Homework #7: ISYE6501x. Introduction to Analytics Modeling

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!]*

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.]

Part 1

```
import pulp #for building/solving optimization models

import pandas as pd #for file reading

#Read data

data=pd.read_excel("dietSummer2018.xls")

#print(data)

data=data[0:64]
```

```
#Convert dataframe to list

data=data.values.tolist()
```

```
#----- Extract individual vectors of data for each nutrient

foods = [x[0] for x in data] # list of food names



calories = dict([(x[0], float(x[3])) for x in data]) # calories for each food
cholesterol = dict([(x[0], float(x[4])) for x in data])
totalFat = dict([(x[0], float(x[5])) for x in data]) # total  fat for each food
sodium = dict([(x[0], float(x[6])) for x in data])
carbohydrate = dict([(x[0], float(x[7])) for x in data])
fiber = dict([(x[0], float(x[8])) for x in data])
protein = dict([(x[0], float(x[9])) for x in data])
vitA = dict([(x[0], float(x[10])) for x in data])
vitC = dict([(x[0], float(x[11])) for x in data])
calcium = dict([(x[0], float(x[12])) for x in data])
iron = dict([(x[0], float(x[13])) for x in data])


cost=dict([(x[0], float(x[1])) for x in data]) # cost for each food
```

The stated task is minimization problem to find the cheapest diet.

```
#-----Create the minimization Lp problem

problem1=pulp.LpProblem('Diet1', pulp.LpMinimize)
```

Only one decision variable is needed to represent how much food we eat.

```
#-----Define the variables

#This decision variable represents how much food we eat

foodVars=pulp.LpVariable.dicts("Foods", foods, 0)
```

The objective function for this task is the cost of the diet.

```
#-----Create objective function

problem1 += pulp.lpSum([cost[f] * foodVars[f]] for f in foods), 'Total costs'
```

Constraints for this task are upper and lower bounds for each nutrient.

```
#----Add constraints upper and lower bounds for each nutrient
problem1 += pulp.lpSum([calories[f] * foodVars [f] for f in foods]) >=1500, 'min Calories'
problem1 += pulp.lpSum([calories[f] * foodVars [f] for f in foods]) <=2500, 'max Calories'
```

```
problem1 += pulp.lpSum([cholesterol[f] * foodVars [f] for f in foods]) >=30, 'min Choleste
rol'
problem1 += pulp.lpSum([cholesterol[f] * foodVars [f] for f in foods]) <=240, 'max Cholest
erol'

problem1 += pulp.lpSum([totalFat[f] * foodVars [f] for f in foods]) >=20, 'min Fat'
problem1 += pulp.lpSum([totalFat[f] * foodVars [f] for f in foods]) <=70, 'max Fat'

problem1 += pulp.lpSum([sodium[f] * foodVars [f] for f in foods]) >=800, 'min Sodium'
problem1 += pulp.lpSum([sodium[f] * foodVars [f] for f in foods]) <=2000, 'max Sodium'

problem1 += pulp.lpSum([carbohydrate[f] * foodVars [f] for f in foods]) >=130, 'min Carboh
ydrate'
problem1 += pulp.lpSum([carbohydrate[f] * foodVars [f] for f in foods]) <=450, 'max Carboh
ydrate'

problem1 += pulp.lpSum([fiber[f] * foodVars [f] for f in foods]) >=125, 'min Fiber'
problem1 += pulp.lpSum([fiber[f] * foodVars [f] for f in foods]) <=250, 'max Fiber'

problem1 += pulp.lpSum([protein[f] * foodVars [f] for f in foods]) >=60, 'min Protein'
problem1 += pulp.lpSum([protein[f] * foodVars [f] for f in foods]) <=100, 'max Protein'

problem1 += pulp.lpSum([vitA[f] * foodVars [f] for f in foods]) >=1000, 'min vit A'
problem1 += pulp.lpSum([vitA[f] * foodVars [f] for f in foods]) <=10000, 'max vit A'

problem1 += pulp.lpSum([vitC[f] * foodVars [f] for f in foods]) >=400, 'min vit C'
problem1 += pulp.lpSum([vitC[f] * foodVars [f] for f in foods]) <=5000, 'max vit C'

problem1 += pulp.lpSum([calcium[f] * foodVars [f] for f in foods]) >=700, 'min Calcium'
problem1 += pulp.lpSum([calcium[f] * foodVars [f] for f in foods]) <=1500, 'max Calcium'

problem1 += pulp.lpSum([iron[f] * foodVars [f] for f in foods]) >=10, 'min Iron'
problem1 += pulp.lpSum([iron[f] * foodVars [f] for f in foods]) <=40, 'max Iron'



#-----Solve the optimization problem

problem1.solve()
print ("Status:", pulp.LpStatus[problem1.status])
```

```
Status: Optimal
```

