Homework 11: ISYE 6501 - Introduction to Analytics Modeling

Question 14.1

Prompt

In the videos, we saw the "diet problem". (The diet problem is one of the first large-scale optimization problems to be studied in practice. Back in the 1930's and 40's, the Army wanted to meet the nutritional requirements of its soldiers while minimizing the cost.) In this homework you get to solve a diet problem with real data. The data is given in the file `diet.xls`.

1. Formulate an optimization model (a linear program) to find the cheapest diet that satisfies the maximum and minimum daily nutrition constraints, and solve it using PuLP.
Turn in your code and the solution. (The optimal solution should be a diet of air-popped popcorn, poached eggs, oranges, raw iceberg lettuce, raw celery, and frozen broccoli. UGH!)

2. Please add to your model the following constraints (which might require adding more variables) and solve the new model:

a. If a food is selected, then a minimum of 1/10 serving must be chosen. (Hint: now you will need two variables for each food i: whether it is chosen, and how much is part of the diet. You'll also need to write a constraint to link them.)

b. Many people dislike celery and frozen broccoli. So at most one, but not both, can be selected.

c. To get day-to-day variety in protein, at least 3 kinds of meat/poultry/fish/eggs must be selected. [If something is ambiguous (e.g., should bean-and-bacon soup be considered meat?), just call it whatever you think is appropriate – I want you to learn how to write this type of constraint, but I don't really care whether we agree on how to classify foods!]

If you want to see what a more full-sized problem would look like, try solving your models for the file `diet_large.xls`, which is a low-cholesterol diet model (rather than minimizing cost, the goal is to minimize cholesterol intake). I don't know anyone who'd want to eat this diet – the optimal solution includes dried chrysanthemum garland, raw beluga whale flipper, freeze-dried parsley, etc. – which shows why it's necessary to add additional constraints beyond the basic ones we saw in the video! [Note: there are many optimal solutions, all with zero cholesterol, so you might get a different one.
It probably won't be much more appetizing than mine.]

14.1 Part 1: Optimize Cheapest Diet

PuLP is an linear and mixed integer programming modeler written in Python [1]. PuLP can generate MPS or LP files and call GLPK, COIN CLP/CBC, CPLEX, and GUROBI to solve linear problems.

For optimization, a linear program is developed. A linear program is the simplest type of mathematical program. To determine the mathematical program is a linear program the decision variables must be real variables, the objective must be a linear expression, and the constraints must be linear expressions.

Load and Inspect Data

```
import pandas as pd
from pulp import *

# Load the diet data into a DataFrame
path_data = "~/projects/ISYE6501/HW11/data/diet.xls"
diet_data = pd.read_excel(path_data)

# Show first few rows
diet_data.head(5)
```

```
##                 Foods  Price/ Serving  ... Calcium mg  Iron mg
## 0      Frozen Broccoli            0.16  ...      159.0      2.3
## 1          Carrots,Raw            0.07  ...       14.9      0.3
## 2          Celery, Raw            0.04  ...       16.0      0.2
## 3          Frozen Corn            0.18  ...        3.3      0.3
## 4  Lettuce,Iceberg,Raw            0.02  ...        3.8      0.1
##
## [5 rows x 14 columns]
```

At first glance, the data looks adequate. Check if there are any missing values and output the quantity missing for each column.

The first 5 values look good (Index 0-4). Look at the last 5 rows. We can see that `Foods` are only listed for 64 rows (0-63).

```
# Check for nan values in the diet data
nan_vals = diet_data.isna().sum()
print(nan_vals)
```

```
## Foods              3
## Price/ Serving     3
## Serving Size       1
## Calories           1
## Cholesterol mg     1
## Total_Fat g        1
## Sodium mg          1
## Carbohydrates g    1
## Dietary_Fiber g    1
## Protein g          1
## Vit_A IU           1
## Vit_C IU           1
```

```
## Calcium mg         1
## Iron mg            1
## dtype: int64
```

```
diet_data.tail(5)
```

```
##                    Foods  Price/ Serving  ... Calcium mg  Iron mg
## 62  Crm Mshrm Soup,W/Mlk            0.65  ...      178.6      0.6
## 63  Beanbacn Soup,W/Watr            0.67  ...       81.0      2.0
## 64                   NaN             NaN  ...        NaN      NaN
## 65                   NaN             NaN  ...      700.0     10.0
## 66                   NaN             NaN  ...     1500.0     40.0
##
## [5 rows x 14 columns]
```

