Kevin Ou
MSDS 460
Assignment 1

The Diet Problem is a known example of constrained optimization introduced by Stigler (1945) and later expanded upon by Dantzig (1990). The goal of the problem is to find the minimum-cost diet that meets daily nutritional requirements, such as caloric intake, sodium limits, and essential nutrients like protein, vitamins, and minerals. I have constructed a personalized diet of things I ate often in order to save money, time or effort (while not usually the healthiest options) to see if I was able to meet the daily nutritional requirements. I used the Python PuLP library to formulate and solve the problem as a linear programming task.

The five packaged food items I selected are Shin Ramen, Kimchi, Eggs, Vienna Sausage, and Arizona Green Tea, and the nutritional labels provided the required information for constructing the diet problem. The prices are calculated here:

- Shin Ramen: $1.11 per pack (1 serving) 18 pack for $20
- Kimchi: $5.49 per container (54 servings), $0.10 per serving
- Eggs: $0.42 per egg (1 serving) 24 eggs for $20
- Vienna Sausage: $0.83 per can (1 serving) 18 cans per box for $15
- Arizona Green Tea: $0.75 per can (1 serving)

I obtained these prices by dividing the total cost of each product by the number of servings listed on the nutritional labels and these prices are used as coefficients in the objective function, which will minimize the total cost of the diet hopefully. The nutritional values are obtained from Figures 1 through 5 which details each item and their nutritional values.

In simple terms, the goal of this problem is to figure out how many servings of five different packaged food items in my diet I should eat in a week to meet my nutritional needs, while also keeping my food costs as low as possible. The decision variables in this problem represent the number of servings of each food item consumed per week. These variables are denoted as x_1 for Shin Ramen, $x_2$for Kimchi, $x_3$for Eggs, $x_4$for Vienna Sausages, and $x_5$for Arizona Green Tea.

The objective function tries to minimize the total cost of the weekly diet with the formula shown as:

$$Minimize\ C\ =\ 1.11x_1 + 0.10x_2 + 0.42x_3 + 0.83x_4 + 0.75x_5$$

The constraints are to ensure that the diet meets the weekly nutritional requirement for calories, protein, sodium, and any other that may be necessary like Calcium, Iron, or Potassium.

- Calories: $510x_1 + 10x_2 + 70x_3 + 160x_4 + 130x_5 \geq 14,000$
- Protein: $10x_1 + 1x_2 + 6x_3 + 10x_4 \geq 350$
- Sodium: $1390x_1 + 180x_2 + 70x_3 + 790x_4 \leq 35,000$
- Vitamin D: $x_3 \geq 140$
- Calcium: $30x_3 + 130x_4 \geq 9,100$
- Iron: $2.9x_1 + 0.9x_3 + 1.1x_4 \geq 126$
- Potassium: $260x_1 + 74x_2 + 70x_3 + 110x_4 \geq 32,900$

These coefficients were gathered from the nutritional labels that are found in the appendix. I implemented the linear programming model using Python's PuLP library as shown in Figure 6 and 7 and the model provided the optimal number of servings for each food item to meet the nutritional requirements at the lowest cost. At first glance, I think that I can achieve a balanced diet mostly through a combination of Shin Ramen, Eggs, and Vienna Sausages as they contribute significantly more than the other components. But the model had other answers and that was to forgo almost everything in the diet and stick with just eggs and kimchi. The solution (from Figure 8) showed that I have 0 servings of Arizona Green Tea, 0 servings of Shin Ramen, 0 servings of Vienna Sausages, and 449.0566 servings of eggs, and 19.8911321 servings of Kimchi. I personally think this solution is hilarious because it is telling me that I can basically solely live off of eggs based on nutritional value. The minimum cost given was $190.62. Now if we were to require at least one serving of each food item or meal during the week, it would not change too much except the servings of Kimchi. The constraint added would be: $x_1, x_2, x_3, x_4, x_5 \geq 1$. This constraint was added to Figures 6 and 7 as Figure 9. The solution (from Figure 10) then would become 1 serving of Arizona Green Tea, 461.8221 servings of Eggs, 2.7358491 servings of Kimchi, 1 serving of Shin Ramen, and 1 serving of Vienna Sausage. By adding this constraint, the revised solution did not add much more variety except for the minimum serving and it slightly increased the total cost to $196.63. The change in cost reflects the trade-off between variety and expense.

