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!]

Solution

First, lets import pandas to make data manipulation a little easier

In [1]:
```
import pandas as pd
import numpy as np
from pulp import *
```

In [2]:
```
data = pd.read_excel('diet.xls')
```

Great, data is imported! Now, lets think back to our videos on optimization. When solving a linear optimization problem, we have three things we need to identify. we need to identify 1. the variables 2. the constraints and 3. the objective function we want to maximize/minimize. So let's do that together one step at a time.

First lets get our data out of the Excel file and into a more python friendly format.

First lets drop the bottom few rows, since it looks like they do not contain food info, they contain data on the min and max nutrients

In [3]:
```
dataTable = data[0:64]
foods = dataTable['Foods'].tolist()
data_dict = dataTable.to_dict('records')
```

Let's start by defining the problem we want to solve. This is kind of working backwards, since this is starting with the 3rd step I mentioned above, but this is required for adding constraints and variables to our problem so we will start here.

In [4]:
```
prob = LpProblem("Minimize_Food_Costs", LpMinimize)
```

Ok, now lets jump back up to step 1, defining our variables. What will our variables be? Well in order to minimize costs, we need to choose an amount of food we want to include in the diet. So those will be our variables. So we need to create a bunch of variables, one for each food.

Each variable takes four parameters.

1. An arbitrary variable name
2. A lower bound for the variable
3. An upper bound of the variable amount
4. Whether the variable is continuous or discrete.

So lets just create one variable for the first food in our list to see how this works.

In [5]:
```
food_1 = LpVariable(name='Frozen Broccoli',
                    lowBound=0,
                    upBound=None,
                    cat=LpContinuous)
```

So for the first arg, we gave it a name. The second we set a lower bound of 0, since we cannot eat a negative amount of food. The third argument we put an upper bound of None. This means our model is not constrained by an upper bound amount. Theoretically, we could eat an infinite amount of one type of food. Now that is not true in practice, we can't eat 100 pounds of broccoli, our stomachs could not hold that much! We will handle those constraints in the aggregate when we define our constraints. The last arg, cat, is whether it is a continuous or integer variable. Since we don't need to eat only one broccoli, we can eat half of a broccoli or 2/3, we will use a continuous variable arg.

Now, instead of typing out variables for all 60+ of our foods, instead we will iterate over the list of our foods, defining a variable for each food and storing it in a list.

In [6]:
```
foodQuant = {food['Foods']: LpVariable(name=food['Foods'],
                                       lowBound=0,
                                       upBound=None,
                                       cat=LpContinuous
                                       )
             for food in data_dict}
```

Now we need to create another variable, which is whether the food is eaten or not. In our function, this would mean a value of 0 if the food is not selected, and 1 if it is selected. Why is that? Well, the cost of a specific food is equal to the cost of the food, time the variable amount of the food. Then, if the food is not selected, we multiply it by 0, so that there is no cost associated with it. We also need to create a link to the food type, and we'll do that by using a dictionary, with the key being the food name and the value being a variable for that food type of either 0 or 1.

In [7]:
```
foodSelected = {food['Foods']: LpVariable(('selected_' + food['Foods']),
                                          lowBound=0,
                                          upBound=1,
                                          cat=LpInteger)
                for food in data_dict}
```

Now that we have the variables defined, it's time to define the objective function. Again this is a little out of order from what is intuitive, since we are required to define the objective function before we can start adding constraints. Our objective function is just food quantity * food cost.

First lets create a cost dictionary to make these values a little easier to access.

In [8]:
```
foodCosts = dict(zip(dataTable.Foods, dataTable['Price/ Serving']))

prob += lpSum([foodCosts[food] * foodQuant[food] for food in foods]), \
        'Total Cost'
```

Now it is time to add our constraints. The first constraint the homework asks us to consider is that the food selections meet the minimum and maximum instrumental requirements. Let's start by structuring our data in a way that enables us to access those nutritional requirements.

In [9]:
```
nutrients = data.columns.to_list()[3:]
nut_min = {nutrient: data[nutrient].iloc[65] for nutrient in
           nutrients}

nut_max = {nutrient: data[nutrient].iloc[66] for nutrient in
           nutrients}
```

Now that our caloric mins and maxes are in a dictionary, lets define our constraints in a way that does not allow the sum of the selected foods to be less than the min or greater than the max for each nutrition.

