Question 15.2
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!]

1. Using Python's PuLP, I found the optimal solution of air-popped popcorn, poached eggs, oranges, raw iceberg lettuce, raw celery, and frozen broccoli. I found this difficult because I have never used PuLP before and have never done optimization before, so I am not sure if I did mine in the most 'optimal' way. I have worked with pandas before, so I used a lot of that for these questions. I loaded the information from the .xls file into a pandas dataframe to start.

```
import pandas as pd
!pip install pulp
import pulp

# Step 1: Load the data from the Excel file
file_path = 'diet.xls'  # Path to Excel file
data = pd.read_excel(file_path, sheet_name='Sheet1')
```

```
9. data = data[0:64] # Selecting only the relevant food items
```

I did not include the min and max values from the .xls sheet because I figured it would be easier to just hard input them later. After creating the dataframe, I set up the rest of the data.

```
10.             # Step 2: Extract necessary data
11.       foods = data['Foods'].tolist()
12.       prices = data['Price/ Serving'].tolist()
13.
14.       # Nutritional values start from the 3rd column (index 2) to
    the end of the DataFrame
15.       nutritional_data = data.iloc[:, 2:].values
16.
17.       # Define the nutrients based on the columns in the Excel file
18.       nutrient_names = data.columns[2:].tolist()
19.
20.       # Step 3: Define minimum and maximum nutritional requirements
21.       min_vals = [1500, 30, 20, 800, 130, 125, 60, 1000, 400, 700,
    10]  # Minimum requirements
22.       max_vals = [2500, 240, 70, 2000, 450, 250, 100, 10000, 5000,
    1500, 40]  #Maximum reqirements
```

After inputting and formatting all the data I needed, I could then set up the linear programming problem.

```
23.       # Set up the linear programming problem
24.       problem = pulp.LpProblem("DietProblem", pulp.LpMinimize)
```

Now that I had the problem's bones, I thought of how to construct an optimization problem. I needed variables, constraints, and an objective function. The variables for this problem were the amounts of each food needed. The objective function was to minimize the total cost of the constraints. The constraints were {min for each nutrient $\leq x \leq$ max for each nutrient}. The below code does exactly that:

```
25.       # Define decision variables (amounts of each food)
26.       food_vars = pulp.LpVariable.dicts("Food", foods, lowBound=0,
    cat='Continuous')
27.
28.       # Define the objective function (minimize total cost)
29.       problem += pulp.lpSum([prices[i] * food_vars[foods[i]] for i
    in range(len(foods))]), "TotalCost"
30.
31.       # Define nutritional constraints
32.       for i, nutrient in enumerate(nutrient_names):
33.           # Skip 'Serving Size' column as it's not a nutrient
```

```
34.          if nutrient == 'Serving Size':
35.              continue
36.
37.          # Convert nutritional data to numeric, handling potential
   errors
38.          nutrient_values =
   pd.to_numeric(pd.Series(nutritional_data[:, i]),
   errors='coerce').fillna(0).tolist()
39.
40.          # Adjust index to align with min_vals and max_vals
   (skipping 'Serving Size')
41.          nutrient_index = i - 1 if i > 0 else 0
42.
43.          # Minimum nutrient constraint
44.          try:
45.              if not pd.isnull(min_vals[nutrient_index]): # Check
   for NaN in min_vals
46.                  problem += pulp.lpSum([nutrient_values[j] *
   food_vars[foods[j]] for j in range(len(foods))]) >=
   min_vals[nutrient_index], f"Min_{nutrient}"
47.          except IndexError:
48.              print(f"Warning: min_vals index out of range for
   nutrient: {nutrient}, index: {nutrient_index}")
49.
50.          # Maximum nutrient constraint
51.          try:
52.              if not pd.isnull(max_vals[nutrient_index]): # Check
   for NaN in max_vals
53.                  problem += pulp.lpSum([nutrient_values[j] *
   food_vars[foods[j]] for j in range(len(foods))]) <=
   max_vals[nutrient_index], f"Max_{nutrient}"
54.          except IndexError:
55.              print(f"Warning: max_vals index out of range for
   nutrient: {nutrient}, index: {nutrient_index}")
```

