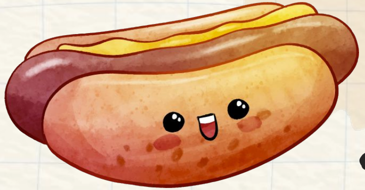


Personalized Diet Recommendation System

Ansh Avi Khanna

Github

Our Motivation

1.

Making Healthy Eating Easy

We're here to make it simple for everyone to eat well by suggesting personalized meals.

2.

Helping You Take Charge:

Our project gives you the tools to manage your health and make informed food choices.

3.

Meeting Different Goals:

Whether it's building muscle or keeping your heart healthy, we've got you covered with tailored recommendations.

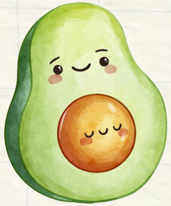
4.

Building a Healthier Future:

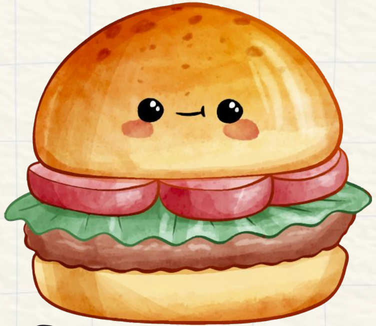
By promoting balanced eating and sustainable habits, we're working towards a healthier tomorrow for everyone.

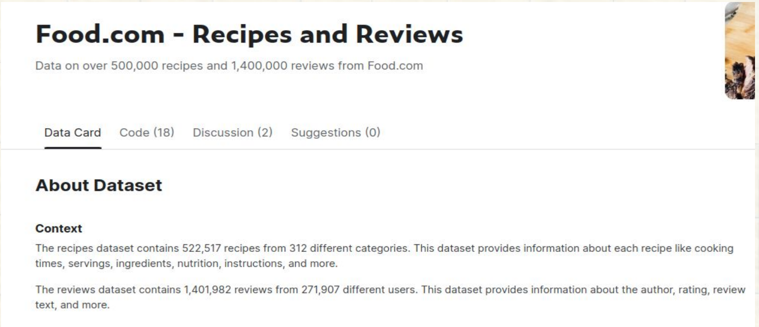
Introduction

- Eating healthily is a challenge in today's world due to busy lifestyles and conflicting dietary advice.
- Our project aims to simplify this by **offering personalized diet recommendations** for breakfast, lunch, and dinner.
- By considering specific user attributes like weight, height, gender as well as individual goals like weight loss, muscle gain, or heart health, we tailor meal plans to meet specific needs.
- With our system, users can make informed choices about their diet, leading to better health outcomes.



Dataset Creation & Scraping





Food.com - Recipes and Reviews

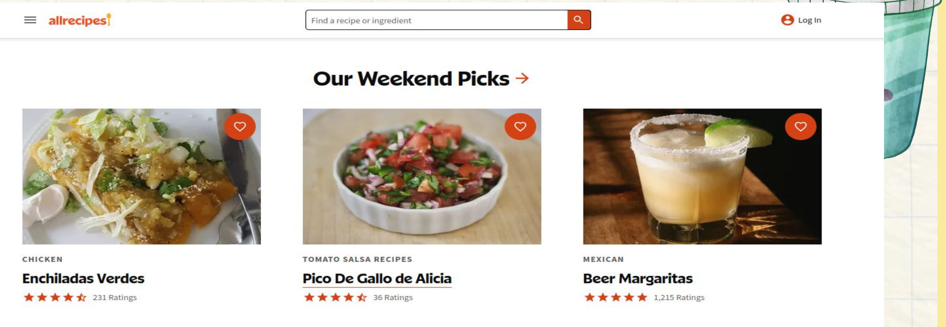
Data on over 500,000 recipes and 1,400,000 reviews from Food.com

About Dataset

Context

The recipes dataset contains 522,517 recipes from 312 different categories. This dataset provides information about each recipe like cooking times, servings, ingredients, nutrition, instructions, and more.

The reviews dataset contains 1,401,982 reviews from 271,907 different users. This dataset provides information about the author, rating, review text, and more.



Our Weekend Picks →

CHICKEN
Enchiladas Verdes
231 Ratings

TOMATO SALSA RECIPES
Pico De Gallo de Alicia
36 Ratings

MEXICAN
Beer Margaritas
1,215 Ratings

Food.com

AllRecipes.com

Initial Dataset Exploration:

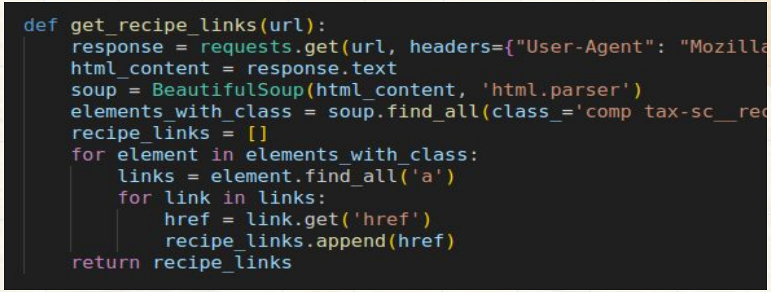
- Initially, we attempted to utilize datasets from both Kaggle and Food.com.
- Encountered challenges with a significant number of NaN values in the dataset.
- Identified difficulties in categorizing meals into distinct categories like breakfast, lunch, and dinner due to the lack of clear labeling.

Alternative Approach: Scraping from AllRecipies.com:

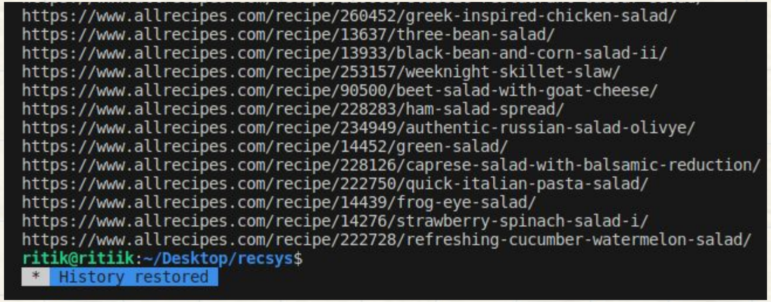
- As an alternative, we decided to scrape data directly from AllRecipies.com
- This approach provided us with a superior dataset characterized by clearer meal labels and a reduction in missing data instances.

Dataset Scraping

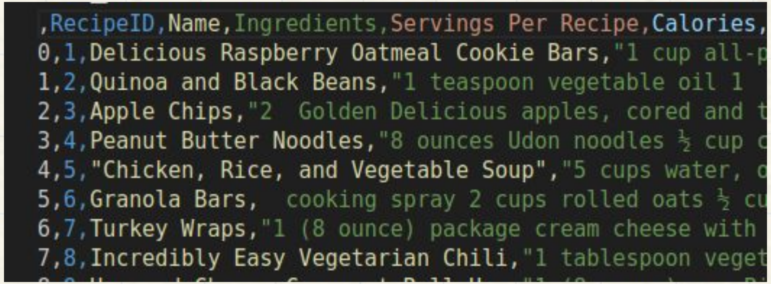
- We utilized the Python library **Beautiful Soup** for scraping data from **AllRecipes.com**.
- Breakfast comprises 1,132 links, lunch comprises 480 links, and dinner comprises 840 links on AllRecipes.com.
- The scraping process took approximately 1.5 – 2 hrs.
- Finally, we created CSV files named **breakfast.csv**, **lunch.csv**, and **dinner.csv**.



```python
def get_recipe_links(url):
    response = requests.get(url, headers={"User-Agent": "Mozilla
    html_content = response.text
    soup = BeautifulSoup(html_content, 'html.parser')
    elements_with_class = soup.find_all(class_='comp tax-sc__rec
    recipe_links = []
    for element in elements_with_class:
        links = element.find_all('a')
        for link in links:
            href = link.get('href')
            recipe_links.append(href)
    return recipe_links
```



```
https://www.allrecipes.com/recipe/260452/greek-inspired-chicken-salad/
https://www.allrecipes.com/recipe/13637/three-bean-salad/
https://www.allrecipes.com/recipe/13933/black-bean-and-corn-salad-ii/
https://www.allrecipes.com/recipe/253157/weeknight-skillet-slaw/
https://www.allrecipes.com/recipe/90500/beet-salad-with-goat-cheese/
https://www.allrecipes.com/recipe/228283/ham-salad-spread/
https://www.allrecipes.com/recipe/234949/authentic-russian-salad-olivye/
https://www.allrecipes.com/recipe/14452/green-salad/
https://www.allrecipes.com/recipe/228126/caprese-salad-with-balsamic-reduction/
https://www.allrecipes.com/recipe/222750/quick-italian-pasta-salad/
https://www.allrecipes.com/recipe/14439/frog-eye-salad/
https://www.allrecipes.com/recipe/14276/strawberry-spinach-salad-i/
https://www.allrecipes.com/recipe/222728/refreshing-cucumber-watermelon-salad/
ritik@ritiik:~/Desktop/recsys$
 * History restored
```