For the next part, I extended my hand to ChatGPT (https://chatgpt.com/), an LLM-based service provided by OpenAI, to specify and solve the diet problem. At first, I asked if the LLM was familiar with the Diet Problem and asked it to craft a model where the 5 foods were the prepackaged items I mentioned earlier. It built a model that did not have the correct prices or nutritional values and then updated it with the correct prices but I was unable to upload a picture due to using the free version of ChatGPT. I then asked it to develop a python code using the PuLP library to craft the model in code. The LLM performed well with standard diet problem formulations, but it also assumed many inaccurate assumptions about the nutritional values and minimums regarding the foods. I asked ChatGPT if it had the capabilities to perform the tasks and models that were in question and it responded with actually doing the tasks and making the models as an example and then proceeded to ask if there were any modifications that I could add to it. I was able to continue building upon the conversation and tailor it to my prices that I used, but found some trouble where I would have to type out the entirety of the nutritional values as I was unable to add pictures to the chat. From Figure 11, I noticed that their way of solving the linear programming equation was very crude and essentially, I had to build out the code in a more brute force kind of style. It had written out the constraints and objective function in the whole equation format while my personal style of coding used the pulp library to the fullest while implementing lpSum function. My experience suggests that it can be a helpful tool for initial problem setup but requires more user oversight for detailed customization. The complete conversation and code generation have been included in the GitHub repository.

Appendix:

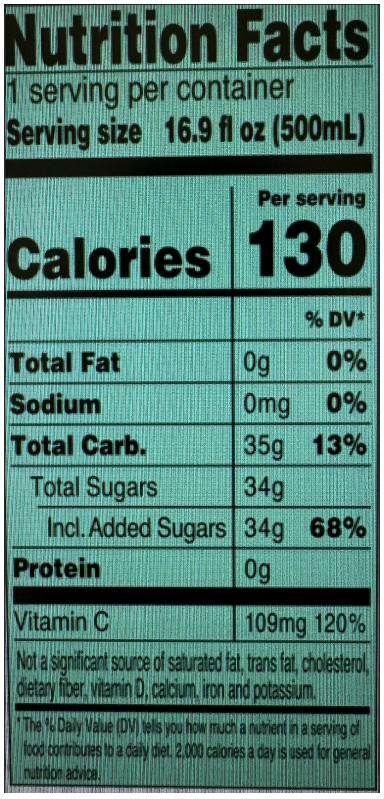

Figure 1: Arizona Green Tea Can

Figure 2: Shin Ramen Packet

Figure 3: Vienna Sausage

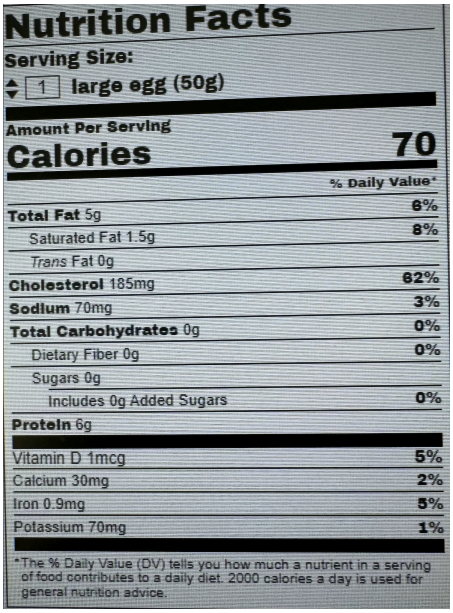

Figure 4: Costco Organic Egg

Figure 5: Kimchi container

```python
import pulp

#Creating the Linear Programming Problem
lp_problem = pulp.LpProblem("Diet_Optimization", pulp.LpMinimize)

#Decision variables: servings of each item
food_items = ['Shin Ramen', 'Kimchi', 'Eggs', 'Vienna Sausage', 'Arizona Green Tea']
costs = [1.11, 5.49 / 54, 0.42, 0.83, 0.75] #Prices based on what I purchase them for usually at local Costco
servings = pulp.LpVariable.dicts("Servings", food_items, lowBound=0, cat='Continuous')

#Nutrition data from label (per serving)
calories = [510, 10, 70, 160, 130]
protein = [10, 1, 6, 10, 0]
sodium = [1390, 180, 70, 790, 0]
vitamin_d = [0, 0, 1, 0, 0]
calcium = [0, 0, 30, 130, 0]
iron = [2.9, 0, 0.9, 1.1, 0]
potassium = [260, 74, 70, 110, 0]

#Weekly requirements (7 day totals)
min_calories = 7 * 2000
min_protein = 7 * 50
max_sodium = 7 * 5000
min_vitamin_d = 7 * 20
min_calcium = 7 * 1300
min_iron = 7 * 18
min_potassium = 7 * 4700

#Objective function: minimizing cost
lp_problem += pulp.lpSum([costs[i] * servings[food_items[i]] for i in range(len(food_items))]), "Total Cost"

# Constraints

#Calories constraint (minimum)
lp_problem += pulp.lpSum([calories[i] * servings[food_items[i]] for i in range(len(food_items))]) >= min_calories, "MinCalories"

#Protein constraint (minimum)
lp_problem += pulp.lpSum([protein[i] * servings[food_items[i]] for i in range(len(food_items))]) >= min_protein, "MinProtein"

#Sodium constraint (maximum)
lp_problem += pulp.lpSum([sodium[i] * servings[food_items[i]] for i in range(len(food_items))]) <= max_sodium, "MaxSodium"
```