The diet data needs to be parsed since it is not in an adequate format to form a model. First, keep only the valid data from rows 0 to 63.

```
# Keep only the 64 data points with no missing values
diet_data_clean = diet_data[0:64]
print(f"Number of rows in the diet data: {len(diet_data_clean)}")
```

```
## Number of rows in the diet data: 64
```

```
diet_data_clean.head(5)
```

```
##                  Foods  Price/ Serving  ... Calcium mg  Iron mg
## 0      Frozen Broccoli            0.16  ...      159.0      2.3
## 1          Carrots,Raw            0.07  ...       14.9      0.3
## 2          Celery, Raw            0.04  ...       16.0      0.2
## 3          Frozen Corn            0.18  ...        3.3      0.3
## 4  Lettuce,Iceberg,Raw            0.02  ...        3.8      0.1
##
## [5 rows x 14 columns]
```

Looking for missing values this time returns 0 since the data was cleaned. The code should return 0 missing values.

```
# Check for nan values in the diet data
nan_vals_clean = diet_data_clean.isna().sum()
print(nan_vals_clean)
```

```
## Foods              0
## Price/ Serving     0
## Serving Size       0
## Calories           0
## Cholesterol mg     0
## Total_Fat g        0
## Sodium mg          0
## Carbohydrates g    0
## Dietary_Fiber g    0
```

```
## Protein g          0
## Vit_A IU           0
## Vit_C IU           0
## Calcium mg         0
## Iron mg            0
## dtype: int64
```

Define Decision Variables

Create an LP problem with `LPProblem(name, sense)`. The name of the problem is defined to use in the output file, `cheapest_diet.lp`. The second argument is `sense` of the LP problem objective, which can be defined either as `LPMinimize` (Default) or `LPMaximize`.

```
# Initialize by creating an LP problem
diet_prob = LpProblem("cheapest_diet", LpMinimize)
```

A list is generated with all of the food names. A nested list is also created of all diet data that has been cleaned.

```
# Unique list of foods
foods = []
for each in diet_data_clean["Foods"]:
    foods.append(each)
print(foods)
```

```
## ['Frozen Broccoli', 'Carrots,Raw', 'Celery, Raw', 'Frozen Corn', 'Lettuce,Iceberg,Raw', 'Peppers, Sw
```

```
# Create a nested list of diet data values
list_data = diet_data_clean.values.tolist()
```

Our objective is to find the cheapest diet. This means that the problem needs the data for each parameter that has data that is quantitative in either an integer or float. This does not include `Foods` or `Serving Size` as these columns contain non-numerical values. Create dictionaries with each parameter and the corresponding values.

```
# Create dictionaries using dictionary comprehensions
# "Price/ Serving"
cost_dict = {each[0]: each[1] for each in list_data}
# "Calories"
cals_dict = {each[0]: each[3] for each in list_data}
# "Cholesterol mg"
choles_dict = {each[0]: each[4] for each in list_data}
# "Total_Fat g"
fat_dict = {each[0]: each[5] for each in list_data}
# "Sodium mg"
sod_dict = {each[0]: each[6] for each in list_data}
# "Carbohydrates g"
carb_dict = {each[0]: each[7] for each in list_data}
# "Dietary_Fiber g"
fiber_dict = {each[0]: each[8] for each in list_data}
# "Protein g"
```

```
pro_dict = {each[0]: each[9] for each in list_data}
# "Vit_A IU"
VA_dict = {each[0]: each[10] for each in list_data}
# "Vit_C IU"
VC_dict = {each[0]: each[11] for each in list_data}
# "Calcium mg"
calc_dict = {each[0]: each[12] for each in list_data}
# "Iron mg"
iron_dict = {each[0]: each[13] for each in list_data}
```