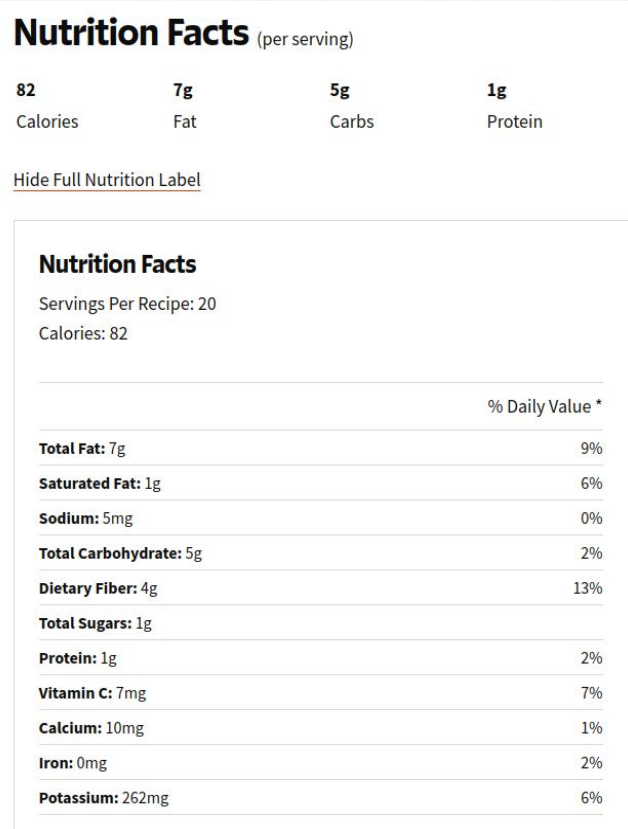


```
,RecipeID,Name,Ingredients,Servings Per Recipe,Calories,
0,1,Delicious Raspberry Oatmeal Cookie Bars,"1 cup all-p
1,2,Quinoa and Black Beans,"1 teaspoon vegetable oil 1
2,3,Apple Chips,"2  Golden Delicious apples, cored and t
3,4,Peanut Butter Noodles,"8 ounces Udon noodles ½ cup c
4,5,"Chicken, Rice, and Vegetable Soup","5 cups water, c
5,6,Granola Bars,  cooking spray 2 cups rolled oats ½ cu
6,7,Turkey Wraps,"1 (8 ounce) package cream cheese with
7,8,Incredibly Easy Vegetarian Chili,"1 tablespoon veget
```

Dataset Scraping

These are the nutrition data available on the site, which best fits our needs. We scraped this data and included it as columns in our dataframe.

***Philosophy:** Our main idea was to keep things real. We wanted to use data that reflects how people actually eat every day. By getting information from AllRecipes.com, we aimed to give practical advice that fits real-life eating habits. This way, our recommendations could be more helpful and relevant to users.*



Nutrition Facts (per serving)

82	**7g**	**5g**	**1g**
Calories	Fat	Carbs	Protein

Hide Full Nutrition Label

Nutrition Facts

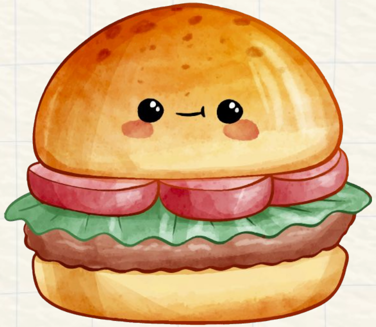
Servings Per Recipe: 20

Calories: 82

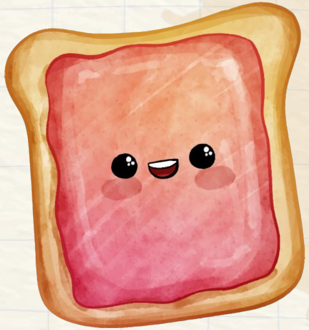
	% Daily Value *
Total Fat: 7g	9%
Saturated Fat: 1g	6%
Sodium: 5mg	0%
Total Carbohydrate: 5g	2%
Dietary Fiber: 4g	13%
Total Sugars: 1g	
Protein: 1g	2%
Vitamin C: 7mg	7%
Calcium: 10mg	1%
Iron: 0mg	2%
Potassium: 262mg	6%