Even though I knew there were no NaN values in the .xls file, I am used to error handling in real life, so I just added those parts. So, at this point I am just hoping that this problem provides the solution the question says that it should. I then printed out the solution:

```
56.      # Solve the problem
57.      problem.solve()
58.
59.      # Print the results
60.      print("Status:", pulp.LpStatus[problem.status])
61.      print("Optimal Diet:")
62.      for food in foods:
```

```
        if food_vars[food].varValue > 0:
            print(f"{food}: {food_vars[food].varValue} servings")

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

This produced the following answer:

Status: Optimal
Optimal Diet:
Frozen Broccoli: 0.25960653 servings
Celery, Raw: 52.64371 servings
Lettuce,Iceberg,Raw: 63.988506 servings
Oranges: 2.2929389 servings
Poached Eggs: 0.14184397 servings
Popcorn,Air-Popped: 13.869322 servings
Total Cost: 4.337116797399999

I doubled checked with the solutions answer and I got the same items that it listed. Looking at this optimal diet, I was trying to figure out what each nutrient would come from mostly - for example I'm assuming most of daily water would come from the lettuce and celery, and protein from the popcorn. If I was a much better coder, I would like to see the breakdown of how much of each nutrient comes from each food item, but this was about the extent of my abilities. I was surprised to see how cheap it is – it takes about $4.3 to survive apparently.

2. Now that I got the very basic optimization problem correct, it was time to add the constraints from the second question to it. I copy and pasted the entire first question to start the second question because all of the information in there is still needed.

```
import pandas as pd
!pip install pulp
import pulp

# Step 1: Load the data from the Excel file
file_path = 'diet.xls'  # Path to your Excel file
data = pd.read_excel(file_path, sheet_name='Sheet1') # Assuming data is in
'Sheet1'
data = data[0:64] # Selecting only the relevant food items

# Step 2: Extract necessary data
foods = data['Foods'].tolist()
prices = data['Price/ Serving'].tolist()

# Nutritional values start from the 3rd column (index 2) to the end of the
DataFrame
```

```
nutritional_data = data.iloc[:, 2:].values

# Define the nutrients based on the columns in the Excel file
nutrient_names = data.columns[2:].tolist()  # Adjust if needed

# Step 3: Define minimum and maximum nutritional requirements
# Replace these values with your actual requirements
min_vals = [1500, 30, 20, 800, 130, 125, 60, 1000, 400, 700, 10]  # Minimum
requirements
max_vals = [2500, 240, 70, 2000, 450, 250, 100, 10000, 5000, 1500, 40]

# Set up the linear programming problem
problem = pulp.LpProblem("DietProblem", pulp.LpMinimize)

# Define decision variables (amounts of each food)
food_vars = pulp.LpVariable.dicts("Food", foods, lowBound=0,
cat='Continuous')

# Define binary variables to indicate if a food is chosen (1) or not (0)
chosen_vars = pulp.LpVariable.dicts("Chosen", foods, cat='Binary')

# Define the objective function (minimize total cost)
problem += pulp.lpSum([prices[i] * food_vars[foods[i]] for i in
range(len(foods))]), "TotalCost"

# Define nutritional constraints
for i, nutrient in enumerate(nutrient_names):
  # Skip 'Serving Size' column as it's not a nutrient
  if nutrient == 'Serving Size':
    continue

  # Convert nutritional data to numeric, handling potential errors
  nutrient_values = pd.to_numeric(pd.Series(nutritional_data[:, i]),
errors='coerce').fillna(0).tolist()

  # Adjust index to align with min_vals and max_vals (skipping 'Serving Size')
  nutrient_index = i - 1 if i > 0 else 0

  # Minimum nutrient constraint
  try:
    if not pd.isnull(min_vals[nutrient_index]): # Check for NaN in min_vals
      problem += pulp.lpSum([nutrient_values[j] * food_vars[foods[j]] for j in
range(len(foods))]) >= min_vals[nutrient_index], f"Min_{nutrient}"
  except IndexError:
```

```
      print(f"Warning: min_vals index out of range for nutrient: {nutrient}, index: {nutrient_index}")

   # Maximum nutrient constraint
   try:
      if not pd.isnull(max_vals[nutrient_index]): # Check for NaN in max_vals
         problem += pulp.lpSum([nutrient_values[j] * food_vars[foods[j]] for j in range(len(foods))]) <= max_vals[nutrient_index], f"Max_{nutrient}"
   except IndexError:
      print(f"Warning: max_vals index out of range for nutrient: {nutrient}, index: {nutrient_index}")
```

(I apologize for the formatting on this one – it did not copy over very nicely and I spent way too long trying to get it to, so this was the best I could come up with). So, above is the entirety of the first question just not broken down into parts. Now it was time to handle the first constraint, that if a food was chosen then a minimum of 1/10 serving must be chosen. This was straightforward, I said that if a food was chosen, then at least 0.1 of it must be chosen. I also added a maximum value just in case anything was weird.