Setup Objective Function

```
# Dictionary 'food_vars' is created to contain the referenced Variables
food_vars = LpVariable.dicts("Foods",foods,0)

# Objective function is added to diet_prob
diet_prob += lpSum([cost_dict[i]*food_vars[i] for i in foods]), "Total Food Cost"
```

Define Constraints

```
# Diet constraints
diet_prob += lpSum([cals_dict[i] * food_vars[i] for i in foods]) >= 1500, "Min_Calorie_Requirement"
diet_prob += lpSum([cals_dict[i] * food_vars[i] for i in foods]) <= 2500, "Max_Calorie_Requirement"
diet_prob += lpSum([choles_dict[i] * food_vars[i] for i in foods]) >= 30, "Min_Cholesterol_Requirement"
diet_prob += lpSum([choles_dict[i] * food_vars[i] for i in foods]) <= 240, "Max_Cholesterol_Requirement
diet_prob += lpSum([fat_dict[i] * food_vars[i] for i in foods]) >= 20, "Min_Fat_Requirement"
diet_prob += lpSum([fat_dict[i] * food_vars[i] for i in foods]) <= 70, "Max_Fat_Requirement"
diet_prob += lpSum([sod_dict[i] * food_vars[i] for i in foods]) >= 800, "Min_Sodium_Requirement"
diet_prob += lpSum([sod_dict[i] * food_vars[i] for i in foods]) <= 2000, "Max_Sodium_Requirement"
diet_prob += lpSum([carb_dict[i] * food_vars[i] for i in foods]) >= 130, "Min_Carb_Requirement"
diet_prob += lpSum([carb_dict[i] * food_vars[i] for i in foods]) <= 450, "Max_Carb_Requirement"
diet_prob += lpSum([fiber_dict[i] * food_vars[i] for i in foods]) >= 125, "Min_Fiber_Requirement"
diet_prob += lpSum([fiber_dict[i] * food_vars[i] for i in foods]) <= 250, "Max_Fiber_Requirement"
diet_prob += lpSum([pro_dict[i] * food_vars[i] for i in foods]) >= 60, "Min_Protein_Requirement"
diet_prob += lpSum([pro_dict[i] * food_vars[i] for i in foods]) <= 100, "Max_Protein_Requirement"
diet_prob += lpSum([VA_dict[i] * food_vars[i] for i in foods]) >= 1000, "Min_VA_Requirement"
diet_prob += lpSum([VA_dict[i] * food_vars[i] for i in foods]) <= 10000, "Max_VA_Requirement"
diet_prob += lpSum([VC_dict[i] * food_vars[i] for i in foods]) >= 400, "Min_VC_Requirement"
diet_prob += lpSum([VC_dict[i] * food_vars[i] for i in foods]) <= 5000, "Max_VC_Requirement"
diet_prob += lpSum([calc_dict[i] * food_vars[i] for i in foods]) >= 700, "Min_Calcium_Requirement"
diet_prob += lpSum([calc_dict[i] * food_vars[i] for i in foods]) <= 1500, "Max_Calcium_Requirement"
diet_prob += lpSum([iron_dict[i] * food_vars[i] for i in foods]) >= 10, "Min_Iron_Requirement"
diet_prob += lpSum([iron_dict[i] * food_vars[i] for i in foods]) <= 40, "Max_Iron_Requirement"
```

Solve Linear Program

```
# write the problem to an LP file
diet_prob.writeLP("cheapest_diet.lp")
```

```
## [Foods_2%_Lowfat_Milk, Foods_3.3%_Fat,Whole_Milk, Foods_Apple,Raw,W_Skin, Foods_Apple_Pie, Foods_Bag
```

```
# call the solver and check the status
diet_prob.solve()
```

```
## 1
```

```
print("Status:", LpStatus[diet_prob.status])
```

```
## Status: Optimal
```

```
print(f"\nTotal Cost of the Cheapest Nutritional Diet: ${value(diet_prob.objective):.2f}")
```

```
##
## Total Cost of the Cheapest Nutritional Diet: $4.34
```