For each nutrient, we need to loop over each nutrient and create a new constraint.

In [10]:
```
for nutrient in nutrients:
    nutrient_value = dict(zip(dataTable.Foods, dataTable[nutrient]))
    prob += lpSum([nutrient_value[food] * foodQuant[food] for food in
                   foods]) >= nut_min[nutrient], (nutrient + ' min '
                                                  'requirement')

    prob += lpSum([nutrient_value[food] * foodQuant[food] for food in
                   foods]) <= nut_max[nutrient], (nutrient + ' max '
                                                  'requirement')
```

Now let's solve the optimization problem. We will need to do this more than once, so lets wrap it in a function to call it again later.

In [11]:
```
def solve(prob_):
    prob_.solve()
    print("Status:", LpStatus[prob_.status])
    for v in prob_.variables():
        if v.varValue > 0:
            print(v.name, "=", v.varValue)
        else:
            pass
    print("Total Cost of Ingredients per meal = ",
          round(value(prob_.objective), 2))
```

In [12]:
```
solve(prob)
```

```
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /Users/chrismesser/opt/anaconda3/lib/python3.9/site-packages/pulp/solverdir/cbc/osx/64/cbc /var/folders/f3/g1154c6d5hb7y4_s6bzsyw8m0000gn/T/50e354dd62a84f84a886
7e646d232831-pulp.mps timeMode elapsed branch printingOptions all solution /var/folders/f3/g1154c6d5hb7y4_s6bzsyw8m0000gn/T/50e354dd62a84f84a8867e646d232831-pulp.sol (default
strategy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 27 COLUMNS
At line 1286 RHS
At line 1309 BOUNDS
At line 1310 ENDATA
Problem MODEL has 22 rows, 64 columns and 1194 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 22 (0) rows, 64 (0) columns and 1194 (0) elements
0  Obj 0 Primal inf 21.63092 (11)
9  Obj 4.3371168
Optimal - objective value 4.3371168
Optimal objective 4.33711681 - 9 iterations time 0.002
Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.00   (Wallclock seconds):       0.02

Status: Optimal
Celery,_Raw = 52.64371
Frozen_Broccoli = 0.25960653
Lettuce,Iceberg,Raw = 63.988506
Oranges = 2.2929389
Poached_Eggs = 0.14184397
Popcorn,Air_Popped = 13.869322
Total Cost of Ingredients per meal =  4.34
```

Now to add the constraints for parts B and C. Link the food selected constraint to the quantity of food. If a food is selected, then it must have some food. Also, if a food is selected, it must have at least .1 servings

In [13]:
```
for food in foods:
    prob += foodQuant[food] >= foodSelected[food] * .1
    prob += foodSelected[food] >= foodQuant[food] * .0000001
```

Now add a constraint that there can only be raw broccoli or celery

In [14]:
```
prob += foodSelected['Frozen Broccoli'] + foodSelected['Celery, Raw'] <= 1
```

And now add the meat constraint. Lets make a list of meats, then loop over it and say that the total number of meats must be greater than 3

In [16]:
```
meats = ['Roasted Chicken', 'Poached Eggs', 'Scrambled Eggs', 'Bologna,Turkey',
         'Frankfurter, Beef', 'Ham,Sliced,Extralean', 'Kielbasa,Prk',
         'Pizza W/Pepperoni', 'Hamburger W/Toppings', 'Hotdog, Plain', 'Pork',
         'Sardines in Oil', 'White Tuna in Water', 'Chicknoodl Soup',
         'Splt Pea&Hamsoup']

prob += lpSum([foodSelected[meat] for meat in meats]) >= 3
```

Now finally, solve the complete problem!