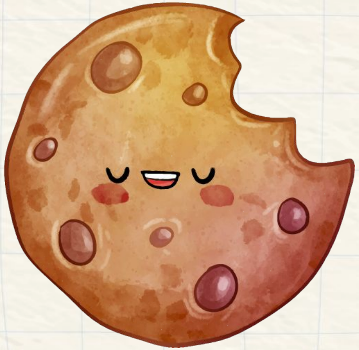
Approach - 1

Contextual Multi-Armed Bandits, Linear UCB, Epsilon-Greedy



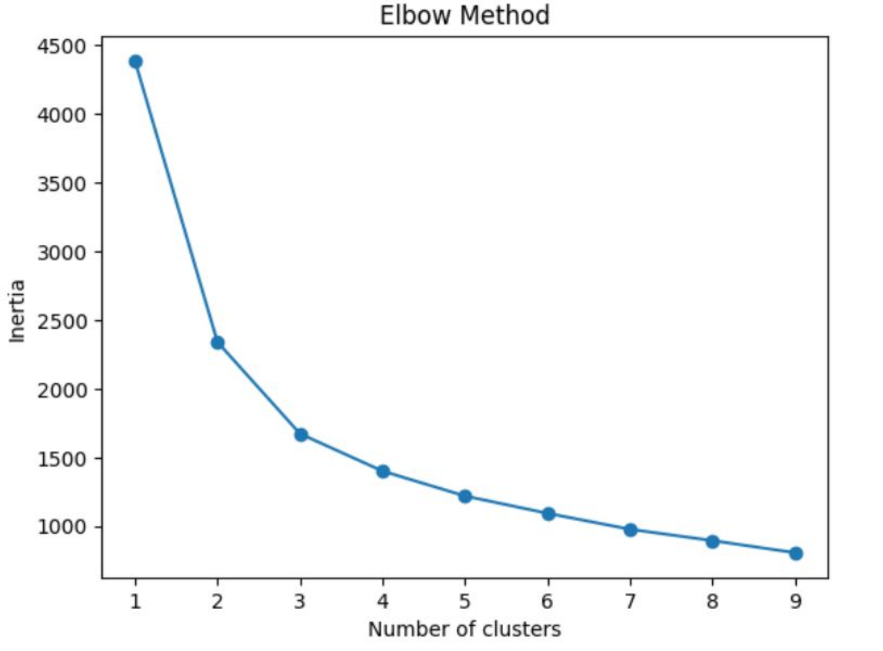
Data Preprocessing





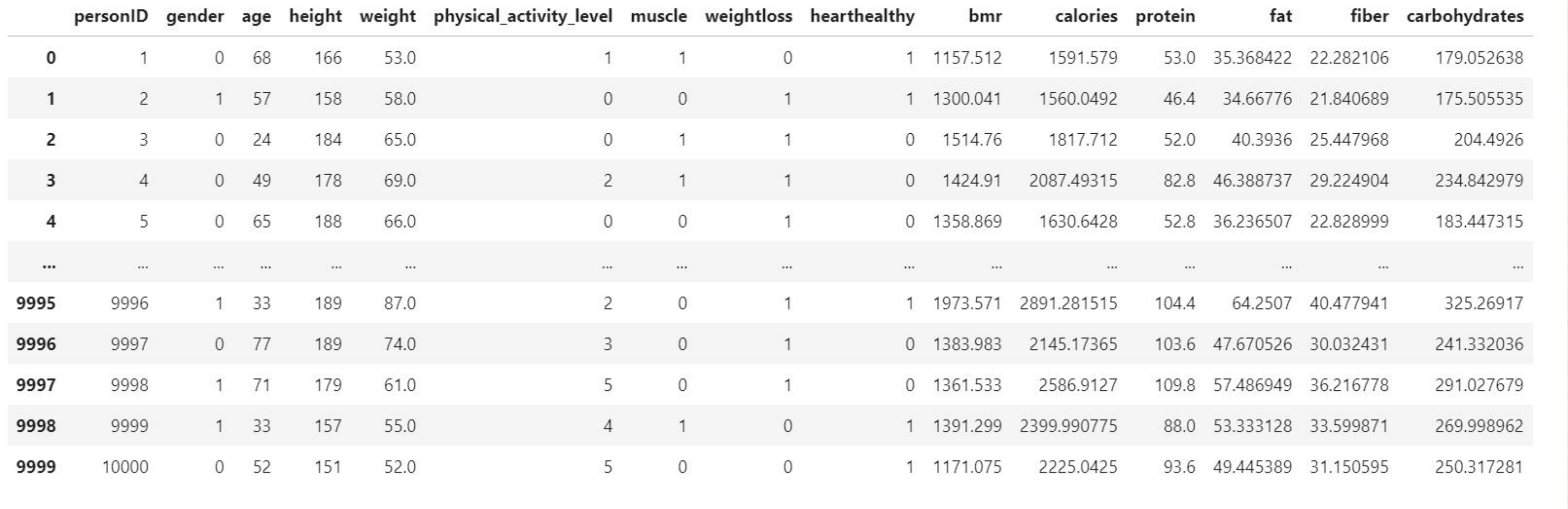
Dataset Preprocessing

- **Datasets**: We have three datasets for breakfast, lunch and dinner recipes. The columns in these datasets are as follows – 'RecipeID', 'Name', 'Ingredients', 'Servings Per Recipe', 'calories', 'protein', 'carbohydrates', 'fat', 'saturated_fat', 'cholesterol', 'fiber', 'Sodium', 'Rating', 'Number of Ratings', 'URL'
- **Clustering**: To do clustering in these three datasets, we first calculate the quantities of nutrients in each recipe for a 1000 calories (this is done in order to make the recipes comparable). Once this is done, the recipes are then clustered into 3, 4 and 4 clusters respectively for breakfast, lunch and dinner using *K Means*. The elbow graph shown is for clustering in breakfast dataframe.



Dataset Preprocessing

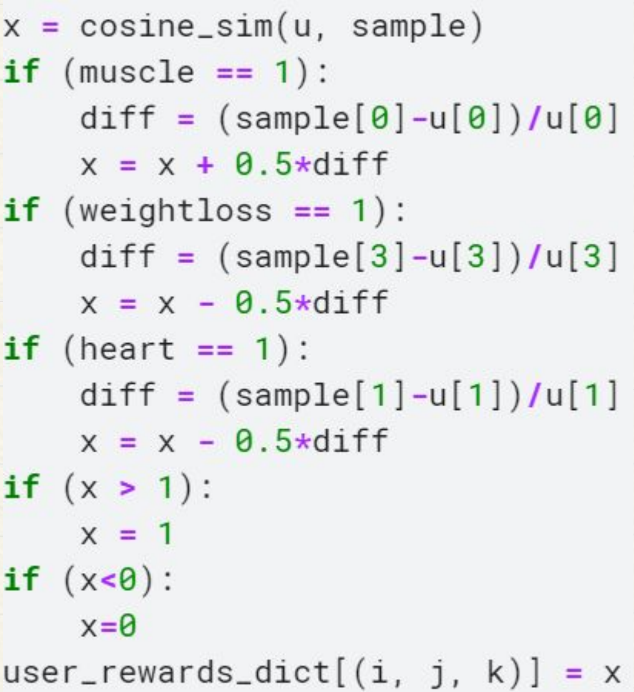
- **User Dataset**: User profiles are being synthetically generated.
- **Nutrient Calculation**: The optimal nutrients for each user are calculated and appended to the row corresponding to the user in the dataset. This nutrient calculation has been done using BMR (basal metabolic rate) and physical activity level, and other formulas found upon research online.



	personID	gender	age	height	weight	physical_activity_level	muscle	weightloss	hearthealthy	bmr	calories	protein	fat	fiber	carbohydrates
0	1	0	68	166	53.0	1	1	0	1	1157.512	1591.579	53.0	35.368422	22.282106	179.052638
1	2	1	57	158	58.0	0	0	1	1	1300.041	1560.0492	46.4	34.66776	21.840689	175.505535
2	3	0	24	184	65.0	0	1	1	0	1514.76	1817.712	52.0	40.3936	25.447968	204.4926
3	4	0	49	178	69.0	2	1	1	0	1424.91	2087.49315	82.8	46.388737	29.224904	234.842979
4	5	0	65	188	66.0	0	0	1	0	1358.869	1630.6428	52.8	36.236507	22.828999	183.447315
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9996	1	33	189	87.0	2	0	1	1	1973.571	2891.281515	104.4	64.2507	40.477941	325.26917
9996	9997	0	77	189	74.0	3	0	1	0	1383.983	2145.17365	103.6	47.670526	30.032431	241.332036
9997	9998	1	71	179	61.0	5	0	1	0	1361.533	2586.9127	109.8	57.486949	36.216778	291.027679
9998	9999	1	33	157	55.0	4	1	0	1	1391.299	2399.990775	88.0	53.333128	33.599871	269.998962
9999	10000	0	52	151	52.0	5	0	0	1	1171.075	2225.0425	93.6	49.445389	31.150595	250.317281

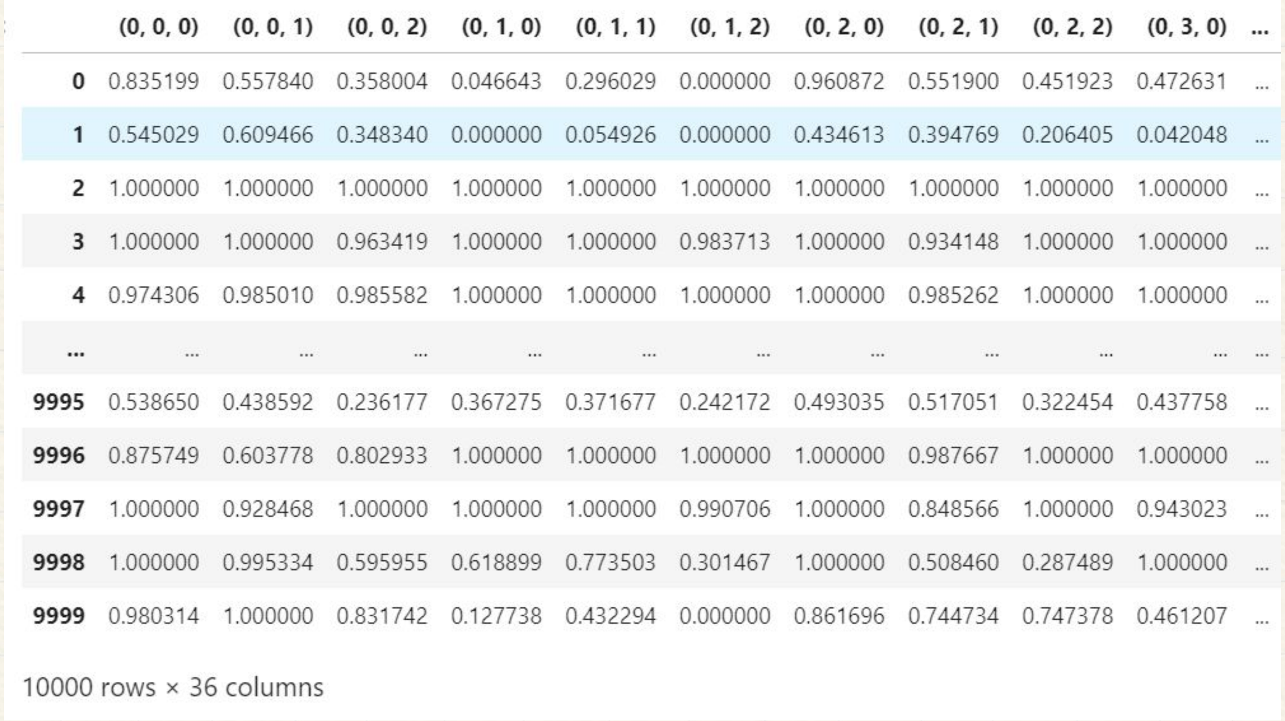
Reward Generation

- An arm is defined by a combination of clusters from breakfast, lunch and dinner (i.e we have 3 x 4 x 3 = 36 combos or arms).
- For each user, we iterate over all arms (combos). Each arm is defined by a combination of clusters as mentioned above. So, arm 0 is defined by the cluster combo (0, 0 , 0). Similarly, arm 1 is defined as (0, 0 ,1), arm 2 as (0, 0, 2), arm 3 as (0, 1, 0) etc. We sample a recipe randomly from clusters i , j and k for an arm defined by (i, j , k).
- We then calculate the reward as shown in the code on left.



```
x = cosine_sim(u, sample)
if (muscle == 1):
    diff = (sample[0]-u[0])/u[0]
    x = x + 0.5*diff
if (weightloss == 1):
    diff = (sample[3]-u[3])/u[3]
    x = x - 0.5*diff
if (heart == 1):
    diff = (sample[1]-u[1])/u[1]
    x = x - 0.5*diff
if (x > 1):
    x = 1
if (x<0):
    x=0
user_rewards_dict[(i, j, k)] = x
```

