Week 12 Homework

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 thistype 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.

Solution Part 1

Optimization Model Definitions:

Variables:
x_i = amount of food i in daily diet

Constraints:
For each nutrient j, $min_j <= \sum_{i=1}^{m} a_{ij}x_i <= max_j$
Minimum x_i = 0

Objective Function:
Minimize $\sum_{i=1}^{n} c_i x_i$

Where:
n = Number of different foods
m = Number of different nutrients
x_i = amount of food i in daily diet
a_{ij} = amount of nutrient j in food i
min_j = Minimum daily intake of Nutrient j
max_j = Maximum daily intake of Nutrient j
c_i = cost of food i

In [1]:
```
import pandas as pd
# Read Created Contraints file
const_df = pd.read_csv("diet_constraints.csv")
# Display Constraints df
const_df
```

Out[1]:

	Nutrient	Min	Max
0	Calories	1500	2500
1	Cholesterol	30	240
2	Total_Fat	20	70
3	Sodium	800	2000
4	Carbohydrates	130	450
5	Dietary_Fiber	125	250
6	Protein	60	100
7	Vit_A	1000	10000
8	Vit_C	400	5000
9	Calcium	700	1500
10	Iron	10	40

In [2]:

```python
# Read Created Foods file
food_df = pd.read_csv("diet_foods.csv")
# Display Constraints df
food_df
```

Out[2]:

	Foods	Price/ Serving	Serving Size	Calories	Cholesterol	Total_Fat	Sodium	Carbohydrates	Dietary_Fiber
0	Frozen Broccoli	0.16	10 Oz Pkg	73.8	0.0	0.8	68.2	13.6	8.5
1	Carrots,Raw	0.07	1/2 Cup Shredded	23.7	0.0	0.1	19.2	5.6	1.6
2	Celery, Raw	0.04	1 Stalk	6.4	0.0	0.1	34.8	1.5	0.7
3	Frozen Corn	0.18	1/2 Cup	72.2	0.0	0.6	2.5	17.1	2.0
4	Lettuce,Iceberg,Raw	0.02	1 Leaf	2.6	0.0	0.0	1.8	0.4	0.3
...	...	...	...	...	...	...	...	...	...
59	Neweng Clamchwd	0.75	1 C (8 Fl Oz)	175.7	10.0	5.0	1864.9	21.8	1.5
60	Tomato Soup	0.39	1 C (8 Fl Oz)	170.7	0.0	3.8	1744.4	33.2	1.0
61	New E Clamchwd,W/Mlk	0.99	1 C (8 Fl Oz)	163.7	22.3	6.6	992.0	16.6	1.5
62	Crm Mshrm Soup,W/Mlk	0.65	1 C (8 Fl Oz)	203.4	19.8	13.6	1076.3	15.0	0.5
63	Beanbacn Soup,W/Watr	0.67	1 C (8 Fl Oz)	172.0	2.5	5.9	951.3	22.8	8.6

64 rows × 14 columns

In [3]:

```
# Import PuLP modeler functions
from pulp import *
```

Preparing Data

In [4]:

```
# List of all foods
foods = food_df["Foods"].to_list()

# A dictionary of the costs of each type of the food
food_costs = dict(zip(food_df["Foods"], food_df["Price/ Serving"]))

# List of all Nutrients
nutrients = const_df["Nutrient"].to_list()
# Create a dictionary for Min value per nutrient
min_nutrients = dict(zip(const_df["Nutrient"], const_df["Min"]))
# Create a dictionary for Max value per nutrient
max_nutrients = dict(zip(const_df["Nutrient"], const_df["Max"]))

# Create Food Nutrient Dictionary of Dictionaries
food_nutrient_dict = food_df.set_index("Foods").iloc[:,2:].to_dict()
# Example of result dict
food_nutrient_dict["Calories"]["Frozen Broccoli"]
```

Out[4]:

```
73.8
```

Initialize the model

In [5]:

```
# Create the 'problem1' variable to contain the problem data
problem1 = LpProblem("The_Diet_Problem_P1", LpMinimize)
```

Create the Variables

In [6]:

```
# A dictionary called 'food_vars' is created to contain the referenced Variables with a
food_vars = LpVariable.dicts("item", foods, lowBound=0)
```

Define the Objective function

In [7]:

```
problem1 += (lpSum([food_costs[i] * food_vars[i] for i in foods]), "Total Cost of Foods'
```

Define the Constraints

In [8]:

```
for nutrient in nutrients:
    # Define Min Constraint
    problem1 += (lpSum([food_nutrient_dict[nutrient][i] * food_vars[i] for i in foods])
                 nutrient+"_min",)
    # Define Max Constraint
    problem1 += (lpSum([food_nutrient_dict[nutrient][i] * food_vars[i] for i in foods])
                 nutrient+"_max",)
```

