USC University of Southern California

Project 2 Meal Planning Problem

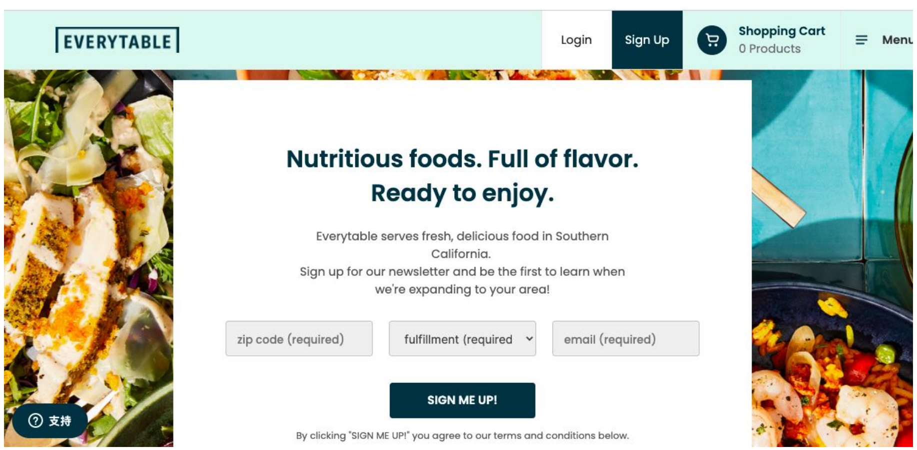
Group 3: Jiacheng Yu, Ting Pan, Lei Lei, Ruixin Wu

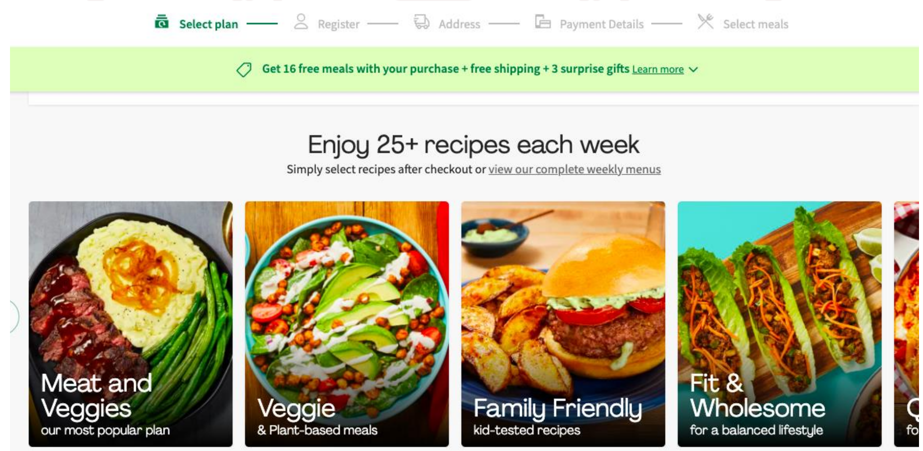
Project Overview

In this project, we are going to generate weekly meal plans for our team members based on the personal ratings information and constraints. It can be divided into two parts:

1) **Matrix Completion**: solving MAP problem to fulfill the sparse ratings data by Coordinate Ascent Algorithm.

2) **Mixed Integer Programming:** constructing a Mixed Integer Programming (MIP) model to maximize the sum of ratings based on our predicted matrix and constraints as well as making the arrangement of our chefs.





USC

Part I

Matrix Completion Problem

Overview

In part one, our aim is to achieve matrix completion to obtain the predicted ratings matrix MP.

First, we convert the json format data into pandas dataframe, build the rating matrix M and extract the personal ratings information (rows: customers, columns: dishes, cells: ratings).

Second, we enlarge the matrix M with additional rows being the ratings from each team member

Third, we fulfill the sparse matrix through matrix completion technique(MAP) which references Coordinate Ascent Algorithm.

