Optimization Models

Problem:

Inspired by the diet optimization problem of the 1930s/1940s US Army, the goal here is to solve several different optimization problems with regards to diet, two different data sets, and different constraints.

Models:

1. `Basic Optimization Problem` : find the cheapest diet that satisfies the minimum and maximum daily nutritional constraints.
2. `Additional Constraints Problem` : find the cheapest diet that satisfies the minimum and maximum daily nutritional constraints and the below constraints:
   - If a food is selected, then a min of 1/10 serving must be chosen
   - Only one of celery or broccoli can be selected
   - At least three kinds of meat/poultry/fish/eggs should be selected
3. `More Complex Data Problem` : find the lowest-cholesterol diet
4. `Maximization Problem` : find the highest-protein diet

In [163…
```
#! pip install pulp
#! pip install xlrd
```

In [164…
```
# Import PuLP modeler functions
from pulp import *
import pandas as pd
import numpy as np

pd.set_option('display.max_rows', None)

import warnings
warnings.filterwarnings("ignore")
```

In [165…
```
# read in the data
diet = pd.read_excel('diet_folder/diet.xls', sheet_name='Sheet1')
```

Optimization Model 1

Basic Optimization problem: find the cheapest diet that satisfies the minimum and maximum daily nutritional constraints.

In [166…
```
# grab only the food rows
diet = diet[0:64]
```

In [167…
```
# convert dataframe to list
diet=diet.values.tolist()
```

In [168…
```
# extract vectors of data for each nutrient
foods = [x[0] for x in diet] #list of food names
cost = dict([(x[0], float(x[1])) for x in diet]) # cost for each food
calories = dict([(x[0], float(x[3])) for x in diet]) # calories for each food
cholesterol = dict([(x[0], float(x[4])) for x in diet]) # cholesterol for each food
totalFat = dict([(x[0], float(x[5])) for x in diet]) # total fat for each food
sodium = dict([(x[0], float(x[6])) for x in diet]) # sodium for each food
carbohydrates = dict([(x[0], float(x[7])) for x in diet]) # carbohydrates for each food
dietaryFiber = dict([(x[0], float(x[8])) for x in diet]) # fibre for each food
protein = dict([(x[0], float(x[9])) for x in diet]) # protein for each food
vitaminA = dict([(x[0], float(x[10])) for x in diet]) # vitamin A for each food
vitaminC = dict([(x[0], float(x[11])) for x in diet]) # vitamin C for each food
calcium = dict([(x[0], float(x[12])) for x in diet]) # calcium for each food
iron = dict([(x[0], float(x[13])) for x in diet]) # iron for each food
```

In [169…
```
# create LP problem; this problem is a minimization problem to find the lowest cost
prob1 = LpProblem(name='Food optimization', sense=LpMinimize)
```

In [170…
```
# define the variable for each food, with a lower limit of zero since you can't eat any negative amounts
foodVars = LpVariable.dicts("Foods", foods, 0)
```

In [171…
```
# Note that the first function we add is taken to be the objective function
prob1 += lpSum([cost[f] * foodVars[f] for f in foods]), 'Total Cost'
```