Results

```
# Print the main foods of the diet
results = {}
for v in diet_prob.variables():
    if v.varValue > 0:
        print(f"{v.name}: {v.varValue}")
        # Save the results in a dictionary
        results[v.name] = v.varValue
```

```
## Foods_Celery,_Raw: 52.64371
## Foods_Frozen_Broccoli: 0.25960653
## Foods_Lettuce,Iceberg,Raw: 63.988506
## Foods_Oranges: 2.2929389
## Foods_Poached_Eggs: 0.14184397
## Foods_Popcorn,Air_Popped: 13.869322
```

```
# Create a DataFrame from the dictionary
results_df = pd.DataFrame(list(results.items()), columns=['Food', 'Servings'])
results_df
```

```
##                         Food   Servings
## 0          Foods_Celery,_Raw  52.643710
## 1      Foods_Frozen_Broccoli   0.259607
## 2  Foods_Lettuce,Iceberg,Raw  63.988506
## 3              Foods_Oranges   2.292939
## 4         Foods_Poached_Eggs   0.141844
## 5   Foods_Popcorn,Air_Popped  13.869322
```

Table 1: Part 1. Cheapest Cost $4.34

Food	Servings
Foods_Celery,_Raw	52.643710
Foods_Frozen_Broccoli	0.259607
Foods_Lettuce,Iceberg,Raw	63.988506
Foods_Oranges	2.292939
Foods_Poached_Eggs	0.141844
Foods_Popcorn,Air_Popped	13.8693226

Part 2: Optimize Cheapest Diet with Constraints

Part 2a: Linking Two Variables with Constraint

If food is selected, then a minimum of 1/10 serving must be chosen.

For this portion of the problem, a second dictionary is added that adds in the binary variable `Selected` if the food is selected. A constraint is added considering you must eat at least 0.1 servings and if food is eaten, binary value is 1. *Only a snippet of the code is shown below, the full code is in the appendix.*

```
# Part 2a: Constraint

# Dictionary called 'food_vars' is created to contain the referenced Variables
food_vars = LpVariable.dicts("Foods",foods,0)

# Constraint A
# Binary variable if food is selected
food_vars_selected = LpVariable.dicts("Selected",foods,0,1,LpBinary)

# If a food is eaten, must eat at least 0.1 serving
for food in foods:
    diet_prob += food_vars[food] >= 0.1 * food_vars_selected[food]

# If any of a food is eaten, its binary variable must be 1
for food in foods:
    diet_prob += food_vars_selected[food] >= food_vars[food]*0.0000001
```

```
## [Foods_2%_Lowfat_Milk, Foods_3.3%_Fat,Whole_Milk, Foods_Apple,Raw,W_Skin, Foods_Apple_Pie, Foods_Bag
```

```
## 1
```

```
## Status: Optimal
```

```
##
## Total Cost of the Cheapest Nutritional Diet: $4.34
```

```
## Foods_Celery,_Raw: 52.64371
## Foods_Frozen_Broccoli: 0.25960653
## Foods_Lettuce,Iceberg,Raw: 63.988506
## Foods_Oranges: 2.2929389
## Foods_Poached_Eggs: 0.14184397
## Foods_Popcorn,Air_Popped: 13.869322
## Selected_Celery,_Raw: 1.0
## Selected_Frozen_Broccoli: 1.0
## Selected_Lettuce,Iceberg,Raw: 1.0
## Selected_Oranges: 1.0
## Selected_Poached_Eggs: 1.0
## Selected_Popcorn,Air_Popped: 1.0
```

```
##                            Food   Servings
## 0            Foods_Celery,_Raw  52.643710
## 1         Foods_Frozen_Broccoli   0.259607
```

```
## 2      Foods_Lettuce,Iceberg,Raw  63.988506
## 3                  Foods_Oranges   2.292939
## 4             Foods_Poached_Eggs   0.141844
## 5       Foods_Popcorn,Air_Popped  13.869322
## 6           Selected_Celery,_Raw   1.000000
## 7       Selected_Frozen_Broccoli   1.000000
## 8   Selected_Lettuce,Iceberg,Raw   1.000000
## 9               Selected_Oranges   1.000000
## 10        Selected_Poached_Eggs   1.000000
## 11   Selected_Popcorn,Air_Popped   1.000000
```

Table 2: Part 2a. Cheapest Cost $4.34

Food	Servings
Foods_Lettuce,Iceberg,Raw	63.988506
Foods_Oranges	2.292939
Foods_Poached_Eggs	0.141844
Foods_Popcorn,Air_Popped	13.869322

Part 2b: Celery or Broccoli Not Both

Many people dislike celery and frozen broccoli. So at most one, but not both, can be selected.