Data Cleaning

- Read ‘result.json’ file in Python
- Remove NaN values in ‘title’
- Standardize values in ‘personal_rating’

	title	rating	calories	sodium	fat	protein	personal_rating
32	Skillet Chicken and Zucchini Enchiladas with T...	5.0	458.0	1825.0	21.0	31.0	[[enpeco2 from New Orleans, LA , 5], [clararaa...
33	Slow-Cooker Green Chicken Chili	5.0	NaN	NaN	NaN	NaN	[[kwkennedy from Denver, CO , 3], [wheedle fro...
34	NaN	NaN	NaN	NaN	NaN	NaN	NaN
35	Roasted Apricot Chicken with Mint and Sage But...	4.5	730.0	1231.0	46.0	58.0	[[iambenjago from Washington, DC , 3], [ellen2...
36	22-Minute Pad Thai	4.0	551.0	2117.0	21.0	26.0	None
37	3-Ingredient Creamy Pumpkin Pasta	4.0	694.0	508.0	29.0	17.0	[[wannabecook79 , 4]]
38	Savory Dutch Baby for Two	5.0	349.0	377.0	22.0	15.0	[[websherpa.ca8425 from Burlington, ON , 5], [...

USC

Create the Sparse Matrix M

Generate the sparse matrix M (2884*289) from original dataframe

```
a=[]
for i in range(0,len(df2)):
    for j in range(0,len(df2[i])):
        a.append(df2[i][j][0])
a=pd.DataFrame(a)
b=a.drop_duplicates()
```

```
b=b.values.tolist()
```

```
M=np.zeros((len(b),len(df2)))
gp3=-1
for i in range(0,len(df2)):
    for j in range(0,len(df2[i])):
        for t in range(len(b)):
            if(df2[i][j][0]==b[t][0]):
                gp3=t
        M[gp3][i]=df2[i][j][1]
        gp3=-1
```

```
print(M)
M.shape
```

```
[[5. 0. 0. ... 0. 0. 0.]
 [5. 0. 0. ... 0. 0. 0.]
 [5. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 5.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 4.]]

(2884, 289)
```

Enlarge the Sparse Matrix M

- Each team member rate 8 different dishes
- 2884+4 = 2888 rows and 289 columns

```
print(M)
M.shape
```

```
[[5. 0. 0. ... 0. 0. 0.]
 [5. 0. 0. ... 0. 0. 0.]
 [5. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

(2888, 289)
```

Complete the Matrix M

- Follow the Matrix Factorization instructions on Page 13 to write functions

Initialize each v_j. For example, generate $v_j \sim N(0, \lambda^{-1}I)$.

for each iteration **do**

- **for** $i = 1, \ldots, N_1$ **update user location**

$$u_i = \left(\lambda\sigma^2 I + \sum_{j\in\Omega_{u_i}} v_j v_j^T\right)^{-1}\left(\sum_{j\in\Omega_{u_i}} M_{ij} v_j\right)$$

- **for** $j = 1, \ldots, N_2$ **update object location**

$$v_j = \left(\lambda\sigma^2 I + \sum_{i\in\Omega_{v_j}} u_i u_i^T\right)^{-1}\left(\sum_{i\in\Omega_{v_j}} M_{ij} u_i\right)$$

```python
v=np.zeros((10,289))
for i in range(10):
    v[i]=np.random.normal(0,0.5,289)
v=v.T #10*289->289*10
u=np.zeros((2888,10))
for t in range(200):
    for i in range(2888):
        u[i]=(1/(2*1+sum_v(i,v)))*(sum_Mv(i,v))
    for j in range(289):
        v[j]=(1/(2*1+sum_u(j,u)))*(sum_Mu(j,u))
```

We have 200 iterations to find the Ui, Vj, and they converge in the end

USC

Complete the Matrix M

- Get the completed predicted Matrix: (MP = U*V.T)
- Ensure ratings of user i rounded to the closest integer

```
MP=np.dot(u,v.T)
MP=np.around(MP)
```