In [172…
```
# add the nutritional constraints for each variable
prob1 += lpSum([calories[f] * foodVars[f] for f in foods]) >= 1500, 'min Calories'
prob1 += lpSum([calories[f] * foodVars[f] for f in foods]) <= 2500, 'max Calories'

prob1 += lpSum([cholesterol[f] * foodVars[f] for f in foods]) >= 30, 'min Cholesterol'
prob1 += lpSum([cholesterol[f] * foodVars[f] for f in foods]) <= 240, 'max Cholesterol'

prob1 += lpSum([totalFat[f] * foodVars[f] for f in foods]) >= 20, 'min Fat'
prob1 += lpSum([totalFat[f] * foodVars[f] for f in foods]) <= 70, 'max Fat'

prob1 += lpSum([sodium[f] * foodVars[f] for f in foods]) >= 800, 'min Sodium'
prob1 += lpSum([sodium[f] * foodVars[f] for f in foods]) <= 2000, 'max Sodium'

prob1 += lpSum([carbohydrates[f] * foodVars[f] for f in foods]) >= 130, 'min Carbohydrates'
prob1 += lpSum([carbohydrates[f] * foodVars[f] for f in foods]) <= 450, 'max Carbohydrates'

prob1 += lpSum([dietaryFiber[f] * foodVars[f] for f in foods]) >= 125, 'min Fiber'
prob1 += lpSum([dietaryFiber[f] * foodVars[f] for f in foods]) <= 250, 'max Fiber'

prob1 += lpSum([protein[f] * foodVars[f] for f in foods]) >= 60, 'min Protein'
prob1 += lpSum([protein[f] * foodVars[f] for f in foods]) <= 100, 'max Protein'

prob1 += lpSum([vitaminA[f] * foodVars[f] for f in foods]) >= 1000, 'min Vit_A'
prob1 += lpSum([vitaminA[f] * foodVars[f] for f in foods]) <= 10000, 'max Vit_A'

prob1 += lpSum([vitaminC[f] * foodVars[f] for f in foods]) >= 400, 'min Vit_C'
prob1 += lpSum([vitaminC[f] * foodVars[f] for f in foods]) <= 5000, 'max Vit_C'

prob1 += lpSum([calcium[f] * foodVars[f] for f in foods]) >= 700, 'min Calcium'
prob1 += lpSum([calcium[f] * foodVars[f] for f in foods]) <= 1500, 'max Calcium'

prob1 += lpSum([iron[f] * foodVars[f] for f in foods]) >= 10, 'min Iron'
prob1 += lpSum([iron[f] * foodVars[f] for f in foods]) <= 40, 'max Iron'
```

In [173…
```
# solve the optimization problem
prob1.solve()
```

Out[173… 1

In [174…
```
# print the output
print()
print("---------The solution to this diet problem is----------")
for var in prob1.variables():
    if var.varValue > 0:
        print(str(var.varValue)+" units of "+str(var).replace('Foods_','') )
print()
print("Total cost of food = $%.2f" % value(prob1.objective))
```

```
---------The solution to this diet problem is----------
52.64371 units of Celery,_Raw
0.25960653 units of Frozen_Broccoli
63.988506 units of Lettuce,Iceberg,Raw
2.2929389 units of Oranges
0.14184397 units of Poached_Eggs
13.869322 units of Popcorn,Air_Popped

Total cost of food = $4.34
```

Optimization Model 1 - Alternative Method

Basic Optimization problem: find the cheapest diet that satisfies the minimum and maximum daily nutritional constraints.

In [175…
```
# read in the data
data = pd.read_excel('diet_folder/diet.xls', sheet_name='Sheet1')
```

In [176…
```
# grab only the food rows
dataTable = data[0:64]
```

In [177…
```
# convert dataframe to list
dataTable = dataTable.values.tolist()
```

In [178…
```
# get the nutrient names / column headers
nutrientNames = list(data.columns.values)  # column headers (nutrient names are in columns 3-13; Excel calls them D-N)
```

In [179…
```
# get the min and max nutrient values
minVal = data[65:66].values.tolist() # minimum nutrient values
maxVal = data[66:67].values.tolist() # maximum nutrient values
```

In [180…
```
# extract individual vectors of data using dictionaries
foods = [j[0] for j in dataTable] #list of food names

cost = dict([(j[0], float(j[1])) for j in dataTable]) # cost for each food

nutrients = []
for i in range(0,11): # for loop running through each nutrient: 11 times starting with 0
    nutrients.append(dict([(j[0], float(j[i+3])) for j in dataTable])) # amount of nutrient i in food j
```

In [181…
```
# create LP problem with the lowest cost
prob = LpProblem(name='Food optimization', sense=LpMinimize)
```

In [182…
```
# define the variables - one variable for each food, with a lower limit of zero
foodVars = LpVariable.dicts("Foods", foods, 0)
```

In [183…
```
# create the objective function
prob += lpSum([cost[f] * foodVars[f] for f in foods]), 'Total Cost'
```

In [184…
```
# add constraints for each nutrient
for i in range(0,11): # for loop running through each nutrient: 11 times starting with 0
    prob += lpSum([nutrients[i][j] * foodVars[j] for j in foods]) >= minVal[0][i+3], 'min nutrient ' + nutrientNames[i]
    prob += lpSum([nutrients[i][j] * foodVars[j] for j in foods]) <= maxVal[0][i+3], 'max nutrient ' + nutrientNames[i]
```