Status of the solutions shows that the model is solved and optimal solution is found.

```
#-----Print the output in a readable format

print("----The solution to this diet problem is-----")
for var in problem1.variables():
    if var.varValue> 0 :
        if str(var).find('Chosen'):
            print(str(var.varValue)+ " units of " +str(var))
print("Total cost of food is $%.2f" % pulp.value(problem1.objective))
```

```
----The solution to this diet problem is-----
52.64371 units of Foods_Celery,_Raw
0.25960653 units of Foods_Frozen_Broccoli
63.988506 units of Foods_Lettuce,Iceberg,Raw
2.2929389 units of Foods_Oranges
0.14184397 units of Foods_Poached_Eggs
13.869322 units of Foods_Popcorn,Air_Popped
Total cost of food is $4.34
```

Part 2

In the part 2 for each subtask the solution is printed and the new constraint is concatenated to previous ones.

To solve the 2nd part one more variable is added, to represent whether the food is chosen by solver or not.

```
#This decision variable represents whether we eat a certain food or not
chosenVars=pulp.LpVariable.dicts("Chosen", foods, 0, 1, 'Binary')
```

a) Constraint on serving size is added. If the food is selected then the serving should contain at least 0.1 units of it.

```
for f in foods:
    problem2 += foodVars[f] <= 10000*chosenVars[f]
    problem2 += foodVars[f] >= 0.1*chosenVars[f]
```

The calculated diet is the same as in the Part 1, because serving size of each food already is greater than 0.1.

```
----The solution to this diet problem is-----
52.64371 units of Foods_Celery,_Raw
0.25960653 units of Foods_Frozen_Broccoli
63.988506 units of Foods_Lettuce,Iceberg,Raw
2.2929389 units of Foods_Oranges
0.14184397 units of Foods_Poached_Eggs
13.869322 units of Foods_Popcorn,Air_Popped
Total cost of food is $4.34
```

If the serving size is increased to 0.2 then we can see small changes in the solution. The amount of Poached eggs is increased to 0.2, according to newly added constraint.

```
----The solution to this diet problem is-----
52.39131 units of Foods_Celery,_Raw
0.2491956 units of Foods_Frozen_Broccoli
64.745212 units of Foods_Lettuce,Iceberg,Raw
2.3183218 units of Foods_Oranges
```

```
0.2 units of Foods_Poached_Eggs
13.859897 units of Foods_Popcorn,Air_Popped
Total cost of food is $4.35
```

b) To add a constraint that either Celery of Broccoli is selected the equation that says we require eating at most one from the particular food is included.

```
problem2 += chosenVars['Celery, Raw']+chosenVars['Frozen Broccoli']<=1
```

As you can see Frozen Broccoli is not anymore included in the diet.

```
----The solution to this diet problem is-----
43.154119 units of Foods_Celery,_Raw
80.919121 units of Foods_Lettuce,Iceberg,Raw
3.0765161 units of Foods_Oranges
2.0464575 units of Foods_Peanut_Butter
0.14184397 units of Foods_Poached_Eggs
13.181772 units of Foods_Popcorn,Air_Popped
Total cost of food is $4.49
```

c) To get day-to-day variety in protein the quation that says we require to eat at least 3 from the particular food.

```
problem2 += chosenVars['Roasted Chicken']+chosenVars['Poached Eggs']+chosenVars['Scrambled
Eggs']+chosenVars['Bologna,Turkey']+chosenVars['Frankfurter, Beef']+chosenVars['Ham,Sliced
,Extralean']+chosenVars['Kielbasa,Prk']+chosenVars['Pizza W/Pepperoni']+chosenVars['Taco']
+chosenVars['Hamburger W/Toppings']+chosenVars['Hotdog, Plain']+chosenVars['Pork']+chosenV
ars['Sardines in Oil']+chosenVars['White Tuna in Water']+chosenVars['Chicknoodl Soup']+cho
senVars['Splt Pea&Hamsoup']+chosenVars['Vegetbeef Soup']+chosenVars['Neweng Clamchwd']>=3
```

As you can see now solution contains 3 protein sources.

```
----The solution to this diet problem is-----
42.399358 units of Foods_Celery,_Raw
0.1 units of Foods_Kielbasa,Prk
82.802586 units of Foods_Lettuce,Iceberg,Raw
3.0771841 units of Foods_Oranges
1.9429716 units of Foods_Peanut_Butter
0.1 units of Foods_Poached_Eggs
13.223294 units of Foods_Popcorn,Air_Popped
0.1 units of Foods_Scrambled_Eggs
Total cost of food is $4.51
```

To get more "human" diet constraints should be added to limit upper bounds of serving sizes.