Run the Model

In [9]:

```
problem1.solve()
# The status of the solution is printed to the screen
print("Status:", LpStatus[problem1.status])
```

```
Status: Optimal
```

In [10]:

```
# Each of the variables is printed with it's resolved optimum value
for var in problem1.variables():
    # Filter out the 0 variables
    if var.varValue != 0:
        print(var.name, "=", var.varValue)
```

```
item_Celery,_Raw = 52.643689
item_Frozen_Broccoli = 0.25963403
item_Lettuce,Iceberg,Raw = 63.987845
item_Oranges = 2.2928841
item_Poached_Eggs = 0.14184397
item_Popcorn,Air_Popped = 13.869357
```

In [11]:

```
print("Total Cost of Diet per person = ", value(problem1.objective))
```

```
Total Cost of Diet per person =  4.3371003174
```

As a result, The optimal solution is a diet of air-popped popcorn, poached eggs, oranges, raw iceberg lettuce, raw celery, and frozen broccoli.

Solution Part 2

1. 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 thistype of constraint, but I don't really care whether we agree on how to classify foods!

In [12]:

```
# Copy Initial model with it's assumptions
problem2 = problem1.copy()
```

Part A

In [13]:

```
# Create a binary variable to store whether a food was selected
food_selection_vars = LpVariable.dicts("Selected", foods, cat = "Binary")
# Add Constraints that if the food is selected then the minimum is 0.1 serving
for a_food in foods:
    problem2 += (food_vars[a_food]>= food_selection_vars[a_food]*0.1,
                 a_food+"_presence",)
    problem2 += (food_vars[a_food]<= food_selection_vars[a_food]*100000,
                 a_food+"_limit",)
```

Part B

In [14]:

```
problem2 += (food_selection_vars["Celery, Raw"]+food_selection_vars["Frozen Broccoli"] <
            "celery and frozen broccoli Choice",)
```

Part C

In [15]:

```
problem2 += (food_selection_vars['Roasted Chicken'] + food_selection_vars['Poached Eggs
              + food_selection_vars['Scrambled Eggs'] + food_selection_vars['Bologna,Turk
              + food_selection_vars['Frankfurter, Beef'] + food_selection_vars['Ham,Slice
              + food_selection_vars['Kielbasa,Prk'] + food_selection_vars['Pizza W/Pepper
              + food_selection_vars['Hamburger W/Toppings'] + food_selection_vars['Hotdog
              + food_selection_vars['Pork'] + food_selection_vars['Sardines in Oil']
              + food_selection_vars['White Tuna in Water'] )>=3, ""
```

Run The Model

In [16]:
```
problem2.solve()
# The status of the solution is printed to the screen
print("Status:", LpStatus[problem2.status])
```

```
Status: Optimal
```

In [17]:
```
# Each of the variables is printed with it's resolved optimum value
for var in problem2.variables():
    # Filter out the 0 variables
    if var.varValue != 0 and "Select" not in var.name:
        print(var.name, "=", var.varValue)
```

```
item_Celery,_Raw = 42.399358
item_Kielbasa,Prk = 0.1
item_Lettuce,Iceberg,Raw = 82.802586
item_Oranges = 3.0771841
item_Peanut_Butter = 1.9429716
item_Poached_Eggs = 0.1
item_Popcorn,Air_Popped = 13.223294
item_Scrambled_Eggs = 0.1
```

In [18]:
```
print("Total Cost of Diet per person = ", value(problem2.objective))
```

```
Total Cost of Diet per person =  4.512543427000001
```

Summary:

1. Minimum is 0.1 per serving
2. Only Celeray was selected (No Broccoli)
3. 3 types of Protein were selected (Kielbasa,Prk & Poached_Eggs & Scrambled_Eggs)

More conditions lead to more expensive meal compared to part 1

Additional Problem

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.

In [19]:
```
# Read Created Contraints file (Note Removed duplicated energy column)
const_large_df = pd.read_csv("diet_large_constraints.csv")
# Display Constraints df
const_large_df.head()
```

Out[19]:

	Nutrient	Min	Max
0	Protein	56.0	1000000
1	Carbohydrate	130.0	1000000
2	Energy	2400.0	1000000
3	Water	3700.0	1000000
4	Calcium	1000.0	2500

In [20]:
```
# Read Created Foods file
food_large_df = pd.read_csv("diet_large_foods.csv")
# replace missing with 0
food_large_df.fillna(0, inplace=True)
# Drop Fatty Acids columns
food_large_df = food_large_df.iloc[:,:-2]
# Display Constraints df
food_large_df.head()
```