For this portion of the problem, a constraint of either frozen broccoli or raw celery is added. *Only a snippet of the code is shown below, the full code is in the appendix.*

```
# Part 2b: Celery or Broccoli

# Constraint B
# Include at most 1 of celery and frozen broccoli
diet_prob += food_vars_selected['Frozen Broccoli'] + food_vars_selected['Celery, Raw'] <= 1
```

```
## [Foods_2%_Lowfat_Milk, Foods_3.3%_Fat,Whole_Milk, Foods_Apple,Raw,W_Skin, Foods_Apple_Pie, Foods_Bag
```

```
## 1
```

```
## Status: Optimal
```

```
##
## Total Cost of the Cheapest Nutritional Diet: $4.49
```

```
## Foods_Celery,_Raw: 43.154119
## Foods_Lettuce,Iceberg,Raw: 80.919121
## Foods_Oranges: 3.0765161
## Foods_Peanut_Butter: 2.0464575
## Foods_Poached_Eggs: 0.14184397
## Foods_Popcorn,Air_Popped: 13.181772
## Selected_Celery,_Raw: 1.0
## Selected_Lettuce,Iceberg,Raw: 1.0
## Selected_Oranges: 1.0
## Selected_Peanut_Butter: 1.0
## Selected_Poached_Eggs: 1.0
## Selected_Popcorn,Air_Popped: 1.0
```

```
##                             Food   Servings
## 0             Foods_Celery,_Raw  43.154119
## 1     Foods_Lettuce,Iceberg,Raw  80.919121
## 2                 Foods_Oranges   3.076516
## 3           Foods_Peanut_Butter   2.046457
## 4            Foods_Poached_Eggs   0.141844
## 5      Foods_Popcorn,Air_Popped  13.181772
## 6          Selected_Celery,_Raw   1.000000
## 7   Selected_Lettuce,Iceberg,Raw   1.000000
## 8              Selected_Oranges   1.000000
## 9        Selected_Peanut_Butter   1.000000
## 10        Selected_Poached_Eggs   1.000000
## 11   Selected_Popcorn,Air_Popped   1.000000
```

Table 3: Part 2b. Cheapest Cost $4.49

Food	Servings
Foods_Celery,_Raw	43.154119
Foods_Lettuce,Iceberg,Raw	80.919121
Foods_Oranges	3.076516
Foods_Peanut_Butter	2.046457
Foods_Popcorn,Air_Popped	13.181772

Part 2c: Protein Variety

For a day-to-day variety in protein, at least 3 kinds of meat/poultry/fish/eggs must be selected. If ambiguous, make an assumption (e.g., bean and bacon soup).