In [185…
```
# solve the optimization problem
prob.solve()
```

Out[185… 1

In [186…
```
# print output
print()
print("---------The solution to the diet problem is----------")
for var in prob.variables():
    if var.varValue > 0:
        print(str(var.varValue)+" units of "+str(var).replace('Foods_','') )
print()
print("Total cost of food = $%.2f" % value(prob.objective))
```

```
---------The solution to the diet problem is----------
52.64371 units of Celery,_Raw
0.25960653 units of Frozen_Broccoli
63.988506 units of Lettuce,Iceberg,Raw
2.2929389 units of Oranges
0.14184397 units of Poached_Eggs
13.869322 units of Popcorn,Air_Popped

Total cost of food = $4.34
```

Optimization Model 2

2. Additional constraints problem: find the cheapest diet that satisfies the minimum and maximum daily nutritional constraints and the below constraints:
   - If a food is selected, then a min of 1/10 serving must be chosen
   - Only one of celery or broccoli can be selected
   - At least three kinds of meat/poultry/fish/eggs should be selected

In [187…
```
# read in the data
data = pd.read_excel('diet_folder/diet.xls', sheet_name='Sheet1')
```

In [188…
```
# grab only the food rows
dataTable = data[0:64] # rows 0:64 (Excel calls them 1-65) is the food data table
dataTable = dataTable.values.tolist() # Convert dataframe to list
```

In [189…
```
# get the nutrient names / column headers
nutrientNames = list(data.columns.values)
```

In [190…
```
# get the min/max values of the nutrients
minVal = data[65:66].values.tolist() # minimum nutrient values
maxVal = data[66:67].values.tolist() # maximum nutrient values
```

In [191…
```
# extract individual vectors of data
foods = [j[0] for j in dataTable] #list of food names

cost = dict([(j[0], float(j[1])) for j in dataTable]) # cost for each food

nutrients = []
for i in range(0,11): # for loop running through each nutrient: 11 times starting with 0
    nutrients.append(dict([(j[0], float(j[i+3])) for j in dataTable])) # amount of nutrient i in food j
```

In [192…
```
# This problem is a minimization problem (find the *lowest* cost)
prob = LpProblem(name='Food optimization', sense=LpMinimize)
```

In [193…
```
# define the variables
foodVars = LpVariable.dicts("Foods", foods, 0) # lower limit of zero
foodVars_selected = LpVariable.dicts("food_select",foods,0,1,LpBinary) # create binary integer variables for whether a food is eaten
```

In [194…
```
# create objective function
prob += lpSum([cost[f] * foodVars[f] for f in foods]), 'Total Cost'
```

In [195…
```
# add nutritional constraints
for i in range(0,11): # for loop running through each nutrient: 11 times starting with 0
    prob += lpSum([nutrients[i][j] * foodVars[j] for j in foods]) >= minVal[0][i+3], 'min nutrient ' + nutrientNames[i]
    prob += lpSum([nutrients[i][j] * foodVars[j] for j in foods]) <= maxVal[0][i+3], 'max nutrient ' + nutrientNames[i]
```

In [196…
```
# add additional constraints

# 1. If a food is selected, then a min of 1/10 serving must be chosen
for food in foods:
    prob += foodVars[food] >= 0.1 * foodVars_selected[food]
# If any of a food is eaten, its binary variable must be 1
for food in foods:
    prob += foodVars_selected[food] >= foodVars[food]*0.0000001
```

In [197…
```
# 2. Only one of celery or broccoli can be selected
prob += foodVars_selected['Frozen Broccoli'] + foodVars_selected['Celery, Raw'] <= 1
```