Reward Generation

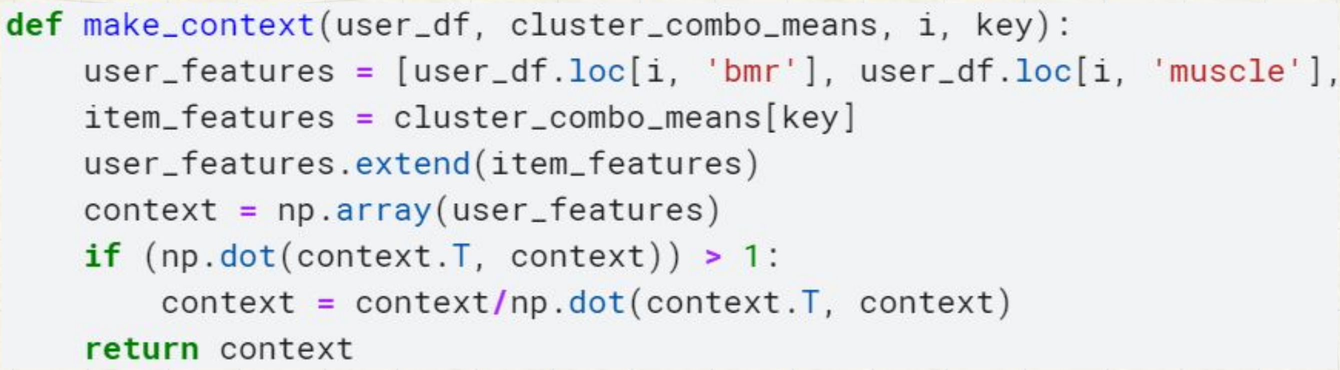


	(0, 0, 0)	(0, 0, 1)	(0, 0, 2)	(0, 1, 0)	(0, 1, 1)	(0, 1, 2)	(0, 2, 0)	(0, 2, 1)	(0, 2, 2)	(0, 3, 0)	...
0	0.835199	0.557840	0.358004	0.046643	0.296029	0.000000	0.960872	0.551900	0.451923	0.472631	...
1	0.545029	0.609466	0.348340	0.000000	0.054926	0.000000	0.434613	0.394769	0.206405	0.042048	...
2	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...
3	1.000000	1.000000	0.963419	1.000000	1.000000	0.983713	1.000000	0.934148	1.000000	1.000000	...
4	0.974306	0.985010	0.985582	1.000000	1.000000	1.000000	1.000000	0.985262	1.000000	1.000000	...
...	...	...	...	...	...	...	...	...	...	...	...
9995	0.538650	0.438592	0.236177	0.367275	0.371677	0.242172	0.493035	0.517051	0.322454	0.437758	...
9996	0.875749	0.603778	0.802933	1.000000	1.000000	1.000000	1.000000	0.987667	1.000000	1.000000	...
9997	1.000000	0.928468	1.000000	1.000000	1.000000	0.990706	1.000000	0.848566	1.000000	0.943023	...
9998	1.000000	0.995334	0.595955	0.618899	0.773503	0.301467	1.000000	0.508460	0.287489	1.000000	...
9999	0.980314	1.000000	0.831742	0.127738	0.432294	0.000000	0.861696	0.744734	0.747378	0.461207	...

10000 rows × 36 columns

Context Formation

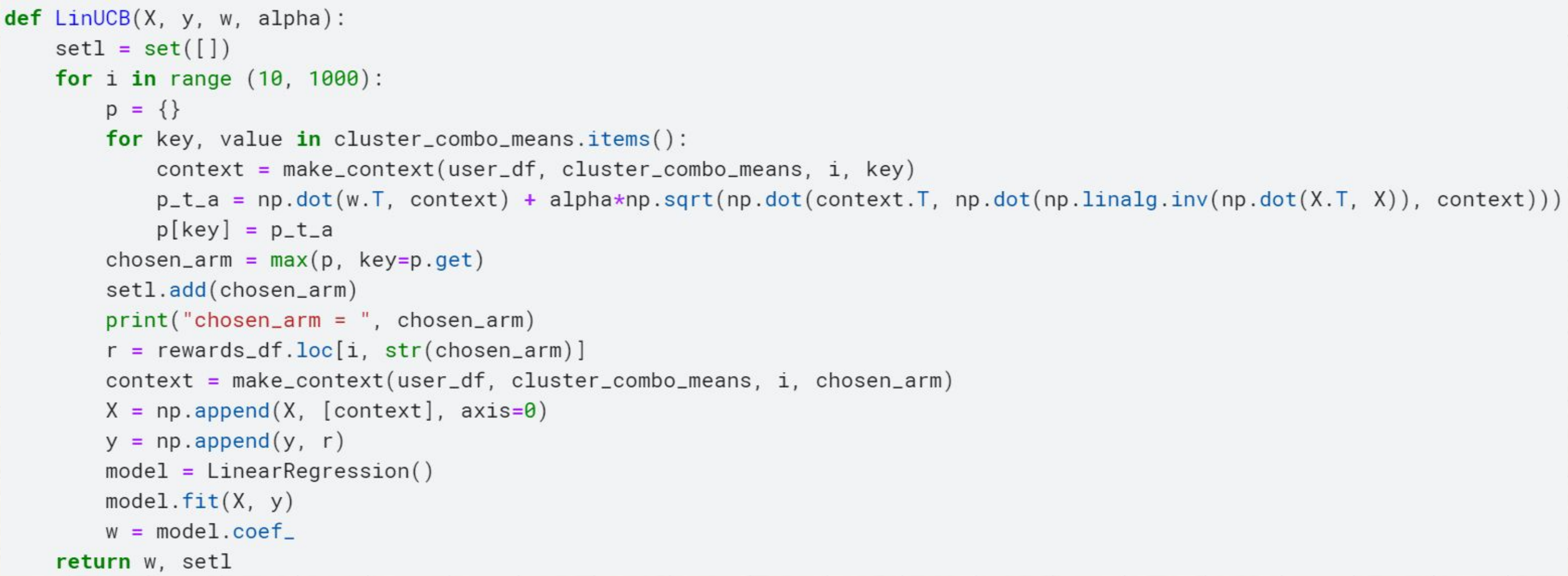
- In contextual bandits, context represents a combination of a user and an arm.
- User’s context is defined by BMR, activity level, muscle goal, weight loss goal, heart healthy goal.
- Arms context is defined by the vector denoting nutrients for a particular combo.
- We concatenate the two vectors to form the context.



```python
def make_context(user_df, cluster_combo_means, i, key):
    user_features = [user_df.loc[i, 'bmr'], user_df.loc[i, 'muscle'],
    item_features = cluster_combo_means[key]
    user_features.extend(item_features)
    context = np.array(user_features)
    if (np.dot(context.T, context)) > 1:
        context = context/np.dot(context.T, context)
    return context
```