For this portion of the problem, meat/poultry/fish/eggs were added to the constraints portion of the code. *Only a snippet of the code is shown below, the full code is in the appendix.*

```
# Part 2c: 3 kinds of protein

# Constraint C
# At least 3 kinds of meat/poultry/fish/eggs
diet_prob += food_vars_selected['Roasted Chicken'] \
        + food_vars_selected['Poached Eggs'] \
        + food_vars_selected['Scrambled Eggs'] \
        + food_vars_selected['Bologna,Turkey'] \
        + food_vars_selected['Frankfurter, Beef'] \
        + food_vars_selected['Ham,Sliced,Extralean'] \
        + food_vars_selected['Kielbasa,Prk'] \
        + food_vars_selected['Pizza W/Pepperoni'] \
        + food_vars_selected['Hamburger W/Toppings'] \
        + food_vars_selected['Hotdog, Plain'] \
        + food_vars_selected['Pork'] \
        + food_vars_selected['Sardines in Oil'] \
        + food_vars_selected['White Tuna in Water'] \
        + food_vars_selected['Chicknoodl Soup'] \
        + food_vars_selected['Splt Pea&Hamsoup'] \
        + food_vars_selected['Vegetbeef Soup'] \
        + food_vars_selected['Neweng Clamchwd'] \
        + food_vars_selected['New E Clamchwd,W/Mlk'] \
        + food_vars_selected['Beanbacn Soup,W/Watr'] >= 3
```

```
## [Foods_2%_Lowfat_Milk, Foods_3.3%_Fat,Whole_Milk, Foods_Apple,Raw,W_Skin, Foods_Apple_Pie, Foods_Bag
```

```
## 1
```

```
## Status: Optimal
```

```
##
## Total Cost of the Cheapest Nutritional Diet: $4.51
```

```
## Foods_Celery,_Raw: 42.399358
## Foods_Kielbasa,Prk: 0.1
## Foods_Lettuce,Iceberg,Raw: 82.802586
## Foods_Oranges: 3.0771841
## Foods_Peanut_Butter: 1.9429716
## Foods_Poached_Eggs: 0.1
## Foods_Popcorn,Air_Popped: 13.223294
## Foods_Scrambled_Eggs: 0.1
## Selected_Celery,_Raw: 1.0
## Selected_Kielbasa,Prk: 1.0
## Selected_Lettuce,Iceberg,Raw: 1.0
## Selected_Oranges: 1.0
```

```
## Selected_Peanut_Butter: 1.0
## Selected_Poached_Eggs: 1.0
## Selected_Popcorn,Air_Popped: 1.0
## Selected_Scrambled_Eggs: 1.0
```

```
##                            Food   Servings
## 0             Foods_Celery,_Raw  42.399358
## 1            Foods_Kielbasa,Prk   0.100000
## 2     Foods_Lettuce,Iceberg,Raw  82.802586
## 3                 Foods_Oranges   3.077184
## 4           Foods_Peanut_Butter   1.942972
## 5            Foods_Poached_Eggs   0.100000
## 6      Foods_Popcorn,Air_Popped  13.223294
## 7          Foods_Scrambled_Eggs   0.100000
## 8          Selected_Celery,_Raw   1.000000
## 9         Selected_Kielbasa,Prk   1.000000
## 10 Selected_Lettuce,Iceberg,Raw   1.000000
## 11             Selected_Oranges   1.000000
## 12       Selected_Peanut_Butter   1.000000
## 13        Selected_Poached_Eggs   1.000000
## 14  Selected_Popcorn,Air_Popped   1.000000
## 15      Selected_Scrambled_Eggs   1.000000
```

Food	Servings
Foods_Celery,_Raw	42.399358
Foods_Kielbasa,Prk	0.100000
Foods_Lettuce,Iceberg,Raw	82.802586
Foods_Oranges	3.077184
Foods_Peanut_Butter	1.942972
Foods_Poached_Eggs	0.100000
Foods_Popcorn,Air_Popped	13.223294
Foods_Scrambled_Eggs	0.100000

Solution

Food	Part 1 Servings	Part 2a Servings	Part 2b Servings	Part 2c Servings
Foods_Celery,_Raw	52.643710		43.154119	42.399358
Foods_Frozen_Broccoli	0.259607			
Foods_Lettuce,Iceberg,Raw	63.988506	63.988506	80.919121	82.802586
Foods_Oranges	2.292939	2.292939	3.076516	3.077184
Foods_Peanut_Butter			2.046457	1.942972
Foods_Poached_Eggs	0.141844	0.141844		0.100000
Foods_Popcorn,Air_Popped	13.8693226	13.869322	13.181772	13.223294
Foods_Kielbasa,Prk				0.100000
Foods_Scrambled_Eggs				0.100000
Cost	**$4.34**	**$4.34**	**$4.49**	**$4.51**

Table: Comparison of food items for cheapest diet with Part 1, 2a, 2b, 2c, constraints.

Appendix A: Full Code

The code below is the full code used to solve Question 14.1. There is a comment above each constraint. The first set of constraints are standard diet constraints for Part 1. Constraints a, b, and c correspond to Part 2a, 2b, and 2c.

