multiple user-centered factors such as cravings, nutrition, cost, and variety. The system simulates real-world behavior, allowing feedback to influence future choices. This could be used in smart meal planning apps, workplace cafeterias, or wellness tools that aim to balance satisfaction with health and budget goals.

2)

This project implements two decision-making strategies:

- Greedy Algorithm – Always selects the meal with the highest score based on a weighted utility function.
- Epsilon-Greedy Bandit Algorithm – Explores new meals with probability epsilon (30%) and exploits known favorites otherwise.

The meal score is calculated using a linear function of craving, nutrition, cost, and a variety penalty:

```
# Compute a weighted score for the meal based on craving, nutrition, cost, and variety
#Linear utility function - combine multiple features that influence the decision
def score(self, current_day, weights):
    penalty = self.variety_penalty(current_day)
    return (weights['crave'] * self.craving_score + #add craving and nutrition (positives)
            weights['nutrition'] * self.nutrition -
            weights['cost'] * self.cost - #sub cost and penalty (negatives)
            weights['variety'] * penalty)
```

```
def greedy(meals, current_day, weights):
    best_score = float('-inf')
    selected = None
    for meal in meals:
        s = meal.score(current_day, weights)
        if s > best_score:
            best_score = s
            selected = meal
    return selected

# Epsilon-Greedy Bandit version:
# Picks a random meal at a set probability. Otherwise, pick greedy best
# Helps try new meals that might turn out better long term
# Models choosing between uncertain options

def bandit(meals, current_day, weights, epsilon):
    if random.random() < epsilon:
        return random.choice(meals)  # Explore: pick a random meal
    return greedy(meals, current_day, weights)  # Exploit: pick best based on score

# After choosing a meal, update its stats based on feedback
# meals evolve over time
def update_meal(meal, feedback_score, current_day):
    meal.total_reward += feedback_score  # total satisfaction score the meal received: tracks how well liked
    #the meal is over time
    meal.times_chosen += 1  # update how many times the meal has been eaten - selection frequency
    meal.last_eaten_day = current_day  # today is last eaten day
    # adapt craving score based on how much the user liked it - adaptive behavior
    meal.craving_score = max(1, min(10, meal.craving_score + (feedback_score - 5) * 0.1))
    #if craving score is positive then it increases and vice versa
```

3)
I ran simulations over 30 days using both Greedy and Bandit strategies. The same starting meal set and scoring weights were used.

Test Case 1: Variety Preference

- Goal: Avoid repeating meals frequently.
- Expected: Meals eaten recently should be deprioritized.
- Actual: Variety penalty reduced selection of repeated meals; most meals were picked only 3–5 times in 30 days.

Test Case 2: Craving Adaptation

- Goal: Meals with good feedback should be chosen more often.
- Expected: Craving score increases with high feedback, making the meal more likely to be picked again.
- Actual: Meals with consistent feedback > 7 saw increasing craving scores and higher frequency later in the simulation.

Test Case 3: Bandit Exploration

- Goal: Try more unique meals with Bandit than Greedy.
- Expected: Bandit should explore low-ranked meals occasionally.
- Actual: Bandit selected every meal at least once; Greedy skipped lowest-ranked meals entirely.

4)
Per-day runtime complexity: O(n) — each day involves scoring n meals.
Total simulation (30 days): O(n * d) = O(30n)
Memory use: O(n) for storing meals and their history.

5)

While the algorithm handles short-term feedback and decision fatigue well, it assumes simulated user feedback and static scoring weights. In reality, user preferences may shift more dynamically. Additionally, meals are treated as independent; there's no modeling of weekly menus or ingredient overlap. The Bandit approach explores but doesn't learn optimal policies beyond basic feedback adaptation.

In the future, this could be extended with personalized learning (user-specific weights), real dietary constraints, or calendar-aware meal planning. Incorporating external factors like weather, budget tracking, or inventory could create a context-aware lunch recommender. Visualization of results would also help users see how and why meals are being selected.