In [17]:
```
solve(prob)
```

```
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /Users/chrismesser/opt/anaconda3/lib/python3.9/site-packages/pulp/solverdir/cbc/osx/64/cbc /var/folders/f3/g1154c6d5hb7y4_s6bzsyw8m0000gn/T/7778f05c5a534182a0c9
44a894b93b8d-pulp.mps timeMode elapsed branch printingOptions all solution /var/folders/f3/g1154c6d5hb7y4_s6bzsyw8m0000gn/T/7778f05c5a534182a0c944a894b93b8d-pulp.sol (default
strategy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 158 COLUMNS
At line 1833 RHS
At line 1987 BOUNDS
At line 2052 ENDATA
Problem MODEL has 153 rows, 128 columns and 1482 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Continuous objective value is 4.38006 - 0.00 seconds
Cgl0003I 0 fixed, 0 tightened bounds, 14 strengthened rows, 0 substitutions
Cgl0004I processed model has 141 rows, 128 columns (64 integer (64 of which binary)) and 870 elements
Cbc0038I Initial state - 8 integers unsatisfied sum - 1.63213
Cbc0038I Pass   1: suminf.    0.52908 (2) obj. 5.37039 iterations 51
Cbc0038I Solution found of 5.37039
Cbc0038I Relaxing continuous gives 5.37039
Cbc0038I Before mini branch and bound, 55 integers at bound fixed and 53 continuous
Cbc0038I Full problem 141 rows 128 columns, reduced to 31 rows 20 columns
Cbc0038I Mini branch and bound improved solution from 5.37039 to 4.51296 (0.02 seconds)
Cbc0038I Round again with cutoff of 4.50011
Cbc0038I Pass   2: suminf.    0.61600 (4) obj. 4.50011 iterations 9
Cbc0038I Pass   3: suminf.    0.55610 (2) obj. 4.50011 iterations 58
Cbc0038I Solution found of 4.50011
Cbc0038I Infeasible when relaxing continuous!

Cbc0038I Before mini branch and bound, 56 integers at bound fixed and 54 continuous
Cbc0038I Full problem 141 rows 128 columns, reduced to 29 rows 18 columns
Cbc0038I Mini branch and bound did not improve solution (0.02 seconds)
Cbc0038I Round again with cutoff of 4.47699
Cbc0038I Pass   4: suminf.    0.65246 (4) obj. 4.47699 iterations 0
Cbc0038I Pass   5: suminf.    0.48977 (2) obj. 4.47699 iterations 58
Cbc0038I Solution found of 4.47699
Cbc0038I Infeasible when relaxing continuous!

Cbc0038I Before mini branch and bound, 56 integers at bound fixed and 54 continuous
Cbc0038I Full problem 141 rows 128 columns, reduced to 29 rows 18 columns
Cbc0038I Mini branch and bound did not improve solution (0.02 seconds)
Cbc0038I Round again with cutoff of 4.44926
Cbc0038I Pass   6: suminf.    0.69620 (4) obj. 4.44926 iterations 0
Cbc0038I Pass   7: suminf.    0.41018 (2) obj. 4.44926 iterations 58
Cbc0038I Solution found of 4.44926
Cbc0038I Infeasible when relaxing continuous!

Cbc0038I Before mini branch and bound, 56 integers at bound fixed and 54 continuous
Cbc0038I Full problem 141 rows 128 columns, reduced to 29 rows 18 columns
Cbc0038I Mini branch and bound did not improve solution (0.03 seconds)
Cbc0038I Round again with cutoff of 4.42985
Cbc0038I Pass   8: suminf.    0.72682 (4) obj. 4.42985 iterations 0
Cbc0038I Pass   9: suminf.    0.35447 (2) obj. 4.42985 iterations 58
Cbc0038I Solution found of 4.42985
Cbc0038I Infeasible when relaxing continuous!

Cbc0038I Before mini branch and bound, 56 integers at bound fixed and 54 continuous
Cbc0038I Full problem 141 rows 128 columns, reduced to 29 rows 18 columns
Cbc0038I Mini branch and bound did not improve solution (0.03 seconds)
Cbc0038I Round again with cutoff of 4.41173
Cbc0038I Pass  10: suminf.    0.75539 (4) obj. 4.41173 iterations 0