LinUCB

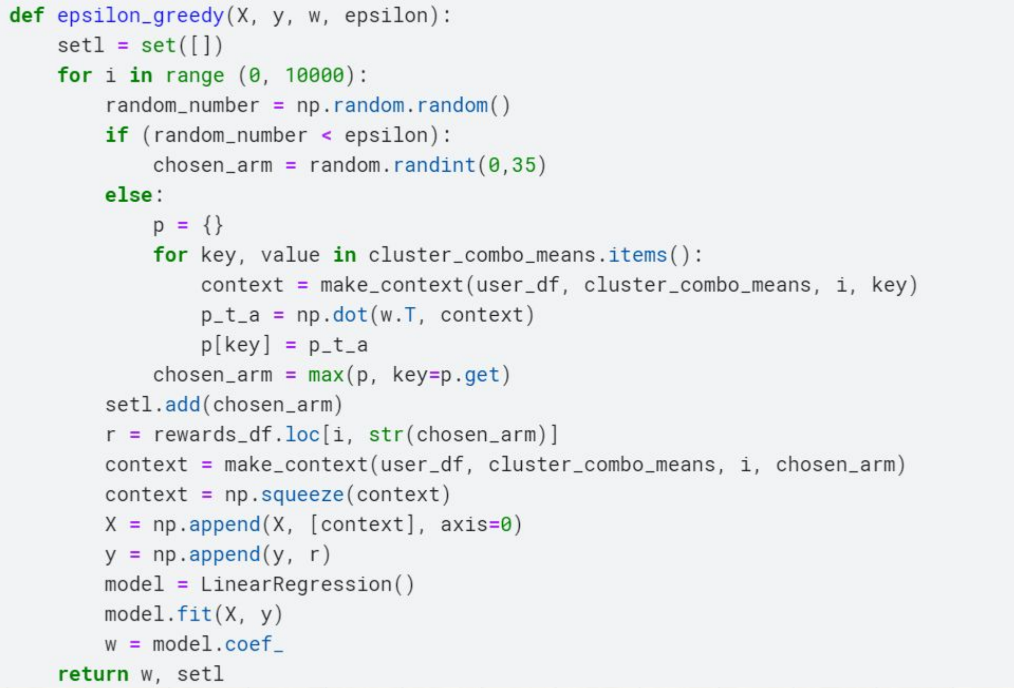
- We first tried using Linear UCB, whose code is shown below.



```
def LinUCB(X, y, w, alpha):
    setl = set([])
    for i in range (10, 1000):
        p = {}
        for key, value in cluster_combo_means.items():
            context = make_context(user_df, cluster_combo_means, i, key)
            p_t_a = np.dot(w.T, context) + alpha*np.sqrt(np.dot(context.T, np.dot(np.linalg.inv(np.dot(X.T, X)), context)))
            p[key] = p_t_a
        chosen_arm = max(p, key=p.get)
        setl.add(chosen_arm)
        print("chosen_arm = ", chosen_arm)
        r = rewards_df.loc[i, str(chosen_arm)]
        context = make_context(user_df, cluster_combo_means, i, chosen_arm)
        X = np.append(X, [context], axis=0)
        y = np.append(y, r)
        model = LinearRegression()
        model.fit(X, y)
        w = model.coef_
    return w, setl
```

Epsilon-Greedy

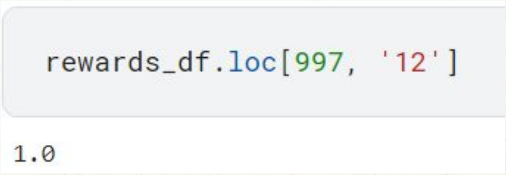
- We then used Epsilon-Greedy, whose code is shown below. We have taken epsilon to be 0.5.
- We chose Epsilon-Greedy over Linear UCB because we realised that even upon experimenting with various values of alpha, the algorithm was not exploring all the arms which is undesirable.



```
def epsilon_greedy(X, y, w, epsilon):
    setl = set([])
    for i in range (0, 10000):
        random_number = np.random.random()
        if (random_number < epsilon):
            chosen_arm = random.randint(0,35)
        else:
            p = {}
            for key, value in cluster_combo_means.items():
                context = make_context(user_df, cluster_combo_means, i, key)
                p_t_a = np.dot(w.T, context)
                p[key] = p_t_a
            chosen_arm = max(p, key=p.get)
        setl.add(chosen_arm)
        r = rewards_df.loc[i, str(chosen_arm)]
        context = make_context(user_df, cluster_combo_means, i, chosen_arm)
        context = np.squeeze(context)
        X = np.append(X, [context], axis=0)
        y = np.append(y, r)
        model = LinearRegression()
        model.fit(X, y)
        w = model.coef_
    return w, setl
```

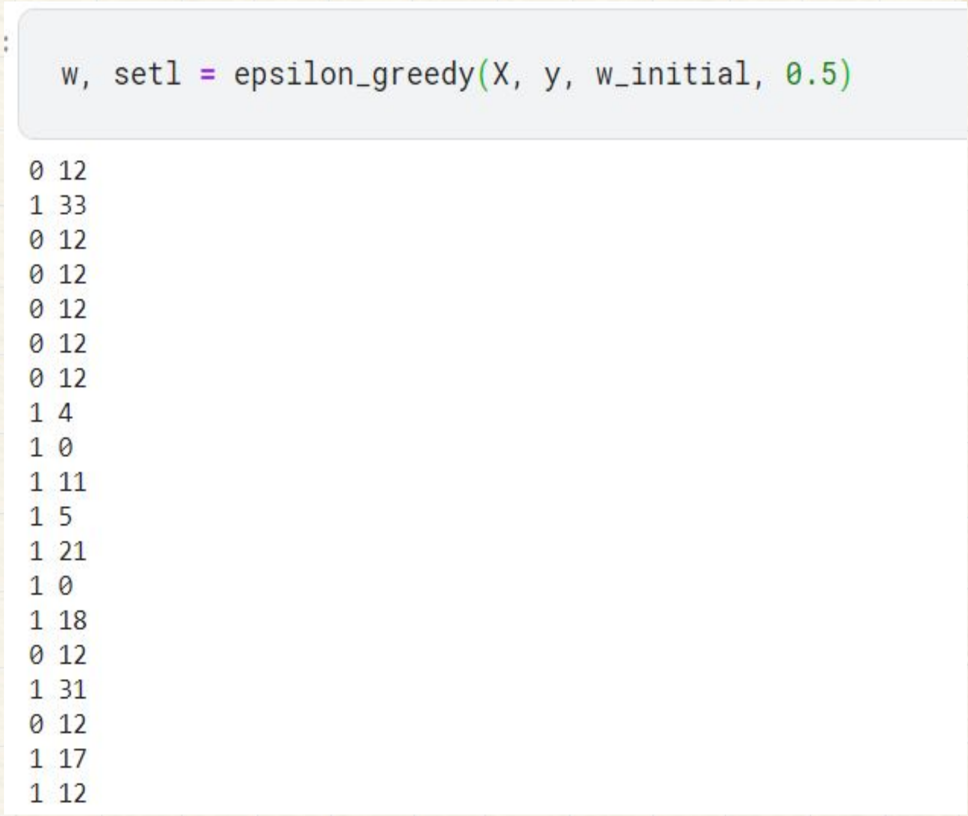
Results

- By the end of the run of the algorithm, if for a user, we are doing exploitation, the best arm should be picked upon exploitation.
- This means that, that particular arm should have the highest reward in the rewards dataframe for that user.
- It can be seen that at the 997th step, the arem chosen upon exploitation (0), is 12. And in rewards dataframe, the rewards for that arm is 1.



```
rewards_df.loc[997, '12']
```

```
1.0
```



```
w, setl = epsilon_greedy(X, y, w_initial, 0.5)
```

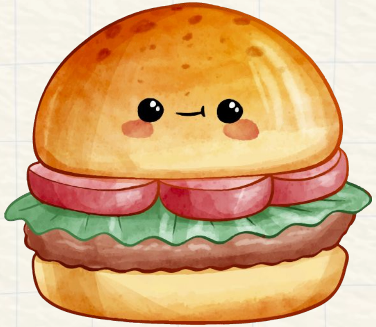
```
0 12
1 33
0 12
0 12
0 12
0 12
0 12
1 4
1 0
1 11
1 5
1 21
1 0
1 18
0 12
1 31
0 12
1 17
1 12
```

Challenges Faced

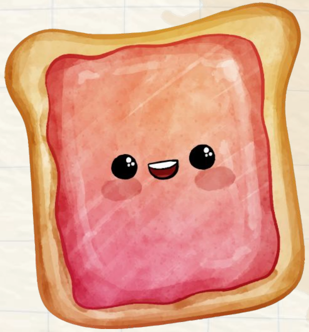
- The main challenge faced was how to form the context vector to make it learnable. We tried various methods for this. Element wise addition of user and arm vector, tensor product between user and arm vector, Hadamard Product between the user and arm vector. But all these methods did not improve the performance of the model, it was almost same as the performance as concatenation of the two vectors.
- Defining rewards was a difficult task. We first tried the converse of the norm of the different between user and sample vector. Then we went on to do the similarity metric with some noise so as to make the rewards stochastic. Finally, random sampling made the most sense since in real life too people would pick random recipes.
- Clustering recipes on what basis was also a challenge.
- We also faced many errors initially during algorithm implementation. Most of them were input shape related.