In [198…
```
# 3. At least three kinds of meat/poultry/fish/eggs should be selected

prob += foodVars_selected['Roasted Chicken'] + foodVars_selected['Poached Eggs'] \
        + foodVars_selected['Scrambled Eggs'] + foodVars_selected['Bologna,Turkey'] \
        + foodVars_selected['Frankfurter, Beef'] + foodVars_selected['Ham,Sliced,Extralean'] \
        + foodVars_selected['Kielbasa,Prk'] + foodVars_selected['Pizza W/Pepperoni'] \
        + foodVars_selected['Hamburger W/Toppings'] \
        + foodVars_selected['Hotdog, Plain'] + foodVars_selected['Pork'] \
        + foodVars_selected['Sardines in Oil'] + foodVars_selected['White Tuna in Water'] \
        + foodVars_selected['Chicknoodl Soup'] + foodVars_selected['Splt Pea&Hamsoup'] \
        + foodVars_selected['Vegetbeef Soup'] + foodVars_selected['Neweng Clamchwd'] \
        + foodVars_selected['New E Clamchwd,W/Mlk'] + foodVars_selected['Beanbacn Soup,W/Watr'] >= 3
```

In [199…
```
# solve the optimization problem
prob.solve()
```

Out[199… 1

In [200…
```
# print output
print()
print("---------The solution to the diet problem is----------")
for var in prob.variables():
    if var.varValue > 0 and "food_select" not in var.name: # Print non binary variables
        print(str(var.varValue)+" units of "+str(var).replace('Foods_','') )
print()
print("Total cost of food = $%.2f" % value(prob.objective))
```

```
---------The solution to the diet problem is----------
42.399358 units of Celery,_Raw
0.1 units of Kielbasa,Prk
82.802586 units of Lettuce,Iceberg,Raw
3.0771841 units of Oranges
1.9429716 units of Peanut_Butter
0.1 units of Poached_Eggs
13.223294 units of Popcorn,Air_Popped
0.1 units of Scrambled_Eggs

Total cost of food = $4.51
```

Optimization Model 3

More Complex Data Problem: find the lowest-cholesterol diet

In [201…
```
# read in the data
data = pd.read_excel("diet_folder/diet_large.xls", skiprows = 1, header = 0) # read all data
```

In [202…
```
# grab food data
dataTable = data[0:7146] # rows 0:7146 (Excel calls them 2-7148; remember we skipped the blank first row in the read call) is the food data table
dataTable = dataTable.values.tolist() # Convert dataframe to list
```

In [203…
```
# get nutrient information
nutrientNames = list(data.columns.values) # column headers (nutrient names are in columns 3-13; Excel calls them D-N)
numNutrients = len(nutrientNames) - 1 # don't count the food-name column
```

In [204…
```
# blank elements are read as 'nan', so need to replace them with zero
for i in range(0,7146):
    for j in range(1,numNutrients):
        if np.isnan(dataTable[i][j]):
            dataTable[i][j] = 0
```

In [205…
```
# get min and max nutrient values
minVal = data[7147:7148].values.tolist() # minimum nutrient values
maxVal = data[7149:7151].values.tolist() # maximum nutrient values
```

In [206…
```
# Extract individual vectors of data
foods = [j[0] for j in dataTable] #list of food names

cost = dict([(j[0], float(j[nutrientNames.index('Cholesterol')])) for j in dataTable]) # cholesterol for each food

nutrients = []
for i in range(0,numNutrients): # for loop running through each nutrient
    nutrients.append(dict([(j[0], float(j[i+1])) for j in dataTable])) # amount of nutrient i in food j
```

In [207…
```
# great lp problem  to minimize the cholesterol
prob = LpProblem(name='Food optimization', sense=LpMinimize)
```

In [208…
```
# define the variables - food with lower limit of zero
foodVars = LpVariable.dicts("Foods", foods, 0)
```

In [209…
```
# create objective function
prob += lpSum([cost[f] * foodVars[f] for f in foods]), 'Total Cost'
```