Cbc0038I Pass  11: suminf.    0.30248 (2) obj. 4.41173 iterations 58
Cbc0038I Solution found of 4.41173
Cbc0038I Infeasible when relaxing continuous!

Cbc0038I Before mini branch and bound, 56 integers at bound fixed and 54 continuous
Cbc0038I Full problem 141 rows 128 columns, reduced to 29 rows 18 columns
Cbc0038I Mini branch and bound did not improve solution (0.03 seconds)
Cbc0038I After 0.03 seconds - Feasibility pump exiting with objective of 4.41173 - took 0.02 seconds
Cbc0012I Integer solution of 4.5216303 found by feasibility pump after 0 iterations and 0 nodes (0.03 seconds)
Cbc0012I Integer solution of 4.5175584 found by DiveCoefficient after 0 iterations and 0 nodes (0.04 seconds)
Cbc0038I Full problem 141 rows 128 columns, reduced to 29 rows 18 columns
Cbc0012I Integer solution of 4.5125434 found by DiveCoefficient after 29 iterations and 0 nodes (0.04 seconds)
Cbc0031I 4 added rows had average density of 17.75
Cbc0013I At root node, 4 cuts changed objective from 4.3845731 to 4.5125434 in 6 passes
Cbc0014I Cut generator 0 (Probing) - 2 row cuts average 2.0 elements, 56 column cuts (56 active)  in 0.001 seconds - new frequency is 1
Cbc0014I Cut generator 1 (Gomory) - 8 row cuts average 31.6 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is 1
Cbc0014I Cut generator 2 (Knapsack) - 2 row cuts average 24.0 elements, 0 column cuts (0 active)  in 0.001 seconds - new frequency is 1
Cbc0014I Cut generator 3 (Clique) - 0 row cuts average 0.0 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 4 (MixedIntegerRounding2) - 1 row cuts average 36.0 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 5 (FlowCover) - 0 row cuts average 0.0 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 6 (TwoMirCuts) - 10 row cuts average 32.2 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is 1
Cbc0001I Search completed - best objective 4.512543398463767, took 29 iterations and 0 nodes (0.04 seconds)
Cbc0035I Maximum depth 0, 1 variables fixed on reduced cost
Cuts at root node changed objective from 4.38457 to 4.51254
Probing was tried 6 times and created 58 cuts of which 0 were active after adding rounds of cuts (0.001 seconds)
Gomory was tried 6 times and created 8 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Knapsack was tried 6 times and created 2 cuts of which 0 were active after adding rounds of cuts (0.001 seconds)
Clique was tried 6 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 6 times and created 1 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
FlowCover was tried 6 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
TwoMirCuts was tried 6 times and created 10 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
ZeroHalf was tried 1 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Result - Optimal solution found

Objective value:                4.51254340
Enumerated nodes:               0
Total iterations:               29
Time (CPU seconds):             0.03
Time (Wallclock seconds):       0.04

Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.03   (Wallclock seconds):       0.05

Status: Optimal
Celery,_Raw = 42.399358
Kielbasa,Prk = 0.1
Lettuce,Iceberg,Raw = 82.802586
Oranges = 3.0771841
Peanut_Butter = 1.9429716
Poached_Eggs = 0.1
Popcorn,Air_Popped = 13.223294
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
Total Cost of Ingredients per meal =  4.51
```