```
# Constraint (a): Minimum serving size if chosen
for food in foods:
   problem += food_vars[food] >= 0.1 * chosen_vars[food], f"MinServing_{food}"
   problem += food_vars[food] <= 10000 * chosen_vars[food], f"MaxServing_{food}"
```

To handle the second constraint, at most one celery or frozen broccoli must be chosen if selected, it was also straightforward. The code below pretty much says if one of their values equals 1 (meaning chosen) then the other one must equal 0. Meaning only one can be chosen.

```
# Constraint (b): Celery and Frozen Broccoli restriction
problem += chosen_vars['Frozen Broccoli'] + chosen_vars['Celery, Raw'] <= 1, "CeleryBroccoli"
```

The third constraint, at least three kinds of protein must be chosen, was a little more difficult than the first two. What made it more difficult was the discrepancies in what I deemed to be ‘meat’ or not. For example, I considered something like bean and bacon soup to be meat, but not pepperoni pizza. I kind of just picked the ones that jumped out to me when I would look at them the first time and, in my head, would say “that is meat.” I went with the following as the protein sources:

```
# Constraint (c): Protein variety
protein_sources = ['Roasted Chicken', 'Beanbacn Soup,W/Watr', 'Poached
Eggs', ‘Scrambled Eggs’, 'Bologna,Turkey', 'Frankfurter, Beef',
'Ham,Sliced,Extralean', 'Hotdog, Plain', 'Hamburger W/Toppings', 'Pork',
'Sardines in Oil', "White Tuna in Water"]
problem += pulp.lpSum([chosen_vars[protein_food] for protein_food in
protein_sources]) >= 3, "ProteinVariety"
```

Now that I had the three constraints added, in addition to the original constraint of {min for each nutrient $\leq$ x $\leq$ max for each nutrient}, I ran the new optimization problem.

```
# Solve the problem
problem.solve()

# Print the results
print("Status:", pulp.LpStatus[problem.status])
print("Optimal Diet:")
for food in foods:
   if food_vars[food].varValue > 0:
      print(f"{food}: {food_vars[food].varValue} servings")

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

This resulted in the following:

Optimal Diet:
Celery, Raw: 42.423026 servings
Lettuce,Iceberg,Raw: 82.673927 servings
Oranges: 3.0856009 servings
Poached Eggs: 0.1 servings
Scrambled Eggs: 0.1 servings
Bologna,Turkey: 0.1 servings
Peanut Butter: 1.9590978 servings
Popcorn,Air-Popped: 13.214473 servings
Total Cost: 4.5129554810000005

Most of what was in the first optimal diet was in this diet, but there were now some differences. For one, there is no frozen broccoli. This is due to the constraint of if either frozen broccoli or raw celery are selected, then the other cannot be selected. This optimization found that the raw celery was better to select, so in turn frozen broccoli cannot be in the optimal diet. Per the third constraint, scrambled eggs and bologna, turkey is now in the optimal diet. This is because that constraint said there needs to be at least three different meats, where in the first solution there was just poached eggs from the ‘protein’ list I constructed. It does seem like it wants to minimize

the usage of these though because it is really hanging onto the first constraint for them, that if something is selected, at least 0.1 of a serving of it must be used. That means that these are probably expensive, and since the optimization problem wants to minimize cost, it wants to use as little as them as possible. The new optimization problem also added some peanut butter which was not in the first one. It was interesting to see that the total cost, even with the addition of three more constraints, is still about the same as the first one. The old total cost was about $4.3, and the new one is about $4.5. Now multiply $.2 by x number of mouths they need to feed and I'm sure the number would be much more than I think, but for now it does not look like much of a difference.