	0	1	2	3	4	5	6	7	8	9	...	279	280	281	282	283	284	285	286	287	288
0	5.0	3.0	3.0	2.0	2.0	4.0	4.0	0.0	0.0	2.0	...	0.0	4.0	-0.0	4.0	4.0	4.0	4.0	4.0	0.0	1.0
1	5.0	3.0	3.0	2.0	2.0	4.0	4.0	0.0	0.0	2.0	...	0.0	4.0	-0.0	4.0	4.0	4.0	4.0	4.0	0.0	1.0
2	5.0	3.0	3.0	2.0	2.0	4.0	4.0	0.0	0.0	2.0	...	0.0	4.0	-0.0	4.0	4.0	4.0	4.0	4.0	0.0	1.0
3	4.0	2.0	3.0	2.0	2.0	3.0	3.0	0.0	0.0	2.0	...	0.0	4.0	-0.0	4.0	4.0	4.0	4.0	3.0	0.0	0.0
4	5.0	3.0	3.0	2.0	2.0	4.0	4.0	0.0	0.0	2.0	...	0.0	4.0	-0.0	4.0	4.0	4.0	4.0	4.0	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2883	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	-0.0	1.0	...	0.0	1.0	0.0	1.0	1.0	2.0	1.0	1.0	-2.0	4.0
2884	3.0	2.0	3.0	2.0	2.0	3.0	3.0	0.0	-0.0	2.0	...	0.0	4.0	0.0	3.0	4.0	3.0	3.0	3.0	-0.0	1.0
2885	4.0	3.0	3.0	2.0	2.0	4.0	4.0	0.0	-0.0	2.0	...	0.0	4.0	0.0	4.0	4.0	4.0	4.0	4.0	-0.0	1.0
2886	4.0	3.0	4.0	3.0	3.0	4.0	4.0	0.0	-0.0	3.0	...	0.0	5.0	0.0	4.0	5.0	5.0	5.0	4.0	-0.0	1.0
2887	4.0	3.0	3.0	2.0	2.0	4.0	4.0	0.0	-0.0	2.0	...	0.0	4.0	0.0	4.0	4.0	4.0	4.0	4.0	-0.0	1.0

2888 rows × 289 columns

Further Data Cleaning

- We wanted to have more known ratings to predict unknown ratings at the beginning

- Remove 33 rows with NaN values in ‘calories’, ‘sodium’, ‘fat’, ‘protein’ now (Also remove those columns in Matrix)

- 289-33 = 256 rows → 256 dishes

	index	title	rating	calories	sodium	fat	protein	personal_rating
0	0	Curried Lentil, Tomato, and Coconut Soup	5.0	437.0	667.0	28.0	13.0	[[rhaeredekop from Winnipeg, MB , 5], [lizgold...
1	1	Roasted Butternut Squash with Herb Oil and Goa...	4.5	175.0	576.0	9.0	4.0	[[ilyssa2 from NY, NY , 5], [dory92064 from Sa...
2	2	Pumpkin Muffins	4.0	364.0	183.0	11.0	6.0	[[mtnmeye , 3], [greenstein.rebecca9820 from N...
3	3	Chopped Salad with Shallot Vinaigrette, Feta, ...	5.0	170.0	413.0	13.0	6.0	[[brushjl from solon, oh , 5], [tatyana_poirie...
4	4	Grain Salad with Olives and Whole-Lemon Vinaig...	3.5	330.0	483.0	19.0	8.0	[[auntwebbie from Allen, TX , 5], [krf from Be...
...	...	...	...	...	...	...	...	...
251	292	White Chicken Chili	5.0	534.0	968.0	14.0	45.0	[[lmanderson from Maryland , 5], [debkane from...
252	293	One-Pot Curried Cauliflower with Couscous and ...	4.5	606.0	1365.0	15.0	27.0	[[mkopke from Toronto, Canada , 4], [akraemer1...
253	295	Hummus Dinner Bowls with Spiced Ground Beef an...	4.5	329.0	327.0	26.0	20.0	[[bradley2 from Foodietown, ,PA , 4], [chaurie...
254	296	Autumn Kale Salad	5.0	287.0	329.0	22.0	4.0	[[dottie60 from Boston , 5], [zeta from Victor...
255	297	Pumpkin Icebox Pie With Snickerdoodle Crust	4.5	3597.0	1869.0	223.0	33.0	[[Lois33 from EGR, Michigan , 5], [mccoyj25 fr...

256 rows × 8 columns

```
print(MP)
MP.shape
```

```
[[ 5.  3.  4. ...  4. -0.  1.]
 [ 5.  3.  4. ...  4. -0.  1.]
 [ 5.  3.  4. ...  4. -0.  1.]
 ...
 [ 4.  3.  3. ...  4. -0.  0.]
 [ 4.  3.  3. ...  4. -0.  1.]
 [ 4.  3.  3. ...  4. -0.  1.]]

(2888, 256)
```

Part II

Mixed Integer Programming
(Meal Plan Recommendation)

Overview

In part two, our aim is to maximize the sum of team members' ratings with given constraints and decide who to cook each dinner

First, based on the predicted matrix MP we got from part one, we choose the last four rows. Then we set up the variables, objective function and constraints in order to maximize ratings and satisfy requirements (e.g. dishes arrangement, nutrients, budget).

By solving the problem, finally we got the optimal dishes arrangement for each dinner with maximal total ratings, and meets all constraints and requirements.