Figure 6: Linear Programming Code Pt. 1

```
#Sodium constraint (maximum)
lp_problem += pulp.lpSum([sodium[i] * servings[food_items[i]] for i in range(len(food_items))]) <= max_sodium, "MaxSodium"

#Vitamin D constraint (minimum)
lp_problem += pulp.lpSum([vitamin_d[i] * servings[food_items[i]] for i in range(len(food_items))]) >= min_vitamin_d, "MinVitaminD"

#Calcium constraint (minimum)
lp_problem += pulp.lpSum([calcium[i] * servings[food_items[i]] for i in range(len(food_items))]) >= min_calcium, "MinCalcium"

#Iron constraint (minimum)
lp_problem += pulp.lpSum([iron[i] * servings[food_items[i]] for i in range(len(food_items))]) >= min_iron, "MinIron"

#Potassium constraint (minimum)
lp_problem += pulp.lpSum([potassium[i] * servings[food_items[i]] for i in range(len(food_items))]) >= min_potassium, "MinPotassium"

lp_problem.solve()

#Results
print(f"Status: {pulp.LpStatus[lp_problem.status]}")
for v in lp_problem.variables():
    print(f"{v.name} = {v.varValue}")

#Total cost
print(f"Total Cost of the Diet: ${pulp.value(lp_problem.objective):.2f}")

```

Figure 7: Linear Programming Code Pt. 2

```
Status: Optimal
Servings_Arizona_Green_Tea = 0.0
Servings_Eggs = 449.0566
Servings_Kimchi = 19.811321
Servings_Shin_Ramen = 0.0
Servings_Vienna_Sausage = 0.0
Total Cost of the Diet: $190.62
```

Figure 8: Output of Linear Programming Code without minimum variety constraint

```
for item in food_items:
    lp_problem += servings[item] >= 1, f"Min_One_Serving_{item}"

```

Figure 9: Addition to Linear Programming for minimum variety constraint

```
Status: Optimal
Servings_Arizona_Green_Tea = 1.0
Servings_Eggs = 461.8221
Servings_Kimchi = 2.7358491
Servings_Shin_Ramen = 1.0
Servings_Vienna_Sausage = 1.0
Total Cost of the Diet: $196.93
```

Figure 10: Output of Linear Programming Code with minimum variety constraint

```python
#ChatGPT assisted Code

import pulp

# Create a linear programming problem
diet_problem = pulp.LpProblem("DietProblem", pulp.LpMinimize)

# Decision variables
x1 = pulp.LpVariable("Shin_Ramen", lowBound=0, cat='Continuous')
x2 = pulp.LpVariable("Arizona_Green_Tea", lowBound=0, cat='Continuous')
x3 = pulp.LpVariable("Kimchi", lowBound=0, cat='Continuous')
x4 = pulp.LpVariable("Eggs", lowBound=0, cat='Continuous')
x5 = pulp.LpVariable("Vienna_Sausages", lowBound=0, cat='Continuous')

# Objective function
diet_problem += (1.11 * x1 + 0.75 * x2 + 0.10 * x3 + 0.42 * x4 + 0.83 * x5, "Total Cost")

# Constraints
diet_problem += (380 * x1 + 120 * x2 + 50 * x3 + 70 * x4 + 200 * x5 >= 2000, "Calories")
diet_problem += (10 * x1 + 0 * x2 + 1 * x3 + 6 * x4 + 10 * x5 >= 60, "Protein")
diet_problem += (1 * x1 + 0 * x2 + 2 * x3 + 0 * x4+ 0 * x5 >= 25, "Fiber")

# Solve the problem
diet_problem.solve()

# Print the results
print("Status:", pulp.LpStatus[diet_problem.status])
for variable in diet_problem.variables():
    print(f"{variable.name}: {variable.varValue}")

print("Total Cost: $", pulp.value(diet_problem.objective))
```

Figure 11: ChatGPT provided code for Linear Programming