And there we have it, our final answer. We are to select Celery,_Raw = 42.399358 Kielbasa,Prk = 0.1 Lettuce,Iceberg,Raw = 82.802586 Oranges = 3.0771841 Peanut_Butter = 1.9429716 Poached_Eggs = 0.1 Popcorn,Air_Popped = 13.223294 Scrambled_Eggs = 0.1

And our total cost is 4.51

Bonus Question

Now, lets solve the large diet problem, optimizing on cholesterol. I won't annotate this one much, since it is largely the same as above.

In [20]:
```
data = pd.read_excel('diet_large.xls', skiprows=1)
data = data.fillna(0)

dataTable = data[0:7146]
dataTable = dataTable.fillna(0)

foods = dataTable['Long_Desc'].tolist()
data_dict = dataTable.to_dict('records')

prob2 = LpProblem("Minimize_Large_Food_Carbs", LpMinimize)

foodQuant = {food['Long_Desc']: LpVariable(name=food['Long_Desc'],
                                           lowBound=0,
                                           upBound=None,
                                           cat=LpContinuous
                                           )
             for food in data_dict}

foodSelected = {
    food['Long_Desc']: LpVariable(('selected_' + food['Long_Desc']),
                                  lowBound=0,
                                  upBound=1,
                                  cat=LpInteger)
    for food in data_dict}

foodCholesterol = dict(zip(dataTable.Long_Desc, dataTable['Cholesterol']))

prob2 += lpSum([foodCholesterol[food] * foodQuant[food] for food in foods]), \
         'Total Cholesterol'

nutrients = data.columns.to_list()[1:]

nut_min = {nutrient: data[nutrient].iloc[7147] for nutrient in
           nutrients}

nut_max = {nutrient: data[nutrient].iloc[7149] for nutrient in
           nutrients}

for nutrient in nutrients:
    if nut_max[nutrient] > 0 and nut_min[nutrient] > 0:
        nutrient_value = dict(zip(dataTable.Long_Desc, dataTable[nutrient]))
        prob2 += lpSum([nutrient_value[food] * foodQuant[food] for food in
                        foods]) >= nut_min[nutrient], (nutrient + ' min '
                                                       'requirement'
                                                      )

        prob2 += lpSum([nutrient_value[food] * foodQuant[food] for food in
                        foods]) <= nut_max[nutrient], (nutrient + ' max '
                                                       'requirement'
                                                      )


solve(prob2)
```

```
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /Users/chrismesser/opt/anaconda3/lib/python3.9/site-packages/pulp/solverdir/cbc/osx/64/cbc /var/folders/f3/g1154c6d5hb7y4_s6bzsyw8m0000gn/T/a2afcc37517541899e40
93ff4d071bfc-pulp.mps timeMode elapsed branch printingOptions all solution /var/folders/f3/g1154c6d5hb7y4_s6bzsyw8m0000gn/T/a2afcc37517541899e4093ff4d071bfc-pulp.sol (default
strategy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 59 COLUMNS
At line 270558 RHS
At line 270613 BOUNDS
At line 270614 ENDATA
Problem MODEL has 54 rows, 6286 columns and 267476 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 48 (-6) rows, 6217 (-69) columns and 238362 (-29114) elements
Perturbing problem by 0.001% of 23376.962 - largest nonzero change 0.00068163095 ( 0.07499568%) - largest zero change 0.00068154335
0  Obj 0 Primal inf 309.95886 (26)
25  Obj 0.00025167115
Optimal - objective value 0
After Postsolve, objective 0, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective 0 - 25 iterations time 0.042, Presolve 0.03
Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.12   (Wallclock seconds):       0.13

Status: Optimal
Beans,_adzuki,_mature_seeds,_raw = 0.21057627
Cocoa_mix,_no_sugar_added,_powder = 0.47860409
Coriander_(cilantro)_leaves,_raw = 0.067451479
Egg,_white,_dried,_flakes,_glucose_reduced = 0.095970311
Infant_formula,_MEAD_JOHNSON,_ENFAMIL,_NUTRAMIGEN,_with_iron,_p = 0.8
Infant_formula,_NESTLE,_GOOD_START_ESSENTIALS__SOY,__with_iron, = 0.29000292
Margarine,_industrial,_non_dairy,_cottonseed,_soy_oil_(partiall = 0.0022115549
Oil,_vegetable,_sunflower,_linoleic,_(hydrogenated) = 0.99621953
Sauce,_mole_poblano,_dry_mix,_single_brand = 0.082093778
Snacks,_potato_chips,_barbecue_flavor = 0.41173509
Snacks,_potato_chips,_plain,_salted = 0.059152845
Snacks,_potato_sticks = 0.18720554
Soybeans,_mature_seeds,_roasted,_no_salt_added = 0.22690017
Spaghetti,_spinach,_dry = 0.12931103
Tomato_powder = 0.42495277
Tomatoes,_sun_dried = 0.086944469
Veal,_variety_meats_and_by_products,_pancreas,_cooked,_braised = 0.0092759904
Water,_bottled,_non_carbonated,_CALISTOGA = 9999.7849
Total Cost of Ingredients per meal =  0.0
```