Then we set up another Mixed Integer Model to assign our team members to cook for each dinner based on equity and availability.

Model Set-up

- **Basic Assumption**

a) For our group members, all members eat one same dish for each dinner from Monday to Friday(5-day)
b) We eat 1 meal(only dinner) together one day cooked by one member.
c) Goal is to find out the arrangement of the 256 dishes for each dinner and the optimal cooking arrangement

- **Binary Variables**

$$x[\underbrace{i}_{\text{day}},\ \underbrace{t}_{\text{dish}}] \in \{0, 1\}$$

```
# Variables definition
@variable(model, x[1:5,1:256], Bin)
# Recommend one dish for each meal
```

$i \in [1:5]$ $t \in [1:256]$

$x[i,t]=1$ means we choose dish t on day i

$x[i,t]=0$ means we don't choose dish t on day i

USC

Model Set-up

- **Objective Function**

To maximize the **<u>Total sum</u>** of the **<u>total ratings made by four members</u>** of **<u>each dish we choose</u>** for **<u>each dinner</u>**.

```
@objective(model, Max, sum(x[i,t]*(sum(df1[i,t] for i in 1:4))
for i in 1:5 for t in 1:256))
```

- **Constraints (Arrangement)**

a) We recommend only one dish for each meal.

```
for i in 1:5
    @constraint(model,sum(x[i,t] for t in 1:256)==1)
end ✓
```

a) Ensure that each dish only appears once in our recommendation.

```
for t in 1:256
    @constraint(model,
    sum(x[i,t] for i in 1:5)<=1)
end ✓
```

USC

Model Set-up

- **Constraints(nutrients)**

a) We recommend only one dish for each meal.
b) Ensure that each dish only appears once in our recommendation.
c) The nutritional requirements per meal are shown as follows:

	min	max
Calories	450	800
Fat	44/3	67/3
Sodium	-	2300/3
Protein	50/3	175/3

```
for i in 1:5
#calories: From 450 to 800 calories per meal .
    @constraint(model,sum(x[i,t]*calories[t] for t in 1:256)<=800)
    @constraint(model,sum(x[i,t]*calories[t] for t in 1:256)>=450)

# A high fat intake is more than 35 percent of your calories, while a low intake is less than
# We have a lower bound and upper bound for fat

    @constraint(model,sum(x[i,t]*fat[t] for t in 1:256)>=44/3)
    @constraint(model,sum(x[i,t]*fat[t] for t in 1:256)<=77/3)

# The sodium RDI is less than 2,300 milligrams per day for adults
    @constraint(model,sum(x[i,t]*sodium[t] for t in 1:256)<=2300/3)

#the protein is between 50 and 175 for an adult per day
    @constraint(model,sum(x[i,t]*protein[t] for t in 1:256)>=50/3)
    @constraint(model,sum(x[i,t]*protein[t] for t in 1:256)<=175/3)
```

USC

Model Set-up

- **Constraints(others)**

a) We recommend only one dish for each meal.
b) Ensure that each dish only appears once in our recommendation.
c) The nutrients requirement is shown as follows.
d) Our budget per person per meal is $20.
We assumed the price of each dish ranges from $6 to $25.
Therefore, we generated 256 numbers range between 6 to 25 randomly to represent the prices.
Then we build budget constraints

Arrangement

Nutrients

Budget

```
price = rand(6:25,256)
```

```
#our budget per person per day is 20 dollars
    @constraint(model,sum(x[i,t]*price[t] for t in 1:256)<=20)
end
```

USC

Running Result

```
@show objective_value(model)  75.0
```

- **Optimal Ratings**

  The optimal objective value of the model is **75**, indicating that the sum of ratings for dishes we choose is 75. Because we have 4 people, 5 days, so the average rating per meal we recommend is 75/20=**3.75**

- **Meal Plan Recommendation**

```
1   73
2   91
3   100
4   203
5   213
```

```
73                  Quick Chicken Tikka Masala
91                Roasted Beet Tzatziki Salad
100     Easy Lamb Tagine with Pomegranate
203                 Easy General Tso's Chicken
213                   Winter Squash Agrodolce
Name: title, dtype: object
```

Cooking time (Considering Equity)

In this part, we set up another Mixed Integer Model to assign our team members to cook for each dish based on **availability** and **we also consider equity (Creative idea)(Linear)**

Time	Monday	Tuesday	Wednesday	Thursday	Friday	Sum	Coefficient
Jiacheng	0	0	0	0	1	1	1-1/5=4/5
Ting	1	0	1	0	1	3	1-3/5=2/5
Ruixin	1	0	1	0	1	3	1-3/5=2/5
Lei	0	1	1	1	1	4	1-4/5=1/5

Constraint: Members with more time are assigned with more work.