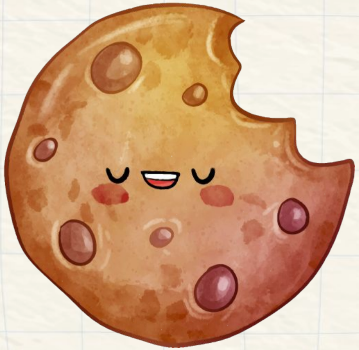
Approach - 2

Recommendations, Ratings, Epsilon Greedy, Decaying epsilon



Data Preprocessing





Filtering According to Diet Goal

Filtering recipes based on diet goals by setting appropriate thresholds on nutrient values.

Muscle Gain

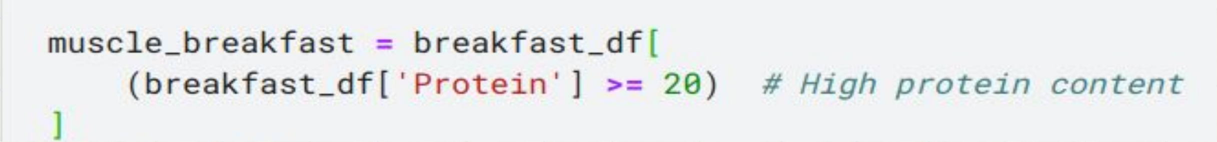
Focus on high protein intake.

Heart Healthy

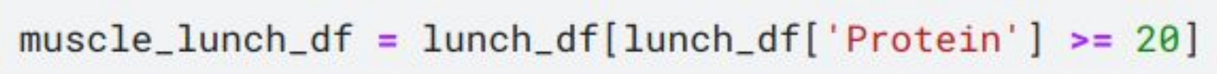
Promote foods low in saturated fats and rich in fruits, vegetables

Muscle Gain

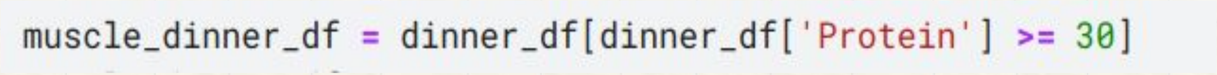
- Below are the constraints for filtering recipes from the dataset for users who want to gain muscle.



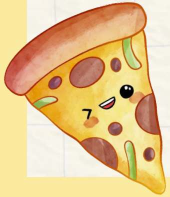
```
muscle_breakfast = breakfast_df[
    (breakfast_df['Protein'] >= 20)  # High protein content
]
```



```
muscle_lunch_df = lunch_df[lunch_df['Protein'] >= 20]
```



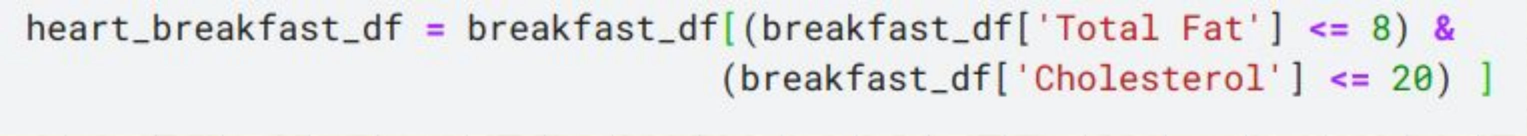
```
muscle_dinner_df = dinner_df[dinner_df['Protein'] >= 30]
```



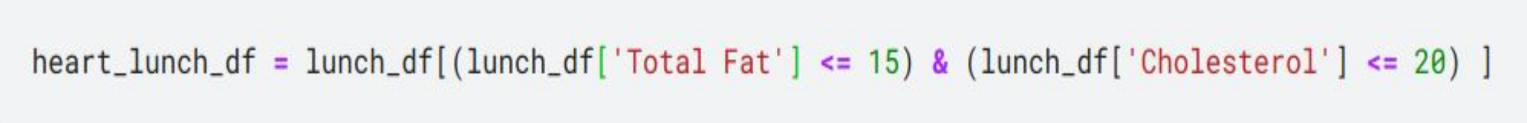
Reference for above thresholds

Heart Healthy

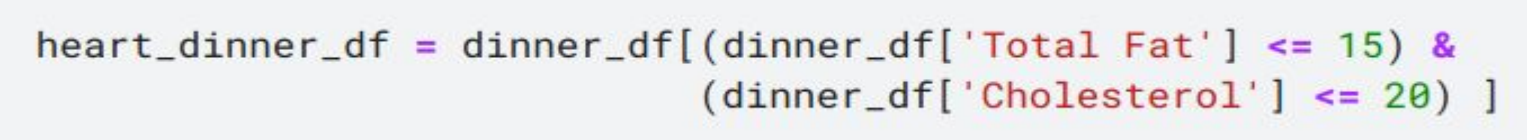
- Below are the constraints for filtering recipes from the dataset for users who want to prioritize heart health.



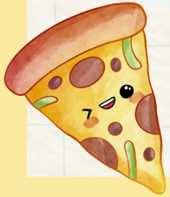
```
heart_breakfast_df = breakfast_df[(breakfast_df['Total Fat'] <= 8) &
                                  (breakfast_df['Cholesterol'] <= 20) ]
```



```
heart_lunch_df = lunch_df[(lunch_df['Total Fat'] <= 15) & (lunch_df['Cholesterol'] <= 20) ]
```



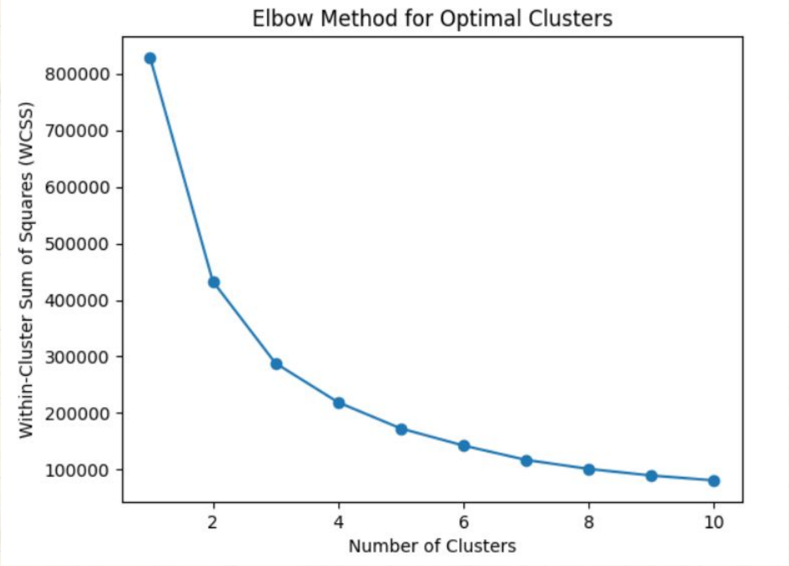
```
heart_dinner_df = dinner_df[(dinner_df['Total Fat'] <= 15) &
                            (dinner_df['Cholesterol'] <= 20) ]
```

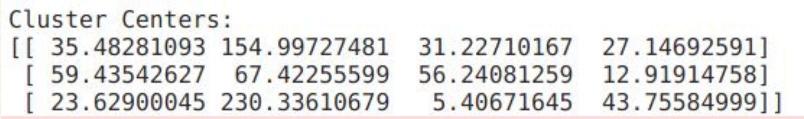


Reference for above thresholds

Clustering

- We did clustering for each of the filtered data frames using **k-means++**.
- And then choose the optimal number of clusters using elbow method.
- According to the graph choosing **3** as the optimal clusters and applying k-means++ with number of clusters as 3