In [210…
```
# add nutritional constraints
for i in range(0,numNutrients): # for loop running through each nutrient
    if (not np.isnan(minVal[0][i+1])) and (not np.isnan(maxVal[0][i+1])): # only write a constraint if upper and lower bounds exist
        print("adding constraint for " + nutrientNames[i+1])
        prob += lpSum([nutrients[i][j] * foodVars[j] for j in foods]) >= minVal[0][i+1], 'min nutrient ' + nutrientNames[i+1]
        prob += lpSum([nutrients[i][j] * foodVars[j] for j in foods]) <= maxVal[0][i+1], 'max nutrient ' + nutrientNames[i+1]
```

```
adding constraint for Protein
adding constraint for Carbohydrate, by difference
adding constraint for Energy
adding constraint for Water
adding constraint for Energy.1
adding constraint for Calcium, Ca
adding constraint for Iron, Fe
adding constraint for Magnesium, Mg
adding constraint for Phosphorus, P
adding constraint for Potassium, K
adding constraint for Sodium, Na
adding constraint for Zinc, Zn
adding constraint for Copper, Cu
adding constraint for Manganese, Mn
adding constraint for Selenium, Se
adding constraint for Vitamin A, RAE
adding constraint for Vitamin E (alpha-tocopherol)
adding constraint for Vitamin D
adding constraint for Vitamin C, total ascorbic acid
adding constraint for Thiamin
adding constraint for Riboflavin
adding constraint for Niacin
adding constraint for Pantothenic acid
adding constraint for Vitamin B-6
adding constraint for Folate, total
adding constraint for Vitamin B-12
adding constraint for Vitamin K (phylloquinone)
```

In [211…
```
# solve the optimization problem
prob.solve()
```

Out[211… 1

In [212…
```
# print output
print()
print("---------The solution to the diet problem is----------")
for var in prob.variables():
    if var.varValue > 0:
        print(str(var.varValue)+" units of "+str(var).replace('Foods_','') )
print()
print("Total cholesterol = %f" % value(prob.objective))
```

```
---------The solution to the diet problem is----------
0.059863415 units of Beans,_adzuki,_mature_seeds,_raw
0.069514608 units of Broccoli_raab,_raw
0.42866218 units of Cocoa_mix,_no_sugar_added,_powder
0.14694398 units of Egg,_white,_dried,_flakes,_glucose_reduced
0.73805891 units of Infant_formula,_MEAD_JOHNSON,_ENFAMIL,_NUTRAMIGEN,_with_iron,_p
0.4258564 units of Infant_formula,_NESTLE,_GOOD_START_ESSENTIALS__SOY,__with_iron,
0.050114149 units of Infant_formula,_ROSS,_ISOMIL,_with_iron,_powder,_not_reconstitu
0.15033656 units of Margarine_like_spread,_approximately_60%_fat,_tub,_soybean_(hyd
0.25918767 units of Mung_beans,_mature_seeds,_raw
0.18052856 units of Nuts,_mixed_nuts,_dry_roasted,_with_peanuts,_with_salt_added
1.184482 units of Oil,_vegetable,_sunflower,_linoleic,_(hydrogenated)
0.10375187 units of Seeds,_sunflower_seed_kernels,_dry_roasted,_with_salt_added
0.031866196 units of Snacks,_potato_chips,_fat_free,_made_with_olestra
0.070710308 units of Spices,_paprika
0.55106575 units of Tomatoes,_sun_dried
9999.6864 units of Water,_bottled,_non_carbonated,_CALISTOGA

Total cholesterol = 0.000000
```

Optimization Model 4

Maximization Problem: find the highest-protein diet

In [213…
```
# read in data
data = pd.read_excel("diet_folder/diet_large.xls", skiprows = 1, header = 0)
```

In [214…
```
# get food data
dataTable = data[0:7146] # rows 0:7146 (Excel calls them 2-7148; remember we skipped the blank first row in the read call) is the food data table
dataTable = dataTable.values.tolist() # Convert dataframe to list
```

In [215…
```
# get nutrient names
nutrientNames = list(data.columns.values) # column headers (nutrient names are in columns 3-13; Excel calls them D-N)
numNutrients = len(nutrientNames) - 1 # don't count the food-name column
```

In [216…
```
# blank elements are read as 'nan', so need to replace them with zero
for i in range(0,7146):
    for j in range(1,numNutrients):
        if np.isnan(dataTable[i][j]):
            dataTable[i][j] = 0
```

In [217…
```
# get min/max values of nutrients
minVal = data[7147:7148].values.tolist() # minimum nutrient values
maxVal = data[7149:7151].values.tolist() # maximum nutrient values
```