You can run this code but change your directory path for the data. If you want to perform a specific portion of Question 14.1, remove the unecessary constraints. For example, if you want to model without the broccoli or celery constraint, remove:

```
# Constraint B
# Include at most 1 of celery and frozen broccoli
diet_prob += food_vars_selected['Frozen Broccoli'] + food_vars_selected['Celery, Raw'] <= 1
```

```
import pandas as pd
from pulp import *

# Load the diet data into a DataFrame
path_data = "~/projects/ISYE6501/HW11/data/diet.xls"
diet_data = pd.read_excel(path_data)

# Check for nan values in the diet data
nan_vals = diet_data.isna().sum()
print(f"Number of Missing Values by Column:\n{len(diet_data)}")
print(f"Number of rows in the diet data: {len(diet_data)}")

# Keep only the 64 data points with no missing values
diet_data_clean = diet_data[0:64]
print(f"Number of rows in the diet data: {len(diet_data_clean)}")

# Model ----------------------------------------------------------------------------
# Initialize by creating an LP problem
diet_prob = LpProblem("cheapest_diet", LpMinimize)

# Unique list of foods
foods = []
for each in diet_data_clean["Foods"]:
    foods.append(each)

# Create a nested list of diet data values
list_data = diet_data_clean.values.tolist()

# Create dictionaries using dictionary comprehensions
# "Price/ Serving"
cost_dict = {each[0]: each[1] for each in list_data}
# "Calories"
cals_dict = {each[0]: each[3] for each in list_data}
# "Cholesterol mg"
choles_dict = {each[0]: each[4] for each in list_data}
# "Total_Fat g"
fat_dict = {each[0]: each[5] for each in list_data}
# "Sodium mg"
sod_dict = {each[0]: each[6] for each in list_data}
# "Carbohydrates g"
```

```
carb_dict = {each[0]: each[7] for each in list_data}
# "Dietary_Fiber g"
fiber_dict = {each[0]: each[8] for each in list_data}
# "Protein g"
pro_dict = {each[0]: each[9] for each in list_data}
# "Vit_A IU"
VA_dict = {each[0]: each[10] for each in list_data}
# "Vit_C IU"
VC_dict = {each[0]: each[11] for each in list_data}
# "Calcium mg"
calc_dict = {each[0]: each[12] for each in list_data}
# "Iron mg"
iron_dict = {each[0]: each[13] for each in list_data}

# Dictionary 'food_vars' contains the optimization variables
food_vars = LpVariable.dicts("Foods",foods,0)

# First, the objective function is added to the problem
diet_prob += lpSum([cost_dict[i]*food_vars[i] for i in foods]), "Total Food Cost"

# Diet constraints
diet_prob += lpSum([cals_dict[i] * food_vars[i] for i in foods]) >= 1500, "Min_Calorie_Requirement"
diet_prob += lpSum([cals_dict[i] * food_vars[i] for i in foods]) <= 2500, "Max_Calorie_Requirement"
diet_prob += lpSum([choles_dict[i] * food_vars[i] for i in foods]) >= 30, "Min_Cholesterol_Requirement"
diet_prob += lpSum([choles_dict[i] * food_vars[i] for i in foods]) <= 240, "Max_Cholesterol_Requirement
diet_prob += lpSum([fat_dict[i] * food_vars[i] for i in foods]) >= 20, "Min_Fat_Requirement"
diet_prob += lpSum([fat_dict[i] * food_vars[i] for i in foods]) <= 70, "Max_Fat_Requirement"
diet_prob += lpSum([sod_dict[i] * food_vars[i] for i in foods]) >= 800, "Min_Sodium_Requirement"
diet_prob += lpSum([sod_dict[i] * food_vars[i] for i in foods]) <= 2000, "Max_Sodium_Requirement"
diet_prob += lpSum([carb_dict[i] * food_vars[i] for i in foods]) >= 130, "Min_Carb_Requirement"
diet_prob += lpSum([carb_dict[i] * food_vars[i] for i in foods]) <= 450, "Max_Carb_Requirement"