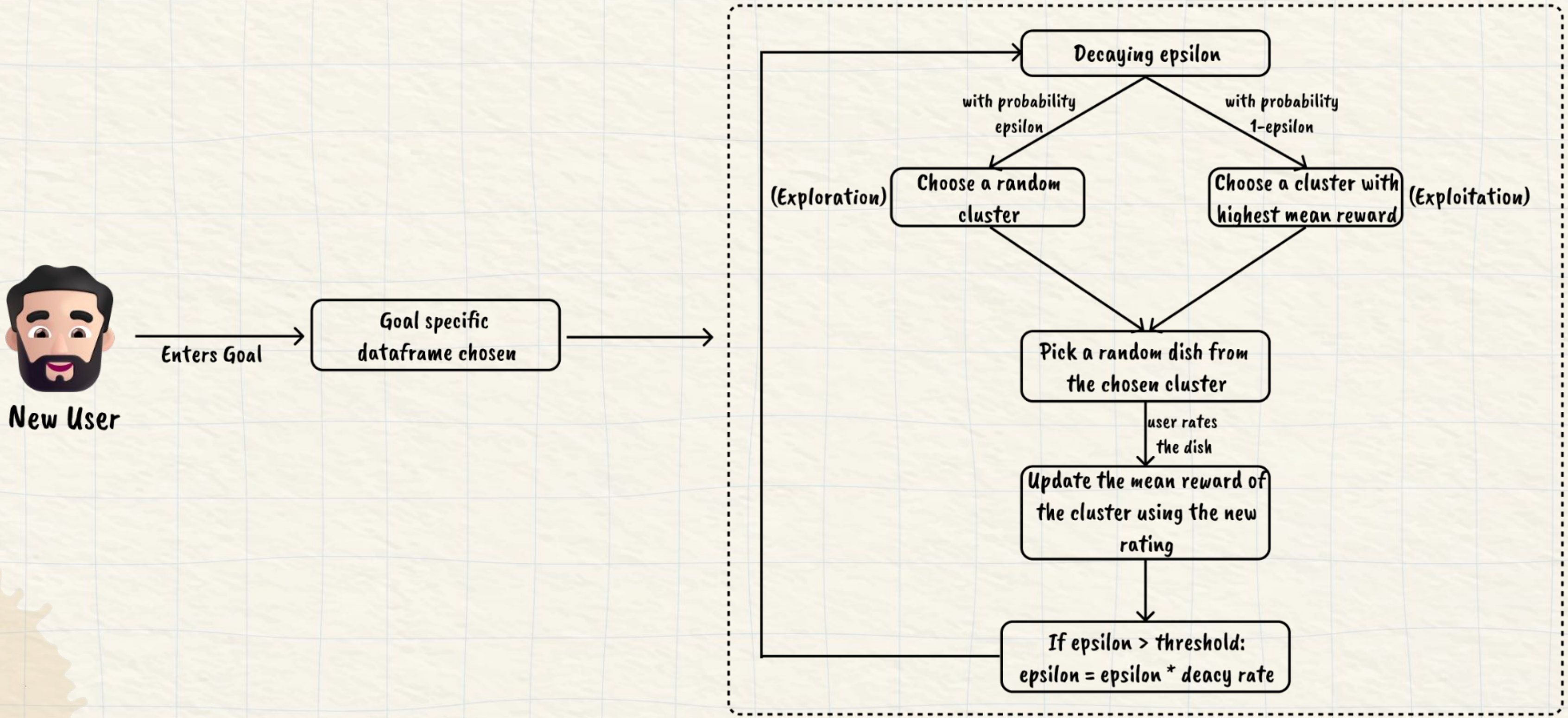




```
Cluster Centers:
[[ 35.48281093 154.99727481  31.22710167  27.14692591]
 [ 59.43542627  67.42255599  56.24081259  12.91914758]
 [ 23.62900045 230.33610679   5.40671645  43.75584999]]
```

Greedy Epsilon MAB

Greedy-epsilon MAB



New User

Enters Goal

Goal specific dataframe chosen

Decaying epsilon

with probability epsilon

with probability 1-epsilon

(Exploration)

Choose a random cluster

Choose a cluster with highest mean reward

(Exploitation)

Pick a random dish from the chosen cluster

user rates the dish

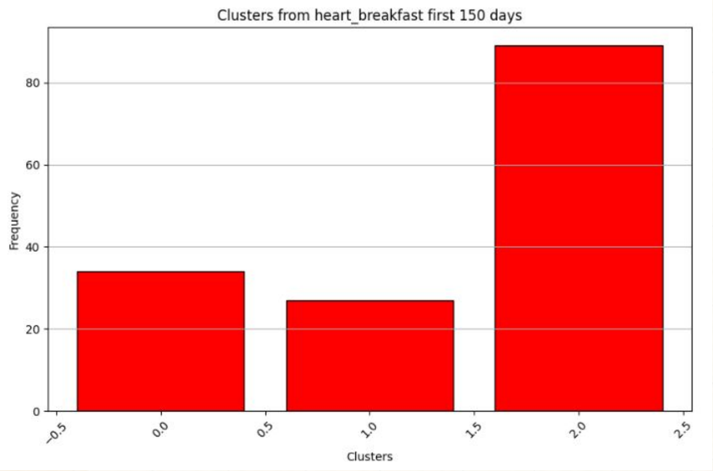
Update the mean reward of the cluster using the new rating

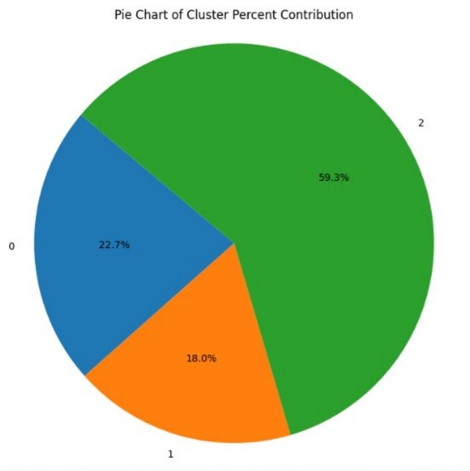
If epsilon > threshold:
epsilon = epsilon * deacy rate

Results

To test run our system, we have generated 365 random ratings that would be given for breakfast recommendations of each day for 365 days. Below are the visualizations depicting the frequency of chosen arms (clusters).

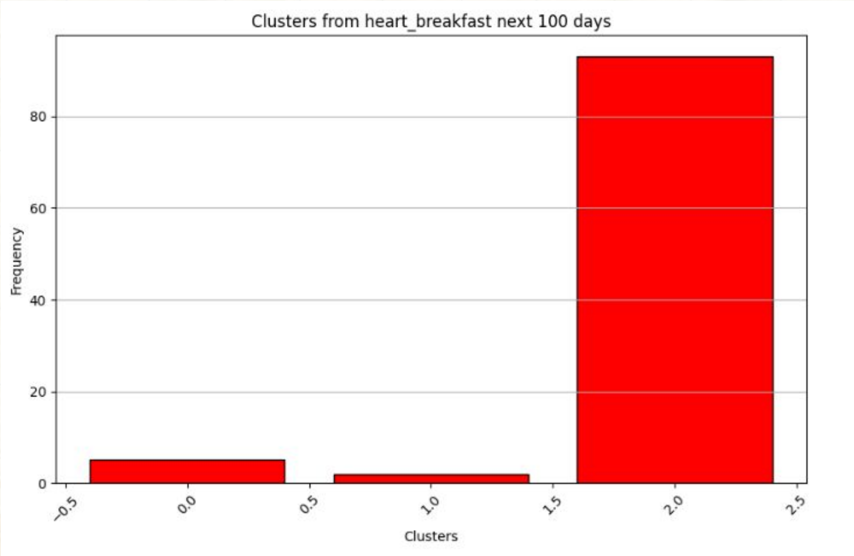
1. For first 150 days.

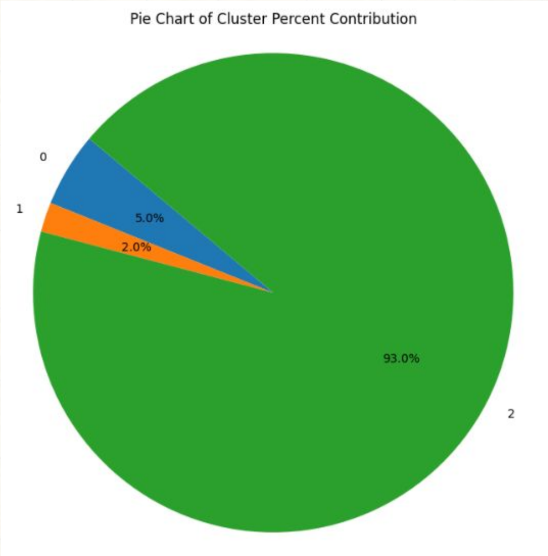




Results

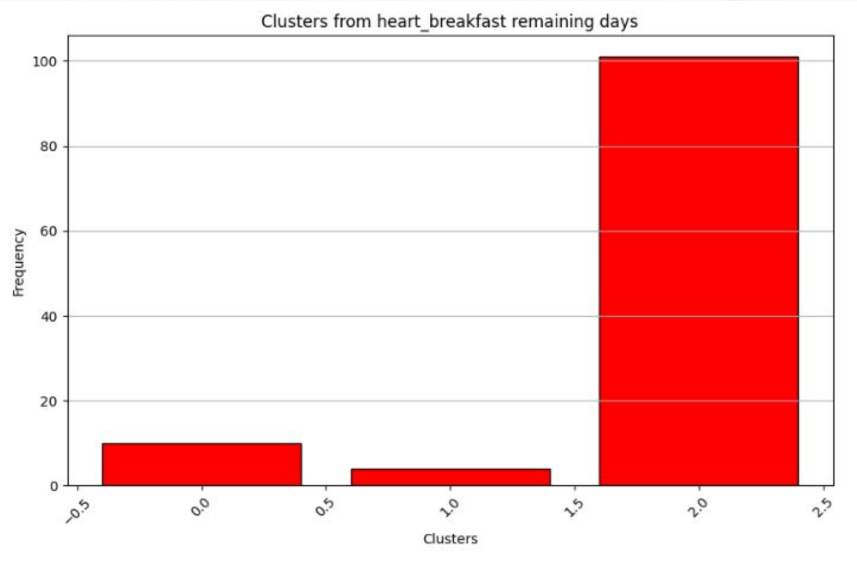
2. For Next 100 days.