In [218…
```
# extract individual vectors of data
foods = [j[0] for j in dataTable] #list of food names

cost = dict([(j[0], float(j[nutrientNames.index('Protein')])) for j in dataTable]) # protein for each food

nutrients = []
for i in range(0,numNutrients): # for loop running through each nutrient
    nutrients.append(dict([(j[0], float(j[i+1])) for j in dataTable])) # amount of nutrient i in food j
```

In [219…
```
# create problem - a maximization of protein
prob = LpProblem(name='Food optimization', sense=LpMaximize)
```

In [220…
```
# define the variables
foodVars = LpVariable.dicts("Foods", foods, 0)
```

In [221…
```
# create objective function
prob += lpSum([cost[f] * foodVars[f] for f in foods]), 'Total Cost'
```

In [222…
```
# add nutritional constraints
for i in range(0,numNutrients): # for loop running through each nutrient
    if (not np.isnan(minVal[0][i+1])) and (not np.isnan(maxVal[0][i+1])): # only write a constraint if upper and lower bounds exist
        print("adding constraint for " + nutrientNames[i+1])
        prob += lpSum([nutrients[i][j] * foodVars[j] for j in foods]) >= minVal[0][i+1], 'min nutrient ' + nutrientNames[i+1]
        prob += lpSum([nutrients[i][j] * foodVars[j] for j in foods]) <= maxVal[0][i+1], 'max nutrient ' + nutrientNames[i+1]
```

```
adding constraint for Protein
adding constraint for Carbohydrate, by difference
adding constraint for Energy
adding constraint for Water
adding constraint for Energy.1
adding constraint for Calcium, Ca
adding constraint for Iron, Fe
adding constraint for Magnesium, Mg
adding constraint for Phosphorus, P
adding constraint for Potassium, K
adding constraint for Sodium, Na
adding constraint for Zinc, Zn
adding constraint for Copper, Cu
adding constraint for Manganese, Mn
adding constraint for Selenium, Se
adding constraint for Vitamin A, RAE
adding constraint for Vitamin E (alpha-tocopherol)
adding constraint for Vitamin D
adding constraint for Vitamin C, total ascorbic acid
adding constraint for Thiamin
adding constraint for Riboflavin
adding constraint for Niacin
adding constraint for Pantothenic acid
adding constraint for Vitamin B-6
adding constraint for Folate, total
adding constraint for Vitamin B-12
adding constraint for Vitamin K (phylloquinone)
```

In [223…
```
# solve the problem
prob.solve()
```

Out[223… 1

In [224…
```
# print output
print()
print("---------The solution to the diet problem is----------")
for var in prob.variables():
    if var.varValue > 0:
        print(str(var.varValue)+" units of "+str(var).replace('Foods_','') )
print()
print("Total protein = %f" % value(prob.objective))
```

```
---------The solution to the diet problem is----------
7.0117007 units of BANQUET_Salisbury_Steak_Meal,_Gravy_and_Salisbury_Steak_with_Ma
0.20365743 units of Cereals_ready_to_eat,_KASHI_Heart_to_Heart_by_KELLOGG
0.23412086 units of Collards,_raw
25.855235 units of Fish,_devilfish,_meat_(Alaska_Native)
31.46708 units of Fish,_lingcod,_meat,_raw_(Alaska_Native)
0.02 units of Fish_oil,_cod_liver
2.2140307 units of Gelatins,_dry_powder,_unsweetened
0.037489833 units of Mollusks,_oyster,_eastern,_canned
57.437865 units of Rhubarb,_wild,_leaves_(Alaska_Native)
621.79859 units of Sweeteners,_tabletop,_aspartame,_EQUAL,_packets
9.5089609 units of Tea,_brewed,_prepared_with_distilled_water
9552.2849 units of Water,_bottled,_non_carbonated,_CALISTOGA
276.5536 units of Water,_bottled,_non_carbonated,_DANNON
0.076732592 units of Whale,_beluga,_flipper,_raw_(Alaska_Native)
9.6405544 units of Whale,_beluga,_liver,_raw_(Alaska_Native)
1.7353546 units of Whale,_beluga,_meat,_air_dried,_raw_(Alaska_Native)

Total protein = 2994.899576
```