diet_prob += lpSum([fiber_dict[i] * food_vars[i] for i in foods]) >= 125, "Min_Fiber_Requirement"
diet_prob += lpSum([fiber_dict[i] * food_vars[i] for i in foods]) <= 250, "Max_Fiber_Requirement"
diet_prob += lpSum([pro_dict[i] * food_vars[i] for i in foods]) >= 60, "Min_Protein_Requirement"
diet_prob += lpSum([pro_dict[i] * food_vars[i] for i in foods]) <= 100, "Max_Protein_Requirement"
diet_prob += lpSum([VA_dict[i] * food_vars[i] for i in foods]) >= 1000, "Min_VA_Requirement"
diet_prob += lpSum([VA_dict[i] * food_vars[i] for i in foods]) <= 10000, "Max_VA_Requirement"
diet_prob += lpSum([VC_dict[i] * food_vars[i] for i in foods]) >= 400, "Min_VC_Requirement"
diet_prob += lpSum([VC_dict[i] * food_vars[i] for i in foods]) <= 5000, "Max_VC_Requirement"
diet_prob += lpSum([calc_dict[i] * food_vars[i] for i in foods]) >= 700, "Min_Calcium_Requirement"
diet_prob += lpSum([calc_dict[i] * food_vars[i] for i in foods]) <= 1500, "Max_Calcium_Requirement"
diet_prob += lpSum([iron_dict[i] * food_vars[i] for i in foods]) >= 10, "Min_Iron_Requirement"
diet_prob += lpSum([iron_dict[i] * food_vars[i] for i in foods]) <= 40, "Max_Iron_Requirement"

# Constraint A
# Add binary variable for the constraint of selecting a food
food_vars_selected = LpVariable.dicts("Selected",foods,0,1,LpBinary)

# If a food is eaten, must eat at least 0.1 serving
for food in foods:
    diet_prob += food_vars[food] >= 0.1 * food_vars_selected[food]

# If any of a food is eaten, its binary variable must be 1
```

```
for food in foods:
    diet_prob += food_vars_selected[food] >= food_vars[food]*0.0000001

# Constraint B
# Include at most 1 of celery and frozen broccoli
diet_prob += food_vars_selected['Frozen Broccoli'] + food_vars_selected['Celery, Raw'] <= 1

# Constraint C
# At least 3 kinds of meat/poultry/fish/eggs
diet_prob += food_vars_selected['Roasted Chicken'] \
        + food_vars_selected['Poached Eggs'] \
        + food_vars_selected['Scrambled Eggs'] \
        + food_vars_selected['Bologna,Turkey'] \
        + food_vars_selected['Frankfurter, Beef'] \
        + food_vars_selected['Ham,Sliced,Extralean'] \
        + food_vars_selected['Kielbasa,Prk'] \
        + food_vars_selected['Pizza W/Pepperoni'] \
        + food_vars_selected['Hamburger W/Toppings'] \
        + food_vars_selected['Hotdog, Plain'] \
        + food_vars_selected['Pork'] \
        + food_vars_selected['Sardines in Oil'] \
        + food_vars_selected['White Tuna in Water'] \
        + food_vars_selected['Chicknoodl Soup'] \
        + food_vars_selected['Splt Pea&Hamsoup'] \
        + food_vars_selected['Vegetbeef Soup'] \
        + food_vars_selected['Neweng Clamchwd'] \
        + food_vars_selected['New E Clamchwd,W/Mlk'] \
        + food_vars_selected['Beanbacn Soup,W/Watr'] >= 3

# write the problem to an LP file
diet_prob.writeLP("cheapest_diet.lp")

# call the solver and check the status
diet_prob.solve()

print("Status:", LpStatus[diet_prob.status])

print(f"\nTotal Cost of the Cheapest Nutritional Diet: ${value(diet_prob.objective):.2f}")

# Print the main foods of the diet
results = {}
for v in diet_prob.variables():
    if v.varValue > 0:
        print(f"{v.name}: {v.varValue}")
        # Save the results in a dictionary
        results[v.name] = v.varValue

# Create a DataFrame from the dictionary
results_df = pd.DataFrame(list(results.items()), columns=['Food', 'Servings'])

results_df
```

References

[1] "PuLP documentation: Optimization with PuLP." https://www.coin-or.org/PuLP/pulp.html.