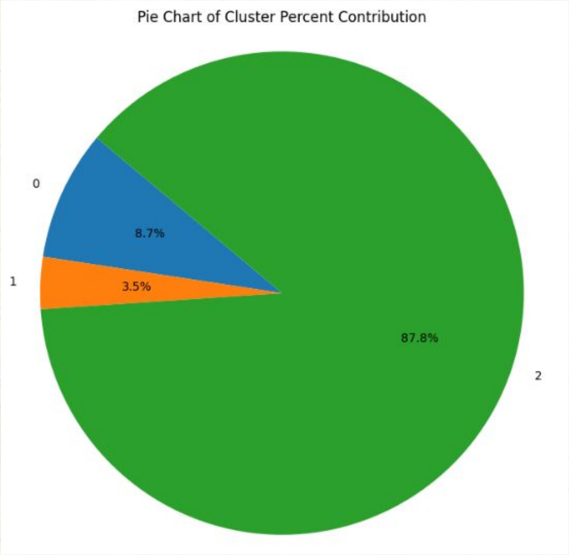




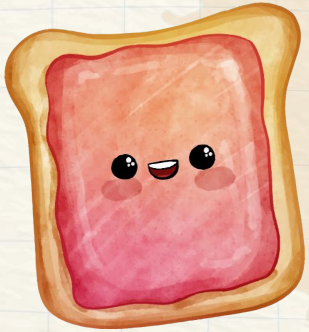
Results

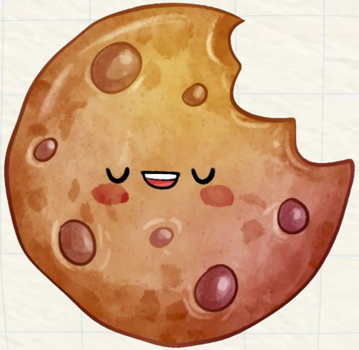
3. For Remaining days.





Challenges & Improvements

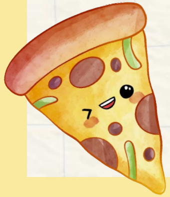




1. Clustering of recipes: Clustering the recipes was a challenge as we wanted the clustering to be relevant to the taste of the food item. According to the research paper [1] nutrients like protein, carbohydrates, fats, etc influence the taste of the food item. Hence we applied kmeans on the nutrient values after scaling them to per 1000 calories.

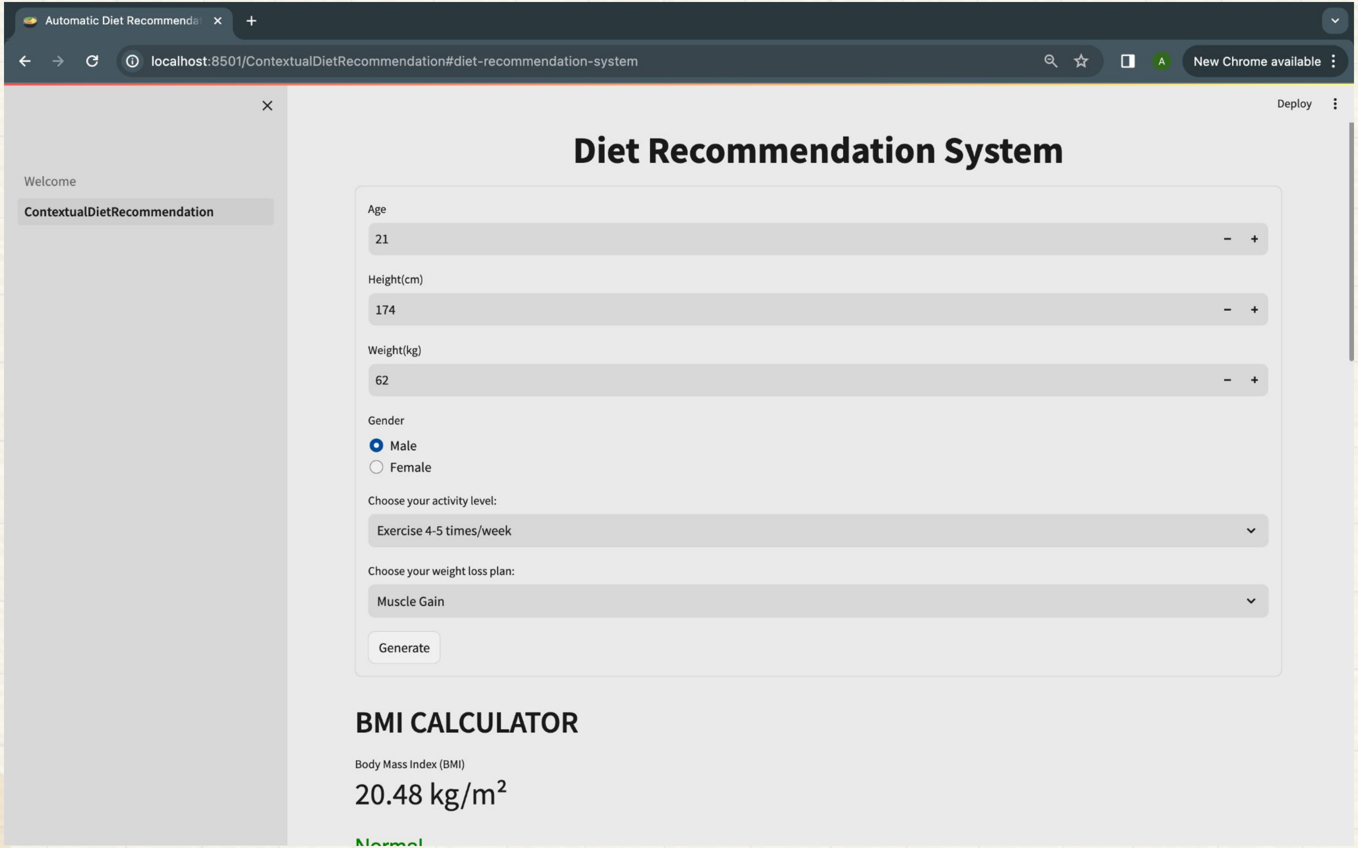
2. Non desirable cluster being chosen majority of times during exploration initially: It so may happen during the initial phase where exploration is given priority, a specific cluster might get recommended everytime randomly and even though the user doesn't prefer that cluster, the same cluster would be recommended during exploitation too as others mean would still be 0. Hence to tackle this initial mean ratings have been set to 2 instead of 0.

3. Epsilon becoming too small: Due to decaying epsilon, epsilon may become too small and exploration would never happen. This would lead to similar recommendations from there on. Hence we have put a lower bound on the value of epsilon such that it won't decay below the lower bound so that a minimal amount of exploration would always persist.



[1] van Langeveld, Astrid & Gibbons, Shannon & Koelliker, Yvonne & Civille, Gail & de Vries, Jeanne & Graaf, Cees & Mars, Monica. (2016). The relationship between taste and nutrient content in commercially available foods from the United States. Food Quality and Preference. 57. 10.1016/j.foodqual.2016.10.012.

Deployment on Streamlit



Automatic Diet Recommenda
localhost:8501/ContextualDietRecommendation#diet-recommendation-system
New Chrome available
Deploy
Welcome
ContextualDietRecommendation

Diet Recommendation System

Age

21

Height(cm)

174

Weight(kg)

62

Gender

● Male

○ Female

Choose your activity level:

Exercise 4-5 times/week

Choose your weight loss plan:

Muscle Gain

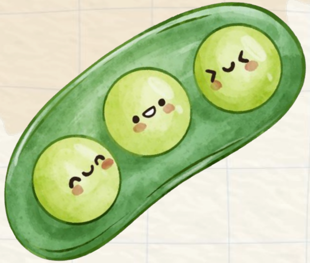
Generate

BMI CALCULATOR

Body Mass Index (BMI)

20.48 kg/m²

Normal



Thanks!