Out[20]:

	Foods	Protein	Carbohydrate	Energy	Water	Calcium	Iron	Magnesium	Phosphorus	Potassium	...	Vit
0	Butter, salted	0.85	0.06	717	15.87	24.0	0.02	2.0	24.0	24.0	...	
1	Butter, whipped, with salt	0.85	0.06	717	15.87	24.0	0.16	2.0	23.0	26.0	...	
2	Butter oil, anhydrous	0.28	0.00	876	0.24	4.0	0.00	0.0	3.0	5.0	...	
3	Cheese, blue	21.40	2.34	353	42.41	528.0	0.31	23.0	387.0	256.0	...	
4	Cheese, brick	23.24	2.79	371	41.11	674.0	0.43	24.0	451.0	136.0	...	

5 rows × 28 columns

Preparing Data

In [21]:
```
# List of all foods
foods = food_large_df["Foods"].to_list()

# A dictionary of the costs of each type of the food
food_Cholesterol = dict(zip(food_large_df["Foods"], food_large_df["Cholesterol"]))

# List of all Nutrients
nutrients = const_large_df["Nutrient"].to_list()
# Create a dictionary for Min value per nutrient
min_nutrients = dict(zip(const_large_df["Nutrient"], const_large_df["Min"]))
# Create a dictionary for Max value per nutrient
max_nutrients = dict(zip(const_large_df["Nutrient"], const_large_df["Max"]))

# Create Food Nutrient Dictionary of Dictionaries
food_nutrient_dict = food_large_df.set_index("Foods").iloc[:,:-1].to_dict()
# Example of result dict
food_nutrient_dict["Protein"]["Butter, salted"]
```

Out[21]:
```
0.85
```

Initialize the Model

In [22]:
```
# Create the 'problem1' variable to contain the problem data
problem_large = LpProblem("The_large_diet_problem", LpMinimize)
```

Add the Variables

In [23]:
```
# A dictionary called 'food_vars' is created to contain the referenced Variables with a
food_vars = LpVariable.dicts("item", foods, lowBound=0)
```

Define the Objective Function

In [24]:
```
problem_large += (lpSum([food_Cholesterol[i] * food_vars[i] for i in foods]), "Total Cho
```

Define the Constraints

In [25]:
```
for nutrient in nutrients:
    # Define Min Constraint
    problem_large += (lpSum([food_nutrient_dict[nutrient][i] * food_vars[i] for i in foo
                    nutrient+"_large_min",)
    # Define Max Constraint
    problem_large += (lpSum([food_nutrient_dict[nutrient][i] * food_vars[i] for i in foo
                    nutrient+"_large_max",)
```

Run the Model

In [26]:
```
problem_large.solve()
# The status of the solution is printed to the screen
print("Status:", LpStatus[problem_large.status])
```

```
Status: Optimal
```

In [27]:
```
# Each of the variables is printed with it's resolved optimum value
for var in problem_large.variables():
    # Filter out the 0 variables
    if var.varValue != 0:
        print(var.name, "=", var.varValue)
```

```
item_Infant_formula,_MEAD_JOHNSON,_ENFAMIL,_NUTRAMIGEN,_with_iron,_p = 0.8
item_Infant_formula,_NESTLE,_GOOD_START_ESSENTIALS__SOY,__with_iron, = 0.026559039
item_Mung_beans,_mature_seeds,_raw = 0.097250444
item_Nuts,_almonds,_oil_roasted,_with_salt_added = 0.085543597
item_Oil,_vegetable,_sheanut = 1129.4851
item_Peppers,_hot_chile,_sun_dried = 0.46214551
item_Radishes,_oriental,_dried = 0.057356542
item_Snacks,_potato_chips,_plain,_salted = 0.78948828
item_Soup,_clam_chowder,_manhattan_style,_dehydrated,_dry = 0.050789628
item_Soybeans,_mature_seeds,_raw = 0.19034869
item_Soybeans,_mature_seeds,_roasted,_no_salt_added = 0.31430039
item_Spices,_mustard_seed,_yellow = 0.10993031
item_Tofu,_fried,_prepared_with_calcium_sulfate = 0.32586109
item_Tomatoes,_sun_dried = 0.21088097
item_Water,_bottled,_non_carbonated,_CALISTOGA = 9999.6745
```

In [28]:
```
print("Total Cholesterol of Diet per person = ", value(problem_large.objective))
```

```
Total Cholesterol of Diet per person =  0.0
```

The Optimization proposed 14 items. Although the objective is fulfilled, i.e. 0 Cholesterol intake, yet some items have been selected effectively twice e.g. Infant_formula. This shows the need for more constraints to optimize the model.