Objective function: $$Max\ \frac{4}{5} * S_1 + \frac{2}{5} * S_2 + \frac{2}{5} * S_3 + \frac{1}{5} * S_4$$

S_i : the sum of dinners made by member i

We try to prioritize assigning work to members with less cooking time when they are available

USC

Cooking time

1. Availability

```
# Constraints
# To ensure there is someone cooking for each dinner
for j in 1:5
    @constraint(model1,sum(deci[i,j]  for i in 1:4)==1)
end  ✓
```

```
# To ensure the member is available to cook for that dinner
for j in 1:5
    for i in 1:4
    @constraint(model1,deci[i,j]<=time1[i][j])
    end
end  ✓
```

Cooking time (Considering Equity)

1. Considering Equity

```
# Time matrix
time1 = [[0,0,0,0,1],
[1,0,1,0,1],
[1,0,1,0,1],
[0,1,1,1,1]]  4-element Vector{Vector{Int64}}:

#avali is a 1D array of sum of dinners made by each members
avail=[0.0,0.0,0.0,0.0]  4-element Vector{Float64}:
for i in 1:4
    for j in 1:5
        avail[i]=avail[i]+time1[i][j]
    end
end  ✓
avail  4-element Vector{Float64}:
```

```
# To rank user time availability with their index
rank=[-1,-1,-1,-1]  4-element Vector{Int64}:
function ranking(rank,avail)
    for i in 1:4
        min=999
        for j in 1:4
            if(check_index(rank,j))
                if(avail[j]<min)
                    rank[i]=j
                    min=avail[j]
                end
            end
        end
    end
    return rank
end  ranking (generic function with 1 method)
rank=ranking(rank,avail)  4-element Vector{Int64}:
```

```
# To ensure members with more time do more cooking
for i in 1:3
    t1=rank[i]
    t2=rank[i+1]
    @constraint(model1,sum(deci[t1,j]  for j in 1:5)<=sum(deci[t2,j]  for j in 1:5))
end  ✓
```

Cooking time (Considering Equity)

```
@objective(model1, Max, sum(avail[i]*
    sum(deci[i,j] for j in 1:5) for i in 1:4))
optimize!(model1)  √
@show objective_value(model1)  2.0
deci1 = value.(deci)  4×5 Matrix{Float64}:
```

$$Max\ \frac{4}{5} * S_1 + \frac{2}{5} * S_2 + \frac{2}{5} * S_3 + \frac{1}{5} * S_4$$

S_i : the sum of dinners made by member i

Result:

```
4×5 Matrix{Float64}:
 0.0  0.0  0.0  0.0  1.0
 0.0  0.0  1.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0
 0.0  1.0  0.0  1.0  0.0
```

Assignment

Time	Monday	Tuesday	Wednesday	Thursday	Friday
Jiacheng	0	0	0	0	1
Ting	1	0	1	0	1
Ruixin	1	0	1	0	1
Lei	0	1	1	1	1

Decision	Monday	Tuesday	Wednesday	Thursday	Friday
Jiacheng	0	0	0	0	1
Ting	0	0	1	0	0
Ruixin	1	0	0	0	0
Lei	0	1	0	1	0

Our Dinner Menu and Chief Arrangement

	Dinner	Chief
Monday	Quick Chicken Tikka Masala	Ruixin Wu
Tuesday	Roasted Beet Tzatziki Salad	Lei Lei
Wednesday	Easy Lamb Tagine with Pomegranate	Ting Pan
Thursday	Easy General Tso’s Chicken	Lei Lei
Friday	Winter Squash Agrodolce	Jiacheng Yu

Future work and innovation

- Although our groups don't have any food allergies, we can enrich our MIP model by taking more practical requirements into consideration. For example, we can add distance restrictions (Group members' and markets' addresses).

- Given that we generated the price of each dishes randomly within a range, we can do further research on the prices of dishes to make them more realistic.

- For all 256 dishes in our dataset, some of them might not suitable for dinner (like soup, salad) and we have arranged "Chicken" in two days of a week. Thus, in the future, we can try to categorize those dishes to make our suggestion more diversified.

- Based on the theory, we can design a smart application to recommend meal plans for more and more people.

USC

Thanks for listening!